Linking Morphology and Reactivity: Growth and Ligand-Assisted Dissolution of Cobalt Oxyhydroxide

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Jason C. Myers

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Dr. R. Lee Penn, Advisor

August 2010

Acknowledgments

I would like to thank my advisor, Professor R. Lee Penn, for her constant support and especially for the opportunities she gave me to follow my interests along the way. I would also like thank past and present members of the Penn research group, especially Sara Isley, Anthony Ratkovich, and Kirsten Moore for being there to talk to, whether about research ideas or random nonsense, and for the occasional help with emergency sanity retention. I thank my parents for helping out in the rough times and for always caring. I would like to thank Michelle Driessen for helping me with all things teaching related and for the occasional lunch break when I desperately needed one. I also thank the people who served as my constant support system: Kevin and Kari, who were here from the beginning; Buck and Jen, who were always there even when they weren't; Burand, Rivard, Tommy, and Mandi, who occasionally did the dishes and always kept the ice tray full; and Beau, with the perpetual offer of a spot on the futon and a cold drink. Finally, I would like to thank Teresa for always supporting me, for reading over everything I couldn't stand to look at anymore, and for always being Teresa.

Abstract

Reactions at the interface of solid materials have a significant role in many fields of study, ranging from environmental science to industrial manufacturing. Identifying and quantifying the reactive surface area of these materials is vital to understanding the reactions in which they participate. The most basic effect of reactive surface area is governing the reaction rate at the surface but, in some cases, it is necessary to have a far more detailed understanding of the surface structure. Many reactions occur most efficiently, or even exclusively, at specific types of surface site. The ability to identify and measure these sites could dramatically improve the design of many applications, such as heterogeneous catalysts or waste remediation systems.

One proposed method of measuring reactive surface area is the use of carefully selected probe molecules that are specifically reactive with the surface sites of interest. This work focuses on the development of a method for analyzing the surface characteristics of heterogenite (β-CoOOH) using the ligand iminodiacetic acid (IDA) as a probe. To investigate this system, first a range of model materials were necessary. The method of heterogenite synthesis was explored, revealing that a surprising amount of control can be exerted over the final particle morphology by altering simple factors such as reaction temperature or choice of oxidizing agent.

The ligand-assisted dissolution of heterogenite by IDA produces a mixture of *s-fac* and *u-fac* isomers of $Co(IDA)_2^-$, and the relative amount of each isomer depends upon the surface characteristics of the heterogenite. When heterogenite particles were aged in suspension at room temperature, a rapid evolution of the number and type of surface site present was observed. This change was tracked by reacting the particles with

IDA then separating and quantifying the resulting $Co(IDA)_2^-$ isomers. Through this method, it was found that the surface evolution occurs more slowly when aged in lower pH buffer. The connection between particle morphology and reactivity was strengthened when a link was found between the height of cylindrical heterogenite plates and the ratio of isomers formed during the dissolution reaction. From this, an empirical relationship between particle height and the relative amount of *s-fac* isomer was derived. This relationship allowed the tracking of particle growth via dissolution reactions rather than direct measurements.

The final connection between morphology and reactivity was discovered when kinetic and thermodynamic studies were undertaken. The rate of reaction when the reactant concentrations are altered suggests that both surface diffusion and product desorption processes are involved in determining the overall reaction rate. Finally, it was hypothesized that the reactivity of some sites varies with temperature. Thus, the ratio of products produced depends not only on the number and type of surface site present, but also on the temperature of the reaction. Additional work is necessary to quantify this temperature dependence.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ads	adsorbed to a surface
aq	aqueous
a.u.	arbitrary units
BET	Brunauer, Emmett, Teller gas adsorption model
BGE	background electrolyte
CCD	charge coupled device
CE	capillary electrophoresis
DTO	delay to oxidation
EDTA	ethylenediaminetetraacetic acid
EDX	energy-dispersive X-ray spectroscopy
EOF	electroosmotic flow
HDPE	high-density polyethylene
ICDD	International Centre for Diffraction Data
IDA	iminodiacetic acid
K_{ads}	equilibrium constant for adsorption
mer	meridional
mIDA	N-methyliminodiacetic acid
PDF	powder diffraction file
PTFE	polytetrafluoroethylene
RNG	(pseudo)random number generator
SA	surface area
s-fac	symmetrical facial

TEM	transmission electron microscopy
TTAB	tetradecyltrimethylammonium bromide
XRD	X-ray diffraction
u-fac	unsymmetrical facial

1

Introduction and Background

The major focus of this work is the use of probe molecules to investigate the surface reactivity of solid materials *in situ*. The dissolution reaction of one model system, the cobalt-containing mineral heterogenite and the chelating ligand iminodiacetic acid, are investigated to determine how changes to the morphology and structure of the solid particles affect reactions at their surfaces. The remainder of this chapter will detail some of the more important background factors that motivate the project itself, the selection of model materials, and the methods used to analyze the results.

1.1 Reactive Surface Area

Reactions at solid surfaces play a huge role in the chemistry of natural, biological, and manufactured systems,[1] and the identification and measurement of reactive surface area is a vital aspect of understanding these reactions. At its simplest, the available surface area of a solid governs the rate of reactions with that solid; more surface area translates into more potential sites for reaction. This can be seen in biological systems such as mitochondria,[2] chloroplasts,[3] and intestines,[4] where membranes are folded to provide increased surface area in a limited volume. This is also a key reason for the recent interest in nanoparticle reactivity: their small size means that nanoparticles have a surface area to volume ratio that is substantially higher than that of bulk materials,

making them vastly more reactive. The surface of most materials, however, is not homogeneous, and many reactions take place only at specific surface sites.

Many factors can affect the reactivity of a solid surface. This is especially true for nanoparticle materials, where the large surface area to volume ratio makes surface reactivity all the more important. The efficiency of many catalysts, for example, has been shown to depend strongly on both the particular polymorph[5-7] and the morphology[8,9] of the catalyst particles. Particles with different shapes may exhibit reactivity differences because of changes to the types of crystallographic faces exposed. Even small size changes, especially at the nanoparticle scale, can modify the surface characteristics by changing the relative amounts of each surface type exposed (Figure 1). Changes in size may also increase the number of step, terrace, or kink sites present.[10] Understanding how morphology impacts a given reaction is essential to such vital undertakings as catalyst design[11-13] and environmental waste remediation.[14,15]

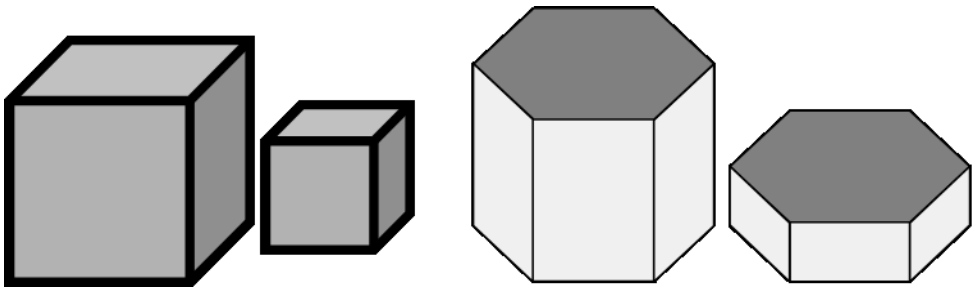

Figure 1.1 Illustration of some effects of minor size and shape changes. Left: reducing the size increases the SA/volume ratio, but also the increases the relative number of "corner" sites (solid black) at edges of the cube. Right: a change in aspect ratio has little effect on the amount of basal (dark) surface, but significantly increases the edge surface area.

Quantifying the reactive surface area in a given reaction is far from trivial. The surface sites that are important for one reaction may be completely different from those

involved in another. Furthermore, reactive surface area frequently scales erratically with total surface area.[16-18] Particularly when only specific sites are involved in a reaction, traditional methods of surface area determination, such as BET gas adsorption, are not well-suited for quantifying the reactivity of many solid materials.[16,19] Techniques such as electron and scanning-probe microscopy can be used to determine the geometric surface area of a particle, but this too has shortcomings. First, microscopy-based methods provide little information about surfaces that may be contained in pores in the material. Second, even for non-porous materials, geometric surface area is most useful if there is already an understanding of how specific surface types participate in reactions. These methods can be used very effectively to monitor the evolution of surfaces, but statistically significant results can require thousands of individual measurements. It is, therefore, important to explore new methods of measuring the type and number of relevant sites in a reaction system. One possible method is the use of probe molecules that are sensitive to specific types of reactive sites. In this way, surface reactivity could be quantified by reacting these probe molecules with the material and using commonly available analytical chemistry methods to evaluate the reaction products. Chelating ligands are one class of molecules that show promise as probes for surface reactivity.

1.2 Ligand-Assisted Dissolution

Ligand-assisted dissolution is the process by which solid materials, typically metal (hydr)oxides, are dissolved through reaction with a complexing agent. The process plays an important part in the weathering of natural materials as well as the transport of

contaminant materials through environmental systems. The introduction of ligands, typically as an acid anion, into natural systems can provide a significant increase in the dissolution rate of solid materials through the formation of stable coordination complexes.[20,21] This can result in the rapid liberation of waste materials that would otherwise remain adsorbed to mineral surfaces. This can be a particularly dangerous issue in the case of radioactive waste, as chelating agents such as EDTA were frequently used in the cleaning of reactor components.[22] In one incident, the simultaneous release of EDTA and metal cations resulted in the transport of radioactive cobalt-60 several kilometers away from the initial disposal pit.[21]

The process of ligand-assisted dissolution is believed to begin via initial adsorption of the ligand to the material, where it displaces surface hydroxyl groups and weakens metal-oxygen bonds. The coordinated metal center is then released into the solution.[23-25] Because of the inner sphere nature of adsorption to the surface, the early stages of this dissolution are believed to be sensitive to the coordination environment of the surface metal centers. Because of this sensitivity, chelating ligands are good candidates for probing the reactivity of metal oxides surfaces.

1.3 Cobalt: Applications and Coordination Complexes

The work presented here focuses on the factors affecting the growth and reactivity of heterogenite (β-CoOOH), and related cobalt hydroxides. Cobalt ores are common in nature, but widely dispersed, with a Co concentration of ~20 mg/kg in the earth's crust.[26] CoOOH itself is frequently found with copper oxide ores and so is often mined with

copper; the Co is then reduced, and separated via solvent extraction.[27] Most cobalt is produced as a by-product of copper and nickel mining operations in central Africa.[28] Nevertheless, Co has several important uses: Co has been used as a blue coloring agent in glass and ceramics for centuries, radioactive Co^{60} is used as a gamma ray source for applications such as surgical gamma knives[29] and Mössbauer spectroscopy, and Co alloys are used in applications where superior corrosion-[30] or heat-resistance[31] is necessary. There has been a recent interest in nanoparticles of the cobalt oxides due to their beneficial properties when included in nickel-metal hydride batteries, including enhanced capacity and longer cycling lifetimes.[32-34] Although cobalt is added to the battery electrodes in a variety of forms, it is subsequently converted into CoOOH during charging.[35] The microstructure of the CoOOH produced is also of special interest: samples with higher defect concentrations are believed to offer further improvements to the electrodes.[33,34]

The coordination complexes of cobalt, especially Co^{3+}, have played an important role in the history of chemistry. Octahedral Co^{3+} complexes typically feature a low-spin d^6 electron configuration, making them kinetically inert towards ligand exchange. For this reason, these complexes played an important role in Werner's pioneering study of coordination chemistry.[36] This same stability also makes reaction of cobalt(III) materials with coordinating ligands a near ideal model system for attempting to probe reactive surface area. A slow exchange rate means that there is ample time for analysis of reaction products without fear of decomposition or equilibration. Likewise, any

information contained in the geometric arrangement of the ligands will not be lost due to rearrangement.

1.4 Particle Growth

Differences in reactivity are, in many cases, a result of the growth history of the particles themselves. The number and types of defect present in a material may vary considerably based on the mechanisms that contributed to its growth. There are two primary mechanisms of nanoparticle growth in suspension: coarsening (or Ostwald ripening) and aggregation. Coarsening is a process by which small particles get smaller and large particles get larger (Figure 1.2, top). The driving force for coarsening is the difference in energy between surface molecules and those in the interior of a particle. This makes smaller particles, with their higher surface area to volume ratio, thermodynamically unfavorable compared to larger particles. Coarsening growth occurs with a constant time dependence and can be described by the equation[37]

$$D_t = D_0 + k_c t^{1/n} \tag{1.1}$$

where D_t is the particle diameter at time t, D_0 is the diameter at time zero, k_c is the rate constant for growth, and n is an integer parameter that depends on the limiting factor for coarsening growth in the particular system. In cases where coarsening is the primary growth mechanism, crystal structure defects are relatively unlikely.

Aggregation is the attachment of primary particles to form larger secondary particles (Figure 1.2, bottom). This typically results in polycrystalline clusters of particles in random orientations. Oriented aggregation is a special case of aggregation, in

which the primary particles are crystallographically aligned at the particle interface.[38] The primary particles in oriented aggregates may remain spatially separated (mesocrystals) or become a monolithic crystal. Growth by oriented aggregation, when it occurs, tends to dominate the early phases of crystal growth. An initial rapid increase in volume is observed,[39] but the rate of aggregation slows as the primary particles are depleted. Systems where growth occurs primarily by oriented aggregation typically show the presence of defects and dislocations in the crystal lattice.

By varying the relative contribution of these mechanisms to particle growth, a variety of different particle morphologies can be generated as model materials. Comparing the reactivity of different morphologies with the same probe ligand will allow empirical relationships between the material structure and reactivity.

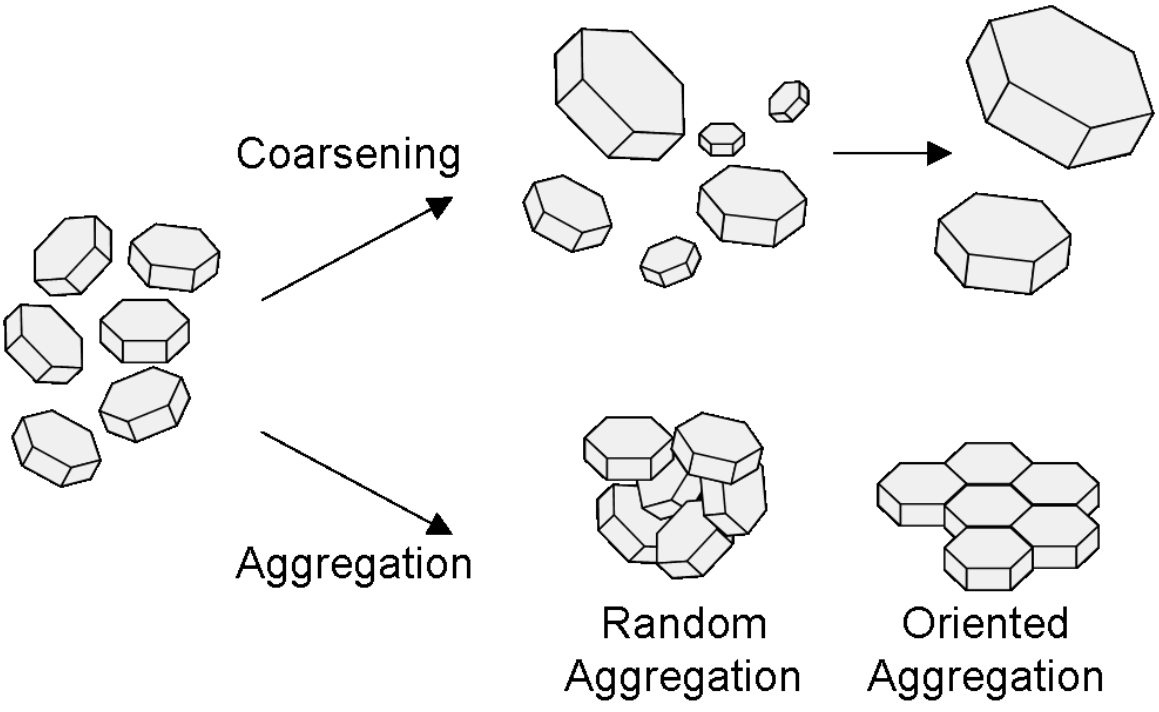

Figure 1.2 Representation of nanoparticle growth mechanisms. Top: growth by coarsening. Bottom: growth by random (left) or oriented (right) aggregation.

1.5 Capillary Electrophoresis

To accurately measure the results of a dissolution reaction, analytical methods must be employed to separate and quantify reaction products. In the case of dissolved Co complexes, capillary electrophoresis (CE) is an excellent tool. CE is a separation technique that functions by applying an electric potential across a buffer-filled capillary. The resulting electric field draws ions towards the anode or cathode at a rate that depends upon the both the charge and the hydrodynamic radius of the species,[40-43] allowing for the separation of analytes based upon the ratio of charge to solvated size. The separation takes place within a thin (typically 20-120 μm inner diameter) capillary,[43] which also serves as the detection cell.

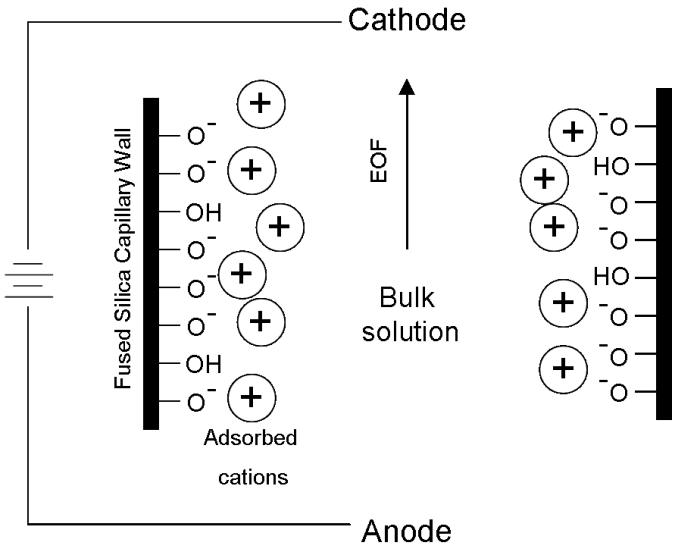

Figure 1.3 Schematic of the electric double layer and resulting electroosmotic flow (EOF) during a capillary electrophoresis separation.

One of the most significant parts of a CE separation is the electroosmotic flow (EOF), a net movement of the bulk solution towards one end of the electric field.[44]

Control over this flow is essential in controlling the retention time of an analyte in a separation procedure. EOF is a caused by the presence of deprotonated silanol groups on the interior of a silica capillary, which impart a negative charge to the capillary wall. This attracts the cationic components of the BGE and forms an electric double layer. These cations are attracted towards the cathode when an electric field is applied and, because their hydration spheres contact the bulk solution, this results in a movement of the entire BGE solution.

Reversal of the EOF is vital for successful separation of negatively charged analytes. When the polarity of the electric field is reversed, placing the anode on the detector side of the capillary, anions will be drawn towards the detector but the EOF still acts towards the cathode. This can result in excessively long separation times and peak broadening, causing a loss in resolution.[45] To combat this effect, small amounts of cationic surfactant can be added to the BGE mixture.[46,47] The surfactants, typically ammonium salts, form bilayers or micelles along the capillary wall,[47,48] effectively replacing the negatively charged silica surface with a positively charged one and reversing the EOF direction. The resolution and speed of a CE separation can be dramatically improved by adjusting EOF direction and rate to maximize the difference in analyte mobility; indeed, with sufficient optimization, even isotopic separation can be attained via differences in diffusion rate.[49-51]

1.6 Scope of the Dissertation

The remainder of this work is organized into six additional chapters and an appendix. Chapter 2 describes the synthesis of heterogenite nanoparticles for use as model materials. Characterization of these materials revealed that the shape and levels of disorder vary widely with differences in synthetic parameters such as reaction temperature and the type of oxidizing agent employed. Chapter 3 details the first investigation into the evolution of heterogenite surface reactivity as a function of aging time. Aging is performed under four different buffering conditions, and the reactivity of the particles is monitored via ligand-assisted dissolution. Chapters 4 and 5 investigate the mechanistic details of the reaction that takes place between the chelating ligand iminodiacetic acid (IDA) and the heterogenite surface. Chapter 4 focuses on the identification of two specific reactive sites on the surface and how the coordinative environment at each site influences the reaction products. Chapter 5 uses adsorption, kinetic, and thermodynamic data to further clarify the nature of the reaction between IDA and the heterogenite surface. Chapter 6 provides concluding remarks regarding the heterogenite-IDA system and suggestions for the continuation of the research presented herein. Chapter 7 describes the details of a simulation program designed to model particle growth by random and oriented aggregation, and appendix A contains the current source code of the simulation program.

1.7 References

1. Stumm, W. *Chemistry of the Solid-Water Interface*; John Wiley & Sons, Inc.: New York, 1992.

2. Scheffler, I.E. *Mitochondria*; John Wiley and Sons, 1999.

3. Booth, G.; McDuell, G.R.; Sears, J. *World of science*; Kogan Page Publishers, 1999.

4. Kent, M. *Advanced biology*; Oxford University Press US, 2000.

5. Jones, A.; Rule, J.; Moore, J.; White, S.; Sottos, N., "Catalyst morphology and dissolution kinetics of self-healing polymers." *Chem. Mater* **2006**, *18*, 1312.

6. Gao, L.; Zhang, Q., "Effects of amorphous contents and particle size on the photocatalytic properties of TiO2 nanoparticles." *Scripta materialia* **2001**, *44*, 1195.

7. Isley, S.L.; Anderson, E.R.; Penn, R.L., "Influence of Ionic Strength on Brookite Content in Sol-Gel Synthesized Titania before and after Hydrothermal Aging." *Electrochemical Society Transactions* **2006**, *3*, 37.

8. Piccolo, L.; Henry, C.R., "NO-CO Reaction Kinetics on Pd/MgO Model Catalysts: Morphology and Support Effects." *Journal of Molecular Catalysis A: Chemical* **2001**, *167*, 181.

9. Jenkins, D.J.; Alabdulrahman, A.M.S.; Attard, G.A.; Griffin, K.G.; Johnston, P.; Wells, P.B., "Enantioselectivity and catalyst morphology: step and terrace site contributions to rate and enantiomeric excess in Pt-catalysed ethyl pyruvate hydrogenation." *Journal of Catalysis* **2005**, *234*, 230.

10. Vollmer, S.; Witte, G.; Wöll, C., "Determination of site specific adsorption energies of CO on copper." *Catalysis Letters* **2001**, *77*, 97.

11. Roeffaers, M.B.J.; Sels, B.F.; Uji-i, H.; De Schryver, F.C.; Jacobs, P.A.; De Vos, D.E.; Hofkens, J., "Spatially Resolved Observations of Crystal-Face-Dependent Catalysis by Single Turnover Counting." *Nature* **2006**, *439*, 572.

12. Freund, H.-J.; Libuda, J.; Bäumer, M.; Risse, T.; Carlsson, A., "Cluster, facets, and edges: Site-dependent selective chemistry on model catalysts." *The Chemical Record* **2003**, *3*, 181.

13. Arslan, I.; Walmsley, J.C.; Rytter, E.; Bergene, E.; Midgley, P.A., "Toward Three-Dimensional Nanoengineering of Heterogeneous Catalysts." *Journal of the American Chemical Society* **2008**, *130*, 5716.

14. Chun, C.L.; Penn, R.L.; Arnold, W.A., "Kinetic and Microscopic Studies of Reductive Transformations of Organic Contaminants on Goethite." *Environmental Science & Technology* **2006**, *40*, 3299.

15. Handler, R.; Beard, B.; Johnson, C.; Scherer, M., "Atom Exchange between Aqueous Fe (II) and Goethite: An Fe Isotope Tracer Study." *Environmental Science & Technology* **2009**, *43*, 1102.

16. White, A.; Peterson, M. Chemical Modeling of Aqueous Systems II. In *ACS Symposium Series 416*; American Chemical Society: Washington DC., 1990; pp 461.

17. Van Cappellen, P., "Reactive Surface Area Control of the Dissolution Kinetics of Biogenic Silica in Deep-Sea Sediments." *Chemical Geology* **1996**, *132*, 125.

18. Holdren, G.R.; Speyer, P.M., "Reaction rate-surface area relationships during the early stages of weathering. II. Data on eight additional feldspars." *Geochimica et Cosmochimica Acta* **1987**, *51*, 2311.

19. Metz, V.; Raanan, H.; Pieper, H.; Bosbach, D.; Ganor, J., "Towards the establishment of a reliable proxy for the reactive surface area of smectite." *Geochimica et Cosmochimica Acta* **2005**, *69*, 2581.

20. Bolt, G.H. *Interactions at the Soil Colloid-soil solution interface*; Kluwer Academic Publishers: Norwell, MA, 1991.

21. Means, J.L.; Crerar, D.A.; Duguid, J.O., "Migration of radioactive wastes: radionuclide mobilization by complexing agents." *Science* **1978**, *200*, 1477.

22. Mansilla, H.D.; Bravo, C.; Ferreyra, R.; Litter, M.I.; Jardim, W.F.; Lizama, C.; Freer, J.; Fernández, J., "Photocatalytic EDTA degradation on suspended and immobilized TiO2." *Journal of Photochemistry and Photobiology A: Chemistry* **2006**, *181*, 188.

23. Wieland, E.; Wehrli, B.; Stumm, W., "The coordination chemistry of weathering: III. A generalization on the dissolution rates of minerals." *Geochimica et Cosmochimica Acta* **1988**, *52*, 1969.

24. Furrer, G.; Stumm, W., "The coordination chemistry of weathering: I. Dissolution kinetics of alumina and beryllium oxide." *Geochimica et Cosmochimica Acta* **1986**, *50*, 1847.

25. Zinder, B.; Furrer, G.; Stumm, W., "The coordination chemistry of weathering: II. Dissolution of Fe (III) oxides." *Geochimica et Cosmochimica Acta* **1986**, *50*, 1861.

26. Metallic Cobalt Particles (with or without Tungsten Carbide). In *IARC Monographs on the Evaluation of Carcinogenic Risks to Humans*; IARC: Lyons, France, 2006; Vol. 86.

27. Swartz, B.; Donegan, S.; Amos, S.; Reolon, P. "Processing considerations for cobalt recovery from Congolese copperbelt ores"; Hydrometallurgy Conference, 2009, The Southern African Institute of Mining and Metallurgy.

28. Shedd, K.B. Cobalt. In *2006 Minerals Yearbook*; U.S. Geological Survey, 2006.

29. de Lunsford, D.; Flickinger, J.; Lindner, G.; Maitz, A., "Stereotactic radiosurgery of the brain using the first United States 201 cobalt-60 source gamma knife." *Neurosurgery* **1989**, *24*, 151.

30. Antony, K. "Wear-resistant cobalt-base alloys"; International Conference on Cobalt: Metallurgy and Uses,, 1981.

31. Crook, P., "Cobalt and cobalt alloys." *ASM International, Metals Handbook, Tenth Edition* **1990**.

32. Armstrong, R.D.; Briggs, G.W.D.; Charles, E.A., "Some effects of the addition of cobalt to the nickel hydroxide electrode." *J. Appl. Electrochem.* **1988**, *18*, 215.

33. Hu, W.K.; Gao, X.P.; Geng, M.M.; Gong, Z.X.; Noreus, D., "Synthesis of CoOOH Nanorods and Application as Coating Materials of Nickel Hydroxide for High Temperature Ni-MH Cells." *J. Phys. Chem. B Lett.* **2005**, *109*, 5392.

34. Pralong, V.; Delahaye-Vidal, A.; Beaudoin, B.; Leriche, J.-B.; Tarascon, J.-M., "Electrochemical behavior of cobalt hydroxide used as additive in the nickel hydroxide electrode." *J. Electrochem. Soc.* **2000**, *147*, 1306.

35. Yuan, A.; Cheng, S.; Zhang, J.; Cao, C., "The influence of calcium compounds on the behaviour of the nickel electrode." *Journal of Power Sources* **1998**, *76*, 36.

36. Housecroft, C.E.; Sharpe, A.G. *Inorganic Chemistry*; Prentice Hall, 2001.

37. Banfield, J.F.; Zhang, H., "Nanocrystals in the Environment." *Reviews in Mineralogy and Geochemistry* **2001**, *44*, 1.

38. Penn, R.L., "Kinetics of Oriented Aggregation." *Journal of PhysicalChemistry B* **2004**, *108*, 12707.

39. Huang, F.; Zhang, H.; Banfield, J.F., "The Role of Oriented Attachment Crystal Growth in Hydrothermal Coarsening of Nanocrystalline ZnS." *Journal of PhysicalChemistry B* **2003**, *107*, 10470.

40. Baker, D.R. *Capillary Electrophoresis*; Wiley-Interscience: New York, 1995.

41. St. Claire, R.L., "Capillary electrophoresis." *Analytical Chemistry* **1996**, *68*, 569.

42. Monning, C.A.; Kennedy, R.T., "Capillary Electrophoresis." *Analytical Chemistry* **1996**, 280R.

43. Chankvetadze, B. *Capillary electrophoresis in chiral analysis*; Wiley, 1997.

44. Helmholtz, H.Z., "About Electrical Interfaces." *Annal. Phys. Chem.* **1879**, *7*, 337.

45. Landers, J.P. *Handbook of capillary electrophoresis*; CRC Press, 1997.

46. Tsuda, T., "Modification of Electroosmotic Flow with Cetyltrimethylammonium Bromide in Capillary Zone Electrophoresis." *J. high resolut. chromatogr.* **1987**, *10*, 622.

47. Melanson, J.E.; Baryla, N.E.; Lucy, C.A., "Dynamic capillary coatings for electroosmotic flow control in capillary electrophoresis." *Trends in analytical chemistry* **2001**, *20*, 365.

48. Lucy, C.A.; Underhill, R.S., "Characterization of the Cationic Surfactant Induced Reversal of Electroosmotic Flow in Capillary Electrophoresis." *Anal. Chem.* **1996**, *68*, 300.

49. Yeung, K.K.-C.; Lucy, C.A., "Improved resolution of inorganic anions in capillary electrophoresis by modification of the reversed electroosmotic flow and the anion mobility with mixed surfactants." *J. Chromatogr. A* **1998**, *804*, 319.

50. Terabe, S.; Yashima, T.; Tanaka, N.; Araki, M., "Separation of oxygen isotopic benzoic acids by capillary zone electrophoresis based on isotope effects on the dissociation of the carboxyl group." *Analytical Chemistry* **1988**, *60*, 1673.

51. Lucy, C.A.; McDonald, T.L., "Separation of Chloride Isotopes by Capillary Electrophoresis Based on the Isotope Effect on Ion Mobility." *Analytical Chemistry* **1995**, *67*, 1074.

2

Controlling Heterogenite Particle Morphology by Varying Synthetic Conditions*

Control over the morphology and microstructure of nanoparticles is a longstanding goal in many facets of materials research. This chapter studies how modifications to the synthetic preparation method of heterogenite particles alter the characteristics of the resulting product. The particles are prepared by first precipitating cobalt (II) hydroxide, which is subsequently oxidized to form heterogenite. This procedure is carried out at multiple temperatures, and the precipitation is completed at three different rates. The oxidation is accomplished by one of two oxidizing agents: H_2O_2 or NaOCl. Changes to the temperature of the synthesis are found to control the size and identity of the precursor cobalt (II) hydroxide particles generated during the precipitation step. The choice of oxidizing agent controls the transformation from precursor to final product. Characterization demonstrates that oxidization by H_2O_2 results in heterogenite crystallites that are substantially smaller than the precursor $Co(OH)_2$ particles, preserving very little of the original morphology and structure. Oxidization by NaOCl, however, transforms the $Co(OH)_2$ while leaving intact features such as the shape, size, and some aspects of the microstructure of the particles.

*A report on this research project was submitted for publication and is reproduced with permission from Myers, J.C. and Penn, R.L. *Materials Research Bulletin*, **2010**.

2.1 Introduction

The shape, size, and microstructure of nanoparticles are known to influence many of their properties, such as reactivity,[1,2] optical properties,[3,4] magnetic characteristics,[5-7] and catalytic activity.[8-10] For this reason, attaining synthetic control over these particle properties is a longstanding goal of many areas of materials research. Solar cell applications, for example, can see considerable gains in efficiency as a result of relatively small modifications to the synthetic method.[11-13]

Heterogenite (β-CoOOH) has recently become the focus of increased attention due to its use in two applications: as an additive for improving performance in rechargeable battery electrodes[14,15] and as a precursor to Co_3O_4 nanomaterials for use as catalysts in applications such as total combustion or oxygen evolution reactions.[16-18] When heterogenite is co-precipitated with nickel in an electrode of nickel-metal hydride batteries, many of the charging properties of the device are improved. Heterogenite particles with lower crystallinity may actually be preferable for this application because the increase in disorder results in improved formation of non-stoichiometric Co^{4+} containing phases,[19] which potentially increase oxygen transport to further enhance device performance. Both the reaction rate and the selectivity of Co_3O_4 catalysis has been shown to depend on the presence of specific surface facets, so preparation of shapes that maximize the reactive surface area could significantly improve the catalysts.[16,17] The conversion of β-CoOOH to Co_3O_4 is believed to occur topotactically,[18,20] so controlling the morphology of heterogenite could lead to control over Co_3O_4 morphology.

This chapter presents the results of varying the synthetic method used to produce heterogenite nanoparticles. The effects of synthesis temperature, precipitation rate, and oxidation methods—both their individual effects and how their interrelations can be used to control the morphology and the disorder of the resulting particles—are examined. The heterogenite particles are characterized using transmission electron microscopy and powder X-ray diffraction, with a focus on differences in morphology, crystallinity, and homogeneity.

2.2 Experimental Methods

All solutions were prepared using Milli-Q purified water (Millipore Corporation, 18 MΩ•cm resistivity). Glassware was washed in 4 M nitric acid (ACS grade, Mallinckrodt Laboratory Chemicals) before use; it was then rinsed repeatedly with distilled water followed by Milli-Q water. Purchased chemicals were analytical reagent grade and used without further purification.

2.2.1 Preparation of Heterogenite Particles

The methods employed in synthesizing the heterogenite samples were modifications of several methods previously used by Stone and co-workers.[21-23] For each synthesis, 1.0 L of Milli-Q H_2O was purged with nitrogen for ~1 h. From this, 500 mL of 3.2 mM Co^{2+} solution (cobalt (II) chloride hexahydrate, Mallinckrodt) and 500 mL of 6.4 mM OH^- solution (sodium hydroxide, Mallinckrodt) were prepared. The pair of solutions was either cooled to approximately 5 °C in an ice bath (*cold synthesis*) or heated to 60 °C using a recirculating hot water bath (*hot synthesis*). A PTFE-coated stir

bar was added to the Co^{2+} solution and the solution was magnetically stirred continuously and maintained in the constant temperature bath while the OH^- solution was added via one of three methods: (1) peristaltic pump at a rate of 8 mL/min (*slow addition*); (2) peristaltic pump at a rate of 17 mL/min (*medium addition*); (3) direct pouring over approximately 2 min (250 mL/min, *fast addition*).

The resulting mixture was pale blue in color for the cold synthesis and light pink-brown for the hot synthesis. The suspension was oxidized using either 10.0 mL of 5 % sodium hypochlorite solution (LabChem, Inc.) or 10.0 mL of 2.4 % hydrogen peroxide solution (prepared using 30 % H_2O_2, Mallinckrodt). The delay between precipitation and oxidation was also varied. In zero-delay syntheses, the oxidizing agent was added to the OH^- solution before it was combined with the Co^{2+}. Otherwise, the particle suspension was stirred in the temperature bath for either 15 or 90 min, after which the oxidizing agent was added dropwise over 3 min. In all cases, the result was a dark brown suspension. This suspension remained in the temperature bath for 2 h after oxidation and was then allowed to return to room temperature. Approximately 18 h after oxidation, samples were prepared for characterization by powder X-ray diffraction and transmission electron microscopy. Samples are referenced using the following notation: $T_{temperature}$ $OH_{addition\ rate}$ *Oxidant* (*Delay to Oxidation*). Table 2.1 summarizes the values for each reaction variable in the present study.

Table 2.1 Summary of Reaction Variables for the Synthesis of Heterogenite Particles

Temperature	Hydroxide Addition Rate	Oxidant	Delay to Oxidation
Cold: T = 5 °C	Slow: 8 mL / min (over ~1 h)	H_2O_2	simultaneous
	Medium: 17 mL / min (over ~0.5 h)		15 min
Hot: T = 60 °C	Fast: 250 mL / min (over ~2 min)	NaOCl	90 min

2.2.2 Materials Characterization

Powder X-ray diffraction (XRD) was used to identify the phases present in each sample, as well as their relative crystallinity and particle sizes. XRD samples were prepared by centrifuging 1 L of particle suspension in 300 mL batches at 14000 G for 10 min. At the conclusion of each centrifugation, the supernatant solution was decanted and the centrifuge tubes were refilled with additional suspension. The particles were washed by resuspending in Milli-Q water and centrifuging three times and were then air dried in shallow weigh boats over 2-4 days. The dried powders were ground with an agate mortar and pestle before being packed into a zero-background single-crystal quartz sample holder. The diffraction patterns were collected using a PANalytical X'Pert Pro MPD diffractometer equipped with a cobalt anode and an X'Celerator detector. The sample phases were confirmed by comparing peak positions with the ICDD PDF-2 database.

Select samples were examined by transmission electron microscopy (TEM). The previously dried samples were resuspended in Milli-Q water by aquasonication (VWR

Aquasonic model 150HT) for 5-15 min to break up particle aggregates. The suspension was then diluted as necessary, and a single drop of suspension deposited onto a 3 mm copper, holey carbon coated TEM grid (SPI Supplies). Each sample grid was examined using an FEI Tecnai T12 or FEI Tecnai F30 FEG microscope, and images were collected using a Gatan CCD camera and Gatan Digital Micrograph software (version 3.8.2).

2.3 Results

2.3.1 General Description of Heterogenite Morphology

The morphology of the resulting heterogenite particles varied considerably with alterations to the synthetic procedure, but several elements of the basic shape were consistent throughout the samples. Inspection of the TEM images of particles prepared by various synthetic methods shows that, in all cases, the particles are well described as flat plates. This plate morphology has been confirmed by imaging at multiple sample tilt positions. The plates typically lie flat on the sample grid; although they can occasionally be found on edge, the number of edge-on particles diminishes as their aspect ratio (diameter : height) increases. This preferred orientation of the plates can also be observed via XRD, where it is manifested as an increase in the relative intensity of the (003) reflection, located at ~23° 2θ. The (003) planes are parallel to the basal surface of the plates, so plates preferentially lying flat on the sample holder cause a significant increase in the intensity of the (003) peak relative to the other reflections. Although this effect can be somewhat mitigated by careful grinding and sample packing, samples with higher aspect ratios inevitably exhibit substantial preferred orientation effects.

The details of the particle morphology change significantly when the synthetic method is varied. The average diameters of the individual plates range from 5 nm to over 1 μm. The shape of the plates also varies: the basal faces can be hexagonal, nearly circular, or irregularly shaped.

2.3.2 Effect of Oxidizing Agent and Delay to Oxidation

The use of different oxidizing agents caused the most significant changes in the resulting heterogenite particles. In general, using hydrogen peroxide (H_2O_2) as the oxidizing agent produced small platelets, with diameters in the range of 5-40 nm, depending on the other synthetic variables. Conversely, using sodium hypochlorite (NaOCl) as the oxidizing agent produced a wide range of morphologies, from small 10 nm platelets to hexagonal plates over 2 μm in diameter, and offered the greatest control over size and shape in combination with the other reaction variables.

A noteworthy difference in materials produced with either oxidant was the purity of the resulting particles. Half of the syntheses employing H_2O_2 produced other phases in addition to heterogenite; unoxidized cobalt (II) hydroxide (α- and β-$Co(OH)_2$) and cobalt oxide (Co_3O_4) were identified in the diffraction patterns of these samples. These other phases were most prevalent when the base addition rate was slowed and were observed at both hot and cold temperatures (see section 2.3.3). Some samples synthesized using NaOCl also showed deviations from pure heterogenite in their XRD patterns; however, most of these could be attributed to crystallographic disorder rather than additional phases (see discussion of Figures 2.6 and 2.7). Only one pair of NaOCl-oxidized samples contained detectable amounts of another phase: $T_{60}OH_{slow}NaOCl$, which contained Co_3O_4

but in substantially lower amount than the H_2O_2-oxidized samples. A summary of the general oxidizing agent effects can be found in Table 2.2.

Table 2.2 Summary of oxidizing agent effects

	H_2O_2	**NaOCl**
Size	5-40 nm platelets	10 nm – 2 μm plates
Purity	Additional phases in all OH_{slow} and OH_{med} samples	Generally pure heterogenite*

***** minor Co_3O_4 impurity observed in $T_{60}OH_{slow}NaOCl$

Significant differences were observed for samples prepared by simultaneous base and oxidant addition as opposed to samples prepared by adding oxidant after a specified delay period. In contrast, no detectable differences were observed with the increase in the delay to oxidation (DTO). Two simultaneous addition samples were prepared: $T_{60}OH_{fast}NaOCl(0)$ and $T_{60}OH_{fast}H_2O_2(0)$. Both of these syntheses resulted in only amorphous material. It is, therefore, concluded that there is a minimum time required between base addition and oxidation to achieve a crystalline product. All DTO = 15 min and DTO = 90 min samples contained crystalline material, so we conclude that the required delay is less than 15 minutes, even at ice bath temperatures.

In general, the XRD patterns between samples with DTO = 15 min and DTO = 90 min overlap almost perfectly (Figure 2.1). Some small differences in relative peak intensity were observed, but TEM images demonstrated no significant differences as a function of delay time. Thus, it is concluded that the differences in intensity are due to

preferred orientation effects and that these effects are induced by slight differences in XRD sample preparation rather than by real changes in aspect ratio. Because samples with 15 min and 90 min DTO are indistinguishable, DTO is henceforth omitted from sample designations.

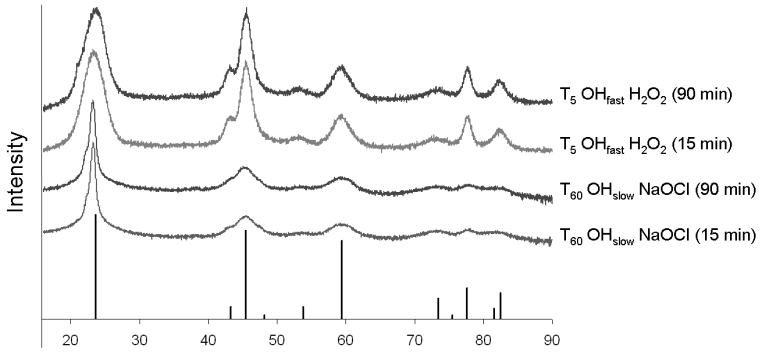

Figure 2.1 XRD patterns of heterogenite particles with different DTO. The patterns correspond to the syntheses (from top to bottom): $T_5OH_{fast}H_2O_2$(90 min), $T_5OH_{fast}H_2O_2$(15 min), $T_{60}OH_{slow}$NaOCl(90 min), $T_{60}OH_{slow}$NaOCl(15 min). The solid black lines are the peak positions for heterogenite (PDF #7-0169). Note: the bottom two patterns are plotted as square root of intensity to enhance visibility of lower intensity peaks.

2.3.3 Effect of Temperature and Base Addition Rate

Substantial differences in heterogenite size and morphology were observed with changing the rate of base addition. All T_{60} NaOCl samples (Figure 2.2) were hexagonal plates, but the size of the plates increased with decreasing rate of base addition. With the fastest addition rate (250 mL/min), the plates were sharply faceted and 100-200 nm in diameter. The medium OH^- addition rate (17 mL/min) resulted in two different types of heterogenite particle. Some of the hexagonal plates were sharply faceted and 0.5-1.5 μm across, similar in appearance to those obtained from the fast addition. The other type of

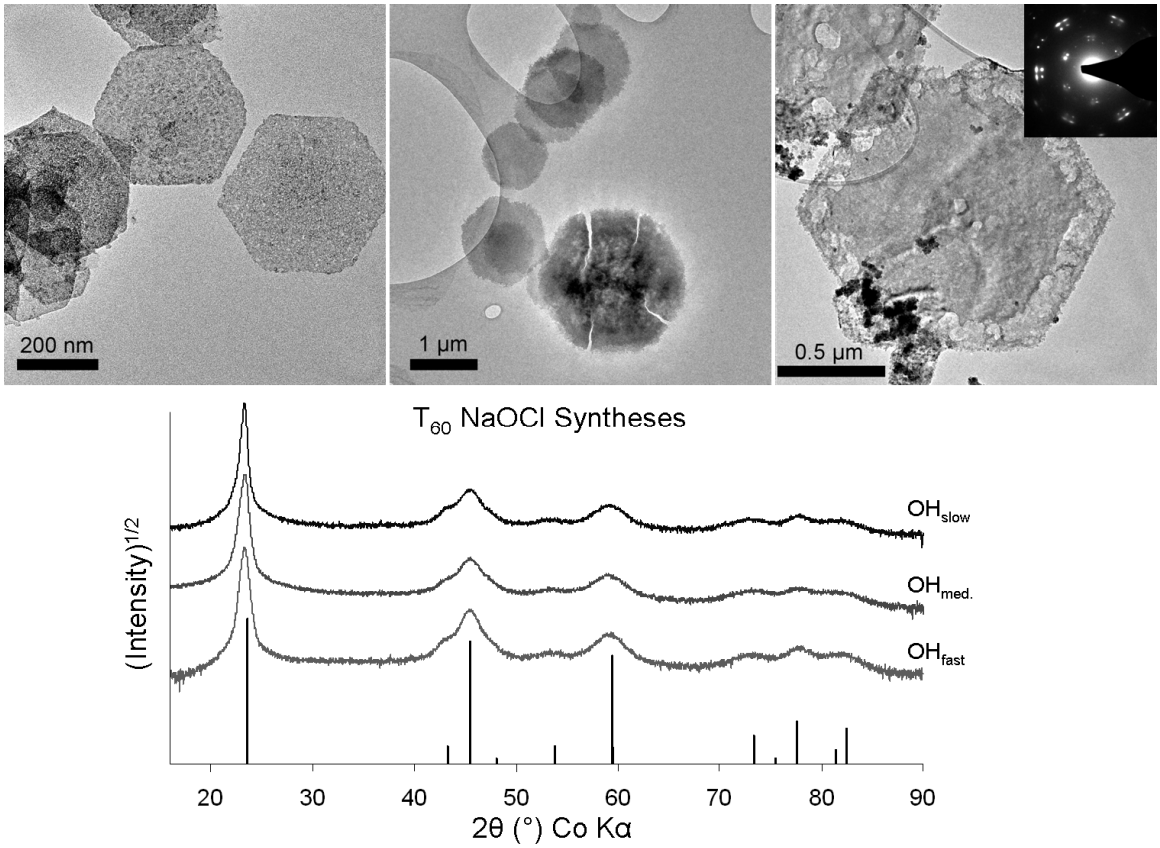

Figure 2.2 Top: TEM images of particles prepared by T_{60} NaOCl syntheses with fast (left), medium (center), and slow (right) base addition rates. The selected area electron diffraction pattern (inset, right) shows the hexagonal heterogenite pattern, with streaking in the (110) spots indicating slight crystallographic misalignment. Bottom: XRD patterns of these particles. The heterogenite peak positions are shown in the black stick pattern at the bottom.

plates observed were typically 2 μm or larger in diameter and exhibited features that appear to be cracks. These crack-like features were not observed in the smaller particles. Finally, when the slowest OH^- addition rate (8 mL/min) was employed, the resulting particles were large plates, 1-2 μm in size, but did not exhibit the crack-like features observed in $T_{60}OH_{med}NaOCl$. These plates appeared thinner along the edges and were very sharply faceted. Higher magnification images and electron diffraction patterns, which show streaking in the (110) spots, indicate that the hexagonal plates produced by

all three hydroxide addition rates are composed of smaller, building block crystallites with slight misorientations in the crystallographic *ab* plane. The similarity in peak breadths in the XRD patterns suggests that these primary crystallites are similarly sized regardless of addition rate. Only $T_{60}OH_{slow}NaOCl$ showed any sign of impurities: a very small amount of Co_3O_4 can be observed in the XRD pattern and small, dark particles are present in the TEM images.

When syntheses were performed at high temperature and oxidized by H_2O_2 (Figure 2.3), the base addition rate had two primary effects. The first effect is that slower base addition resulted in additional phases in the final product. No impurity phases were detected in $T_{60}OH_{fast}H_2O_2$ by XRD, but $T_{60}OH_{med}H_2O_2$ contained minor Co_3O_4 impurity. The $T_{60}OH_{slow}H_2O_2$ sample, however, showed significant amounts of impurities and evidence for incomplete oxidation. The XRD pattern could be identified as mostly β-$Co(OH)_2$, with substantial heterogenite and a small amount of Co_3O_4. Large hexagonal plates were seen infrequently in the TEM images, but selected area electron diffraction reveals that these plates are made up of β-$Co(OH)_2$ and Co_3O_4 and that heterogenite occurs as the smaller particles. Repeating this synthesis with up to 25 times the original H_2O_2 concentration did not increase the amount of heterogenite produced.

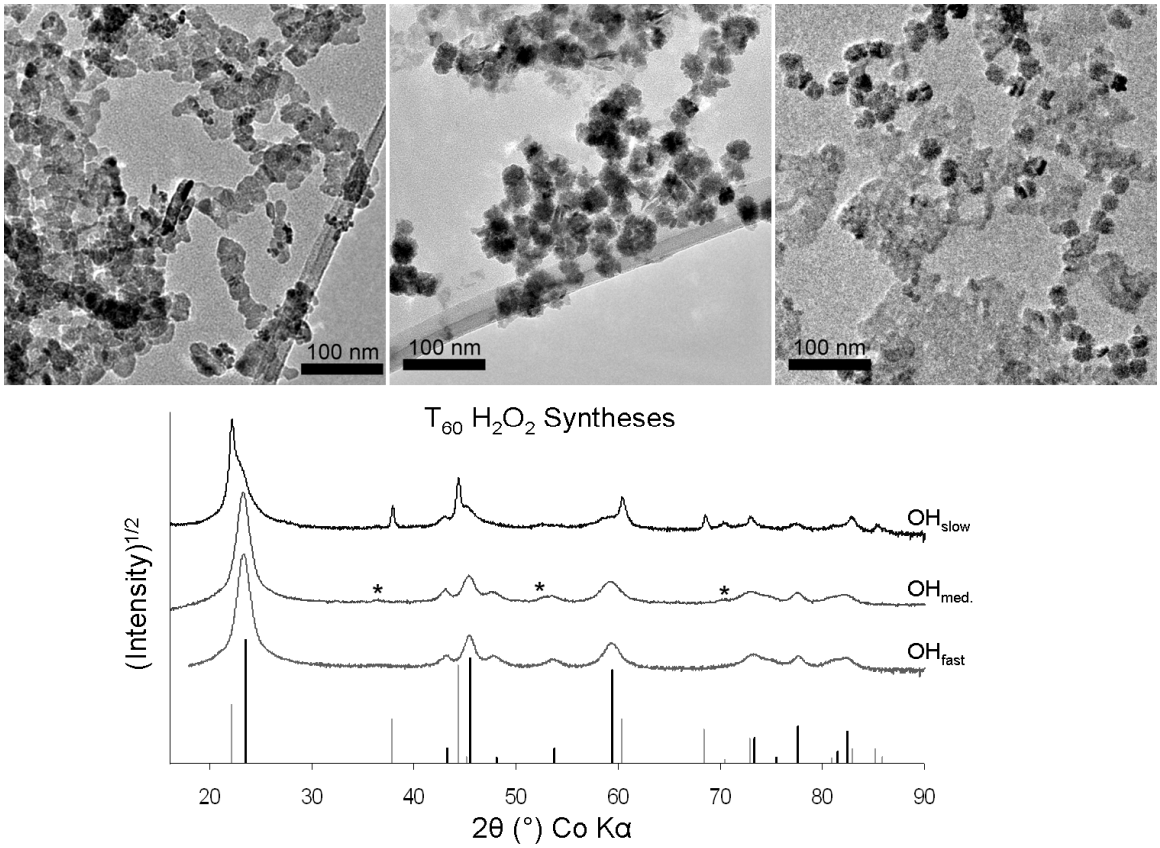

Figure 2.3 Top: TEM images of particles prepared by T_{60} H_2O_2 syntheses with fast (left), medium (center), and slow (right) base addition rates. Bottom: XRD patterns of these particles. The stick patterns at the bottom show the peak positions of heterogenite (black) and β-$Co(OH)_2$ (grey, PDF #30-0443). Peaks attributable to Co_3O_4 are marked with asterisks on the $T_{60}OH_{med}H_2O_2$ pattern.

The second effect of varying the base addition rate was a change in the frequency of secondary particles composed of orientated building-block crystallites. As the base addition rate decreased, the frequency of clusters and rings increased (Figure 2.4). Interestingly, there was no significant change in the size of the primary heterogenite crystallites as a function of base addition rate. Electron diffraction of the aggregates reveals that they are mixtures of heterogenite, β-$Co(OH)_2$, and Co_3O_4 primary particles, which assemble edge-to-edge with slight crystallographic misorientation. This effect was

also observed as an increase in preferred orientation in the XRD patterns. The full widths at half maximum of the heterogenite XRD peaks did not change for samples prepared using the various base addition rates, providing further support for the similarly sized heterogenite primary crystallites.

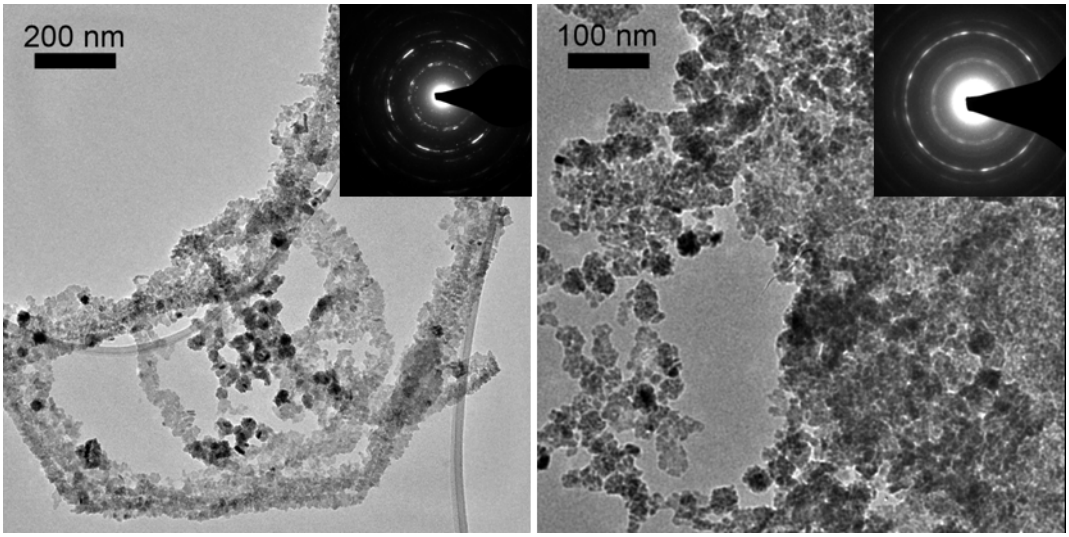

Figure 2.4 TEM images of rings (left) and clusters (right) observed in $T_{60}OH_{slow}H_2O_2$ samples. The selected area electron diffraction patterns (insets) display bright spots indicative of preferred orientation, but the rings and arcs show that there is not perfect crystallographic alignment.

In the low temperature syntheses, the effect of changing the base addition rate was less pronounced. Among the T_5 H_2O_2 samples (Figure 2.5), there was only a very slight difference observed between the XRD patterns for the fast and medium base addition rates: the presence of small peak shoulders at 26.5° and 40° 2θ. When the base was added at the slowest rate, however, these new peaks became the dominant features of the XRD pattern. No appropriate pattern matching these two peaks could be found in the PDF database; however, the observed reflections are very similar to a recently proposed

structure for Cl^- intercalated α-$Co(OH)_2$.[24] After the sample had been stored dry for over a year, a second XRD pattern was collected. In this later pattern, the peaks observed at 26.5° and 40° 2θ were no longer detected, and heterogenite and cobalt chloride hydroxide ($Co_2(OH)_3Cl$) were identified as the only phases present. In the TEM images, $T_5OH_{slow}H_2O_2$ appeared similar to the other T_5 H_2O_2 samples, except that some small (~5 nm), more electron dense objects were occasionally observed. An energy dispersive X-ray (EDX) spectrum collected from a region containing these objects confirmed the presence of chlorine, suggesting that these darker objects are the Cl^- intercalated phase.

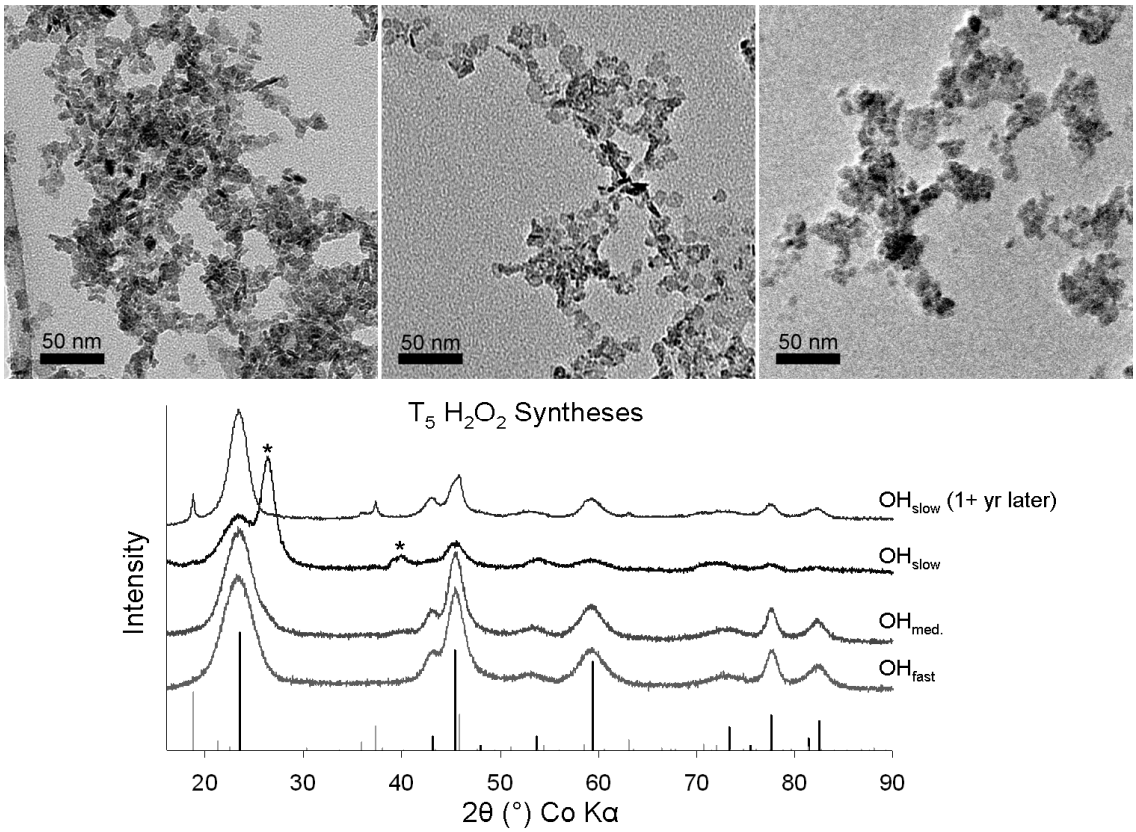

Figure 2.5 Top: TEM images of particles prepared byT_5 H_2O_2 syntheses with fast (left), medium (center), and slow (right) base addition rates. Bottom: XRD patterns of these particles. The topmost pattern is the OH_{slow} particles after dry storage for over a year, showing the transformation of the unknown phase. The stick patterns at the bottom show the peak positions of heterogenite (black) and $Co_2(OH)_3Cl$ (grey, PDF #1-073-2134). Peaks marked with an asterisk belong to an unknown phase similar to those observed in a Cl^- intercalated form of α-$Co(OH)_2$.[24]

The T_5 NaOCl samples (Figure 2.6) likewise displayed only small variations when the base addition rate was changed. In all cases, the XRD patterns showed substantial broadening, irregular peak shape, and a shift towards lower angle in the (003) peak (located at 23° 2θ), although these changes were less pronounced in $T_5OH_{fast}NaOCl$. These shape differences, along with the near extinction of the (015) peak (located at 59.5° 2θ) strongly suggest that stacking disorder is present along the *c* axis of the heterogenite. Variations in the inter-layer spacing of the heterogenite structure would disrupt the regularity of the (003) planes, causing the corresponding peak to become irregularly broadened and decrease in intensity. Likewise, the (015) reflection would also be expected to broaden and weaken with increased stacking disorder. The TEM images appeared similar for all three samples, consisting mostly of irregularly-shaped platelets averaging 10-30 nm across. Some of the platelets were found in a side-on orientation, and examination of the lattice fringes observed in these images revealed that the spacing of the (003) planes varied between different plates and sometimes even within an individual plate. Fringe spacings as small as 4.4 Å and as large as 6.5 Å were observed in the limited number of side-on plates present (Figure 2.7). This disorder could easily explain the peak broadening and intensity decrease observed in the XRD patterns, and the increase in average layer spacing also explains the shift of the (003) peak towards lower angle. This variation is observed in all three samples but is less pronounced in $T_5OH_{fast}NaOCl$. The particles obtained from each synthetic variation are summarized in Table 2.3.

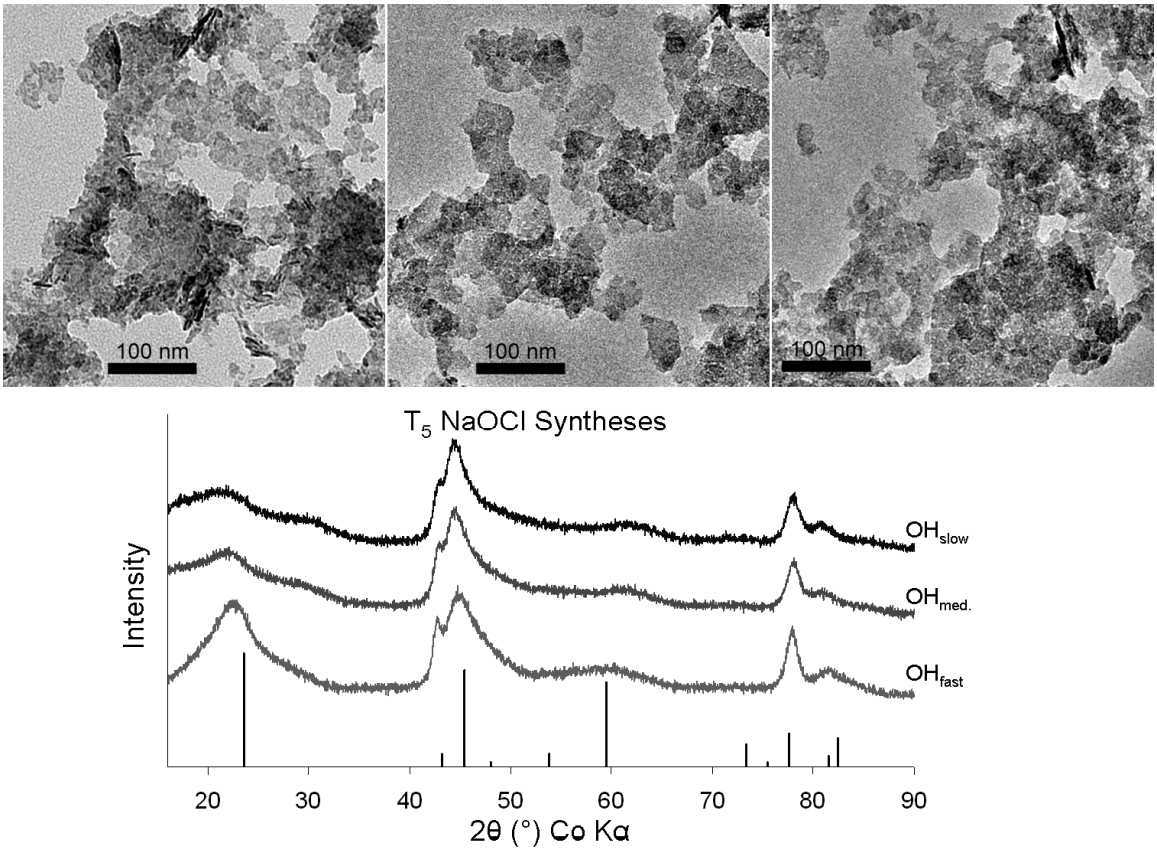

Figure 2.6 Top: TEM images of particles prepared by T_5 NaOCl samples with fast (left), medium (center), and slow (right) base addition rates. Bottom: XRD patterns of these particles. The stick pattern at the bottom shows the heterogenite peak positions.

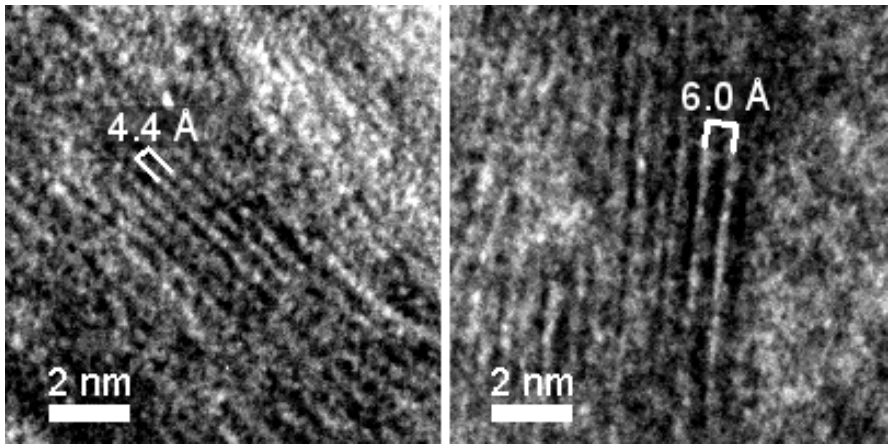

Figure 2.7 Lattice fringe TEM images of heterogenite particles prepared by the $T_5OH_{slow}NaOCl$ method. The white brackets highlight differences in planar spacing observed along the *c* direction.

Table 2.3 Summary of temperature and base addition rate effects

		OH_{fast}	$OH_{med.}$	OH_{slow}
NaOCl	**5 °C**	10-30 nm platelets minor *c* disorder	10-30 nm platelets significant *c* disorder	10-30 nm platelets significant *c* disorder
	60 °C	100-200 nm plates	0.5-2 μm plates crack-like features (larger plates only)	1-2 μm plates minor Co_3O_4
H_2O_2	**5 °C**	5-10 nm platelets	5-10 nm platelets minor Cl-containing phase	5-10 nm platelets Cl-containing phase
	60 °C	10-30 nm platelets	10-30 nm platelets minor Co_3O_4	10-30 nm platelets β-$Co(OH)_2$, Co_3O_4
	Both T	rings very rare	rings rare	occasional rings

2.4 Discussion

2.4.1 Role of the Oxidizing Agent

The most noteworthy differences caused by varying the oxidizing agent were that 1-2 μm plates were only observed when NaOCl was used and that additional phases were more likely to be present in the final product when H_2O_2 was used. The pink intermediate observed before oxidation in the T_{60} syntheses, β-$Co(OH)_2$, is known to form highly-faceted, hexagonal plates 50 nm to 2 μm in diameter,[20,25-27] similar in shape and size to the heterogenite plates obtained from the T_{60} NaOCl syntheses. Because of this, it is hypothesized that reaction with NaOCl preserves the precursor morphology, while reaction with H_2O_2 breaks the particles into smaller platelets.

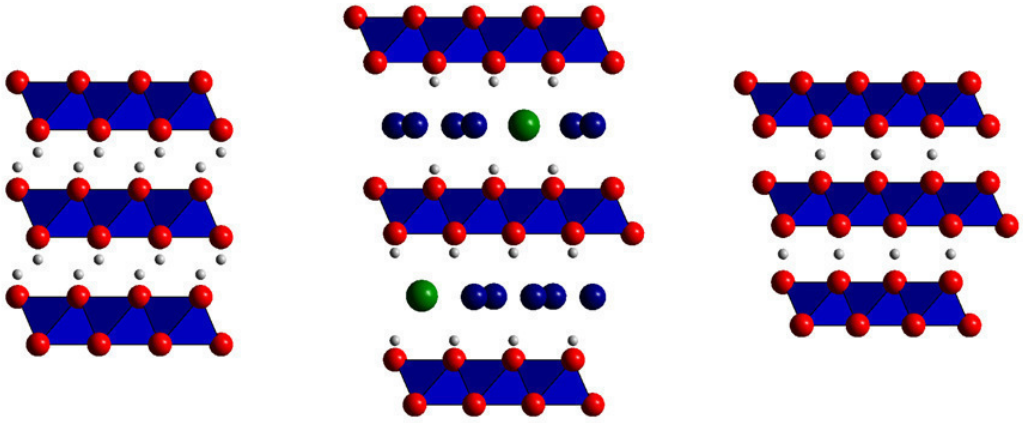

Figure 2.8 Polyhedral representation of the crystal structures of β-$Co(OH)_2$ (left), α-$Co(OH)_2$ (center), and heterogenite (β-CoOOH, right). The octahedra represent CoO_6 units, the red spheres are O atoms and the white spheres are H atoms. The larger spheres in the α-$Co(OH)_2$ structure represent intercalated water molecules (blue) and Cl^- ions (green).

The explanation for this difference lies in the details of the transformation from $Co(OH)_2$ to heterogenite. When comparing the crystal structures of β-$Co(OH)_2$ and heterogenite (Figure 2.8), it is seen that both materials consist of layers of edge-sharing CoO_6 octahedra. Heterogenite, however, has an offset layer stacking vector, so the sheets are required to shear into a new position during the solid-state transformation. The transformation from cobalt (II) hydroxide to heterogenite also involves a 32% increase in Co atom packing density (3.23×10^{22} Co/cm^3 in CoOOH vs. 2.45×10^{22} Co/cm^3 in β-$Co(OH)_2$) as calculated from unit cell parameters so, regardless of oxidant, some structural rearrangement is necessary to accommodate this increase in density. When the oxidant is NaOCl, these factors are most likely the cause of the mosaic structure present in all T_{60} NaOCl and the cracks observed in the larger plates in $T_{60}OH_{17}$NaOCl. This preservation of precursor morphology and the misalignment of the heterogenite

crystallites are in agreement with previously reported results using NaOCl or atmospheric O_2 as an oxidant.[20,26,27] In contrast, reaction with H_2O_2 produces smaller heterogenite platelets. Two factors potentially explain this fragmentation. The first is that the H_2O_2 reaction simply proceeds more rapidly than that of NaOCl. A faster reaction would allow less time for the structure to relax during this transformation and could thus result in a high degree of stress, producing fragmentation. A second possibility arises because oxidation by H_2O_2 is accompanied by the rapid evolution of gas. This may result in gas bubbles forming within pores in the β-$Co(OH)_2$ crystals and the structure being torn apart by the increased pressure.

Oxidation using H_2O_2 also tends to leave some unoxidized Co^{2+}, which appears most frequently in materials prepared at high temperature and using the slowest base addition rate. These conditions favor production of the largest heterogenite particles when oxidized by NaOCl but, when oxidized by H_2O_2, only a small number of large plates are found interspersed with much smaller heterogenite particles (Figure 9). Selected area electron diffraction reveals that these large plates are composed of β-$Co(OH)_2$ and Co_3O_4. The diffraction pattern (Figure 9, inset) also shows streaking in the Co_3O_4 (311) and (440) spots, but no streaking in spots corresponding to β-$Co(OH)_2$. It is, therefore, hypothesized that only the exterior of the original single crystal β-$Co(OH)_2$ plates are fully oxidized and that this outer portion is broken up during oxidation, generating the smaller heterogenite platelets. Beyond a certain depth, the cobalt (II) is not fully oxidized, producing a shell of Co_3O_4 that surrounds the larger β-$Co(OH)_2$ interior and protects it from further reaction. This fragmentation could also explain why

the size of H_2O_2-oxidized heterogenite particles is insensitive to OH^- addition rate. A similar process of oxidation at the surface of β-$Co(OH)_2$ to form smaller heterogenite particles has been reported by Pralong, et al. under conditions of electrochemical and hydrothermal oxidation.[26]

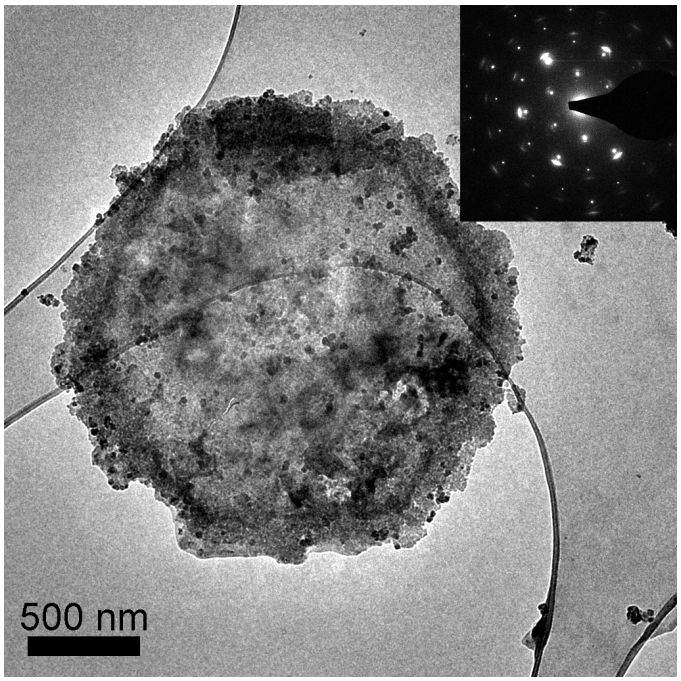

Figure 2.9 TEM image of a large hexagonal plate observed in the $T_{60}OH_{slow}H_2O_2$ sample. The selected area electron diffraction pattern (inset) shows that the plate is composed of well-aligned β-$Co(OH)_2$ and slightly misoriented Co_3O_4 particles.

The inability of H_2O_2 to fully oxidize the large plates can be partially elucidated by the work of Rai and co-workers, who have performed studies on the oxidation of chromium(III) by H_2O_2 and NaOCl.[28,29] They found that the rate of oxidation by H_2O_2 diminished substantially with oligomerization of the chromium species, while oxidation by NaOCl was not significantly affected. The authors proposed that the activated complex for oxidation by H_2O_2 involves breaking oxo-bridges between the Cr ions. If a

similar activated complex is formed during Co oxidation, breakage of the oxo- and hydroxo-bridges could sever the charge-transfer pathways between Co centers in the interior of the particle and those at the reactive surface, resulting in a core of unreacted material. Alternatively, heterogenite is known to react with $Co(OH)_2$ to form Co_3O_4, especially at elevated temperature.[30,31] It is possible that, once the outer layer of the precursor particle is oxidized to heterogenite, part of that layer then reacts with the interior of the particle to form an unreactive Co_3O_4 shell.

2.4.2 Cobalt Hydroxide Precursor

The water bath temperature and base addition rate have the greatest impact on the properties of the cobalt (II) hydroxide precursor formed, and the morphology and microstructure of the resulting heterogenite particles is strongly influenced by those of this precursor. The synthesis temperature controls which $Co(OH)_2$ polymorph is formed as the precursor, and the crystallinity and size of the precursor particles depends most strongly upon the rate of base addition.

The effect of temperature can be readily seen from the color differences between hot and cold syntheses. When the 60 °C water bath is used, the pink-brown precursor formed is β-$Co(OH)_2$, the most well understood form of cobalt hydroxide.[25,32,33] The form produced at ice-bath temperatures is blue-green α-$Co(OH)_2$.[25,32,33] The α-$Co(OH)_2$ structure (Figure 8, center) consists of sheets similar to those found in the beta form, but with an offset stacking vector like that of heterogenite. The structure frequently features intercalated ions between the layers, leading to a significantly larger inter-layer spacing and increased disorder along the *c*-axis.[24,25,33,34]

Changing the base addition rate controls the nucleation of the cobalt hydroxide particles. If the base is added rapidly, a relatively high concentration of aqueous cobalt hydroxide is produced. Since neither form of $Co(OH)_2$ is appreciably soluble, the solution is highly supersaturated and fast nucleation is expected to occur, resulting in a large number of small $Co(OH)_2$ particles. Conversely, when the base is added more slowly, the degree of supersaturation is lower and the result is fewer, larger particles. This means that, at higher temperature, slower addition is expected to produce larger, more highly faceted plates of β-$Co(OH)_2$. If these plates are oxidized by NaOCl, the size of the resulting heterogenite particles is directly related to the precursor size. If they are instead oxidized by H_2O_2, the precursor morphology is not preserved and the resulting heterogenite is fragmented into smaller particles. Larger precursor particle size also increases the likelihood of incomplete oxidation by H_2O_2, which leads to a more heterogeneous mixture of phases in the final product.

In the low temperature syntheses, there is no evidence that large or faceted plates of α-$Co(OH)_2$ are ever formed, but slower base addition would still result in a lower degree of supersaturation and more gradual particle growth. This could be expected to produce α-$Co(OH)_2$ particles with a higher degree of crystallinity and, therefore, a greater degree of stability and more regular dispersion of intercalated Cl^-. Similar increases in stability have been shown when slower hydrolysis agents, such as hexamethylenetetramine, are used to precipitate the cobalt hydroxide.[25] If NaOCl is used to oxidize the Co^{2+}, the product particles retain the precursor structure and thus the variation in layer spacing caused by intercalation, resulting in heterogenite with

substantial disorder along the *c* axis. If H_2O_2 is used, the precursor structure is not preserved, producing small heterogenite particles that do not show appreciable signs of inter-layer disorder.

The sole exception to lack of intercalation in H_2O_2-oxidized heterogenite is $T_5OH_{slow}H_2O_2$, the XRD pattern of which shows a phase nearly matched by a recently proposed structure for Cl^- intercalated α-$Co(OH)_2$.[24] This structure has a *c* lattice constant of 24.1 Å; if this spacing is contracted to 23.6 Å, the unidentified peaks located at 26.3°, 39.1°, and 39.9° 2θ correspond to the (006), (101), and (009) reflections of the α-$Co(OH)_2$ structure. After more than a year of exposure to atmospheric oxygen, the amount of heterogenite increased, but the sample still contains Co^{2+} and Cl^- in the form of β-$Co_2(OH)_3Cl$. This transformation supports the hypothesis that the unknown is the Cl-intercalated α-$Co(OH)_2$, and suggests that further oxidation had occurred to produce the additional heterogenite. It also indicates a structural transformation from the alpha to the more stable beta phase of $Co(OH)_2$, with the Cl^- regularly incorporated into the structure rather than intercalated. During this conversion, the intercalated water molecules are presumably lost to evaporation, but the chloride anions are retained and integrated into the new structure. It is, therefore, likely that this represents another example of incomplete oxidation by H_2O_2, leaving a core of precursor, in this case α-$Co(OH)_2$, as an impurity in the final product.

2.5 Conclusions

The influence of the synthesis temperature, oxidizing agent, base addition rate, and delay to oxidation on heterogenite particle structure and morphology has been examined by systematically changing each factor. By modifying synthetic conditions, substantial control over the purity, size, and morphology was achieved: for example, particle size varied from under 10 nm to over 2 μm. The presence of disorder along the *c*-axis and of impurity phases were also shown to be controllable by adjusting the synthesis procedures. The temperature of the synthesis was found to be the most important variable in determining the final structure, with the choice of oxidizing agent also playing a significant role. The rate of base addition could be adjusted to further modify the particle size and control the presence of inter-layer disorder in the heterogenite structure. Finally, the delay to oxidation had no significant effect on the resulting heterogenite, aside from the requirement of a minimum delay to produce crystalline material.

The most dramatic control over the heterogenite particles was achieved by changing the synthetic temperature. Depending upon the temperature, the precursor particles could be either the alpha or beta form of $Co(OH)_2$. The structure of the precursor contributed greatly to the structure of the product particles. The low temperature form, α-$Co(OH)_2$, incorporates Cl^- anions from solution into its interlayer structure, resulting in disorder of the layer spacing. This potentially translates into disorder and/or intercalation in the heterogenite product. The high temperature form, β-$Co(OH)_2$, does not allow intercalation; therefore the heterogenite particles derived from it

do not display the *c*-axis disorder found in some of the cold temperature syntheses. The β-$Co(OH)_2$ also has a stronger tendency to form large, hexagonal plates, which results in the possibility of larger heterogenite plates than were observed from α-$Co(OH)_2$.

The choice of oxidizing agent also had a significant effect on the final particle characteristics. Oxidation by NaOCl appears to retain the morphology of the precursor particles, allowing the size of the heterogenite particles to be determined by the cobalt (II) hydroxide size. The stacking disorder produced by intercalation in α-$Co(OH)_2$ can also be retained when during the oxidation, allowing for heterogenite particles that are highly disordered along the *c* axis. Oxidation by H_2O_2, however, preserves few of the precursor characteristics. This resulted in smaller heterogenite particles and the occasional larger fragment. These smaller particles were more likely to assemble into oriented rings and clusters, which could potentially be exploited to create materials with useful magnetic or electronic properties. Particle prepared using H_2O_2 also frequently contained unoxidized Co^{2+} in the form of either $Co(OH)_2$ or Co_3O_4. This incomplete oxidation of the precursors leads to samples containing a mixture of cobalt oxides.

The rate of base addition is significant in that it controls the size and crystallinity of the precursor particles. Changes to the addition rate, therefore, can be used to vary the size of the larger hexagonal plates produced. In the same way, larger plates increase the likelihood of H_2O_2 leaving an unoxidized core of material; a faster OH^- addition rate generates fewer impurity phases in an H_2O_2-oxidized sample. The rate also influences the degree of intercalation in α-$Co(OH)_2$ precursors. Slower base addition produces more stable α-$Co(OH)_2$, which is therefore more likely to contribute to stacking disorder in the

resulting heterogenite.

Finally, the delay to oxidation had the smallest effect on the structure of the heterogenite particles. Changing the delay from 15 min to 90 min produced no apparent change in the size, shape, or structure of the product. Removing the delay altogether, however, yielded amorphous products. It seems, therefore, that crystalline precursor particles are required to generate crystalline heterogenite and that creating crystalline precursors requires a measurable amount of time. The actual time required likely depends on temperature and concentration, but 15 minutes was found to be sufficient under all conditions considered here.

These results show that a high degree of synthetic control is possible by changing only a small number of procedural variables. Extending this study to additional variables such as concentration of Co^{2+}, OH^-, and oxidant; additional choices of oxidizing agent and base; and the inclusion of additional anions for intercalation could yield even more over the resulting heterogenite size, structure, and morphology. Furthermore, by the use of an oxidizing agent, such as NaOCl, that preserves precursor morphology, any structure that could be generated in $Co(OH)_2$ particles could be converted to Co^{3+}, allowing for many potential electronic and catalytic applications.

2.6 Acknowledgments

We thank the University of Minnesota and the National Science Foundation (Career-036385 and MRI EAR-0320641) for funding to support this work. Portions of this work were conducted at the Characterization Facility, University of Minnesota, which receives support from the NSF through the National Nanotechnology Infrastructure Network.

2.7 References

1. Chun, C.L.; Penn, R.L.; Arnold, W.A., "Kinetic and Microscopic Studies of Reductive Transformations of Organic Contaminants on Goethite." *Environmental Science & Technology* **2006**, *40*, 3299.

2. Myers, J.C.; Penn, R.L., "Evolving Surface Reactivity of Cobalt Oxyhydroxide Nanoparticles." *J. Phys. Chem. C* **2007**, *111*, 10597.

3. He, Y.; Liu, S.; Kong, L.; Liu, Z., "A study on the sizes and concentrations of gold nanoparticles by spectra of absorption, resonance Rayleigh scattering and resonance non-linear scattering." *Spectrochimica Acta, Part A: Molecular and Biomolecular Spectroscopy* **2005**, *61*, 2861.

4. Mock, J.J.; Barbic, M.; Smith, D.R.; Schultz, D.A.; Schultz, S., "Shape effects in plasmon resonance of individual colloidal silver nanoparticles." *Journal of Chemical Physics* **2002**, *116*, 6755.

5. Park, S.-J.; Kim, S.; Lee, S.; Khim, Z.G.; Char, K.; Hyeon, T., "Synthesis and Magnetic Studies of Uniform Iron Nanorods and Nanospheres." *Journal of the American Chemical Society* **2000**, *122*, 8581.

6. Nath, S.; Kaittanis, C.; Ramachandran, V.; Dalal, N.S.; Perez, J.M., "Synthesis, Magnetic Characterization, and Sensing Applications of Novel Dextran-Coated Iron Oxide Nanorods." *Chemistry of Materials* **2009**, *21*, 1761.

7. Ichiyanagi, Y.; Yamada, S., "The size-dependent magnetic properties of Co3O4 nanoparticles." *Polyhedron* **2005**, *24*, 2813.

8. Jenkins, D.J.; Alabdulrahman, A.M.S.; Attard, G.A.; Griffin, K.G.; Johnston, P.; Wells, P.B., "Enantioselectivity and catalyst morphology: step and terrace site contributions to rate and enantiomeric excess in Pt-catalysed ethyl pyruvate hydrogenation." *Journal of Catalysis* **2005**, *234*, 230.

9. Piccolo, L.; Henry, C.R., "NO-CO Reaction Kinetics on Pd/MgO Model Catalysts: Morphology and Support Effects." *Journal of Molecular Catalysis A: Chemical* **2001**, *167*, 181.

10. Meusel, I.; Hoffmann, J.; Hartmann, J.; Libuda, J.; Freund, H., "Size Dependent Reaction Kinetics on Supported Model Catalysts: A Molecular Beam/IRAS Study of the CO Oxidation on Alumina-Supported Pd Particles." *J. Phys. Chem. B* **2001**, *105*, 3567.

11. Chirvase, D.; Parisi, J.; Hummelen, J.; Dyakonov, V., "Influence of nanomorphology on the photovoltaic action of polymer–fullerene composites." *Nanotechnology* **2004**, *15*, 1317.

12. Yang, X.; Loos, J.; Veenstra, S.; Verhees, W.; Wienk, M.; Kroon, J.; Michels, M.; Janssen, R., "Nanoscale morphology of high-performance polymer solar cells." *Nano Letters* **2005**, *5*, 579.

13. Park, N.-G.; van de Lagemaat, J.; Frank, A.J., "Comparison of Dye-Sensitized Rutile- and Anatase-Based TiO2 Solar Cells." *Journal of Physical Chemistry B* **2000**, *104*, 8989.

14. Spinolo, G.; Ardizzone, S.; Trasatti, S., "Surface characterization of Co3O4 electrodes prepared by the sol-gel method." *Journal of Electroanalytical Chemistry* **1997**, *423*, 49.

15. Pralong, V.; Delahaye-Vidal, A.; Beaudoin, B.; Leriche, J.-B.; Tarascon, J.-M., "Electrochemical behavior of cobalt hydroxide used as additive in the nickel hydroxide electrode." *J. Electrochem. Soc.* **2000**, *147*, 1306.

16. Xie, X.; Shen, W., "Morphology control of cobalt oxide nanocrystals for promoting their catalytic performance." *Nanoscale* **2009**, *1*, 50.

17. Schubert, M.; Plzak, V.; Garche, J.; Behm, R., "Activity, selectivity, and long-term stability of different metal oxide supported gold catalysts for the preferential CO oxidation in H2-rich gas." *Catalysis Letters* **2001**, *76*, 143.

18. Furlanetto, G.; Formaro, L., "Precipitation of Spherical Co3O4 Particles." *Journal of Colloid and Interface Science* **1995**, *170*, 169.

19. Hu, W.K.; Gao, X.P.; Geng, M.M.; Gong, Z.X.; Noreus, D., "Synthesis of CoOOH Nanorods and Application as Coating Materials of Nickel Hydroxide for High Temperature Ni-MH Cells." *J. Phys. Chem. B Lett.* **2005**, *109*, 5392.

20. Figlarz, M.; Guenot, J.; Fievet-Vincent, F., "Morphological and topotactical aspects of the reactions Co(OH)2 -> CoOOH and CoOOH -> Co3O4." *J. Mater. Sci.* **1976**, *11*, 2276.

21. Stone, A.T.; Ulrich, H.J., "Kinetics and reaction stoichiometry in the reductive dissolution of manganese(IV) dioxide and cobalt(III) oxide by hydroquinone." *J. Colloid Interface Sci.* **1989**, *132*, 509.

22. McArdell, C.S.; Stone, A.T.; Tian, J., "Reaction of EDTA and Related Aminocarboxylate Chelating Agents with Co(III)OOH (Heterogenite) and Mn(III)OOH (Manganite)." *Environ. Sci. Technol.* **1998**, *32*, 2923.

23. Penn, R.L.; Stone, A.T.; Veblen, D.R., "Defects and Disorder: Probing the Surface Chemisry of Heterogenite (CoOOH) by Dissolution Using Hydroquinone and Iminodiacetic Acid." *J. Phys. Chem. B* **2001**, *105*, 4690.

24. Ma, R.; Liu, Z.; Takada, K.; Fukuda, K.; Ebina, Y.; Bando, Y.; Sasaki, T., "Tetrahedral Co (II) Coordination in [alpha]-Type Cobalt Hydroxide: Rietveld Refinement and X-ray Absorption Spectroscopy." *Inorg. Chem* **2006**, *45*, 3964.

25. Liu, Z.; Ma, R.; Osada, M.; Takada, K.; Sasaki, T., "Selective and controlled synthesis of alpha-and beta-cobalt hydroxides in highly developed hexagonal platelets." *Journal of the American Chemical Society* **2005**, *127*, 13869.

26. Pralong, V.; Delahaye-Vidal, A.; Beaudoin, B.; Gerand, B.; Tarascon, J.-M., "Oxidation mechanism of cobalt hydroxide to cobalt oxyhydroxide." *J. Mater. Chem.* **1999**, *9*, 955.

27. Figlarz, M.; Guenot, J.; Tournemolle, J.-N., "Oxidation of cobalt (II) hydroxide to oxide hydroxide: solids evolution during reaction." *J. Mater. Sci.* **1974**, *9*, 772.

28. Rao, L.; Zhang, Z.; Friese, J.I.; Ritherdon, B.; Clark, S.B.; Hess, N.J.; Rai, D., "Oligomerization of Chromium (III) and its impact on the oxidation of Chromium (III) by Hydrogen Peroxide in alkaline solutions." *Journal of the Chemical Society, Dalton Transactions: Inorganic Chemistry* **2002**, *2002*, 267.

29. Jiang, H.; Rao, L.; Zhang, Z.; Rai, D., "Characterization and oxidation of chromium(III) by sodium hypochlorite in alkaline solutions." *Inorganica Chimica Acta* **2006**, *359*, 3237.

30. Dogterom, R.; Oosterbeek, H.; Reynhout, M. Catalytic Process For The Conversion Of Co (II) Hydroxide In Co (III) Hydroxide U.S., 2005.

31. Singh, R.; Mendenhall, R. Heterogenite material for making submicron cobalt powders U.S., 2004.

32. Armstrong, R.D.; Briggs, G.W.D.; Charles, E.A., "Some effects of the addition of cobalt to the nickel hydroxide electrode." *J. Appl. Electrochem.* **1988**, *18*, 215.

33. El-Batlouni, H.; El-Rassy, H.; Al-Ghoul, M., "Cosynthesis, Coexistence, and Self-Organization of [alpha]-and [beta]-Cobalt Hydroxide Based on Diffusion and Reaction in Organic Gels." *Journal of Physical Chemistry A* **2008**, *112*, 7755.

34. Kamath, P.; Therese, A.; Helen, G.; Gopalakrishnan, J., "On the existence of hydrotalcite-like phases in the absence of trivalent cations." *Journal of Solid State Chemistry* **1997**, *128*, 38.

3

Evolving Surface Reactivity of Cobalt Oxyhydroxide Nanoparticles*

Accurate characterization of the reactive surface area of nanocrystalline materials is a vital yet challenging aspect of understanding their properties. This work demonstrates that the ligand-assisted dissolution of heterogenite (β-CoOOH) nanocrystals by iminodiacetic acid (IDA) can serve as a probe of reactive surface area. Specifically, two geometric isomers of $Co(IDA)_2^-$ result, and the relative concentrations of these isomers change with the particle aspect ratio. In experiments in which heterogenite particles were aged under varying pH conditions, the observed size and shape of the particles are strongly influenced by pH; aging under higher pH conditions results in larger particles with a smaller aspect ratio, which is reflected in the ratio of dissolution products. A model for particle growth is proposed, and the corresponding rate law is fitted to the experimental data. This leads to the conclusion that particle growth occurs predominately by the stacking of very thin primary platelets across the (003) surfaces.

*Reproduced in part with permission from Myers, J.C. and Penn, R.L. *J. Phys. Chem. C,* **2007**, *111*, 10597-10602.

3.1 Introduction

The chemical and physical properties of a material often vary dramatically with size, morphology, and microstructure. Studies on the incorporation of cobalt oxides into the electrodes of nickel metal hydride batteries have shown that the size and shape of the particles can alter both capacity and cycling life.[1-5] The use of particles with higher defect concentrations is also believed to enhance the surface activity of the electrode, further improving the device.[3,5] The effectiveness of catalytic materials is also strongly affected by particle morphology. For example, cubic, octahedral, and truncated octahedral ceria particles all display different activities.[6]

Characterizing the reactive surface area of a material is a vital step in understanding its reactivity. Many materials have been shown to react primarily or exclusively at one type of surface. Dissolution of hectorite clay in a strongly acidic medium, for example, proceeds entirely along the edge surfaces of the particles.[7] Reductive degradation of organic contaminants by the iron oxide goethite has been shown to occur predominantly on the (021) surfaces of the material.[8] Nonetheless, the relative reactivities of different crystallographic surfaces and defect structures remain difficult to ascertain using traditional solid-state methods. Dissolution with carefully chosen molecules may provide a means to indirectly measure these differences. If a chelating ligand dissolves a solid via a surface-bound intermediate, the product geometry may be controlled by the surface structure. In such a case, the relative amounts of each product allow quantification of different surface types, while the kinetics of product formation allow comparison of the reactivity of each surface.

The mineral heterogenite (β-CoOOH) and the ligand iminodiacetic acid (IDA) serve as one such system. While not as common as other metal oxide minerals, heterogenite has many properties that make it an excellent model material. It is rapidly synthesized from cheap and readily available precursors, and it can be prepared in a variety of sizes and morphologies. Heterogenite samples can also be prepared with both defect-rich and relatively defect-free structures.[9] Aqueous Co^{III} complexes are not labile, especially below neutral pH,[10] which means that any surface information obtained in the form of relative concentrations or geometries of the products is not lost to rearrangement of the ligands before analysis. Finally, previous studies of the heterogenite-IDA system have shown no evidence of Co reduction or other side reactions.[11]

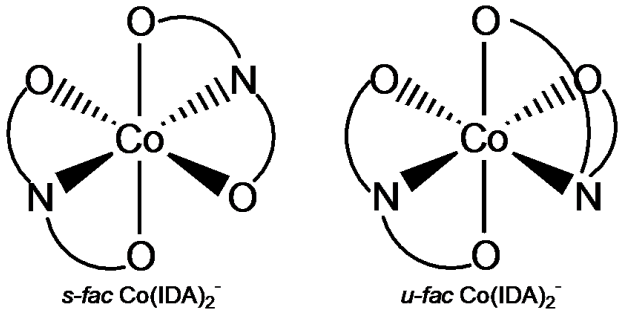

Figure 3.1 Structural representations of the symmetrical (*s-fac*) and unsymmetrical (*u-fac*) facial isomers of $Co(IDA)_2^-$.

Reaction of heterogenite with IDA produces two geometric isomers of $Co(IDA)_2^-$ (Figure 3.1), and samples of heterogenite prepared by differing methods reproducibly generate different ratios of the *s-fac* and *u-fac* products.[9,11] If the products were formed by chelation of free Co^{3+} ions, one would expect to find the thermodynamic ratio of isomers regardless of the synthesis conditions; this variation indicates not only that the reaction proceeds via a surface-bound intermediate, but also that this method of

dissolution is sensitive to the changes in structure caused by different preparation methods.

In this work, it is shown that ligand-assisted dissolution by chelating molecules can be used to probe reactive surfaces. Results tracking the relative concentrations of *s-fac* and *u-fac* isomers produced are presented, and these data are used to develop a model linking the ratio of isomers to the morphology of the CoOOH particles. It is shown that the reactivity of heterogenite particles changes as a function of aging conditions, and that this change can be monitored by way of dissolution using IDA. Finally, a qualitative link between aging pH and the evolution of reactive surface area is proposed, and supporting evidence is presented in the form of physical characterization and kinetic modeling.

3.2 Experimental Methods

3.2.1 Preparation of Heterogenite Particles

Heterogenite particles [$T_5OH_{fast}H_2O_2$(90 min), see chapter 2] were prepared according to a procedure adapted from the method of Stone and Ulrich.[12] A 4 L volume of Milli-Q purified water (Millipore Corporation, 18 MΩ•cm resistivity) was purged with N_2 gas for 3 h. From this, 2 L of 3.2 mM Co^{2+} solution (cobalt (II) chloride hexahydrate, analytical reagent grade, Mallinckrodt Laboratory Chemicals) and 2 L of 6.4 mM OH^- solution (sodium hydroxide, AR, Mallinckrodt) were prepared. These solutions were cooled to ~5 °C in an ice bath.

The hydroxide solution was added to the cobalt (II) solution over ~1 min and shaken to ensure rapid mixing. The resulting pale blue suspension was stirred with a

Teflon-coated magnetic stir bar for 90 min while remaining in the ice bath. The suspension was then oxidized using 40 mL of 0.74 M hydrogen peroxide (30% hydrogen peroxide solution, Mallinckrodt). The suspension was stirred in the ice bath for an additional 1 h, at which time it was removed and buffered to a pH of 3.9, 4.6, or 5.6 using a 10 mM acetate buffer (glacial acetic acid, AR, Mallinckrodt) or a 10 mM MES buffer (MES hydrate, 99%, Acros Ogranics) and adding 1.0 M sodium hydroxide to attain the desired pH. The acetate pH 4.6 and MES pH 5.6 samples maintained a stable pH; the acetate pH 3.9 and 5.6 samples, however, became more basic as they equilibrated over 24 h, resulting in final pH of 4.1 and 5.8. This is likely due to the buffering properties of the particle surfaces, and the lowered buffer capacity as pH becomes more distant from the pKa of acetic acid. The buffered heterogenite samples were aged at room temperature for 30 days.

3.2.2 Particle Dissolution

During the 30 day aging period, the heterogenite particles were periodically sampled and characterized by IDA dissolution. While stirring to ensure uniform distribution of the suspended particles, a 50 mL aliquot of each buffered suspension was removed. The pH was adjusted to 4.6 using either 1 M sodium hydroxide or 1 M hydrochloric acid (Mallinckrodt). The MES-buffered sample required the addition of acetate buffer (to a concentration of 10 mM) before the pH could be stabilized at 4.6. In control experiments where MES-buffered samples were reacted at pH 5.6 with and without acetate addition, the presence of the acetate was found to have no discernable effect on the results. To begin dissolution, 0.100 mL of 0.1 M IDA (98+%, Acros

Organics) was added to each sample. Batch reactors were periodically sampled by extracting 3.0 mL, and samples were quenched by filtering with 0.2 μm nylon membrane syringe filters (Pall Life Sciences) before analysis.

The concentration of each $Co(IDA)_2^-$ isomer was determined using a Beckman Coulter P/ACE MDQ capillary electrophoresis (CE) system equipped with a fused-silica capillary, which had a length of 51 cm (to the detector) and an inner diameter of 73 μm. The separation method was a modification of that employed by Bürgisser and Stone.[13] The background electrolyte consisted of a 50 mM phosphate buffer (pH 7) and 10 mM tetradecyltrimethylammonium bromide (TTAB, 99%, Aldrich) additive. The TTAB adsorbs to the walls of the silica capillary and creates a positively charged layer that results in a reversal of the electroosmotic flow within the capillary.[14,15] The separations were performed at 10 kV, reversed polarity, and using injection pressure of 0.5 psi for 10 s. Prior to analysis, samples were spiked with a $Co(EDTA)^-$ internal standard to account for variations in injection volume. The $Co(EDTA)^-$ has a migration time between that of the two $Co(IDA)_2^-$ isomers and is stable with respect to ligand exchange over the time scale of the experiment. Products were measured using UV photometric detection at a wavelength of 254 nm. Concentrations were determined from five-point calibration curves; pure samples of *s-fac* and *u-fac* isomers were prepared using well-established methods[16,17] and analyzed in the presence of the $Co(EDTA)^-$ standard. Select dissolution experiments were performed in duplicate, and values differed from the mean by no more than 2%.

3.2.3 Materials Characterization

After aging 30 days, the heterogenite samples were characterized by powder X-ray diffraction (XRD). To prepare each sample, approximately 500 mL of particle suspension was centrifuged at 14000 rcf (relative centrifugal force). The supernatant was decanted and the particles were resuspended in Milli-Q water. This was repeated twice to wash the particles, which were then placed in shallow dishes and allowed to air dry over 6 days. The samples were characterized using a PANalytical X'Pert Pro MPD theta-theta diffractometer, equipped with a cobalt anode and an X'Celerator detector. Powder samples were placed on a zero-background quartz sample holder. Data were collected in scanning mode over 18-90° 2θ with a scan rate of 0.005 °/s. Peaks were fitted using a pseudo-Voigt profile function, and integral breadths were determined for the relevant peaks.

Samples were prepared for transmission electron microscopy (TEM) by centrifuging a small amount of each particle suspension and resuspending in Milli-Q water. The rinsed particles were then diluted and aquasonicated (VWR Aquasonic model 150HT) for 5 min. A single drop of suspension was placed on a 3 mm copper TEM grid coated with holey carbon film (SPI Supplies). The grids were examined using FEI Tecnai T12 and FEI Tecnai F30 FEG microscopes. Images were collected via a Gatan charge-coupled device camera and analyzed using Gatan Digital Micrograph software (version 3.8.2).

3.3 Results and Discussion

3.3.1 Dissolution

Reaction between a given heterogenite sample and IDA produces a mixture of *s-fac* and *u-fac* $Co(IDA)_2^-$ isomers in a fixed ratio. The ratio of products formed is changed not only by the preparation method of the heterogenite,[9] but also by the length of aging time between synthesis and dissolution. As aging time increases, the ratio of *s-fac* to *u-fac* (s/u) isomers increases. This effect is strongly dependent on the pH at which the particles were aged (note that all dissolution experiments were conducted at pH 4.6); when the suspension was maintained at higher pH, the s/u ratio was larger than when the particles were aged at lower pH (Figure 3.2). The initial rate of production for total

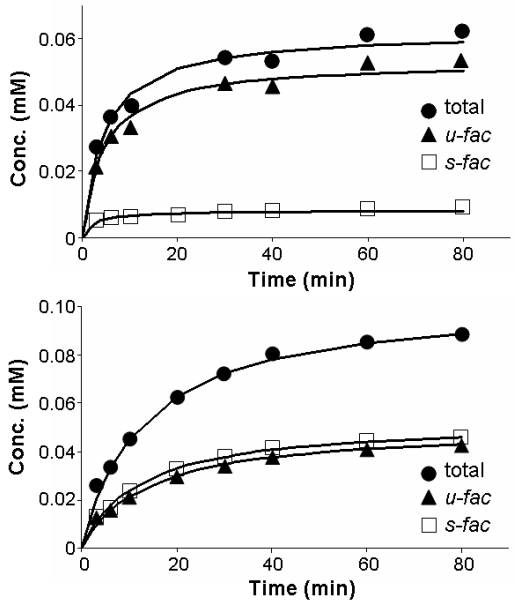

Figure 3.2 Concentration of *u-fac* isomer, *s-fac* isomer, and total $Co(IDA)_2^-$ as a function of time for dissolution of heterogenite samples by IDA. The particles were aged for 2 days in acetate buffer at pH 4.1 (top) or pH 5.8 (bottom) before reaction. The solid lines represent fittings of pseudo-second order integrated rate laws.

dissolved cobalt is lower when particles are aged at pH 5.8 (rate = 8.0×10^{-3} mM/min) than when aged at pH 4.1 (rate = 1.4×10^{-2} mM/min). One possible explanation for this decrease—which is confirmed by the characterization resulted presented below—is that the particles are continuing to grow and the rate of growth is faster at higher pH. Larger particles have less available surface area per volume, so the effective concentration of surface sites would be decreased.

Figure 3.3 shows the s/u ratio versus time for samples aged for 30 days under conditions ranging from pH 4.1 to pH 5.8. In all cases, the trend is a rapid initial increase in s/u ratio, followed by a gradual leveling off towards a final, steady-state value. As the suspensions are aged at higher pH, the ratio of products changes more rapidly and reaches a steady-state value earlier. Experiments performed across a range of

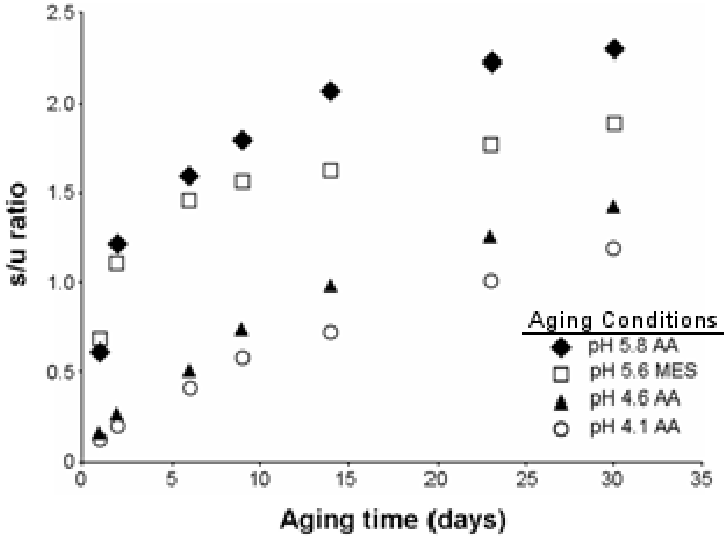

Figure 3.3 Ratio of *s-fac* to *u-fac* isomers produced by dissolution as a function of aging time. The samples were aged at room temperature in acetate buffer (pH 4.1, 4.6, or 5.8) or MES buffer (pH 5.6).

temperatures showed no statistically significant change in activation energy with aging,[18] so the different product ratio is unlikely to be a result of a change in reaction mechanism. These results clearly demonstrate that aging at higher pH leads to more rapid evolution of reactivity. The sample aged in the pH 5.6 MES buffer appears to behave differently from the acetate samples; however, previous experiments (see Figure 3.4) in which particles were aged for 6 days in pH 5.1 acetate or MES buffer showed no difference in product ratio between the two buffers. The most likely explanation derives from the CE separation method used to quantify each isomer. The presence of the MES buffer in the samples decreases the separation efficiency, resulting in some overlap between the standard and *s-fac* peaks for samples with high *s-fac* concentration. Peak intensity was assigned using peak fitting routines, but peak shapes obtained from CE separations are often irregular,[19] so this method is somewhat crude and could introduce error into the later pH 5.6 MES samples. Because of this, the MES data was excluded from the kinetic modeling presented below.

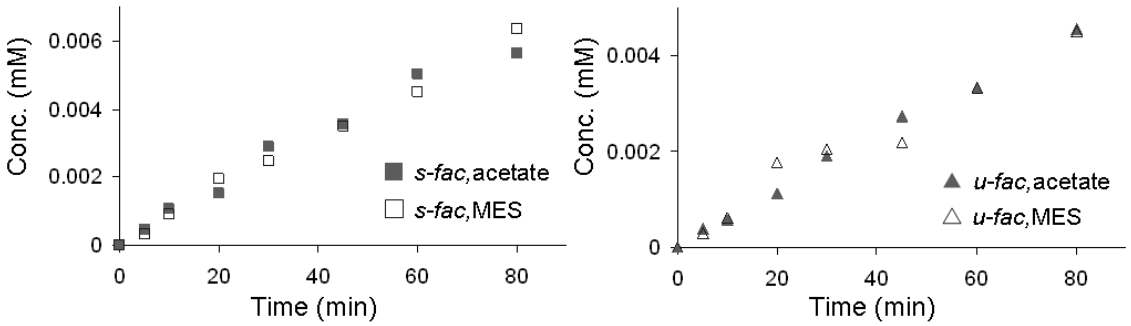

Figure 3.4 Comparison of dissolution results from heterogenite particles aged 145 h in pH 5.1 buffers. Left: *s-fac* isomer production after aging in acetate (■) or MES (□) buffer. Right: *u-fac* isomer production after aging in acetate (▲) or MES (△) buffer.

3.3.2 Characterization

Figure 3.5 shows a representative TEM image from each of the four samples after the full 30 days of aging. All four samples are composed predominately of round platelets, which agrees with observations of heterogenite particles produced via similar methods.[9] As the aging pH increases, the platelets grow in size, and objects appearing darker and elongated are observed more frequently in the images. Imaging these objects at a range of tilts suggests that they are platelets lying on their sides rather than flat on the

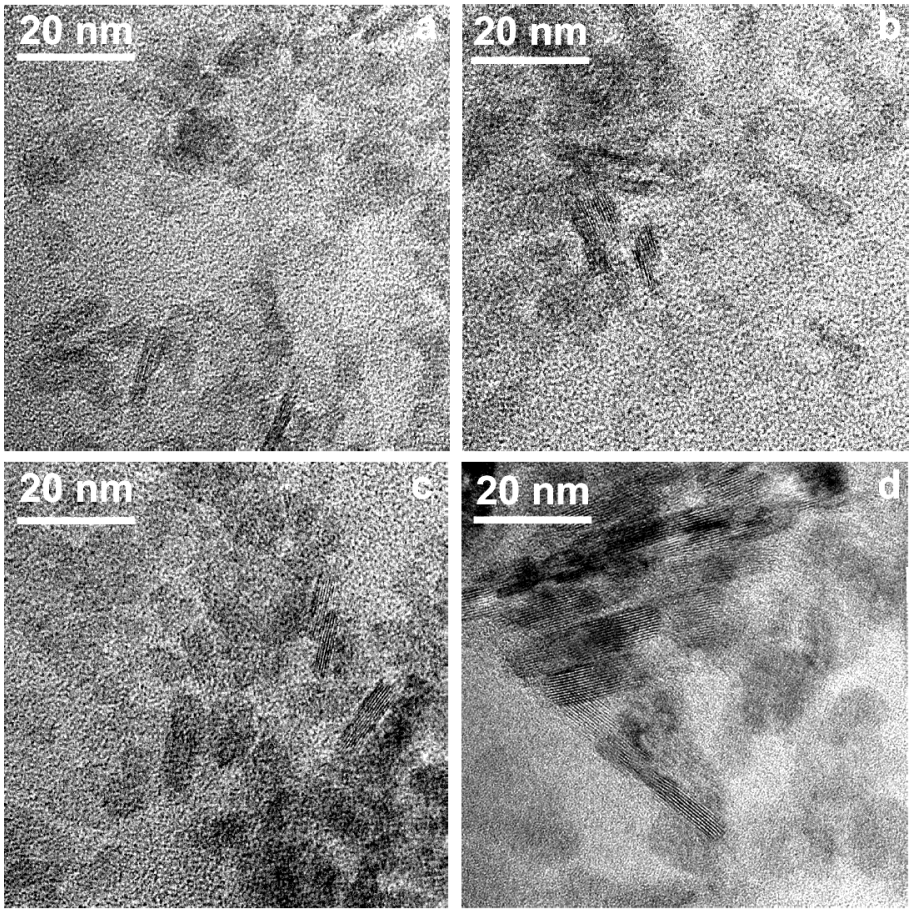

Figure 3.5 Representative TEM images of heterogenite aged 30 days in buffered suspensions: a) pH 4.1 acetate, b) pH 4.6 acetate, c) pH 5.6 MES, d) pH 5.8 acetate

sample grid. The darker color is explained by mass-thickness contrast, due to the electron beam passing through the entire width of the plates rather than the thinner height. At higher magnification, lattice fringes with spacing consistent with (003) are observed (Figure 3.6). The mean particle diameter of each sample was determined by measuring approximately 400 particles. Fewer particles were in a favorable orientation to measure plate thickness, so each mean thickness was calculated from a minimum of 150 measurements.

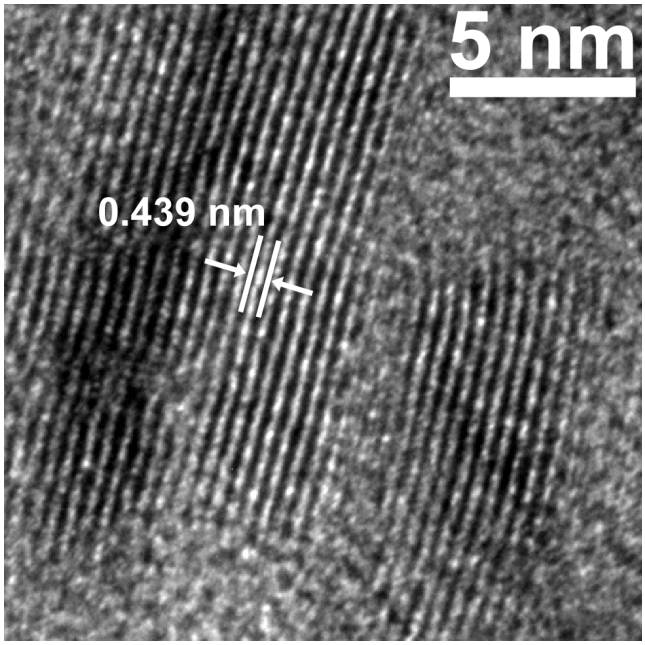

Figure 3.6 Lattice fringe TEM image of heterogenite platelets viewed edge on. The spacing between fringes corresponds to the (003).

Figure 3.7 shows XRD patterns obtained from the heterogenite samples after aging for 30 days. The most noticeable feature is the narrowing of the peak at 23.5° 2θ as aging pH increases. This peak arises from the 003 crystallographic planes, and the narrowing implies an increase in the thickness of the crystallite plates. Many of the other

visible peaks overlap, but the 110 peak, located at 77.6° 2θ, is distinguishable and corresponds to the diameter of the plates. Volume-weighted crystallite sizes were calculated in these two directions by using a single-line integral breadth method.[20] A pseudo-Voigt peak profile was used, instrumental broadening contributions were removed, and sizes were corrected for an assumed cylindrical crystallite shape. A comparison of the sizes determined from TEM measurements and XRD analysis is presented in Table 3.1.

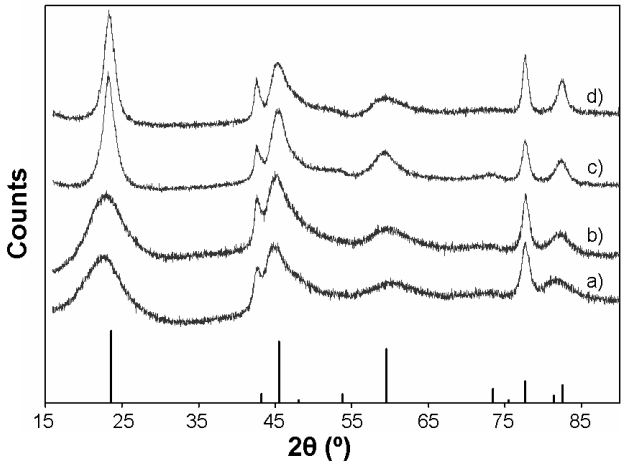

Figure 3.7 XRD patterns of heterogenite particles aged for 30 days in a) pH 4.1 acetate, b) pH 4.6 acetate, c) pH 5.6 MES, or d) pH 4.8 acetate buffer solution. The stick pattern corresponds to the PDF of heterogenite-3R (#007-0169).

Table 3.1 Particle Diameter and Height as Determined by TEM and XRD

	D_{TEM}[a] (nm)	D_{XRD} (nm)	H_{TEM}[a] (nm)	H_{XRD} (nm)
pH 4.1 AA	9.1 ± 1.7	11.5	1.9 ± 0.5	3.3
pH 4.6 AA	10.0 ± 1.7	12.2	2.4 ± 0.5	2.4
pH 5.6 MES	11.6 ± 2.4	16.0	3.8 ± 0.9	4.1
pH 5.8 AA	12.7 ± 2.0	15.4	3.8 ± 0.9	4.3

[a] Range represents the standard deviation of the particle size distribution

3.3.3 Modeling

To link the differences in product ratio over time with the differences in particle morphology observed via XRD and TEM, a model for particle growth is proposed. The most prominent differences were an increase in overall size and a decrease in aspect ratio as the particles were aged at higher pH; therefore, it was hypothesized that the smaller aspect ratio is responsible for the increased s/u product ratio. The heterogenite particle morphology can be generalized to a cylindrical plate shape and described by two basic types of surfaces: basal and edge (Figure 3.8). Examining the crystal structure of heterogenite shows that the coordinative environment of the cobalt centers on the basal surfaces is quite different from those on the edge-type surfaces, so it is reasonable to hypothesize that each surface is primarily responsible for the formation of one of the product isomers. Comparing the particle morphology to the dissolution product ratio shows that the *u-fac* isomer dominates reactions involving thinner platelets, so production of *u-fac* isomer was assigned to the basal surface, while production of *s-fac* isomer was assigned to the edge-type surfaces. This is supported by previous studies, in which the *u-fac* isomer was the primary product from the reaction of large, monolithic plates with a high aspect ratio.[9]

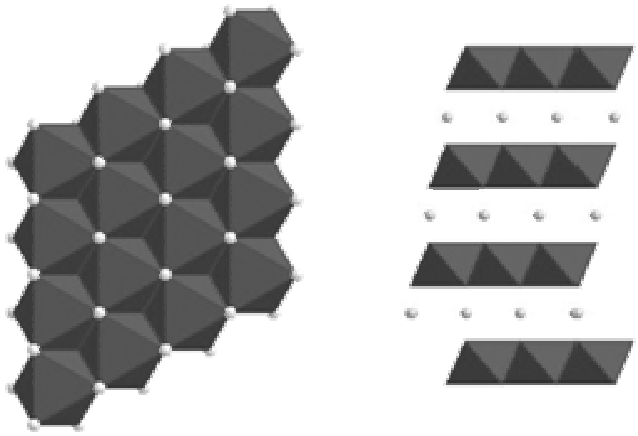

Figure 3.8 Polyhedral representation of the structures of the (left) basal (ab plane) and (right) edge (ac or bc plane) surfaces of heterogenite. Octahedra represent edge-sharing CoO_6 units and spheres represent protons.

As a first approximation, the system was modeled as if the particles rapidly nucleate as very thin platelets that then grow in thickness while maintaining the same width. Furthermore, because the change in s/u ratio with respect to aging time was found to have a general second-order shape, it was assumed that the crystallites grow by stacking across (003) surfaces. This results in a rapid doubling of height and gives rise to second-order kinetics for reduction in basal surface area. As a final element of the model, the rate of crystallite stacking decreases as the stacks approach some ultimate thickness, at which point no further growth occurs. This decrease in rate is supported by DLVO theory, which states that the repulsion between colloidal particles increases with particle size.[21]

Following these simplifications, each time a crystallite is added to a stack, the amount of available basal surface would be halved, with no reduction in edge surface.

The edge surface would then be constant, while the rate of change of the basal surface can be expressed as

$$\frac{-dS^{basal}}{dt} = k'\left(S^{basal} - \frac{1}{n}S_0^{basal}\right)^2 \tag{3.1}$$

Integrating this model gives time-dependant rate expressions for the availability of each type of surface site:

$$S^{edge} = S_0^{edge} \tag{3.2}$$

$$S^{basal} = \frac{1}{k't + \frac{1}{S_0^{basal}\left(1-\frac{1}{n}\right)}} + \frac{1}{n}S_0^{basal} \tag{3.3}$$

Here, the S variables represent the number of reactive surface sites of each type. The S_0 represent the S immediately after nucleation, k' is a pseudo-second order rate constant, t is the aging time in days, and n is the ultimate number of crystallites in a stack before growth stops. The expected final ratio can then be found by relating each surface type to the production of the appropriate isomer and dividing

$$\frac{s}{u} = \frac{D_s S^{edge}}{D_u S^{basal}} \tag{3.4}$$

Here, D_s and D_u are dissolution functions for each isomer (units not defined). They are functions of the aqueous IDA concentration, the adsorption constant for IDA on the surface, and a rate constant for dissolution. These values are likely to differ for the two isomers, but under the conditions of these experiments each D should be constant. By substituting (3.2) and (3.3) into (3.4), the ratio can be represented as

$$\frac{s}{u} = \frac{D_s S_0^{edge}}{\frac{1}{\frac{k'}{D_u}t + \frac{1}{D_u S_0^{basal}\left(1-\frac{1}{n}\right)}} + \frac{1}{n} D_u S_0^{basal}} \tag{3.5}$$

The pH dependence of the growth can be introduced as

$$k' = k_H [\mathrm{H}^+]^m \tag{3.6}$$

where k_H and m are the rate constant and order of the reaction with respect to the proton concentration. With the currently available data, it is not possible to separate the dissolution functions from the surface area, so the parameters $D_s S_0^{edge}$, $D_u S_0^{basal}$, n, $\frac{k_H}{D_u}$, and m were fit to the experimental data for acetate-buffered samples by least-squares regression. The three data sets were fit simultaneously to give a single value for each variable, yielding fitted curves that differ only by H^+ concentration. The n parameter was restricted to integral values, and fitting attempts in which n was allowed to vary between samples did not appreciably improve the quality of fit. The values obtained for $D_u S_0^{basal}$ and $D_s S_0^{edge}$ were 1.06×10^{-2} and 8.99×10^{-4}. The H^+ dependence had a rate constant $\left(\frac{k_H}{D_u}\right)$ of 0.178 and an order of -0.623. The stacking value, n, refined to 32. The resulting fits, along with the experimental data, are shown in Figure 3.9. The small deviations can be attributed to the fact that the edge-type surface area was not truly constant, as the crystallite width did vary between samples.

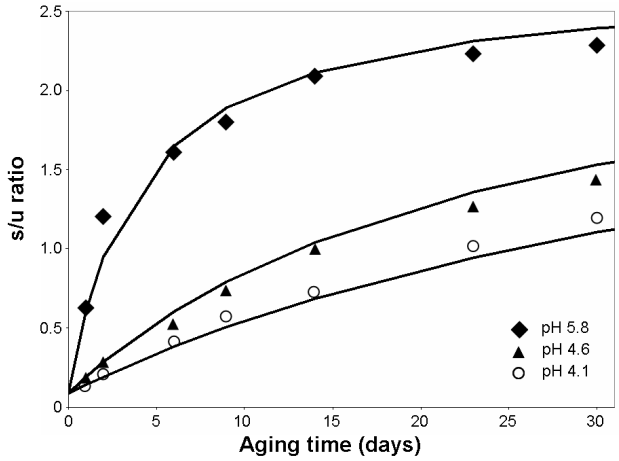

Figure 3.9 Ratio of *s-fac* to *u-fac* isomers produced as a function of aging time in acetate buffer at pH 4.1, 4.6, or 5.8. The solid lines represent fits based on a model of particle growth by stacking (equations 3.5 and 3.6). The fits differ only by H^+ concentration.

The relationship between proton concentration and the growth rate by stacking can be explained in several ways. The most likely explanations are that lower pH results in more positively charged surfaces, increasing the electrostatic repulsion between the surfaces, or that protonation of the surface makes stacking less energetically favorable. The structure of heterogenite consists of layers of edge-sharing CoO_6 octahedra that are stacked in the [003] direction, with each pair of sheets separated by a layer of protons. This structure could facilitate the addition of new layers to an existing crystal. At acidic pH, however, one would expect a higher degree of surface protonation; therefore, excess protons would need to be removed in order to match the heterogenite crystal structure. The additional energy required to remove these protons would make stacking less favorable at lower pH.

The stacking value of 32 crystallites seems unrealistically high, considering that this would produce plates 14 nm thick even if each crystallite is only a single octahedral

sheet. The complex nature of the dissolution functions and the inability to separate them from the surface areas, however, make it difficult to assign physical dimensions to the model crystallites. The freshly nucleated particles are unlikely to be well described by perfect, single-layer platelets, and this discrepancy from the model could easily result in a fractional thickness for the theoretical crystallites. Additional experiments are underway to assign more physically meaningful values to the model parameters and allow direct quantification of the reactive surface area.

3.4 Conclusions

The results of this work show a probe molecule such as IDA can indeed be used as a chemical method to investigate the reactive surface area. By using the IDA dissolution method, it was demonstrated that the reactivity of the heterogenite particles changes rapidly while in suspension and that this evolution can be hastened or slowed by varying the pH at which the sample is aged. This change in reactivity is linked to a change in particle morphology; higher pH leads to platelets that are slightly wider and substantially thicker than those aged in low pH suspensions. Finally, a growth model was proposed, in which crystals expand primarily by stacking platelets atop one another, with the rate of growth showing a strongly inverse relationship to the proton concentration in solution. The good fit of the model to experimental data supports this as the primary mechanism for particle growth. Future work will focus on determining the mechanism of dissolution and quantifying the relative reactivities of different surfaces.

3.5 Acknowledgments

We acknowledge the University of Minnesota and the National Science Foundation (Career-036385 and MRI EAR-0320641) for funding. Portions of this work were conducted at the Characterization Facility, University of Minnesota, which receives support from the NSF through the National Nanotechnology Infrastructure Network.

3.6 References

1. Armstrong, R. D.;Briggs, G. W. D.;Charles, E. A., "Some effects of the addition of cobalt to the nickel hydroxide electrode." *J. Appl. Electrochem.* **1988**, *18*, 215.

2. Butel, M.;Gautier, L.;Delmas, C., "Cobalt oxyhydroxides obtained by 'chimie douce' reactions: structure and electronic conductivity properties." *Solid State Ionics* **1999**, *122*, 271.

3. Hu, W. K.;Gao, X. P.;Geng, M. M.;Gong, Z. X.;Noreus, D., "Synthesis of CoOOH Nanorods and Application as Coating Materials of Nickel Hydroxide for High Temperature Ni-MH Cells." *J. Phys. Chem. B Lett.* **2005**, *109*, 5392.

4. Barde, F.;Palacin, M.-R.;Beaudoin, B.;Delahaye-Vidal, A.;Tarascon, J.-M., "New Approaches for Synthesizing gamma III-CoOOH by Soft Chemistry." *Chem. Mater.* **2004**, *16*, 299.

5. Pralong, V.;Delahaye-Vidal, A.;Beaudoin, B.;Leriche, J.-B.;Tarascon, J.-M., "Electrochemical behavior of cobalt hydroxide used as additive in the nickel hydroxide electrode." *J. Electrochem. Soc.* **2000**, *147*, 1306.

6. Wang, Z. L.;Feng, X., "Polyhedral Shapes of CeO2 Nanoparticles." *J. Phys. Chem. B* **2003**, *107*, 13563.

7. Bosbach, D.;Charlet, L.;Bickmore, B.;Hochella, M. F., Jr., "The dissolution of hectorite: In-situ, real-time observations using atomic force microscopy." *American Mineralogist* **2000**, *85*, 1209.

8. Chun, C. L.;Penn, R. L.;Arnold, W. A., "Kinetic and Microscopic Studies of Reductive Transformations of Organic Contaminants of Goethite." *Environ. Sci. Technol.* **2006**, *40*, 3299.

9. Penn, R. L.;Stone, A. T.;Veblen, D. R., "Defects and Disorder: Probing the Surface Chemisry of Heterogenite (CoOOH) by Dissolution Using Hydroquinone and Iminodiacetic Acid." *J. Phys. Chem. B* **2001**, *105*, 4690.

10. Kawaguchi, H.;Ama, T.;Yasui, T., "The Base-Catalyzed Isomerization of the (Iminodiacetato)(N-methyliminodiacetato)cobaltate(III) and Bis(iminodiacetato)cobaltate(III) Ions." *Bull. Chem. Soc. Jpn.* **1984**, *57*, 2422.

11. McArdell, C. S.;Stone, A. T.;Tian, J., "Reaction of EDTA and Related Aminocarboxylate Chelating Agents with Co(III)OOH (Heterogenite) and Mn(III)OOH (Manganite)." *Environ. Sci. Technol.* **1998**, *32*, 2923.

12. Stone, A. T.;Ulrich, H. J., "Kinetics and reaction stoichiometry in the reductive dissolution of manganese(IV) dioxide and cobalt(III) oxide by hydroquinone." *J. Colloid Interface Sci.* **1989**, *132*, 509.

13. Burgisser, C. S.;Stone, A. T., "Determination of EDTA, NTA, and Other Amino Carboxylic Acids and Their Co(II) and Co(III) Complexes by Capillary Electrophoresis." *Environ. Sci. Technol.* **1997**, *31*, 2656.

14. Reijenga, J. C.;Aben, G. V. A.;Verheggen, T. P. E. M.;Everaerts, F. M., "Effect of Electroosmosis on Detection in Isotachophoresis." *J. Chromatogr.* **1983**, *260*, 241.

15. Tsuda, T., "Modification of Electroosmotic Flow with Cetyltrimethylammonium Bromide in Capillary Zone Electrophoresis." *J. high resolut. chromatogr.* **1987**, *10*, 622.

16. Ama, T.;Kawaguchi, H.;Yasui, T., "Preparation and Reaction of the Unsymmetrical-facial Isomers of the Bis(N-alkyliminodiacetato)cobaltate(III) Ions." *Chem. Lett.* **1981**, 323.

17. Yasui, T.;Kawaguchi, H.;Koine, N.;Ama, T., "Stereochemistry and Properties of unsym-fac-, sym-fac-, and mer- (Iminodiacetato)n(N-alkyliminodiacetato)2-ncobaltate(III) (n=0, 1, or 2) Complexes." *Bull. Chem. Soc. Jpn.* **1983**, *56*, 127.

18. Myers, J. C.;Penn, R. L., "Evolving Surface Reactivity of Cobalt Oxyhydroxide Nanoparticles." *J. Phys. Chem. C* **2007**, *111*, 10597.

19. Gas, B.;Stedry, M.;Kenndler, E., "Peak broadening in capillary zone electrophoresis." *Electrophoresis* **1997**, *18*, 2123.

20. Keijser, T. H. d.;Langford, J. I.;Mittemeijer, E. J.;Vogels, A. B. P., "Use of the Voigt Function in a Single-Line Method for the Analysis of X-ray Diffraction Line Broadening." *J. Appl. Crystallogr.* **1982**, *15*, 308.

21. Jolivet, J.-P. *Metal Oxide Chemistry and Synthesis*; John Wiley & Sons Ltd.: West Sussex, England, 2000.

4

Size- and Shape-Dependent Reactivity of Cobalt Oxyhydroxide with Iminodiacetic Acid*

The use of surface-specific reactions to probe reactive surface area is a promising direction in materials research. The work presented herein examines how the kinetics of dissolution can be used to quantify particle growth as well as the evolution of site-specific reactive surface area. The dissolution of heterogenite by IDA results in two geometric isomers of $Co(IDA)_2^-$, the *s-fac* and *u-fac* isomers. The heterogenite particles studied here can generally be described as cylindrical plates, and the relative amount of *s-fac* isomer produced is found to increase as the height of the plates increases. The quantity of each isomer produced is shown to correlate with the relative number of two different types of surface sites, designated as *edge* and *corner* sites, while basal sites are seemingly unreactive. It is hypothesized that *u-fac* isomer results from the more accessible Co centers at the corner sites, while the *s-fac* isomer results from the less accessible edge sites. An empirical relationship is developed between the fraction of *s-fac* isomer produced and the height of the β-CoOOH particles, and this relationship is used to quantify particle growth by analysis of kinetic data. Finally, this new information is used to modify a previously-proposed pH-dependant growth model, resulting in a significant improvement in the fit and physical relevance of the model.

---

* A report on this research project was submitted for publication and is reproduced with permission from Myers, J.C. and Penn, R.L. *Langmuir*, **2010**.

4.1 Introduction

The chemical and physical properties of materials can be strongly dependent on morphology and microstructure. The surface energy of a given particle depends on the number, type, and size of exposed crystallographic facets. Thus, there can be substantial variations in free energy for crystals of the same material with different particle shapes. This can result in pronounced reactivity differences depending on the specific crystal face at which the reaction occurs. For example, variations in interfacial stability lead to preferential cation-exchange at the ends of CdS nanorods.[1] The reaction between chlorinated organic contaminants and Fe^{2+} adsorbed to the surface of goethite (α-FeOOH) has been demonstrated to occur almost exclusively at the {0 2 1} type surfaces of the goethite crystals.[2] The non-oxidative dissolution of galena nanocrystals by hydrochloric acid occurs most rapidly at {1 1 1} and {1 1 0} facets.[3]

Quantifying reactive surface area continues to present a substantial challenge in materials and geochemical research. Typical measurements of surface area rely on either gas adsorption, such as the Brunauer, Emmet, and Teller (BET) method, or geometrical approximation based on microscopy measurements. The surface area determined by gas adsorption may have little relationship to the surface that actually participates in a given reaction.[4,5] Particularly in circumstances where various surface types exhibit differing reactivity, meaningful connections cannot be drawn between BET measurements and reactive surface area. Measuring surface area based on data from electron or scanning probe microscopy has the advantage that different surface types can be more easily distinguished, but it also has several disadvantages. Microscopy-based measurements

must employ a geometric model to relate measurements to surface area, and these models frequently assume idealized geometries and neglect surface roughness. Additionally, these methods often require time-consuming manual processing of hundreds of images in order to achieve statistically relevant measurements. By using probe molecules that demonstrate highly surface-specific reactivity, however, it may be possible to acquire accurate measurement of reactive surface area without specialized apparatus or extensive data processing.

An excellent model system for quantifying reactive surface area is the mineral heterogenite (β-CoOOH) and the tridentate ligand iminodiacetic acid (IDA). When reacted in aqueous solution, the ligand-assisted dissolution of heterogenite produces two geometric isomers of $Co(IDA)_2^-$: the symmetrical facial (*s-fac*) and unsymmetrical facial (*u-fac*) isomers.[6] The relative amount of each isomer produced during the dissolution of heterogenite varies substantially with the preparation method[7] and aging conditions[8] of the heterogenite particles. This variation is a strong indicator that product isomers are formed at the surface of the particles and that the geometry of the products is influenced by the particular site at which dissolution occurs. This system is ideal for this type of investigation for a variety of reasons. Heterogenite is easily synthesized from solutions of inexpensive and readily available precursors, and it can be prepared in a variety of sizes and morphologies with small variations in the synthetic procedure (see chapter 2). The product isomers, low-spin Co^{III} complexes, are non-labile—especially at acidic pH[9]—so rearrangement during analysis is negligible. The system also shows no evidence

of undesirable side reactions, such as reductive dissolution of the heterogenite,[6] which could complicate quantification of reaction products.

This paper presents a significant advancement towards the goal of quantifying reactive surface area using ligand-assisted dissolution. In particular, results tracking the evolution of reactive surface area as a function of crystal growth are present. Samples of heterogenite particles are aged under various conditions, and powder X-ray diffraction (XRD) is used to identify the material as well as detect changes in the crystallinity and relative size of the particles. Transmission electron microscopy (TEM) was used to quantify the morphology and size of heterogenite particles as a function of aging time. Combining the results of ligand-assisted dissolution with characterization by XRD and TEM demonstrates that the relative quantity of each isomer produced is strongly related to the size and shape of the heterogenite particles. This information is used to evaluate the growth kinetics of the particles, and these results are applied to enhance a previous pH-dependent growth model[8] and arrive at physically meaningful constants for the growth rate of heterogenite particles.

4.2 Experimental Methods

All aqueous solutions and suspensions were prepared using Milli-Q purified water (Millipore Corporation, 18 MΩ•cm resistivity). The glass- and plasticware employed were washed in a 0.2 M oxalic acid bath (analytical reagent grade, Mallinckrodt Laboratory Chemicals) and/or a 4 M nitric acid bath (ACS grade, Mallinckrodt), and rinsed several times with Milli-Q water prior to use. All purchased chemicals were used

without further purification. Unless otherwise specified, mixtures were stirred using PTFE-coated magnetic stir bars.

4.2.1 Preparation of Heterogenite Particles

Heterogenite particles [$T_5OH_{fast}H_2O_2$(90 min), see chapter 2] were prepared by oxidizing a suspension of cobalt (II) hydroxide using hydrogen peroxide. Two liters of Milli-Q water were purged with N_2 gas for 1.5 h. The purged water was used to prepare 1.00 L of 3.2 mM Co^{2+} solution (cobalt (II) chloride hexahydrate, AR, Mallinckrodt) and 1.00 L of 6.4 mM OH^- solution (sodium hydroxide, AR, Mallinckrodt). Both solutions were cooled to 5 °C in an ice bath.

The hydroxide and cobalt (II) solution were combined over 2 min and hand shaken to ensure complete and rapid mixing. The resulting cobalt (II) hydroxide suspension remained in the ice bath while being magnetically stirred. After 90 min, 5.00 mL of 1.57 M hydrogen peroxide (30% hydrogen peroxide solution, Mallinckrodt) was added to oxidize the particles, resulting in a color change from pale-blue to brown. This suspension remained in the ice bath for an additional 1.0 h, after which it was removed and buffered to pH 4.6 using a 10. mM acetate buffer prepared using glacial acetic acid (AR, Mallinckrodt) and sodium hydroxide. The buffered suspension was allowed to age at room temperature and was sampled at specified time intervals for use in dissolution experiments. Some data presented herein were collected using additional heterogenite suspensions prepared as described above but with changes to the buffer identity or pH.[8] These samples were buffered to pH 4.1 or 5.8 using acetate buffer, or to pH 5.6 using 10. mM MES buffer (MES hydrate, 99%, Acros Ogranics).

4.2.2 Particle Dissolution

While stirring to ensure homogeneity of the mixture, 50.0 mL aliquots of particle suspension were transferred to 60 mL HDPE bottles. Dissolution of the heterogenite particles was initiated by adding 0.100 mL of 0.10 M IDA (98+%, Acros Organics) to each sample. The suspension was stirred over the course of the reaction. At selected times, 3.0 mL samples were removed from the reaction vessel and quenched by removing the remaining particles via 0.2 μm PTFE membrane syringe filters (Pall Life Sciences). The concentration of each $Co(IDA)_2^-$ isomer was determined using a Beckman Coulter P/ACE MDQ capillary electrophoresis (CE) system equipped with a fused-silica capillary, which had a length of 51 cm (to the detector) and an inner diameter of 73 μm. The separation method was a modification of that employed by Bürgisser and Stone.[10] The background electrolyte consisted of a 50 mM phosphate buffer (pH 7) and 10 mM tetradecyltrimethylammonium bromide (TTAB, 99%, Aldrich) additive. The TTAB performs two functions: first, it adsorbs to the walls of the silica capillary and creates a positively charged layer that results in a reversal of the electroosmotic flow within the capillary.[11,12] Second, the TTAB is present in excess of its critical micelle concentration[13], so surfactant micelles exist within the capillary. Analytes partition in and out of these micelles, resulting in improved separation efficiency.[14] The separations were performed at 10 kV, reversed polarity, and using an injection pressure of 0.5 psi for 10 s. Prior to analysis, a $Co(EDTA)^-$ internal standard was prepared[15] and added volumetrically to each sample to minimize error due to variations in injection volume. The $Co(EDTA)^-$ shows no evidence of ligand exchange with the $Co(IDA)_2^-$ isomers over

the time scale of the separation, has a migration time between that of the two isomers, and is baseline separated from both under the conditions employed. Five-point calibration curves were constructed from authentic standards of *s-fac* and *u-fac* $Co(IDA)_2^-$ prepared using well-established methods[16,17] and spiked with the $Co(EDTA)^-$ standard. Analyses were performed in duplicate or triplicate and average results are presented.

4.2.3 Materials Characterization

The phase and crystallinity of the heterogenite samples were characterized using powder X-ray diffraction (XRD). Samples were prepared from approximately 300 mL of particle suspension. These suspensions were centrifuged at 14000 G, the supernatant was decanted, and the particles were resuspended in Milli-Q water. This washing procedure was repeated three times and then the particles were dried over 4 h under a stream of nitrogen. The dried particles were ground with a mortar and pestle before being packed into a zero-background single-crystal quartz holder.

Diffraction patterns were collected using a PANalytical X'Pert Pro MPD theta-theta diffractometer equipped with a cobalt anode and an X'Celerator detector. Data were collected over a range of 18-90° 2θ in scanning mode with rates of between 7.8 x 10^{-5} and 1.7 x 10^{-3} °2θ /s. The sample phase was confirmed by comparison with the ICDD PDF-2 database. Several of the sample patterns showed irregular backgrounds at low angle, likely as a result of scattering caused by sample roughness. This severely complicated the line-profile fitting of the pattern; thus, reliable single-line size-broadening analyses could not be conducted.

Samples were prepared for transmission electron microscopy (TEM) by centrifuging a small portion of the particle suspension and resuspending in Milli-Q water. This process was repeated 2-3 times to remove soluble salts. The suspension was further diluted and aquasonicated (VWR Aquasonic model 150HT) for 5 min. A single drop of the resulting suspension was placed on a 3 mm copper TEM grid coated with holey carbon film (SPI Supplies) and allowed to air dry. The grids were examined using an FEI Tecnai T12 or FEI Tecnai F30 FEG TEM. Calibrated images were collected using a Gatan charge-coupled device (CCD) camera and Gatan Digital Micrograph software (version 3.8.2), and particles sizes were measured and recorded using ImageJ software.[18]

4.3 Results and Discussion

4.3.1 Particle Characterization

The heterogenite particles examined in this work were aged for a total time of 18 months. Samples were characterized by TEM after 1, 6, and 45 days of aging, as well as at the conclusion of the 18 months. In the TEM images, the particles appear as a mixture of round objects and darker, elongated objects. It has been previously established through sample tilting that the particle morphology is well described as cylindrical plates with the height of the plates parallel to the c crystallographic axis. The darker objects are particles viewed edge on, which provide a convenient method for determining the height of the cylindrical plates. Table 4.1 shows the height and diameter measured from images of each sample, as well as the number of particles measured. The number of particles available for measurement in the 1 day sample was limited; a combination of particle

aggregation and low thickness contrast resulted in few measurement-quality images. Figures 4.1 and 4.2 show representative TEM images of the particles at various times in the aging process. An examination of the measurements in Table 4.1 shows that from day 1 to day 6, the cylindrical particles more than double in height but show a much smaller increase in diameter. In contrast, at later aging times the height change is much less significant: the increase in the final 18 months is less than that over the first 6 days. The diameter, however, continues to expand gradually with aging time.

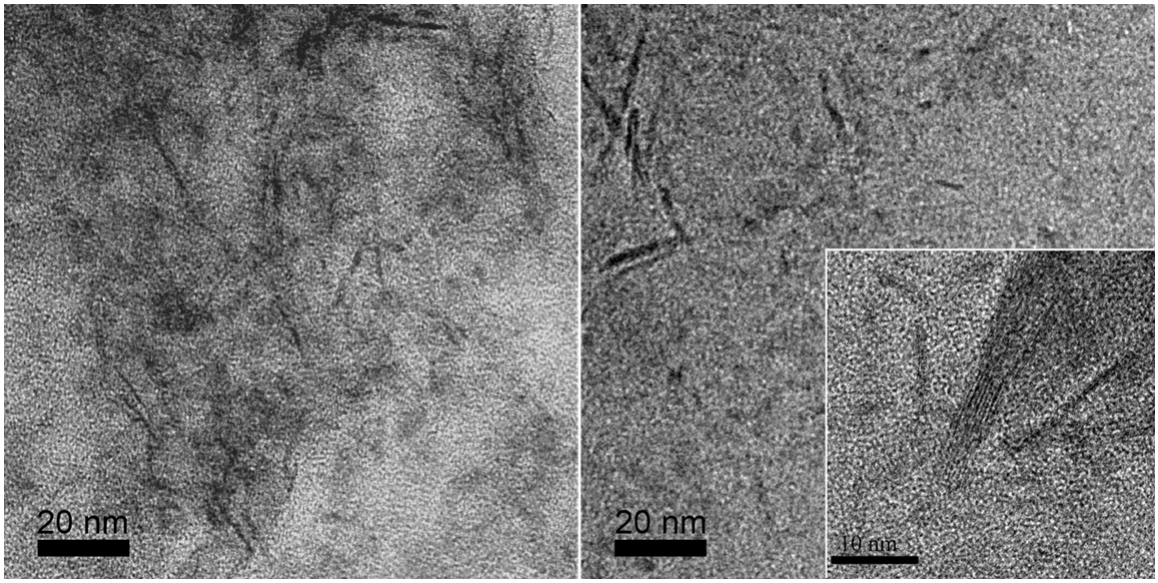

Figure 4.1 TEM images of heterogenite particles during early aging times. The particles were aged for 1 day (left) and 6 days (right) in pH 4.6 acetate buffer. The inset shows a lattice fringe image of a 6 day aged plate viewed edge on.

Table 4.1 Particle Diameter and Height as Determined by TEM

Aging[a] Time	Height[b] (nm)	Number Measured	Diameter[b] (nm)	Number Measured
1 day	1.0 (0.2)	30	4.3 (1.1)	71
6 day	2.5 (0.4)	127	5.4 (1.3)	199
45 day	3.0 (0.5)	119	9.2 (1.8)	316
18 month	3.7 (0.8)	209	11.9 (2.1)	502

[a] Aged in 10 mM acetate buffer at pH 4.6
[b] Parentheses indicate standard deviation of size distribution

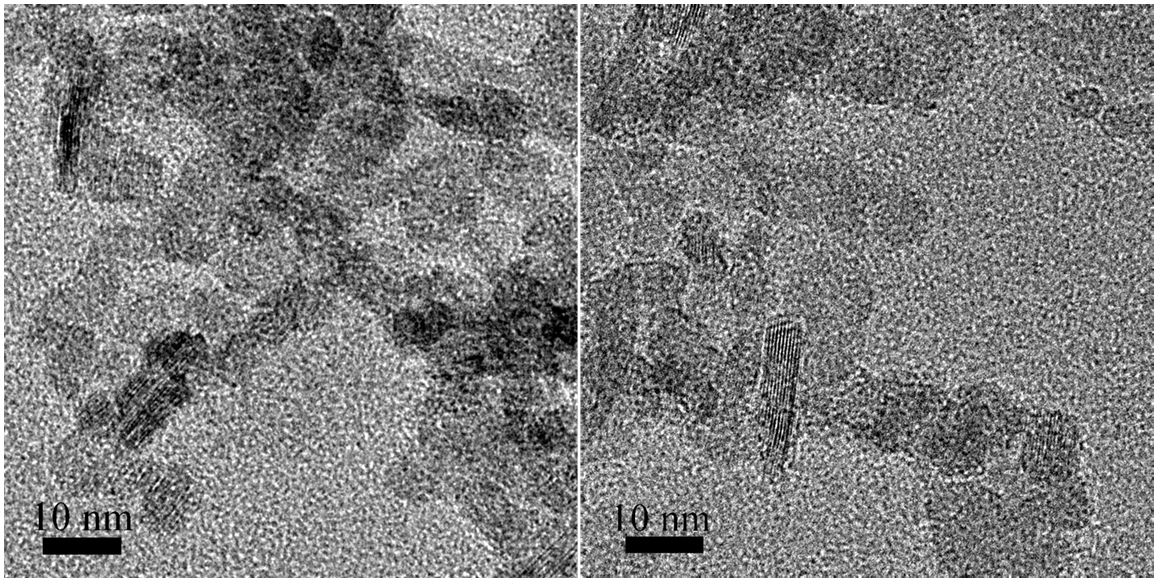

Figure 4.2 TEM images of heterogenite particles after extended aging time. The particles were aged for 45 days (left) and 18 months (right) in pH 4.6 acetate buffer.

X-ray diffraction patterns show that the material produced was heterogenite with no detectable impurities or products of phase transformation. The XRD patterns for samples collected after 3, 8, and 45 days as well as 3 and 18 month of aging are shown in Figure 4.3. The small peak located at approximately 52.5° 2θ in the 45 day and 3 month aged samples is attributable to the aluminum sample holder base. The substantial

increase in preferred orientation in the 18 month sample, evidenced by the comparatively large intensity of the (003) reflection, is likely due to a sample preparation issue: there was very little sample available, so the grinding step was omitted, resulting in a less-random distribution of crystallite orientations. The increased surface roughness of the XRD sample is responsible for the elevated baseline at low angle in that pattern. As the aging time of the particles increases, the most prominent change in the XRD patterns is a narrowing of the peaks, which is indicative of an increase in crystallite size in the direction normal to the diffracting planes that produce each peak. The widths of the (003) and (110) peaks, located at 23.5° and 77.6° 2θ, correspond to the height and diameter of the plates. The very broad (003) peaks at low aging times are consistent with the small (~1 nm) particle heights determined from TEM measurements. With increased aging time, all peaks narrow as the height and diameter of the particles increase.

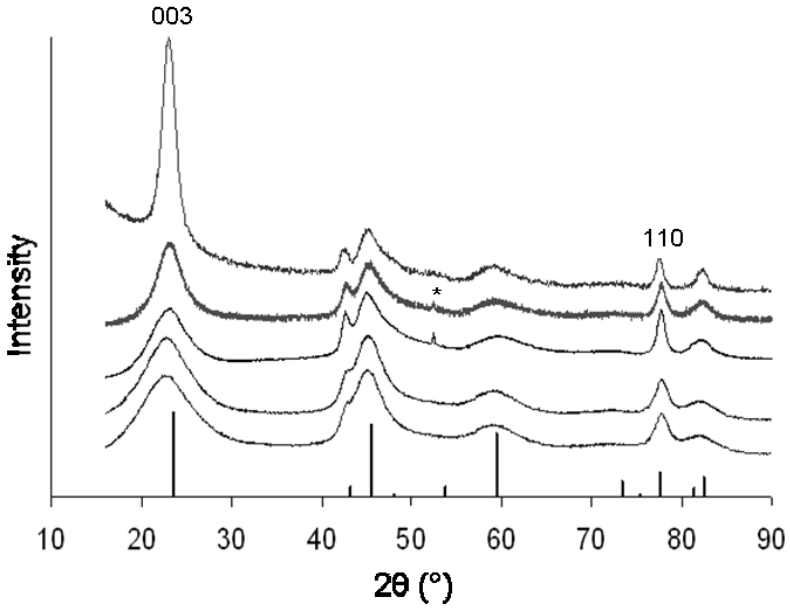

Figure 4.3 XRD patterns of heterogenite particles aged for (from bottom to top) 3 d, 8 d, 45 d, 3 mo., and 18 mo. in pH 4.6 acetate buffer. The peaks marked with an asterisk are attributable to the Al sample holder base. The stick pattern corresponds to the heterogenite PDF (#07-0169).

4.3.2 Dissolution Results

As expected, the dissolution of heterogenite particles by IDA produced a mixture of *s-fac* and *u-fac* geometric isomers of $Co(IDA)_2^-$, and the results of a typical dissolution reaction are presented in Figure 4.4. The relative amount of each isomer produced is essentially constant over the course of the reaction and depends on the suspension aging conditions prior to reaction. The kinetics of heterogenite dissolution by IDA were quantified for particles aged for 3 hours; 3, 7, and 14 days; and 18 months. The data from each dissolution experiment was fit using a pseudo-second-order rate law, and the initial rate of production and relative amount of each isomer were determined (Table 4.2). From these data, it can be seen that increased heterogenite aging time leads to a decrease in the initial rate of dissolution and an increase in the relative amount of *s-fac* isomer produced. The rate of *u-fac* production decreases much more rapidly than that of *s-fac* production, indicating that *u-fac* producing sites are selectively consumed or otherwise rendered unreactive with aging.

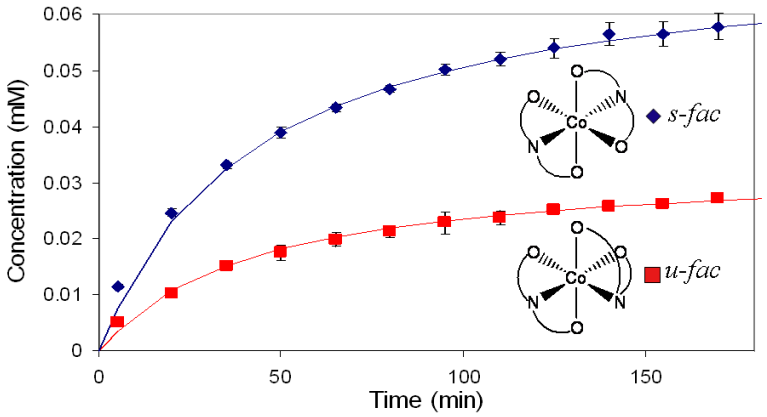

Figure 4.4 Plot of isomer concentration vs. time for a typical reaction between IDA and heterogenite. The structure of the *s-fac* (diamonds) and *u-fac* (squares) isomers are shown as insets. The error bars represent a range of ± one standard deviation from three dissolution trials.

Table 4.2 Initial Reaction Rates and Isomer Production

Aging[a] Time	***s-fac* Rate[b] (mM/min)**	***u-fac* Rate[b] (mM/min)**	**% *s-fac* Isomer[c] Produced**
3 hour	2.2×10^{-3}	2.2×10^{-2}	9.9 ± 0.6%
3 day	2.0×10^{-3}	2.8×10^{-3}	40.3± 0.5%
7 day	1.6×10^{-3}	1.2×10^{-3}	52.7± 0.7%
14 day	1.6×10^{-3}	1.1×10^{-3}	59.4± 0.9%
18 month	1.5×10^{-4}	6.4×10^{-5}	66.6± 0.7%

[a] aged in 10 mM acetate buffer at pH 4.6

[b] determined by 2nd order fit of reaction data

[c] average value of all dissolution data points

To explain the change in relative isomer production as the particle surface evolves, it is necessary to consider both the thermodynamic relationship between the two isomers and the structure of the heterogenite surface. The *s-fac* isomer is thermodynamically favored in aqueous solution. Previous work by Cooke showed that heating a pure sample of either isomer to 80 °C for 0.5 h resulted in a mixture that was 79% *s-fac* isomer.[19] Similarly, allowing pure samples to equilibrate at room temperature and neutral pH for ~1000 h resulted in 77% *s-fac* isomer. All the dissolution reactions produced lower than equilibrium amounts of s-*fac* isomer, so the additional *u-fac* isomer production must arise from kinetic factors, the most likely of which is the surface acting as a template for the products. It has been previously noted that the ratio of $Co(IDA)_2^-$ isomers produced via dissolution seems to depend only on the height of the heterogenite particles;[8] the different isomers, therefore, likely derive from different surface types on the particles. It was hypothesized that the *s-fac* isomer is produced at the edge surfaces

of the plate, while the *u-fac* isomer is produced at the basal surfaces. However, measurements of particle sizes before and after 75% particle dissolution showed no decrease in height,[7] so it is unlikely that significant dissolution occurs on the basal surface of the plates. Further comparison between isomer ratios and different types of surface sites has yielded a substantive refinement to the original hypothesis.

The heterogenite crystal structure comprises layers of edge-sharing CoO_6 octahedra (Figure 4.5). These layers are stacked perpendicularly to the *c*-axis and alternate with layers of protons. This layered structure can be viewed directly in high-resolution TEM images of the particles (Figure 4.6, left). The Co atoms located at the

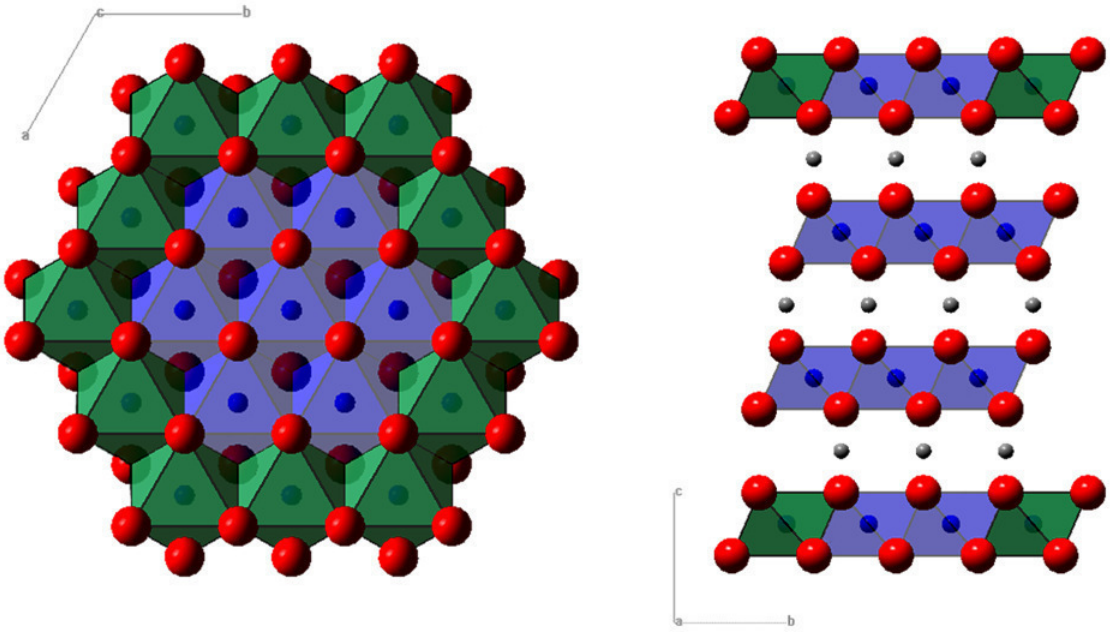

Figure 4.5 Polyhedral representation of the heterogenite crystal structure. The octahedra are edge-sharing CoO_6 units, red spheres are O, blue are Co, and grey are protons. The individual layers extend in the ab-plane and are stacked along the c-axis. Left: basal surface. Right: edge surface. The corner-type sites are highlighted in green.

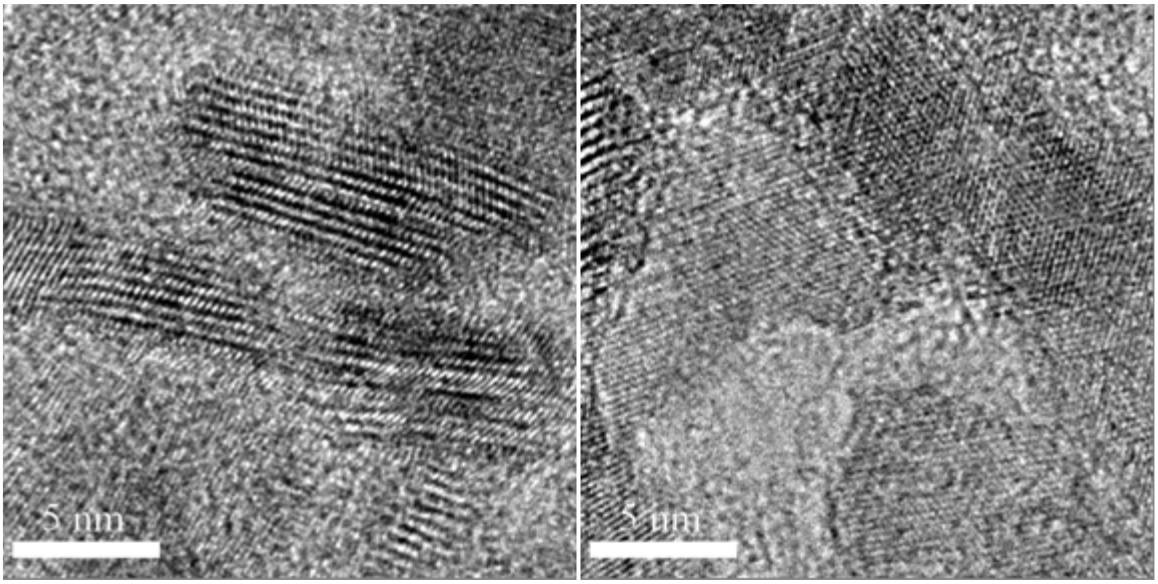

Figure 4.6 High-resolution TEM images of heterogenite particles after 18 months of aging. Left: plates viewed edge on. The alternating dark and light fringes correspond to the 003 crystallographic planes. Right: plates viewed down the *c*-axis.

perimeter of these sheets make up the edge-type surfaces, while the exposed sheets on the top and bottom of the stack make up the basal surface. The Co centers at the edge of the two basal sheets, however, are in a unique arrangement that will hereafter be referred to as *corner* sites (Figure 4.5, right). Comparing the surface area attributable to edge and corner type sites for a given particle height produces a compelling correlation: the ratio of edge to corner surface area closely matches the ratio of *s-fac* to *u-fac* isomers produced by dissolution. As this ratio depends only on the height of the edge surface, it explains why the product ratio is independent of the particle diameter. If the production of *s-fac* isomer is attributed to the edge surface and *u-fac* isomer is attributed to the corner surface, then the relevant surface areas, assuming cylindrical plate geometry, are:

$$SA_s = \pi\, d\, (h - 2l_c) \qquad (4.1)$$

$$SA_u = \pi\, d\, (2l_c + 2l_{ab}) \tag{4.2}$$

where h and d are the height and diameter of the particle, l_c is the Co to Co spacing in the c crystallographic direction (0.4385 nm), and l_{ab} is the Co to Co spacing in the a and b directions (0.2855 nm). The fraction of *s-fac* isomer produced can be calculated by dividing SA_s by the sum of SA_s and SA_u, in which case several factors cancel from the equation, resulting in the diameter-independent equation

$$s = \left(\frac{h - 2l_c}{h + 2l_{ab}} \right) \tag{4.3}$$

with s representing the mole fraction of *s-fac* isomer produced. Conversely, the particle height can then be derived from a known isomer fraction:

$$h = \frac{s\, l_{ab} + l_c}{s - 1} \tag{4.4}$$

Table 4.3 shows measured yield of *s-fac* isomer for heterogenite samples compared with those predicted from equation (4.3) and TEM-measured heights. It is noteworthy that the accuracy of the predictions is consistent across a pH range of 4.1 to 5.8 and in the presence of both acetate and MES buffers.

Table 4.3 Comparison of Predicted and Observed Isomer Production

Aging Conditions[a]	**Height (nm)**	**Predicted % *s-fac***[b]	**Observed % *s-fac***[c]
pH 4.6, 1 day	1.0	7.8%	9.9%
pH 4.6, 6 days	2.5	53%	53%
pH 4.1, 45 days[d]	2.3	50.%	54%
pH 4.6, 45 days	3.0	60.%	62%
pH 5.6 (MES), 45 days[d]	3.4	66%	64%
pH 5.8, 45 days[d]	3.8	67%	69%
pH 4.6, 18 months	3.7	66%	67%

[a] aged in acetate buffer unless otherwise noted

[b] based on TEM measurements and equation (4.3)

[c] average value of all dissolution data points

[d] sample discussed in detail in chapter 3

A structural inspection of the heterogenite basal surface (Figure 4.5, left), reveals a potential explanation for the selectivity of the edge and corner sites. Edge sites are composed of two different types of Co sites: those with three neighboring CoO_6 octahedra and those with four. In both cases, however, there is at least one O atom bound to only a single Co atom. In aqueous suspension, these non-bridging O exist as surface hydroxyl groups that can undergo ligand exchange with the IDA molecules. This would result in a polarization of the remaining Co-O bonds[20] that, combined with the chelate effect from the tridentate IDA, is expected to facilitate the removal of the Co center from the heterogenite crystal. In this process, some of the μ^2 bridging O would be converted into surface OH groups on neighboring Co centers, further promoting ligand exchange at those sites.

In the case of the corner sites, four or five of the six oxygen atoms in the CoO_6 octahedra—1 or 2 terminal O, 2 μ^2 bridging O, and 1 μ^3 O—are exposed to potential attack by an IDA molecule (see Figure 4.5). Once an IDA molecule has complexed the Co center, at least one Co-O bond remains exposed at the interface for displacement by a second IDA. Perhaps more importantly, the corner sites are only bounded by one other layer, removing steric barriers to the approach of the second molecule. At the edge sites, however, at most four O atoms are exposed to the surface; the Co center is bounded by layers both above and below, removing access to the μ^3 bridging oxygen that is available at the corner sites. It is, therefore, likely that the Co center must first be partially liberated from the heterogenite particle before access by a second IDA molecule is possible. It is thus hypothesized that the *u-fac* isomer predominately results from the complexation of Co centers at corner sites by two IDA molecules in a *cis* arrangement and that the *s-fac* isomer results from the partial or complete liberation of a Co-IDA species, which is then rapidly complexed by a second IDA molecule in the thermodynamically-favored *trans* arrangement.

4.3.3 Growth Model

The relationship between isomer production and reactive surface area can be used to examine the changing surface of the particles, and thus particle growth kinetics, by way of dissolution experiments. A previous pH-dependant crystal growth model was proposed using dissolution data,[8] but this model suffered from a dependence on several parameters that could not be directly measured. It was, therefore, not possible to extract physically meaningful information from the results. The model can now be

extended to include equation (4.4) and arrive at a series of new growth curves that not only contain more directly applicable parameters but also provide an improved fit to the experimental data.

This model begins with the assumption that particles rapidly nucleate as very thin cylindrical platelets, which then grow through two different mechanisms: the diameter increases as a result of growth by coarsening, and the height of the plates increases primarily through oriented aggregation across (001) surfaces. This model is supported by the platelets observed in TEM images of particles after 1 day of aging, which are often less than three layers high. The proposed growth mechanism is consistent with the observation that the particle diameter increases steadily with longer aging time, while the particle height has an initial, very rapid increase, followed by a substantial decrease in growth rate and leveling off. This leveling effect is consistent with previous results showing that the kinetics of oriented aggregation decrease with larger crystal sizes.[21] For the model presented here, this decrease is approximated by a linear decay of stacking rate with height

$$\text{rate} = \text{rate}_0\left(1 - \frac{h}{h_{\max}}\right) \tag{4.5}$$

where rate_0 is the growth rate for the initial platelets, h is the average particle height, and $h_{\max}$ is the ultimate height at which growth levels off.

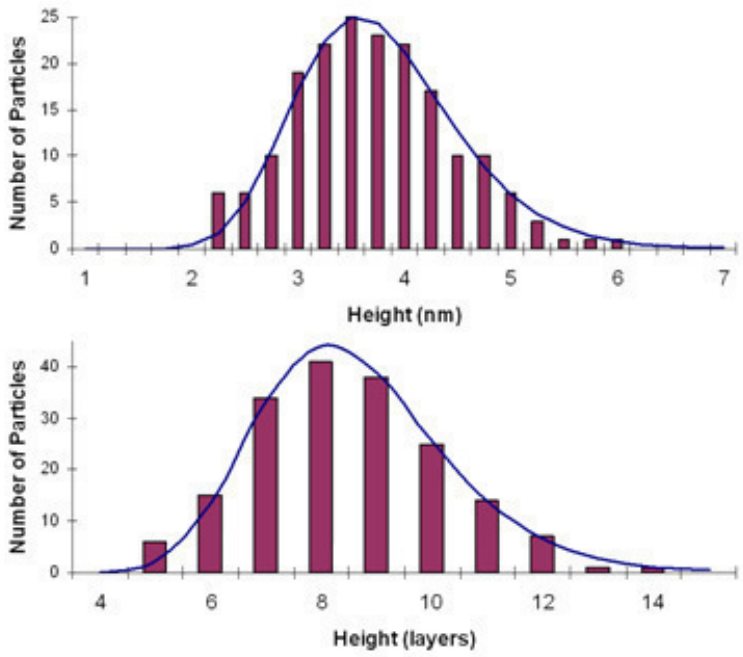

Figure 4.7 Particle height distribution determined from TEM measurements of heterogenite particles aged 18 months in pH 4.6 acetate buffer. Data are binned by nm (top) and number of Co-O layers (bottom). The solid lines represent a fitted log-normal distribution with mean of 3.8 nm (8.6 layers) and standard deviation of 0.76 nm (1.7 layers).

Two further simplifications are incorporated into the model: first, the only mechanism of particle loss (by number) is through oriented aggregation. This ignores any losses occurring due to particles entirely consumed by the coarsening mechanism and decouples the number concentration of particles from the particle diameter. Second, all particles are treated as if they are the mean size of the population. This second point is clearly untrue; however, when measured particle size distributions (example in Figure 4.7) are used in place of mean heights in the model, the average difference in predicted isomer production is less than 1.5%. This, therefore, allows a significant simplification of the model with negligible loss of accuracy.

Based on these assumptions, the decrease in number of free particles can be represented by

$$-\frac{d[\mathrm{P}]}{dt} = k'\left(1-\frac{h}{h_{\max}}\right)[\mathrm{P}]^2 \tag{4.6}$$

where [P] is the number concentration of particles, *k'* is a pseudo-second order rate constant for particle stacking, and *t* is the aging time. At any given time, the number of particles can be expressed as a function of height

$$[\mathrm{P}] = \frac{h_0}{h}[\mathrm{P}]_0 \tag{4.7}$$

$$-\frac{d[\mathrm{P}]}{dt} = \frac{h_0[\mathrm{P}]_0}{h^2}\frac{dh}{dt} \tag{4.8}$$

with $[\mathrm{P}]_0$ representing the number concentration of particles at time $t = 0$, and h_0 being the initial particle height. Due to the assumption that particles are consumed only by stacking, only $[\mathrm{P}]_0$ retains a diameter dependence. Substituting (4.7) and (4.8) into (4.6), integrating, and applying the boundary condition $h(0) = h_0$ yields

$$h(t) = h_{\max} - \left(h_{\max} - h_0\right)\exp\left(k'[\mathrm{P}]_0\frac{h_0}{h_{\max}}t\right) \tag{4.9}$$

This model performs very well when used to fit the growth data from particles aged in pH 4.6 acetate buffer (Figure 4.8). In order to separate the *k'* and $[\mathrm{P}]_0$ variables, equation (4.9) was evaluated as $h(t-1)$, so the h_0 and $[\mathrm{P}]_0$ values could be taken from the day 1 data. The initial concentration of particles, $[\mathrm{P}]_0$, was calculated as 2.0×10^{19} particles/L from the known total [Co] and the measured height and diameter of the particles,

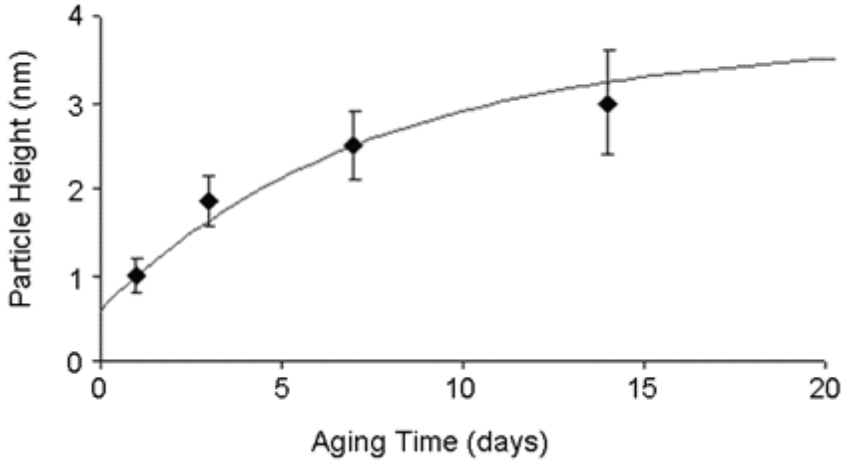

Figure 4.8 Plot of particle height (from TEM measurements) vs. aging time in days for heterogenite particles aged in pH 4.6 acetate buffer. Error bars are one standard deviation of the height distribution in each direction. The line corresponds to a least-squares fit of equation (4.9) to the data, with values of $[P]_0 = 2.0 \times 10^{19}$ particles/L, $h_0 = 1.0$ nm, $h_{max} = 3.7$ nm, and $k = 2.5 \times 10^{-19}$ L/particle·day.

approximating the shape as cylindrical plates, after 1 day of aging. The resulting fit yields a rate constant of $k' = 2.5 \times 10^{-19}$ L/particle·day and particles with a maximum height of 3.7 nm, a close match to the height achieved when the same particles are aged to 18 months. Beyond this, however, by adding a term for pH dependence, the model can be applied to the previously studied heterogenite particles aged at a varying pH values.

$$k' = k[H_3O^+]^n \tag{4.10}$$

where k is the pH-independent rate constant for stacking and n is the order of reaction with respect to hydronium ion concentration $[H_3O^+]$. After substituting (4.10) into (4.9), the parameters h_0, h_{max}, and $k[P]_0$ were refined for particles aged in all three pH buffers. The resulting fits to both % *s-fac* isomer produced and predicted particle height are presented in Figure 4.9. This fit yielded the following values: n = -0.55 (similar to the

-0.62 order obtained using the previous growth model), h_0 = 1.0 nm, h_{max} = 4.1 nm, and $k[P]_0 = 5.7 \times 10^{-4}$ $M^{0.55}$/day.

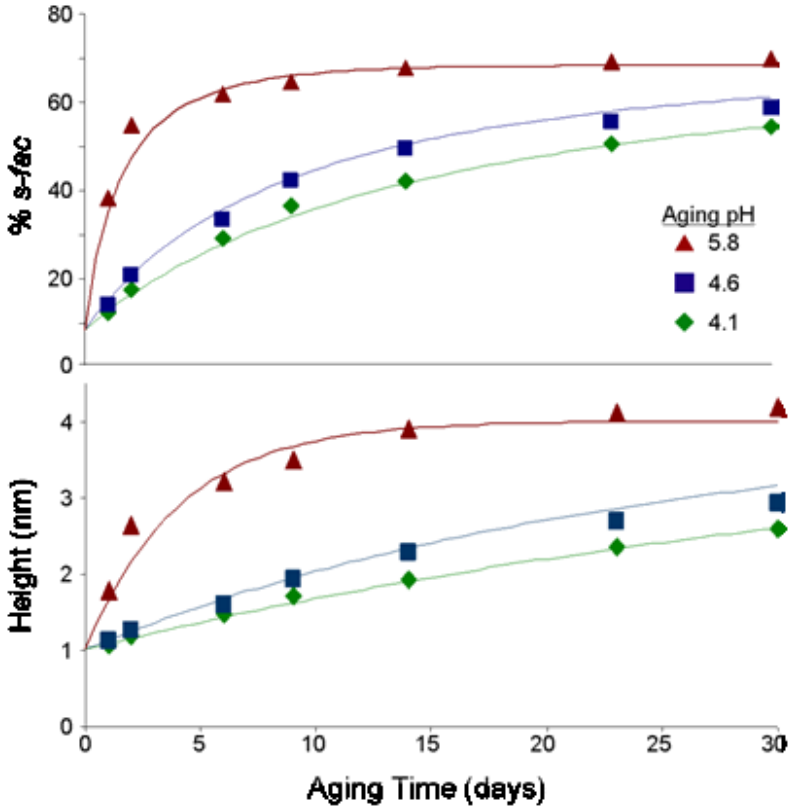

Figure 4.9 Plots of the pH-dependent growth model developed by combining equations (4.9) and (4.10). The data show the evolution of heterogenite particles aged in acetate buffer at pH 4.1 (diamonds), 4.6 (squares) and 5.8 (triangles). Top: plot of *s-fac* isomer yield vs. aging time. Bottom: plot of particle height calculated via equation (4.4) vs. aging time.

4.4 Conclusions

The results presented here further refine the method of employing iminodiacetic acid as a microstructural probe for the reactive surface area of heterogenite particles. It is hypothesized that the *s-fac* isomer is produced by reaction of IDA with the edge-type

heterogenite surfaces and the *u-fac* isomer is produced by reaction with the corner-type surfaces. This is supported by three observations: the relative amount of each isomer produced depends only on particle height, the particle basal surface shows little or no reactivity towards IDA, and the amount of each isomer produced scales with the geometric surface area of the proposed surface type. Comparing predicted and observed *s-fac* fractions using this model results in a match within 5% in all cases examined, with most results within 1-2%. The simplest explanation for this site dependence is that the corner-type sites possess both edge and basal exposed surfaces. Thus, more of the Co-O bonds can be directly displaced by the coordinating groups of the IDA. This allows rapid coordination by two IDA molecules in a *cis* arrangement, resulting in a *u-fac* arrangement of ligands in the product. Conversely, the edge-type sites allow much more limited access to the Co center, so it is likely that they must first be partially or completely liberated from the surface by complexation of a single IDA molecule. The second IDA molecule can then react with the intermediate species to form the favored *s-fac* isomer.

This hypothesis allows both the average height of the heterogenite particles and the ratio of reactive surface area to be determined from the results of an IDA dissolution reaction. When these results are combined with simple particle diameter measurements, such as those obtainable by TEM or XRD, it is possible to arrive at a semi-quantitative description of both reactive and non-reactive surface area. Adding this height relationship to the previously proposed growth model provides a significant enhancement of the original model by both simplifying the model equation and improving the fit to

empirical data. These improved fits further support the hypothesis that the heterogenite particles examined here grow in height through an initial, rapid oriented-stacking process, while the diameter of the plates increases through a more gradual coarsening process. Finally, this refined model can be applied to previously examined pH-dependant aging data. This application provides a pH-independent rate constant for particle growth, and it converges to a theoretical maximum particle height for these conditions that is consistent with the sizes observed for very long aging times. This allows the derivation of growth curves from readily obtainable reaction data, without the need for *ex-situ* characterization or manual particle size measurements.

The use of surface-specific reactions and probe molecules not only allows the quantification of different types of reactive surface area but also provides a convenient method to track the evolution of these surfaces. While the results presented here are specific to the heterogenite–IDA system, the principles should extend to many systems that undergo ligand-assisted dissolution and represents a potentially rich new avenue of materials characterization. A particularly promising direction is the application of these results to environmental nanoparticles, where changes in reactive surface area can have significant implications for the kinetics and selectivity of natural reactions.

4.5 Acknowledgments

We thank the University of Minnesota and the National Science Foundation (Career-036385 and MRI EAR-0320641) for providing financial support for this work. Portions of this work were conducted at the Characterization Facility, University of Minnesota, which receives support from the NSF through the National Nanotechnology Infrastructure Network.

4.6 References

1. Sadtler, B.; Demchenko, D.O.; Zheng, H.; Hughes, S.M.; Merkle, M.G.; Dahmen, U.; Wang, L.-W.; Alivistados, A.P., "Selective Facet Reactivity during Cation Exchange in Cadmium Sulfide Nanorods." *J. Am. Chem. Soc.* **2009**, *131*, 5285.

2. Chun, C.L.; Penn, R.L.; Arnold, W.A., "Kinetic and Microscopic Studies of Reductive Transformations of Organic Contaminants on Goethite." *Environmental Science & Technology* **2006**, *40*, 3299.

3. Liu, J.; Aruguete, D.M.; Jinschek, J.R.; Rimstidt, J.D.; Hochella, M.F., Jr., "The non-oxidative dissolution of galena nanocrystals: Insights into mineral dissolution rates as a function of grain size, shape, and aggregation state." *Geochim. Cosmochim. Acta* **2008**, *72*, 5984.

4. Metz, V.; Raanan, H.; Pieper, H.; Bosbach, D.; Ganor, J., "Towards the establishment of a reliable proxy for the reactive surface area of smectite." *Geochimica et Cosmochimica Acta* **2005**, *69*, 2581.

5. White, A.; Peterson, M. Chemical Modeling of Aqueous Systems II. In *ACS Symposium Series 416*; American Chemical Society: Washington DC., 1990; pp 461.

6. McArdell, C.S.; Stone, A.T.; Tian, J., "Reaction of EDTA and Related Aminocarboxylate Chelating Agents with Co(III)OOH (Heterogenite) and Mn(III)OOH (Manganite)." *Environ. Sci. Technol.* **1998**, *32*, 2923.

7. Penn, R.L.; Stone, A.T.; Veblen, D.R., "Defects and Disorder: Probing the Surface Chemisry of Heterogenite (CoOOH) by Dissolution Using Hydroquinone and Iminodiacetic Acid." *J. Phys. Chem. B* **2001**, *105*, 4690.

8. Myers, J.C.; Penn, R.L., "Evolving Surface Reactivity of Cobalt Oxyhydroxide Nanoparticles." *J. Phys. Chem. C* **2007**, *111*, 10597.

9. Kawaguchi, H.; Ama, T.; Yasui, T., "The Base-Catalyzed Isomerization of the (Iminodiacetato)(N-methyliminodiacetato)cobaltate(III) and Bis(iminodiacetato)cobaltate(III) Ions." *Bull. Chem. Soc. Jpn.* **1984**, *57*, 2422.

10. Burgisser, C.S.; Stone, A.T., "Determination of EDTA, NTA, and Other Amino Carboxylic Acids and Their Co(II) and Co(III) Complexes by Capillary Electrophoresis." *Environ. Sci. Technol.* **1997**, *31*, 2656.

11. Reijenga, J.C.; Aben, G.V.A.; Verheggen, T.P.E.M.; Everaerts, F.M., "Effect of Electroosmosis on Detection in Isotachophoresis." *J. Chromatogr.* **1983**, *260*, 241.

12. Tsuda, T., "Modification of Electroosmotic Flow with Cetyltrimethylammonium Bromide in Capillary Zone Electrophoresis." *J. high resolut. chromatogr.* **1987**, *10*, 622.

13. Lin, C.-E.; Wang, T.-Z.; Chiu, T.-C.; Hsueh, C.-C., "Determination of the Critical Micelle Concentration of Cationic Surfactants by Capillary Electrophoresis." *J. High Resol. Chromatogr.* **1999**, *22*, 265.

14. Terabe, S.; Otsuka, K.; Ichikawa, K.; Tsuchiya, A.; Ando, T., "Electrokinetic separations with micellar solutions and open-tubular capillaries." *Anal. Chem.* **1984**, *56*, 111.

15. Dwyer, F.P.; Gyarfas, E.C.; Mellor, D.P., "The Resolution and Racemization of Potassium Ethylenediaminetetra-acetatocobaltate(III)." *J. Phys. Chem.* **1955**, *59*, 296.

16. Ama, T.; Kawaguchi, H.; Yasui, T., "Preparation and Reaction of the Unsymmetrical-facial Isomers of the Bis(N-alkyliminodiacetato)cobaltate(III) Ions." *Chem. Lett.* **1981**, 323.

17. Yasui, T.; Kawaguchi, H.; Koine, N.; Ama, T., "Stereochemistry and Properties of unsym-fac-, sym-fac-, and mer- (Iminodiacetato)n(N-alkyliminodiacetato)2-ncobaltate(III) (n=0, 1, or 2) Complexes." *Bull. Chem. Soc. Jpn.* **1983**, *56*, 127.

18. Rasband, W.S. ImageJ; U. S. National Institutes of Health: Bethesda, Maryland, USA, http://rsb.info.nih.gov/ij/, 1997-2009.

19. Cooke, D.W., "The Stereochemistry of the Bis Complexes of Cobalt(III) with Iminodiacetic Acid and Methyliminodiacetic Acid." *Inorg. Chem.* **1966**, *5*, 1141.

20. Zinder, B.; Furrer, G.; Stumm, W., "The coordination chemistry of weathering: II. Dissolution of Fe (III) oxides." *Geochimica et Cosmochimica Acta* **1986**, *50*, 1861.

21. Penn, R.L.; Tanaka, K.; Erbs, J., "Size dependent kinetics of oriented aggregation." *J. Cryst. Growth* **2007**, *309*, 97.

5

Insights into the Dissolution of Cobalt Oxyhydroxide by Iminodiacetic acid: Kinetic, Adsorption, and Thermodynamic Studies

The work presented in this chapter continues the investigation into the mechanism of ligand-assisted dissolution of heterogenite (β-CoOOH) particles by iminodiacetic acid (IDA). This dissolution reaction results in a mixture of *s-fac* and *u-fac* $Co(IDA)_2^-$ isomers, and the relative amount of each isomer is a sensitive probe of the reactive surface area of the particles. The adsorption of IDA to the surface is quantified, and the data is fit to a Langmuir isotherm. The heterogenite particles examined are well described as cylindrical plates, and the adsorption data are consistent with IDA adsorbing only to the edge sites—not to the basal surface—of these particles. Kinetic studies at various IDA concentrations and particle loadings reveal that the reaction orders with respect to IDA and surface are approximately 2 and 1.3, but the IDA over is very sensitive to surface saturation. Thermodynamic study reveals a wide difference between the activation energies for *s-fac* and *u-fac* isomers and that both change with aging time. This is attributed to a temperature-dependence for the products from edge-type sites, as well as compensation effects arising from highly T-dependent diffusion and desorption steps. Further evidence for kinetically-significant desorption is provided by particle cycling experiments.

5.1 Introduction

Nanoparticle reactivity is important in many fields, from environmental to industrial and biomedical applications. A high percentage of the atoms in a nanocrystal lie at the particle interface; this makes the particles very reactive in general but may also change the particle reactivity in ways that are different from the bulk material. The exploitation of these unique properties has led to improvements in solar cells,[1,2] batteries,[3,4] and waste remediation,[5,6] among other applications. In many cases, these results depend upon the reactivity of specific surface features or facets,[7,8] and an understanding of how reactivity varies with surface type is essential for further improvement.

The question of how best to measure reactive surface area is a challenging one. Methods of estimating the reactive area include: using all or a fraction of the surface area determined by the BET gas adsorption method,[9,10] calculating geometric area based on particle morphology and size,[11-13] and using probe molecules to selectively measure the desired sites.[14,15] Each of these methods has advantages and disadvantages. BET gas adsorption is a commonly used and relatively simple method of measuring surface area, but it does not typically distinguish between different surface types. Since different surface sites may have dramatically different reactivity, the BET area is insufficient for materials with significant surface heterogeneity. Geometric surface area methods can be used to find the area of specific facets and surface types, but often ignore additional area resulting from porosity and surface roughness. The successful interpretation of geometric surface area requires additional information to ascertain which surface types are reactive.

The probe molecule approach relies on chemical species that react with only specific surface types or features. The products of this reaction can then be used to quantify the surfaces in question. This allows for accurate determination of even rough and porous surfaces, as well as a relatively simple analysis, but it too has a prominent downside: in order to be useful, suitable probe molecules must first be found and methods for their use must be developed. The results promising reactions must first be compared with the geometric area of various surface types to determine exactly what is being measured by the probe reaction.

This work presents a study on the reactions of one such probe molecule, iminodiacetic acid (IDA), with the surface of heterogenite (β-CoOOH) particles. These compounds possess several attractive features that make them desirable as a model system to investigate reactive surface area. The reaction is known to be influenced by the heterogenite surface structure,[14,16] no undesirable side reactions occur, and the products are kinetically inert to ligand exchange, so there is no rearrangement or equilibration of the reaction products before they can be analyzed.

Several relevant details are already known about the heterogenite-IDA system from experiments discussed in chapters 3 and 4 of this work and elsewhere.[14,16,17] The products of the dissolution are two geometric isomers, the *s-fac* and *u-fac* isomers of $Co(IDA)_2^-$. The relative amount of each isomer produced depends upon the synthetic method and aging conditions used to prepare the heterogenite particles prior to reaction. In general longer particle aging time results in a larger amount of *s-fac* produced relative to *u-fac*. This effect is also shown to depend on the pH at which the particles are aged.

Unbuffered particles (pH varies from 7 to 9) display a very rapid increase in *s-fac* isomer production with aging time. Particles that are buffered using acetate or MES buffer evolve more slowly, with the rate of change decreasing at lower pH levels.

The study of this aging effect has led to several conclusions about the connections between particle morphology and the products of the dissolution reaction. First, the heterogenite particles examined here are well described as approximately cylindrical plates, and it was found that the fraction of *s-fac* isomer produced depends strongly on the height of these plates; specifically, increasing height leads to an increase in *s-fac* isomer, but this phenomenon has little or no dependence on the diameter of the plates. Furthermore, transmission electron microscope images of particles before and after dissolution showed no decrease in the height of the particles during reaction.[16] Based on these data, a hypothesis has been proposed attributing the production of each isomer to specific surface types on the heterogenite plates. These surface types (Figure 5.1) consist of the basal and edge surfaces of the plates, as well as a third type, herein referred to as corner sites, that are located at the junction of edge and basal type surfaces. It is currently proposed that the edge sites are primarily responsible for the production of *s-fac* isomer, that corner sites are responsible for the majority of the *u-fac* isomer produced, and that the basal sites do not participate in the dissolution reaction to a significant degree.

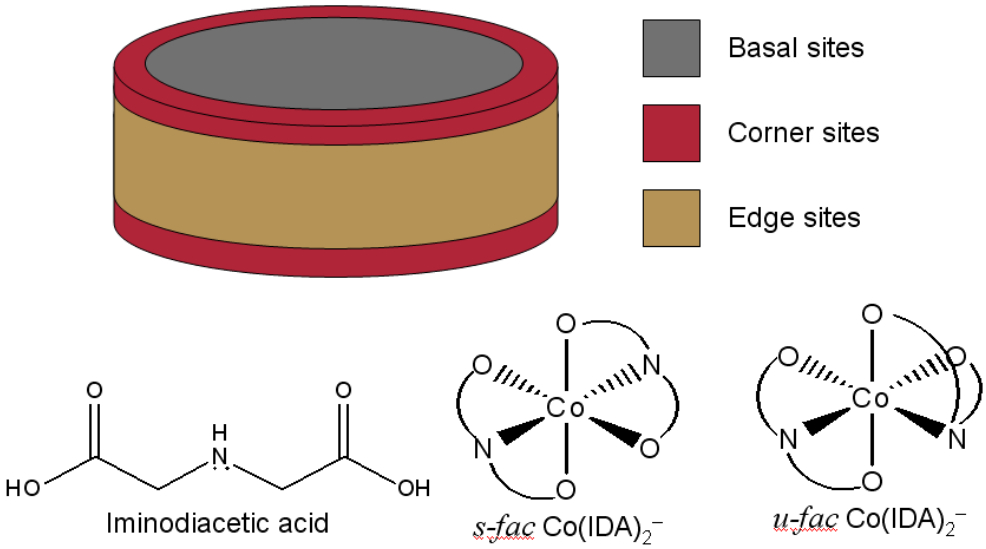

Figure 5.1 Top: representation of heterogenite particles with cylindrical morphology, illustrating the types of surface sites discussed. Bottom: representations of iminodiacetic acid (IDA) and the *s-fac* and *u-fac* isomers of $Co(IDA)_2^-$.

The work presented herein represents a detailed investigation of the mechanism that controls the production of *s-fac* and *u-fac* isomers during heterogenite dissolution. A Langmuir adsorption isotherm is constructed and, from it, values of the area and equilibrium constant for IDA adsorption are obtained. Kinetic studies are used to determine the empirical orders of the dissolution reaction with respect to each of the reactants. These orders help to elucidate the roles of the aqueous and adsorbed forms of IDA in the reaction. Reactions are also performed at several temperatures, yielding information about the activation energy for the production of each isomer. Finally, respiking experiments, where the same particles are reacted with IDA multiple times, provide insight into how the particle surface changes during the reaction.

5.2 Experimental Methods

All solutions were prepared using Milli-Q purified water (Millipore Corporation, 18 MΩ•cm resistivity). All purchased chemicals were analytical reagent grade and used without further purification. Glassware used in these experiments was washed in 0.2 M oxalic acid (Mallinckrodt Laboratory Chemicals) and/or a 4 M nitric acid bath (Mallinckrodt). Glassware was then rinsed several times with Milli-Q water before use. Unless otherwise specified, stirring was performed via PTFE-coated magnetic stir bars.

5.2.1 Preparation of Heterogenite Particles

All heterogenite particles discussed in this chapter were prepared using the method fully detailed in section 4.2.1. Once the particle oxidation was complete, the suspensions remained in an ice bath for an additional 2 h before being removed and allowed to reach room temperature. The aging time and conditions varied substantially between samples, and the exact aging treatments will be specified when the reactions of each set of particles are described. In all cases, the suspensions remained at room temperature until the time of reaction. Unless otherwise noted, the suspension pH was maintained by a 10. mM acetate buffer prepared from glacial acetic acid (Mallinckrodt) with the pH adjusted to 4.6 by addition of sodium hydroxide (Mallickrodt).

5.2.2 Adsorption of IDA

The adsorption of IDA (98+%, Acros Organics) onto the heterogenite surface was measured over a concentration range of 0.05-2.5 mM IDA. Each adsorption sample consisted of a 50. mL aliquot of particle suspension, withdrawn while stirring to maintain

homogeneity and placed in a ~60 mL HDPE bottle, to which was added the appropriate volume of 0.103 M IDA solution for the desired IDA concentration. The difference in final volume caused by the variation in amount of IDA solution added was less than 1 %. The IDA was added rapidly by pipet, and the bottle was hand shaken for ~30 s to mix the IDA and particles. The particles were then removed from the suspension via 0.2 μm PTFE membrane syringe filters (Pall Life Sciences). The short mixing time was necessary to minimize the extent of dissolution that occurred during the experiment. Similar experiments were conducted using N-methyliminodiacetic acid (mIDA, Aldrich), a ligand related to IDA by the addition of a methyl group to the nitrogen atom. Heterogenite is dissolved significantly more slowly by mIDA, so the time before filtering was allowed to range from 10 s to 5 min. These results indicated that 30 s of shaking was sufficient time for adsorption to reach equilibrium. Control experiments conducted without particles present demonstrated that the amount of IDA lost by sorption to the filter membrane was negligible.

The concentration of aqueous IDA in the sample was analyzed by complexation with Cu^{2+} (copper (II) sulfate, Mallinkrodt) to form Cu(IDA), which was then measured by an Agilent 8453 UV-vis spectrophotometer. Cu(IDA) absorbs strongly at 237 nm, but any excess Cu also forms complexes with the acetate and water in the buffer solution. As these complexes also absorb at 237 nm (albeit significantly less strongly), this complicates the analysis. To overcome this problem, the IDA concentration was determined by spectrophotometric titration. A microliter pipet was used to add 25-150 μL (depending on the IDA concentration) increments of 1.00 mM Cu^{2+} solution to the

IDA solution. After each addition, the absorbance at 237 nm was recorded. The aqueous IDA concentration was determined from a plot of absorbance vs. Cu added, as explained in section 5.3.1. The higher concentration (>0.6 mM) IDA samples required dilution with additional buffer to keep the absorbance within the accurately measureable range.

5.2.3 Kinetics and Thermodynamics of Heterogenite Dissolution

Heterogenite particles were reacted with IDA under a variety of different conditions. In each reaction, 50.0 mL aliquots of particle suspension were placed in 60. mL HDPE bottles. Some reactions were carried out at elevated or depressed temperatures; these reaction vessels were placed in ice or water baths, and the temperature was controlled by a recirculating water heater/chiller. Dissolution was initiated by adding 0.10 M IDA to each bottle to reach the desired IDA concentration. The reaction vessels were continuously stirred during dissolution. At various times throughout each reaction, 3.0 mL samples were removed from the reaction vessel for analysis; the reactions were quenched by removing the heterogenite particles via 0.2 μm PTFE membrane syringe filters.

The concentrations of the $Co(IDA)_2^-$ isomers were quantified using capillary electrophoresis (CE, Beckman Coulter P/ACE MDQ) and a modification of a separation method employed by Bürgisser and Stone.[18] The full description of the separation method can be found in section 4.2.2 of this document.

5.3 Results and Discussion

5.3.1 Adsorption of IDA on β-CoOOH

The adsorption equilibrium constant (K_{ads}) for IDA on the surface of heterogenite was determined by allowing IDA to equilibrate with the surface of heterogenite particles that had been aged for 1.5 years in pH 4.6 acetate buffer. The particles that had been aged the longest were selected to minimize the chance of any changes to the particle surface over the course of the adsorption experiments. The high aging time does, however, mean that the available surface area, and thus the maximum adsorption, will be lower than for any other sample considered.[14] The aqueous IDA concentration ($[IDA]_{aq}$) was measured via spectrophotometric titration with Cu^{2+} solution, which was added in increments. The absorbance at 237 nm was measured after each increment, and the results were displayed in a modified Beer's Law plot. Beer's law is usually written as

$$A = bc\varepsilon \tag{5.1}$$

where A is the absorbance, b is the path length through the solution, c is the analyte concentration, and ε is the molar absorptivity of the analyte. Because the titration occurred in a small (~3 mL) cuvette, each addition of Cu^{2+} solution caused a non-trivial change in the total solution volume. To simplify the analysis, equation (5.1) was multiplied by the volume, giving

$$A\,V = bn\varepsilon \tag{5.2}$$

where V is the total solution volume and n is the number of moles of analyte. The resulting data were plotted in the form of $A{\cdot}V$ versus moles of Cu^{2+} added (Figure 5.2).

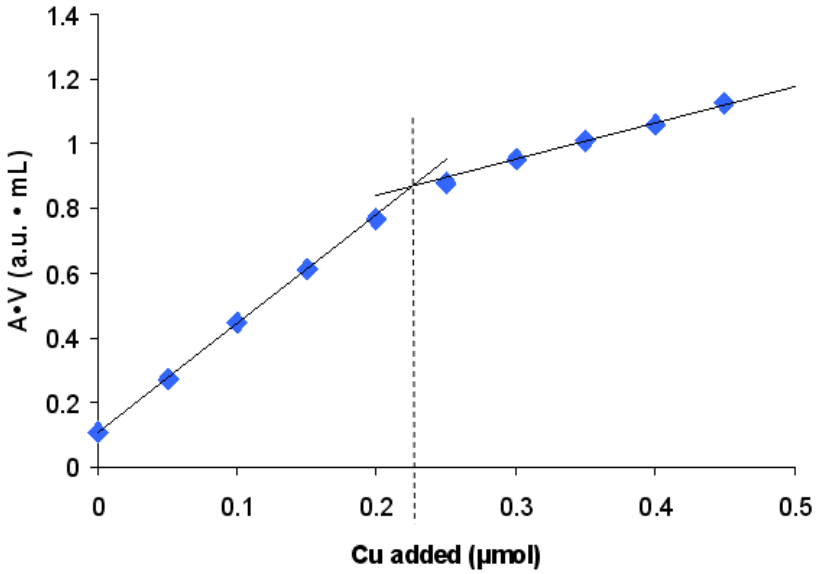

Figure 5.2 Example results from a spectrophotometric titration of IDA with Cu^{2+}. The first linear region corresponds to Cu(IDA) formation, the second to Cu-buffer complex formation. The intersection of the lines indicates the endpoint of the titration (marked by the dashed line).

The plots consist of two linear regions: the first linear region corresponds to the formation of the Cu(IDA) complex; the second corresponds to the formation of additional complexes with the buffer solution. Control experiments showed that no Cu-acetate complex forms when excess IDA is present. This is as expected from the higher affinity of metal ions for chelating ligands such as IDA. As the amount of free IDA approaches zero, however, a mixture of Cu(IDA) and Cu-buffer complexes form, and the plot begins to curve. If the linear portions of each region are extended, the equivalence point of the titration can be found from the intersection of the two linear fits. The number of moles of free IDA present in the original solution can be determined from the amount of Cu^{2+} added. The concentration of adsorbed IDA ($[IDA]_{ads}$) was calculated by difference from the initial IDA concentration.

The concentrations of free and adsorbed IDA at each total IDA concentration were used to construct a Langmuir adsorption isotherm, according to

$$[\mathrm{IDA}]_{\mathrm{ads}} = [\mathrm{IDA}]_{\mathrm{max}} \frac{K_{ads}[\mathrm{IDA}]_{\mathrm{aq}}}{1 + K_{ads}[\mathrm{IDA}]_{\mathrm{aq}}} \tag{5.3}$$

where $[\mathrm{IDA}]_{\mathrm{max}}$ is the maximum possible adsorbed concentration. A plot of $[\mathrm{IDA}]_{\mathrm{ads}}$ versus $[\mathrm{IDA}]_{\mathrm{aq}}$ was constructed, and the values of K_{ads} and $[\mathrm{IDA}]_{\mathrm{max}}$ were determined by least-squares regression of the data (Figure 5.3). The resulting values are given in Table 5.1. This procedure was repeated using N-methyliminodiacetic acid (mIDA) in place of IDA. Like IDA (Figure 5.1), mIDA is a tridentate ligand, but mIDA has a methyl group in place of the amine proton on the central N atom. The determined values of K_{ads} and $[\mathrm{mIDA}]_{\mathrm{max}}$ for mIDA are also reported in Table 5.1.

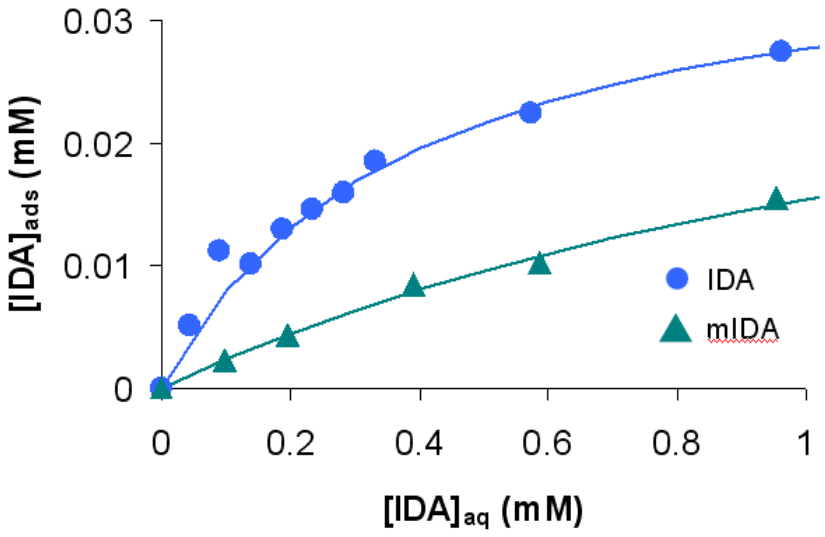

Figure 5.3 Equilibrium adsorption data for IDA and mIDA on the surface of heterogenite particles. The solid lines show fits of the Langmuir adsorption isotherm equation using the values listed in Table 5.1.

Table 5.1 Results of Fitting Adsorption Data for IDA and mIDA to the Langmuir Equation.

	IDA	**mIDA**
K_{ads} (mM^{-1})	2.7	0.66
$[IDA]_{max}$ (mM)	0.038	0.039

Comparing the adsorption results for IDA and mIDA reveals information about the adsorption mechanism. The similarity in the adsorption maximum suggests that the nature of the adsorption, and specifically which sites the molecules adsorb to, is the same for both molecules. The N atom is in the center of the molecule and, thus, there could be considerable steric hindrance for its approach to the surface in both IDA and mIDA molecules. At the pH conditions employed, the particle surface is also likely to be positively charged. These two factors make it likely that at least one of the carboxylate groups attaches to the surface before the N atom. The substantial decrease in K_{ads} for mIDA, however, indicates that the N atom is directly involved in the adsorption and that the presence of the methyl group makes adsorption less favorable. This is most likely a result of additional steric factors: the methyl group in mIDA is substantially larger than the H atom in IDA, and it could interfere with the N coming close enough to the surface to form a coordinate bond with a Co center. This evidence, when combined with the low probability of the N serving as the initial attachment point to the surface, suggests that IDA/mIDA adsorb in a bi- or tridentate fashion.

In order to further interpret the adsorption study results, possible IDA adsorption footprints and the area occupied by each Co center were calculated. The surface area (SA) per IDA molecule was calculated from the value of $[IDA]_{max}$ and the geometric

surface area of the particles, as calculated from measurements of transmission electron microscope images (the details of this measurement are provided in chapter 4). The heterogenite particles were modeled as cylinders with a diameter of 11.9 nm and a height of 3.7 nm, and the nanoparticles were assumed to have a density of 4.93 g/cm^3 (the value calculated from the crystallographic unit cell). Possible adsorption areas were then calculated by assuming that the IDA molecules adsorbed to the entire particle surface, to only the edge surface, or to only the basal surface and dividing the relevant geometric SA by the number of IDA molecules present. These calculated adsorption areas are presented in Table 5.2.

Table 5.2 Calculated Adsorption Area[a] of IDA on Heterogenite Surfaces

	All surfaces	**Basal surface only**	**Edge surface only**
SA / IDA molecule (Å^2)	114	70	44

[a] calculated using $[IDA]_{max}$ from adsorption studies and geometric SA of the specified surface(s) as determined from TEM measurements.

The adsorption area of acetic acid is usually given as 21 Å^2/molecule,[19] and many other molecule similar in size to IDA are reported as having areas of 25-50 Å^2/molecule.[20-22] Based on these sizes, and the observation that there is no evidence of dissolution reaction on the basal surfaces,[16] it is likely that the IDA molecules adsorb mostly to the edge surfaces of the heterogenite particles. Finally, the area that each Co center occupies on the surface was calculated, using the inter-atom spacings from the unit cell: 7.1 Å^2 per Co on basal surfaces, 13.5 Å^2 per Co on edge surfaces. Because the adsorption area of each IDA molecule is larger than the area of a single Co atom, there

will be unoccupied Co centers near the adsorbed molecules. This means that there will always be unoccupied surface sites adjacent to an adsorbed IDA; as movement to an adjacent site is the most common method of surface diffusion, this would facilitate transport of adsorbed IDA across the surface.[23,24]

5.3.2 Analysis of Reaction Kinetics

A kinetic investigation of a reaction can reveal many details of the reaction mechanism, such as number and identities of the species involved in the rate-determining step and the influence of other factors such as pH. The results presented in this work involve analyzing the difference in reaction rates when the concentration of IDA or of surface sites is changed. In order to make a comparison of reactions with different conditions, however, it is necessary to extract measurements of reactivity that are independent of the variables changed. To this end, all kinetic data presented here has been analyzed via the model described below.

The relative amount of each $Co(IDA)_2^-$ isomer produced has been shown to depend upon the morphology and size of the heterogenite particles;[14,16] therefore, the rate of production for each isomer also depends upon the particle characteristics. Because the mechanism of this production is not fully understood, it is useful to have a kinetic model that does not depend on any particular hypothesis about the origins of the isomers. Because the *s-fac* or *u-fac* arrangement of the IDA molecules in $Co(IDA)_2^-$ is only determined when both molecules have attached to the Co center, and the amount of each isomer depends on the nature of the particles, the reaction must occur while both the IDA

molecules are adsorbed or in close proximity to the particle surface. The rate laws were, therefore, formulated as

$$\frac{d[s-fac]}{dt} = k'_s\,[\mathrm{IDA}]_{\mathrm{eff}}^2 \tag{5.1}$$

$$\frac{d[u-fac]}{dt} = k'_u\,[\mathrm{IDA}]_{\mathrm{effs}}^2 \tag{5.2}$$

where k'_s and k'_u are the pseudo-second order rate constants for formation of *s-fac* and *u-fac* isomers, t is time, and $[\mathrm{IDA}]_{\mathrm{eff}}$ is the concentration of IDA capable of participating in the reaction (likely the adsorbed IDA). The ratio of products produced is assumed to remain constant over the course of the reaction and is denoted by

$$r = \frac{[s-fac]}{[u-fac]} \tag{5.3}$$

There are four important assumptions in this formulation: adsorption of the IDA molecules and desorption of the products are not kinetically significant; $[\mathrm{IDA}]_{\mathrm{eff}}$ scales linearly with total IDA concentration; the total number of surface sites remains constant over the course of the reaction; and, by inclusion of the same $[\mathrm{IDA}]_{\mathrm{eff}}$ term in both rate laws, it is implied that the reactive IDA molecules are not distinct prior to reaction. There is some justification for each of these assumptions. The results of the adsorption study show that the time scale for adsorption to reach equilibrium (< 30 s) is significantly faster than the time required for measurable amounts of product to appear (5-30 minutes, depending on reaction conditions), so adsorption is unlikely to contribute to the rate-limiting step. The concentrations of IDA used in most experiments (0.05-0.20 mM) is within the linear region of the absorption isotherm (Figure 5.3), and the particles reacted

were typically much smaller than those used to construct the isotherm, so they have increased surface area and should demonstrate linear adsorption to even higher concentrations. During typical reactions, approximately 6 % of the Co in a heterogenite sample is consumed, so it is reasonable to assume that the change in the number of surface sites is very small. The results of the adsorption study also suggest that adsorption occurs on edge sites, with one IDA occupying one Co center, so all adsorbed IDA should be equivalent. Further, unoccupied adjacent sites allow for facile diffusion of adsorbed IDA across the surface, which is consistent with the hypothesis that the rate-determining step involves the meeting of two adsorbed IDA molecules.

Integration of (5.1) and (5.2) requires that the $[IDA]_{eff}$ term be expanded to

$$[IDA]_{eff} = [IDA]_0 - 2[s\text{-}fac] - 2[u\text{-}fac] \tag{5.5}$$

where $[IDA]_0$ is the value of $[IDA]_{eff}$ at time $t = 0$. Using the product ratio r, as defined in (5.3), allows $[IDA]_{eff}$ to be expressed in terms of only *s-fac* or *u-fac* isomer

$$[IDA]_{eff} = [IDA]_0 - 2[s\text{-}fac]\left(1+\frac{1}{r}\right) \tag{5.6}$$

$$[IDA]_{eff} = [IDA]_0 - 2[u\text{-}fac](1+r) \tag{5.7}$$

Substituting (5.6) and (5.7) into (5.1) and (5.2) and integrating gives

$$[s-fac] = \frac{[IDA]_0}{2\left(1+\frac{1}{r}\right)} - \left(4\left(1+\frac{1}{r}\right)^2 k'_s\, t + \frac{2\left(1+\frac{1}{r}\right)}{[IDA]_0}\right)^{-1} \tag{5.8}$$

$$[u-fac] = \frac{[IDA]_0}{2(1+r)} - \left(4(1+r)^2 k'_u\, t + \frac{2(1+r)}{[IDA]_0}\right)^{-1} \tag{5.9}$$

In this case, t = 0 represents the onset of the dissolution reaction. Because adsorption/desorption is assumed to be rapid with respect to dissolution, $[IDA]_0$ has a non-zero value, which represents the result of a rapid equilibrium between reactive IDA and free IDA molecules before the onset of dissolution. Because the total reactive surface area of heterogenite particles can not be easily quantified, true concentrations of adsorbed IDA are not available for most samples. Therefore, the values of $[IDA]_0$, k'_s and k'_u for a particular reaction were determined by simultaneously minimizing the unsigned mean error between the values of equations (5.8) and (5.9) and the experimental data. The values of r and $[IDA]_0$ were constrained to be equal for both isomers. To allow comparison between reactions using different reaction conditions, the time-derivative rate equations (5.1) and (5.2) were evaluated at t = 0 to determine initial rates, which served as a basis for comparison.

5.3.3 Kinetics at Variable IDA Concentration

To determine the effect of changing the IDA and surface site concentrations on the reactivity of the IDA-CoOOH system, dissolution experiments were performed using different initial IDA concentrations and different particle loadings. The data were fit to the kinetic model discussed in 5.3.2 and the initial rates were determined for *s-fac* and *u-fac* isomer production in each reaction. The initial rates were plotted against the IDA concentration or solid loading to determine the order with respect to each reactant, according to an empirical rate law equation

$$\frac{d[\mathrm{I}]}{dt} = k[\mathrm{IDA}]^{m}[\mathrm{S}]^{n} \qquad (5.10)$$

where [I] is the concentration of the isomer in question, *k* is the rate constant, [IDA] is the total IDA concentration, [S] is the concentration of surface sites, and *m* and *n* are the empirical reaction orders with respect to [IDA] and [S].

The order with respect to the surface was determined first. The particle suspension used in this reaction was prepared as described previously and aged unbuffered for approximately 3 days. The suspension was buffer to pH 4.6 using acetate buffer immediately before reaction. The original particle suspension was diluted to create suspensions with loadings of 0.5 and 0.2 times the original concentration (1.6 mM Co^{3+} as heterogenite), and each suspension was reacted with 0.2 mM IDA. The surface site concentration was assumed to be proportional to the number of particles present and, thus, dilution by a factor of 2 should result in half the original surface site concentration. Based on TEM measurements of particles prepared under similar conditions, these particles are estimated to have approximately 15-20 times the specific surface area of the particles used for the adsorption study. The determined initial rates are shown in Table 5.3, and Figure 5.4 shows the resulting fit for the 0.5x loading sample.

Table 5.3 Kinetic Data from Variable Particle Loading Reactions

Particle Loading	***s-fac* rate$_0$ (mM/min)**	***u-fac* rate$_0$ (mM/min)**	**total rate$_0$ (mM/min)**
0.2x	4.18×10^{-4}	2.45×10^{-4}	6.63×10^{-4}
0.5x	1.37×10^{-3}	8.36×10^{-4}	2.21×10^{-3}
1.0x	3.11×10^{-3}	2.05×10^{-3}	5.16×10^{-3}

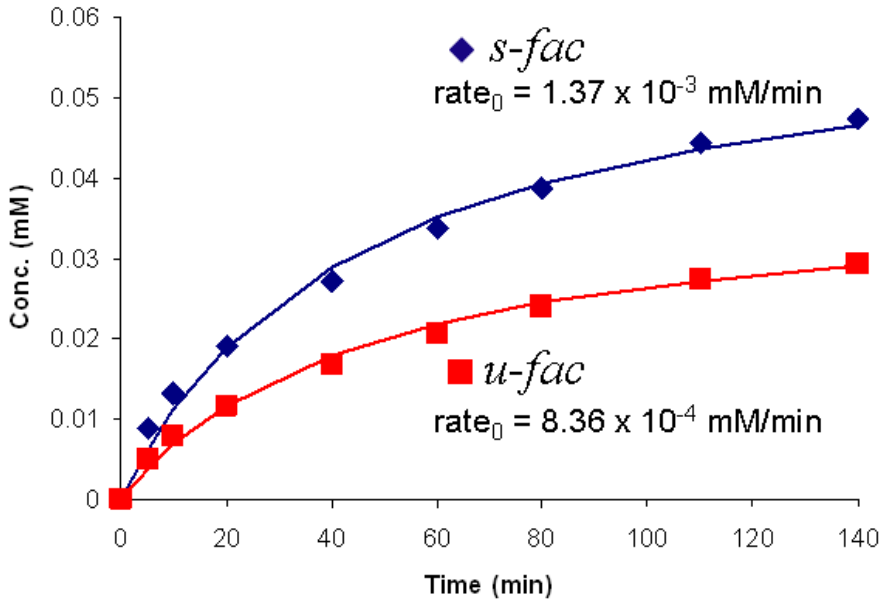

Figure 5.4 Results from dissolution of the 0.5x heterogenite particle loading sample with 0.2 mM IDA. The solid lines represent fittings of equations (5.8) and (5.9) to the data. The $rate_0$ values are derivatives at time zero for each isomer.

Comparing the production rate for each isomer with the particle loading factor revealed that the reaction has an order with respect to surface sites of 1.25 for *s-fac* and 1.32 for *u-fac* isomers. The rate of total $Co(IDA)_2^-$ production, denoted as total $rate_0$ in Table 5.3, had an order of 1.28 with respect to [S]. This increase from the expected order of 1.00 implies that the correspondence between total [IDA] and $[IDA]_{eff}$ is not linear. This most likely means that, despite the increase in surface area compared to the adsorption study, adsorption is not fully linear in this concentration range. Furthermore, because of the second-order dependence on [IDA], deviations from linear adsorption would be magnified. A second possible explanation arises because, although only a single Co center directly participates in the reaction, it is expected that the reaction occurs between two adsorbed IDA molecules diffusing across the surface. Therefore, even

within the linear adsorption range, an increase in Co sites may contribute a greater rate increase than a simple first-order dependence on site concentration, thereby legitimately elevating the surface order.

To determine the order with respect to IDA, reactions were carried out at concentrations of 0.05, 0.10, 0.20, and 0.40 mM IDA. In this case, the reactions were performed on two separate particle samples. The first was the same sample used in the particle loading study after an additional 6 days of aging in acetate buffer (for a total of 9 days aging: 3 days unbuffered and 6 days at pH 4.6). The second sample was buffered to pH 4.6 at 3 h after the completion of the synthesis and reacted approximately 16 h later. The results of these reactions are presented in Table 5.4.

Table 5.4 Kinetic Data from Variable [IDA] Reactions

	[IDA] (mM)	***s-fac* rate$_0$ (mM/min)**	***u-fac* rate$_0$ (mM/min)**	**total rate$_0$ (mM/min)**
19 h aged	0.05	4.0×10^{-4}	1.2×10^{-3}	1.5×10^{-3}
(3 h unbuffered)	0.10	1.0×10^{-3}	2.9×10^{-3}	3.9×10^{-3}
(16 h pH 4.6)	0.20	5.6×10^{-3}	1.6×10^{-2}	2.1×10^{-2}
	0.40	1.8×10^{-2}	4.7×10^{-2}	6.5×10^{-2}
9 day aged	0.05	6.5×10^{-4}	4.2×10^{-4}	1.1×10^{-3}
(3 d unbuffered)	0.10	1.8×10^{-3}	9.9×10^{-4}	2.8×10^{-3}
(6 d pH 4.6)	0.20	2.9×10^{-3}	1.5×10^{-3}	4.4×10^{-3}
	0.40	3.3×10^{-3}	1.7×10^{-3}	5.0×10^{-3}

Several differences are immediately apparent between the two sets of particles. First, the total rate of $Co(IDA)_2^-$, shown as "total $rate_0$" in Table 5.4, is significantly higher for the particles aged 19 h. This is related to the surface area of the particles: the younger particles are smaller, so there is more accessible surface per volume of particles. There is also a substantial difference in the relative amounts of each isomer produced. The younger particles produce mostly *u-fac* isomer, while the particles aged for longer produce mostly *s-fac* isomer. This is consistent with the relationship between reactivity and aging discussed in chapters 3 and 4.

When attempts are made to determine the empirical reaction order for IDA via a log-log plot, a third difference becomes apparent (Figure 5.5). The 19 h particles give a fairly linear plot, yielding a reaction order of 1.88 for the *s-fac* isomer and 1.83 for the *u-fac* isomer. The 9 d particles, on the other hand, are far from linear and give reaction orders that vary from 1.4 to 0.15. It is hypothesized that these particles, through a combination of coarsening growth and random aggregation, have far less surface area than anticipated and the surface is approaching saturation at the higher IDA concentrations. This would explain the diminishing effect of doubling the [IDA]: as the particles near saturation, only very small increases in $[IDA]_{ads}$ are observed for large increases in total [IDA]. It is also possible that either desorption is kinetically significant, in contrast to previous expectations, or that there is some non-trivial relaxation step between release of the product and generation of a new reactive site. Either of these possibilities would mean that reactive surface could become depleted as the reaction progresses, causing an apparent saturation at lower-than-expected IDA concentration.

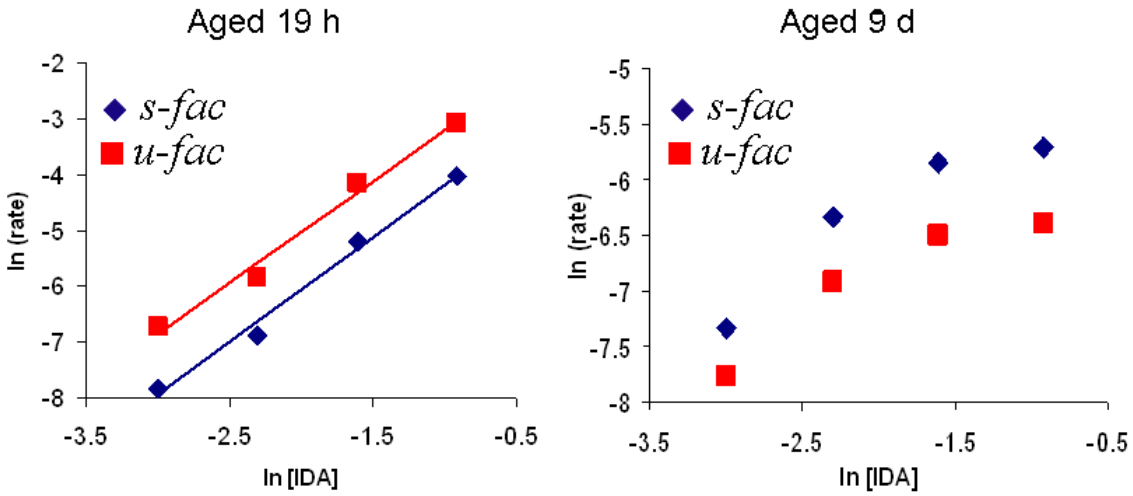

Figure 5.5 Log-log plots of [IDA] vs. $rate_0$ for each isomer. Left: reactions with particles aged 19 h (3 h unbuffered, 16 h pH 4.6 acetate). The solid lines are best fit lines with slopes of 1.83 for *u-fac* and 1.88 for *s-fac* isomer. Right: reactions with particles aged 9 d (3 d unbuffered, 6 d pH 4.6 acetate). The curving of the plot indicates a decrease in the fraction of reactive IDA molecules as total [IDA] increases.

5.3.4 Thermodynamics

The energetics of the dissolution reaction can do much to reveal mechanistic details. To investigate the thermodynamic relationships involved, a fresh batch of heterogenite particles was synthesized and aged in pH 4.6 acetate buffer for 24 days. Using samples aged for 3, 7, 14, and 24 day, dissolution reactions were carried out at four temperatures, and the kinetic data from each individual reaction were collected and analyzed using the methods discussed above. The rate information from the different temperatures and aging times is presented in Table 5.5. These data were then analyzed according to the Arrhenius equation

$$k = Ae^{-E_a/RT} \tag{5.11}$$

where k is the rate constant, A is the pre-exponential factor, E_a is activation energy for the reaction, R is the gas constant, and T is absolute temperature. Because the same particles

Table 5.5 Kinetic Data from Variable Temperature Reactions

Aging Time	Temperature (°C)	*s-fac* rate$_0$ (mM/min)	*u-fac* rate$_0$ (mM/min)	total rate$_0$ (mM/min)	% *s-fac*
3 days	5.2	3.5×10^{-4}	1.1×10^{-3}	1.4×10^{-3}	25%
	22.5	2.1×10^{-3}	3.3×10^{-3}	5.4×10^{-3}	40%
	29.8	4.9×10^{-3}	5.0×10^{-3}	9.9×10^{-3}	47%
	38.2	8.6×10^{-3}	7.4×10^{-3}	1.6×10^{-2}	53%
7 days	3.4	3.1×10^{-4}	5.2×10^{-4}	8.3×10^{-4}	37%
	21.7	1.8×10^{-3}	1.6×10^{-3}	3.5×10^{-3}	53%
	31.8	4.3×10^{-3}	2.7×10^{-3}	7.0×10^{-3}	60%
	37.5	6.7×10^{-3}	3.5×10^{-3}	1.0×10^{-2}	64%
14 days	3.4	3.7×10^{-4}	4.9×10^{-4}	8.6×10^{-4}	43%
	22	2.0×10^{-3}	1.4×10^{-3}	3.4×10^{-3}	59%
	30	3.3×10^{-3}	2.0×10^{-3}	5.2×10^{-3}	64%
	39.3	6.2×10^{-3}	2.9×10^{-3}	9.1×10^{-3}	69%
24 days	2.2	2.9×10^{-4}	3.0×10^{-4}	5.9×10^{-4}	45%
	22.8	1.7×10^{-3}	8.1×10^{-4}	2.5×10^{-3}	62%
	30	2.4×10^{-3}	1.1×10^{-3}	3.5×10^{-3}	66%
	39.7	4.5×10^{-3}	1.8×10^{-3}	6.3×10^{-3}	71%

and [IDA] are used for the experiments at each aging time, the rate$_0$ values are used in place of k in equation (5.11). If ln(rate$_0$) is plotted against $1/T$, the activation energy and pre-exponential factor can be determined from the slope and intercept of the resulting graph (Figure 5.6). The resulting activation energies will be independent of the surface concentration term, which is incorporated into the rate; differences in reactive surface area among the particles will, however, manifest themselves in the intercept of the graph, and thus in the value of A.

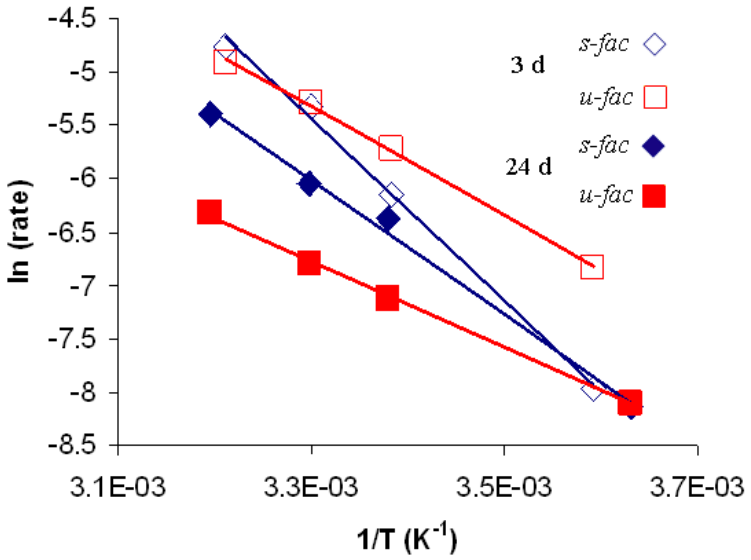

Figure 5.6 Arrhenius plot constructed from temperature-dependent reactions of IDA with heterogenite at aging times of 3 days and 24 days. Similar plots were constructed for 7 and 14 days of aging time but were omitted here for clarity.

The activation energy (Table 5.6) decreased steadily with aging for both the *s-fac* and *u-fac* isomers. A decline in activation energy of dissolution is most commonly attributable tom either of two factors: a decrease in stability of the crystal structure or the reaction proceeding through a new, lower energy transition state. There is no evidence of a change in transition state with aging, and it is improbable that the heterogenite crystal structure would become less stable with aging time; therefore, the difference is most likely not a change in the true activation energy, but rather in other temperature-dependent factors in the reaction. This implies that the dissolution reaction has multiple kinetically-significant steps.[25] Therefore, the combination of all the temperature-dependent factors will influence the apparent activation energy, while the apparent pre-exponential factor will include all temperature-independent factors in the reaction. It is

hypothesized that the temperature-rate correlation results primarily from two factors: the mobility of IDA molecules on the surface and the geometry of IDA attachment to Co centers. A third factor, the equilibrium constant for adsorption (K_{ads}), is also a temperature-dependent aspect of the reaction. However, as K_{ads} should not depend on aging time, it is unlikely to contribute to the change in activation energy with aging.

Table 5.6 Summary of Arrhenius Parameters

Aging time (days)	***E_a* (*s-fac*) (kJ/mol)**	***A* (*s-fac*) (mM/min)**	***E_a* (*u-fac*) (kJ/mol)**	***A* (*u-fac*) (mM/min)**	***E_a* total (kJ/mol)**	***A* total (mM/min)**
3	71.3	8.87×10^9	42.3	9.44×10^4	53.3	1.44×10^7
7	64.7	5.27×10^8	40.0	1.92×10^4	52.6	7.20×10^6
14	56.4	1.75×10^7	35.6	2.64×10^3	47.1	6.98×10^5
24	52.3	2.53×10^6	33.5	6.79×10^2	44.8	1.90×10^5

The first factor is the simpler of the two: the mobility of adsorbed IDA molecules. An increase in temperature imparts higher kinetic energy to the entire system, which means that some or all of the coordinating arms of IDA are more likely to detach from a Co center. This, in turn, means that they are more likely to reattach to a different Co atom on the surface, resulting in an increase in movement across the particle. If, as is proposed, the dissolution requires the meeting of two adsorbed IDA molecules at a single Co center, then increasing the mobility will cause a general increase in reaction rate. As aging time increases, the amount of available surface area decreases. This means that, all else being equal, an IDA molecule has a greater chance of meeting a second molecule, somewhat negating the need for enhanced mobility on the surface. This would cause a

decrease in the temperature dependence of the reaction with aging, which would appear as a decrease in apparent E_a for both isomers.

The second factor is the geometry of IDA attachment to Co centers on the surface. It was proposed in chapter 4 that *u-fac* isomer results from corner-type sites while *s-fac* results from edge-type sites. The results of the thermodynamic study, however, suggest a likely modification of this hypothesis. The formation of *u-fac* isomer at corner sites is based largely on the crystal structure acting as a sort of template for isomer formation. If a single IDA molecule is adsorbed to a corner-type Co, the most accessible locations for a second IDA to adsorb will most likely place the N atoms in a *cis* arrangement (see discussion in section 4.3.2). When an edge-type Co atom is complexed by a single IDA, however, the site is less accessible to approach by a second IDA. An edge-type Co site has, at most, 4 bonds to (hydr)oxo groups that are accessible to replacement by IDA. This means that a second IDA molecule most likely attaches by only a single carboxyl arm before the Co is partially liberated from the crystal structure. In this case, the geometry of the final product is not directed by the surface, but rather is determined by the attachment of the other two coordinating groups of the second IDA molecule. The *s-fac* isomer is thermodynamically favored,[26] but attachment of the N atom at a *cis* site, yielding a *u-fac* product, is statistically favored at lower temperatures. This implies that the edge sites, rather than generating exclusively *s-fac* product, are responsible for a distribution of products that favors *u-fac* at low temperature and *s-fac* at higher temperature. Unfortunately, the instability of uncomplexed Co^{3+} in aqueous solution

makes it difficult to directly measure the effect of temperature on the isomer ratio in the absence of a particle surface.

A temperature-dependence in the distribution of products generated at edge sites would also explain the large disparity between the activation energies of the *s-fac* and *u-fac* isomers. The rate of reaction for both isomers will increase as temperature increases, but this effect will be modified by the change in the amount of each isomer produced from edge sites. The *s-fac* isomer would experience both the standard temperature-dependent rate increase and an additional increase because more of the reactive sites are producing *s-fac* isomers. The effect of this combination will be to inflate the temperature dependence of the *s-fac* isomer, causing the apparent activation energy to be significantly higher than the true E_a. The *u-fac* isomer experiences the opposite effect; the increased temperature raises the reaction rate, but a decrease in the number of *u-fac* isomers produced at edge sites works in opposition to this increase. This reduces the magnitude of the *u-fac* temperature dependence and, thus, produces a deflated apparent activation energy.

The change in the pre-exponential factor, A, with aging has a relatively clear explanation, depending upon two factors: surface area and adsorption. The prefactor is also known as the frequency factor, because it scales with the number of collisions that potentially result in successful reaction. A decrease in A implies that there are fewer collisions between the reactants. The most obvious cause for A to decrease with aging is that the growth of the heterogenite particles results in less surface area with time. This decrease in surface area translates to fewer reactive sites and lowers the prefactor.

Another explanation that possibly contributes to the changes in A is the so-called compensation effect that is commonly found in heterogeneous reactions.[25,27-30] In systems, that exhibit a compensation effect, E_a and A are found to vary together, such that a change in one compensates for the increase or decrease in rate caused by the change in the other. There are many potential explanations for this effect, but in general they arise from situations where several steps contribute to the reaction rate, and one or more secondary processes (in this case adsorption and diffusion of the IDA molecules) have stronger temperature dependences than the primary reaction process[29,30] (liberation of the Co center). These secondary processes are highly affected by surface coverage, and thus show a large change in the activation energy with aging. The fundamental process, however, is not significantly changed, so the rate changes less than might be predicted from the temperature-dependence. The value of A, calculated from equation (5.11), reflects this difference between the T-dependent rate and the true rate, resulting in compensating behavior.

5.3.5. Particle Cycling

As a method of determining what changes occur at the heterogenite surface during the course of a dissolution reaction, two different particle-cycling experiments were undertaken. In each case, particles were reacted with 0.2 mM IDA as per the standard reaction procedure but, after a given time, the amount of IDA was refreshed and the results of this second spike of IDA were compared to those from the previous addition. The first experiment used newly synthesized particles: they were buffered to pH 4.6 1 h

after completion of the synthesis and reacted 14 h later. The second batch of particles were aged without buffer for 25 days and buffered immediately before reaction.

During dissolution of the younger particles (aged for 15 h), the IDA concentration was renewed to 0.2 mM after every 80 min of reaction time. This amounted to each IDA spike being 85-95% consumed before the next spike was added. The resulting kinetic data were plotted as both true concentration (Figure 5.7) and an adjusted concentration, which subtracted the amounts of each isomer at the time of each spike addition. These adjusted concentration plots were used to determine initial rates for each IDA addition. The reaction of the older particles (aged 25 d) was undertaken somewhat differently. The [IDA] remained 0.2 mM, but in this case 24 h passed between initiation of the first and second IDA spikes. Results from each spike were also kinetically analyzed using adjusted concentration plots. The resulting rate data for both sets of particles are shown in Table 5.7.

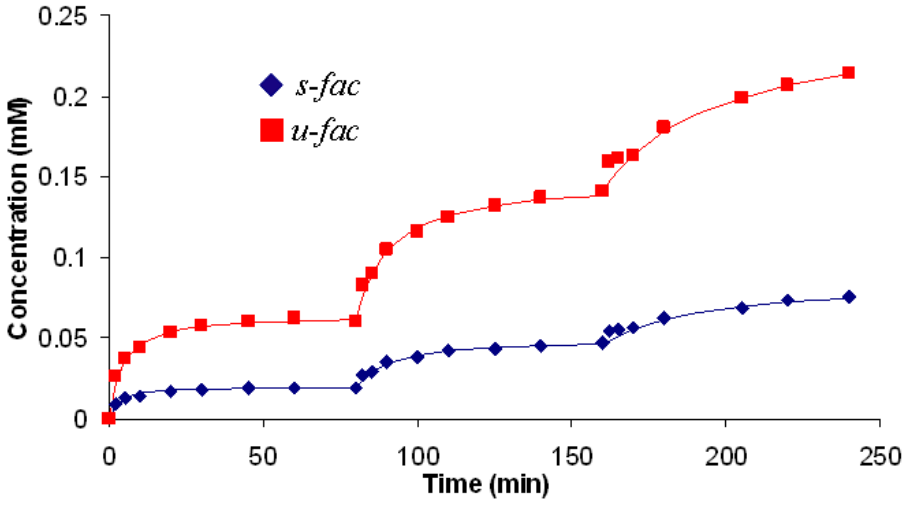

Figure 5.7 Results from the young particle cycling experiments. The solid lines represent fittings of equations (5.8) and (5.9) to the data.

Table 5.7 Kinetic Data from Particle Cycling Experiments

Aging time	IDA Spike #	*s-fac* rate$_0$ (mM/min)	*u-fac* rate$_0$ (mM/min)	total rate$_0$ (mM/min)	% *s-fac* produced
	1	6.9×10^{-3}	1.8×10^{-2}	2.4×10^{-2}	24.6 ± 0.9
19 h	2	3.0×10^{-3}	8.6×10^{-3}	1.2×10^{-2}	25.9 ± 0.6
	3	1.3×10^{-3}	2.8×10^{-3}	4.1×10^{-3}	28.5 ± 0.5
25 d	1	8.5×10^{-4}	3.0×10^{-4}	1.2×10^{-3}	74.7 ± 0.5
	2	8.9×10^{-4}	3.1×10^{-4}	1.2×10^{-3}	74.6 ± 0.7

The younger particles showed a steady decrease in the rate of $Co(IDA)_2^-$ production with each additional IDA spike. The second spike saw the rate drop by approximately half, while the third spike experienced another decrease of nearly two-thirds. The total amount of IDA added over all three spikes, 0.6 mM, would be expected to dissolve just under 20 % of the heterogenite sample. The decrease in surface area over the course of the reaction is, therefore, a possible explanation, although it cannot account for the magnitude of the rate decreases. This suggests that reaction changes the surface and depletes the reactive sites. When these results are compared to those of the older particles, it is seen that no similar decline in reactivity occurs. While it is possible that this results from the aging difference in the particles, the older particles should have less available surface area, and therefore see a greater effect from any change to the surface. The possibility also exists that the young particles continue growing during the reaction, causing this difference. The size of the rate decrease, however, is much greater than would be expected from any potential change in size over 4 h (see Chapter 4).

Therefore, it appears that the decrease in reaction rate only occurs when the subsequent spikes occur soon after the each other, not when there is a substantial delay between spikes. This leads to the hypothesis that there is a time delay between the loss of a Co center from the surface and the generation of a new reactive site to replace it. This could be explained by either a kinetically significant $Co(IDA)_2^-$ desorption step or by the requirement for some sort of structural rearrangement before the newly exposed Co sites become reactive towards IDA. There is some evidence for desorption in this case, in that the younger particles consistently show a similar amount of IDA that is unaccounted for at the end of the reaction: 0.033 mM after the first spike, 0.024 mM after the second, and 0.023 mM after the third. Based on the adsorption isotherm shown in Figure 5.3 and the certainty that the young particles have higher surface area than the older particles (aged 18 months) used for the isotherm determination, it seems unlikely that this concentration is sufficient to saturate the particle surface. However, the $Co(IDA)_2^-$ isomers are substantially larger than IDA molecules, and little is known about their adsorption characteristics. It is possible that this amount is sufficient to passivate a large portion of the heterogenite surface.

5.4 Conclusions

The study presented in this chapter clarifies several points about the reaction between IDA and heterogenite. First, an adsorption isotherm was prepared for IDA using the Langmuir equation. This isotherm allows predictions about the amount on IDA present on the heterogenite surface at any time. The adsorption is very rapid—

equilibrium can be reached before any reaction product is detected—and, under the reaction conditions typically employed, less than 10% of the total IDA is adsorbed to particle surfaces. It was also determined, based upon a calculation of adsorption area and comparison with mIDA, that IDA adsorbs in a bi- or tridentate fashion on Co centers, and that it adsorbs only to the edges of the cylindrical heterogenite plates.

Reactions with variable concentrations of IDA and surface sites were used to determine orders for the empirical reaction rate law. The order with respect to surface sites was found to be higher than 1.00. It is believed that, although only a single Co site is involved in the actual production of a $Co(IDA)_2^-$ molecule, the increased order reflects the secondary importance of other Co sites in the adsorption and surface diffusion of the IDA molecules. The order with respect to IDA was found to vary considerably with the reaction conditions: at most, it was 1.88 and at least it was 0.15. The order decreased at higher IDA concentrations and for particles with less available surface area, suggesting that the surface was becoming saturated. As the particle surfaces near saturation, the concentration of adsorbed IDA would become independent of the aqueous concentration, shifting the apparent reaction order towards zero.

Thermodynamic reactions were performed on heterogenite particles at 4 different aging times. Each set of reactions consisted of dissolving the particles at 4 temperatures between 3 and 40 °C. These reactions revealed several important details about the reaction. The apparent activation energy of the *s-fac* isomer was significantly higher than that of the *u-fac* isomer. There was also a substantial increase in the percentage of *s-fac* isomer as temperature increased. To explain these observations, it is proposed that the

edge sites, rather than producing primarily *s-fac* products, produce a temperature dependent amount of each isomer. This could be explained by the liberation of edge-type Co as $Co(IDA)^+$ or a related compound, which then reacts with another surface-bound or near-surface IDA molecule to form a $Co(IDA)_2^-$ isomer. In this case, the arrangement would be controlled by the thermodynamics of attachment rather than the surface itself. The *s-fac* isomer is the more thermodynamically favored, so higher temperature would increase the proportion of *s-fac* produced by edge-type sites. Thus, at low temperature, both the corner and edge sites contribute to the production of *u-fac* isomer, while at high temperature the corner sites alone produce it. If this hypothesis is correct, the greatest surface sensitivity would be obtained at the highest reaction temperature that does not introduce interconversion of the aqueous isomers, because at this limit corner-type Co would form exclusively *u-fac* isomer and edge-type would form exclusively *s-fac*.

Finally, particle cycling experiments revealed that there is a steady decrease in the reactivity of the surface with subsequent spikes of IDA, but that this decrease is not permanent; particles respiked with IDA 24 h after the initial reaction showed no sign of the rate decrease. This suggests that either the desorption of $Co(IDA)_2^-$ is slow, preventing the regeneration of surface sites, or that there is a non-trivial relaxation step required between the release of one Co site and the previously interior sites beneath it becomes reactive. Further testing of these hypotheses require that cycling experiments be conducted on identical particles with varying delays between the addition of fresh IDA. The presence of a relatively slow $Co(IDA)_2^-$ desorption step could be assessed by

monitoring both the free IDA and isomer concentrations over the course of a dissolution reaction.

These results have significantly increased the amount that is understood about the heterogenite-IDA dissolution reaction, but there are additional avenues to explore. Several of the reactions should be repeated under conditions of very low IDA concentration to completely eliminate the possibility of surface saturation. Further particle cycling tests are necessary to further clarify the nature of site regeneration. And finally, variable temperature reactions should be carried out on particles with a both very high and very low proportion of edge-type sites in order to quantify how this affects the temperature-dependence of the *s-fac* to *u-fac* ratio. But even without additional data, it is clear that the amounts of *s-fac* and *u-fac* isomers produced depend on several different surface factors and that, with a well designed assessment procedure, the probe molecule method could potentially evaluate them all.

5.5 Acknowledgments

We thank the University of Minnesota and the National Science Foundation (Career-036385 and MRI EAR-0320641) for subsidizing the work presented here.

5.6 References

1. Yang, X.; Loos, J.; Veenstra, S.; Verhees, W.; Wienk, M.; Kroon, J.; Michels, M.; Janssen, R., "Nanoscale morphology of high-performance polymer solar cells." *Nano Letters* **2005**, *5*, 579.

2. Baxter, J.; Aydil, E., "Nanowire-based dye-sensitized solar cells." *Applied Physics Letters* **2005**, *86*, 053114.

3. Poizot, P.; Laruelle, S.; Grugeon, S.; Depont, L.; Tarascon, J.-M., "Nano-sized transition-metal oxides as negative-electrode materials for lithium-ion batteries." *Nature* **2000**, *407*, 496.

4. Wang, Z.; Huang, X.; Chen, L., "Characterization of Spontaneous Reactions of LiCoO with Electrolyte Solvent for Lithium-Ion Batteries." *Journal of the Electrochemical Society* **2004**, *151*, A1641.

5. Zhang, W.; Elliott, D., "Applications of iron nanoparticles for groundwater remediation." *Remediation Journal* **2006**, *16*, 7.

6. Liu, W., "Nanoparticles and their biological and environmental applications." *Journal of bioscience and bioengineering* **2006**, *102*, 1.

7. Lee, I.; Morales, R.; Albiter, M.A.; Zaera, F., "Synthesis of heterogeneous catalysts with well shaped platinum particles to control reaction selectivity." *Proceedings of the National Academy of Sciences* **2008**, *105*, 15241.

8. Verziu, M.; Cojocaru, B.; Hu, J.; Richards, R.; Ciuculescu, C.; Filip, P.; Parvulescu, V., "Sunflower and rapeseed oil transesterification to biodiesel over different nanocrystalline MgO catalysts." *Green Chemistry* **2008**, *10*, 373.

9. Brosse, E.; Magnier, C.; Vincent, B., "Modelling Fluid-Rock Interaction Induced by the Percolation of CO2-Enriched Solutions in Core Samples: the Role of Reactive Surface Area." *Oil & Gas Science and Technology* **2005**, *60*, 287.

10. Zhang, L.; Lüttge, A., "Morphological evolution of dissolving feldspar particles with anisotropic surface kinetics and implications for dissolution rate normalization and grain size dependence: A kinetic modeling study." *Geochimica et Cosmochimica Acta* **2009**, *73*, 6757.

11. White, A.F.; Brantley, S.L., "The effect of time on the weathering of silicate minerals: why do weathering rates differ in the laboratory and field?" *Chemical Geology* **2003**, *202*, 479.

12. Brantley, S.L.; Mellot, N.P., "Surface area and porosity of primary silicate minerals." *American Mineralogist* **2000**, *85*, 1767.

13. Hodson, M., "Does reactive surface area depend on grain size? Results from pH 3, 25° C far-from-equilibrium flow-through dissolution experiments on anorthite and biotite." *Geochimica et Cosmochimica Acta* **2006**, *70*, 1655.

14. Myers, J.C.; Penn, R.L., "Evolving Surface Reactivity of Cobalt Oxyhydroxide Nanoparticles." *J. Phys. Chem. C* **2007**, *111*, 10597.

15. Washton, N.; Brantley, S.; Mueller, K., "Probing the molecular-level control of aluminosilicate dissolution: A sensitive solid-state NMR proxy for reactive surface area." *Geochimica et Cosmochimica Acta* **2008**, *72*, 5949.

16. Penn, R.L.; Stone, A.T.; Veblen, D.R., "Defects and Disorder: Probing the Surface Chemisry of Heterogenite (CoOOH) by Dissolution Using Hydroquinone and Iminodiacetic Acid." *J. Phys. Chem. B* **2001**, *105*, 4690.

17. McArdell, C.S.; Stone, A.T.; Tian, J., "Reaction of EDTA and Related Aminocarboxylate Chelating Agents with Co(III)OOH (Heterogenite) and Mn(III)OOH (Manganite)." *Environ. Sci. Technol.* **1998**, *32*, 2923.

18. Burgisser, C.S.; Stone, A.T., "Determination of EDTA, NTA, and Other Amino Carboxylic Acids and Their Co(II) and Co(III) Complexes by Capillary Electrophoresis." *Environ. Sci. Technol.* **1997**, *31*, 2656.

19. Aloko, D.; Afolabi, E.A., "Titanium Dioxide as a Cathode Material in a Dry Cell." *Leonardo Electronic Journal of Practices and Technologies* **2007**, *11*, 97.

20. Peeters, M.P.J., "An NMR Study of MeTMS Based Coatings Filled with Colloidal Silica." *Journal of Sol-Gel Science and Technology* **2000**, *19*, 131.

21. Sellers, H.; Ulman, A.; Shnidman, Y.; Eilers, J.E., "Structure and binding of alkanethiolates on gold and silver surfaces: implications for self-assembled monolayers." *Journal of the American Chemical Society* **2002**, *115*, 9389.

22. Schwarz, J.; Contescu, C. *Surfaces of nanoparticles and porous materials*; CRC, 1999.

23. Gilliland, E.R.; Baddour, R.F.; Perkinson, G.P.; Sladek, K.J., "Diffusion on Surfaces. I. Effect of Concentration on the Diffusivity of Physically Adsorbed Gases." *Industrial & Engineering Chemistry Fundamentals* **1974**, *13*, 95.

24. Dacey, J.R., "SURFACE DIFFUSION OF ADSORBED MOLECULES." *Industrial & Engineering Chemistry* **1965**, *57*, 26.

25. Galwey, A.K.; Brown, M.E., "Application of the Arrhenius equation to solid state kinetics: can this be justified?" *Thermochimica Acta* **2002**, *386*, 91.

26. Cooke, D.W., "The Stereochemistry of the Bis Complexes of Cobalt(III) with Iminodiacetic Acid and Methyliminodiacetic Acid." *Inorg. Chem.* **1966**, *5*, 1141.

27. Bligaard, T.; Honkala, K.; Logadottir, A.; Norskov, J.K.; Dahl, S.; Jacobsen, C.J.H., "On the Compensation Effect in Heterogeneous Catalysis." *The Journal of Physical Chemistry B* **2003**, *107*, 9325.

28. Zhdanov, V., "Arrhenius parameters for rate processes on solid surfaces." *Surface Science Reports* **1991**, *12*, 185.

29. Garn, P., "An examination of the kinetic compensation effect." *Journal of Thermal Analysis and Calorimetry* **1975**, *7*, 475.

30. Galwey, A., "Compensation effect in heterogeneous catalysis." *Advances in Catalysis* **1977**, *26*, 247.

6

Conclusions on Heterogenite Synthesis and Reaction with Iminodiacetic Acid

Measuring the reactive surface area of nanocrystalline materials is a significant challenge in many fields aspect of environmental and materials chemistry research. The successful use of probe molecules to quantify surface sites through direct reaction would be of tremendous value, both for the increased accuracy compared to geometric calculations and increased specificity compared to gas adsorption methods, and because it would provide detailed information on the behavior of the nanoparticle *in situ*. The work presented in this dissertation demonstrates that, while the results might not initially be straightforward, ligand-assisted dissolution can indeed be used as a probe for the reactivity of surface sites.

The ligand-assisted dissolution of heterogenite (β-CoOOH) by iminodiacetic acid (IDA) is proposed as a model system for developing a prototype method. It was initially known that this reaction produced two geometric isomers of $Co(IDA)_2^-$ and that the dominant product changed as a function of the solid-state properties of the heterogenite. The two products, the symmetrical facial (*s-fac*) and unsymmetrical facial (*u-fac*) isomers, were also known to be kinetically inert to ligand exchange; therefore, they remain stable for sufficient time needed to quantify them.

Over the course of this work, many more details have been discovered about this system and its reactivity. The earliest work focused on the production of model materials. Several variations on a synthetic method for producing heterogenite have been studied, and it has been found that a surprising amount of control can be exerted over the heterogenite particles by varying only three synthetic parameters: the temperature, the rate of base addition, and the choice of oxidizing agent. By adjusting these variables, the size of the heterogenite particles can be controlled, giving a range from small platelets with possible sizes of 5-50 nm in diameter to large, well-faceted plates ranging from 200 nm to over 2 μm. It was also found possible to introduce disorder along the *c*-axis of the crystal; by synthesizing the heterogenite via intercalated α-$Co(OH)_2$ particles, small or large amounts of disorder can be introduced into the final product.

There are several promising avenues for continued work on controlling heterogenite particle morphology. First, many factors in the synthesis remain unexplored; there are other possibilities for the oxidizing agent, many other possible synthesis temperatures, and bases other than sodium hydroxide. Any of these variables could offer further control over different aspects of the particle morphology. A second, especially interesting topic in the synthetic portion of this work is the formation of heterogenite rings composed of crystallographically aligned individual 5-10 nm platelets. These rings were occasionally observed in the TEM images of heterogenite samples, but their existence could not be attributed to any obvious common factor. Nonetheless, the exploration of this ring forming could lead to interesting new conclusions about self-assembly as well as the production of materials with very specifically-controlled

morphology. Finally, the use of surfactants to selectively passivate specific surfaces of the heterogenite particles could allow entirely different morphologies to be produced.

Several relationships were discovered between the morphology and reactivity of heterogenite particles. Through the use of IDA as a probe molecule, changes in reactivity as a function of aging time could be explored. These changes were attributed to a relationship between increasing height of the cylindrical heterogenite plates and an increase in the amount of *s-fac* isomer produced. The relative amounts of *s*- and *u-fac* isomers were compared to the geometric surface area of various surface types on the heterogenite particles. A relationship was found between the ratio of isomers and that of the two surface types: the "pure edge" sites, Co centers that has only edge-type surfaces exposed, and the "corner sites", Co centers that had both edge- and basal-type surfaces exposed. Based on this outcome, it was concluded that the corner sites are responsible for producing *u-fac* isomers and the edge sites are responsible for producing *s-fac* isomers. No evidence for the basal surface participating in the dissolution reaction was observed.

The results of fitting IDA adsorption data to a Langmuir isotherm produced an equilibrium constant for adsorption of 2.7 mM^{-1} for IDA on heterogenite. From the maximum adsorption values, adsorption areas were calculated using differing assumptions about the location of the adsorption. From these, the most reasonable estimate was that IDA adsorbed only on edge of the platelets with an adsorption cross-section of ~44 $Å^2$. Dissolution reactions were also run at variable concentrations of IDA and surface sites; these produced reaction orders of 1.8-1.9 for IDA until it neared

saturation, at which point the order dropped to 0.2; the saturation point, however, came much earlier than would be expected from the adsorption isotherm. The order with respect to surface sites was ~1.3. This deviation from the expected order of 1.00 for surface sites was interpreted to mean that surface diffusion of IDA molecules is also important in determining the rate of reaction.

A set of dissolution reactions was also carried out at variable temperature and aging conditions. The most striking result of these reactions was the large difference in apparent activation energy between *s-fac* isomer, *u-fac* isomer, and total $Co(IDA)_2^-$ production. This difference led to the conclusion that, while corner sites are still the primary source of *u-fac* isomer, edge sites produce a temperature-dependent mixture of *s-fac* and *u-fac* isomers. While this observation calls into question the theoretical derivation of the relationship between particle height and *s-fac* isomer production, the ability of equation (4.3) to predict particle heights from a known *s-fac* ratio remains at worst a useful empirical relationship.

Additional work on this project could proceed in several directions. There is certainly more to learn about the reaction mechanisms present in the heterogenite-IDA system. The use of N-methyliminodiacetic acid (mIDA) presents some interesting possibilities in this regard. The additional methyl group makes two important differences in the reactivity of mIDA. First, it creates additional steric hindrance at the N atom. This means that comparing IDA and mIDA reacts could highlight steps where the N is directly involved, because they will be substantially slower with the mIDA. A second, less obvious benefit to mIDA use is a difference in the isomers it forms with Co. In aqueous

solution, the *u-fac* isomer of $Co(mIDA)_2^-$ is unstable, probably due to repulsion between the methyl groups. Unlike with IDA, however, the *mer* isomer of $Co(mIDA)_2^-$ is stable. Looking for evidence of the *mer* isomer, which, like the *s-fac* isomer, has the N atoms in a *trans* configuration, could give indications about the conformation of the molecule on the surface and during complexation of the Co. Another option would be to expand to other heterogenite syntheses. The NaOCl-oxidized particles, with their tendency to display disorder along the *c*-axis, could provide particularly interesting insight into how steric factors around the Co sites affect the dissolution process. The final direction that the work could take is to change materials altogether. Heterogenite is a wonderful model material, but it has very limited application. Theoretically, the results from this system should be extendable to many metal oxide surfaces. For example, the iron oxide family contains many materials that are more environmentally relevant than heterogenite but, unlike Co^{III}, Fe^{III} complexes tend to be very labile. The problem lies in finding a ligand probe that shows appropriate surface sensitivity and forms sufficiently stable products to allow for analysis without any degradation or exchange. Controlling reaction conditions to favor product stability and the use of more rapid or *in situ* analysis methods could somewhat alleviate these problems, but designing the experimental method would still be the biggest barrier to applying these principles to a new system. Nevertheless, this work has demonstrated that probe molecules offer a very viable method of probing solid surface reactivity in a fast, inexpensive, and *in situ* technique.

7

Aggregation-Based Simulation of Particle Growth

7.1 Introduction

The growth, aggregation, and assembly of nanoparticles is of interest to a variety of fields including pharmaceuticals,[1] the petroleum industry,[2,3] geological engineering,[4] and material design.[5] Investigation of particle aggregation, however, is hampered by the rapid rate at which aggregation often occurs, the difficulty of collecting *in situ* observations of the process, and the complexity of controlling the many factors that affect the aggregation process. Computer simulation provides a means to probe the effect of individual factors on the overall process and resulting aggregates with a level of control that is not possible using empirical methods.

The accuracy and utility of a simulation depends upon the simulation method and the underlying assumptions of the model. Care must be exercised to choose a method that is applicable to the system being considered and, therefore, many different simulation methods have been used for a variety of systems. Molecular dynamics simulations provide a means to track very detailed information about how each atom in a system changes over the course of a reaction, but these simulations are typically computationally expensive. The inclusion of solvent molecules in the simulation

significantly increases the computational overhead. Furthermore, molecular dynamics are typically limited to sub-microsecond time scales, making them inappropriate for the lengthier process of colloidal aggregation. Other approaches attempt to produce generalized equations based on first principles and a series of underlying assumptions. Examples of this include systems based off the Smoluchowski equation[6-8] or DLVO theory.[9,10] These generalized equations, however, treat the averages of large populations of particles and usually include rates of growth but not specific details of the aggregation. This type of simulation is not generally capable of describing many of the unique morphologies found in real systems. A third style of simulation focuses on modeling results by constraining simulated objects to known rules of behavior. In this case, the rules are typically empirically derived and relate to first principles only so far as is necessary to ensure the accuracy of the simulation. Although not appropriate to all situations, this approach can often give accurate and detailed results at relatively low computational cost.[3,11]

This chapter presents the development of a program for simulating two-dimensional particle growth by the aggregation of primary crystallites. The simulation combines a random-walk method of particle movement with simplified diffusion equations to allow the simultaneous modeling of particles with differing sizes and shapes. Aggregation between primary particles can be handled using either a simple sticking probability approach, allowing for a range of diffusion- or reaction-limited aggregation systems to be modeled, or a surface energy approach, which allows the construction of large geometric crystals and rings or chains of particles. Although not complete, the

present form of the simulation can produce many different particle shapes, including those composed of intermediate secondary particles. The design and implementation of the method are detailed in the following sections, and the source code can be found in Appendix A of this document.

7.2 Design and Methodology

7.2.1 Basics of the Simulation

Each simulation involves the construction of particle aggregates or clusters from primary particles. The process begins by introducing one or more seed particles: particles that remain in a fixed position throughout the simulation. Other primary particles are introduced at the edge of the simulation region and move randomly within the area until they collide with another particle or aggregate. At this time, the collision either forms an attachment and the particle becomes part of the aggregate, or it does not and the particle resumes its random motion. The simulation continues until meeting an end criterion, such reaching a specified aggregate size (number of particles or diameter) or performing a certain number of movement steps. A schematic of the simplest form of the simulation procedure can be seen in Figure 7.1.

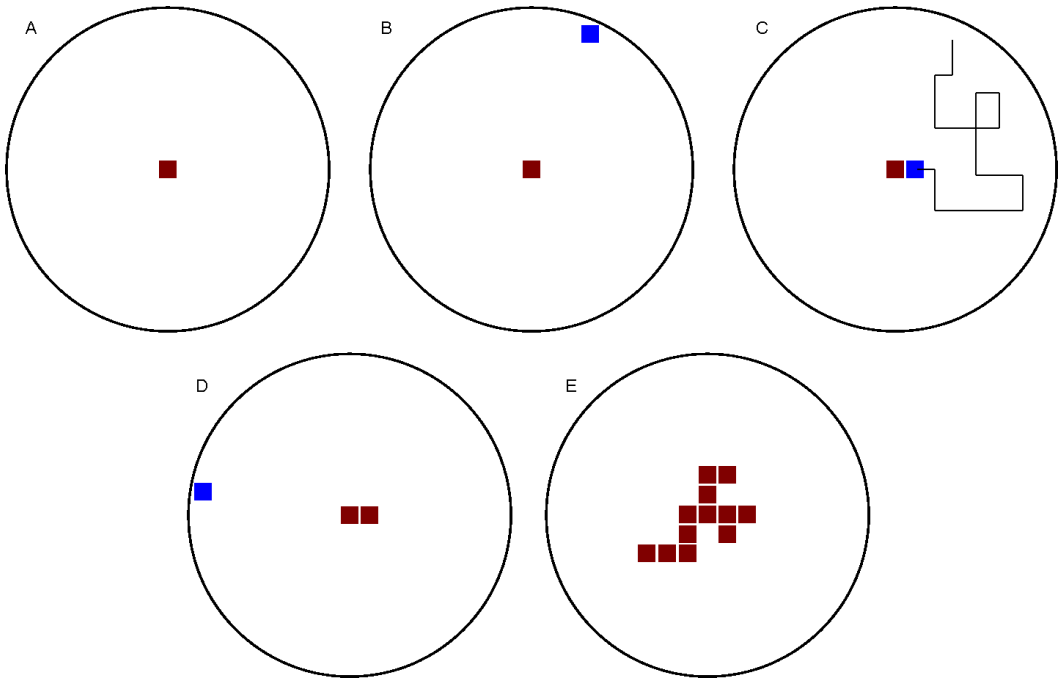

Figure 7.1 Schematic of a basic simulation. A) a fixed seed particle is created B) a primary particle is introduced at the simulation boundary C) the primary particle follows a random walk until meeting the fixed particle D) the particle become fixed and a new primary particle is introduced E) steps A-D repeat until end criterion is reached.

7.2.2 Primary Particles and Simulation Grid

The primary particles in the simulation are considered to be uniform in shape and size. In the current implementation, this shape can be either rectangular or hexagonal. Opposing sides of the shapes are considered to be equivalent surfaces, producing the possibility of two distinct surfaces in the rectangular case and three in the hexagonal. Each particle moves randomly during the simulation until coming into contact with another particle or cluster of particles, at which point the possibility of attachment exists. Particle attachment is considered from two different perspectives: sticking probability and surface energy (see 7.2.3). The movement of the particles can currently be considered either singly or simultaneously. In the single case, each particle is introduced

and moved randomly, one step at a time, until attachment to an aggregate. In the simultaneous case, multiple free particles are introduced and moved pseudo-concurrently. Each of the particles moves with a speed randomly generated from a Gaussian distribution. When all free particles have moved, it is considered to be the end of a "turn" of the simulation. At the end of each movement turn, particles in close proximity are checked for successful attachment. Particles may also be allowed to rotate, although rotation is constrained by the shape (90° intervals for rectangular primaries, 60° intervals for hexagonal).

The simulation takes place on a two-dimensional grid with definable boundaries. Each grid space is the size of one primary particle, and the overall grid shape mirrors that of the particles (rectangular or hexagonal). Constraining the particles to a grid limits the possible attachment geometry of the primary particles but also offers a significant advantage: when determining particle collisions, it is only necessary to examine the adjacent grid spaces for particles, as opposed to checking all possible particles for proximity. This advantage is increasingly important when large numbers of particles are considered, as it reduces the time necessary to check for collisions to a linear scaling rather than quadratic.

There are two possible interpretations of the grid boundaries in the context of the simulation: as a physical border of the system or as the edge of a region of interest, with the system extending beyond the grid. In the physical boundary case, particles are not allowed to move beyond the edge of the grid. In the region of interest case, particles that pass beyond the edge of the grid are removed from the simulation and reintroduced at

another point on the edge. This can be viewed as a compression of time when no free particles are inside the region of interest; the simulation skips directly from the departure of the last free particle to the moment a new particle is introduced from outside the region. When the grid is considered as a region of interest, the distance of the boundaries from the particle aggregates can influence the attachment of new free particles. The simulation currently allows three types of grid boundary: fixed, anisotropic expansion, and isotropic expansion. A fixed grid boundary does not change during the course of the simulation. An anisotropic grid boundary expands as the aggregate nears it so that a minimum distance exists between each side of the grid and the nearest fixed particle. An isotropic grid boundary also expands, but expands equally in all directions when a portion of an aggregate nears any one boundary. The three boundary types are illustrated in Figure 7.2. The isotropic and anisotropic boundary methods allow a more efficient simulation by minimizing the number of steps necessary to reach the aggregate in the early stages but expanding as the simulation progress to avoid introducing artifacts caused by particles being introduced very near the aggregate.

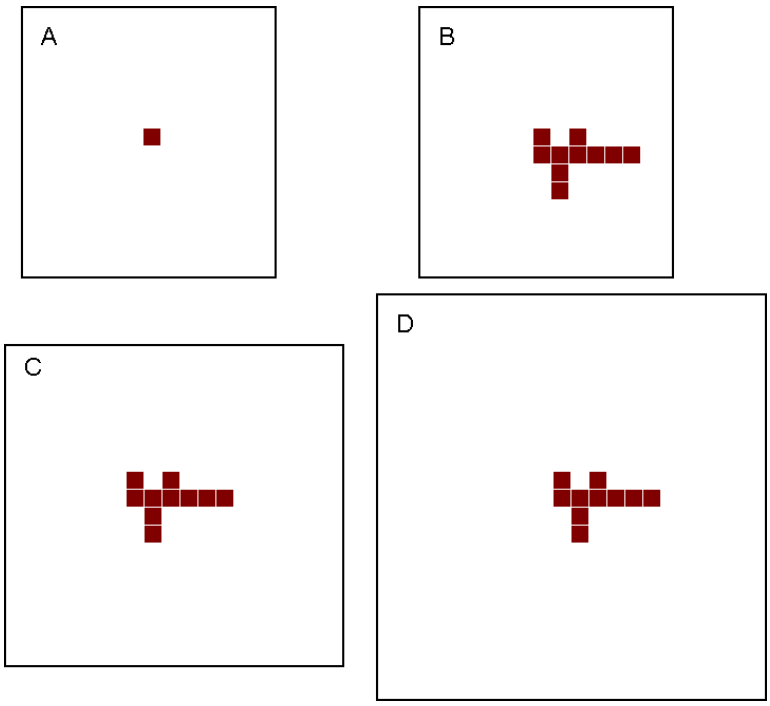

Figure 7.2 Illustration of different grid boundary types. The boundary is shown at the beginning of the simulation (A) and after 9 particles have attached using: fixed boundaries (B), anisotropic expansion (C), or isotropic expansion (D).

7.2.3 Particle Attachment

When two particles collide or move into close proximity to one another, the possibility of attachment occurs. In this model, whether a particle attaches to another particle or cluster is determined by comparing a randomly generated number to the probability of successful attachment. If the random number is less than or equal to the probability, attachment is successful. The probability of attachment can be determined in two different manners: sticking probability and surface energy.

The sticking probability method is the simpler of the two. In this case, a sticking chance is assigned to each possible combination of particle faces. When two particles meet, the appropriate probability is selected based on the particle faces that are in contact.

For example, rectangular primary particles with two different face types, A and B, would have three possible sticking probabilities: A-A, B-B, and A-B. Varying the sticking probabilities allows control over several factors. Decreasing the sticking probability for all faces will result in more compact particle aggregates. In this way, the sticking probability can be viewed as a continuum between two limiting cases: diffusion-limited aggregation (DLA), where the sticking probability is 100%, and reaction-limited aggregation (RLA), where the sticking probability is considerably lower and most collisions do not result in successful attachment (see Figure 7.3). Changing the relative sticking probabilities for different face combinations (A-A or B-B) allows the creation of favored growth directions, producing aggregates with higher aspect ratios. By increasing the probability of dissimilar (A-B) faces sticking and allowing particle rotation, misalignments can be introduced into the aggregate. Depending on the material being modeled, these misalignments can be viewed as structural defects or twinning.

It is also possible to define unsticking probabilities. When non-zero unsticking probabilities are used, the assumption is that attachment to an aggregate is not necessarily irreversible. These probabilities are defined in the same way as sticking probabilities: A-A, B-B, and A-B chances. Any particle that is not on the interior of the aggregate (i.e. has fewer neighbors than it has faces) has a check made against its unsticking probability. If this check is successful, it leaves the aggregate and becomes a free primary particle once more. Unsticking chances can be applied to in two different ways. If unsticking is allowed during the main portion of the simulation, the unsticking check is applied to every attached particle at the end of each movement cycle. In this case, it should be

noted that the sticking and unsticking probabilities are not directly comparable. Because each particle will receive numerous unsticking checks over the course of aggregate formation, these chances should typically be significantly (often orders of magnitude) lower than the corresponding stick chance in order to allow any aggregation to occur. This method approximates the formation of unstable clusters, which readily break apart and reform. Unsticking can also be applied after the end criteria for aggregation have been met. This is akin to a fast aggregation step followed by a slower coalescence stage, in which exterior particles tend to migrate inwards, converting extended fractal aggregates into denser clusters (Figure 7.4).

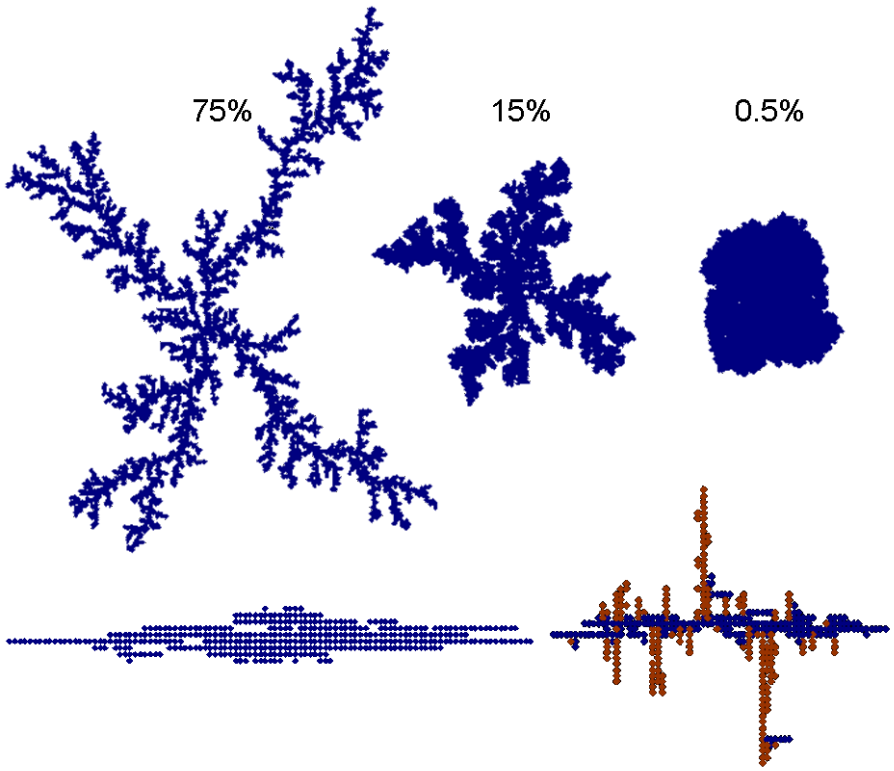

Figure 7.3 Effects of changing sticking probabilities. Top: Aggregates formed from 1200 square particles with all faces equivalent and a 75% (left), 15% (center) or 0.5% (right) chance of successful attachment on collision. Bottom: Aggregates formed of 400 square particles with a 50% A-A sticking probability, 2% B-B and 0% A-B (left) or 0.1% A-B(right).

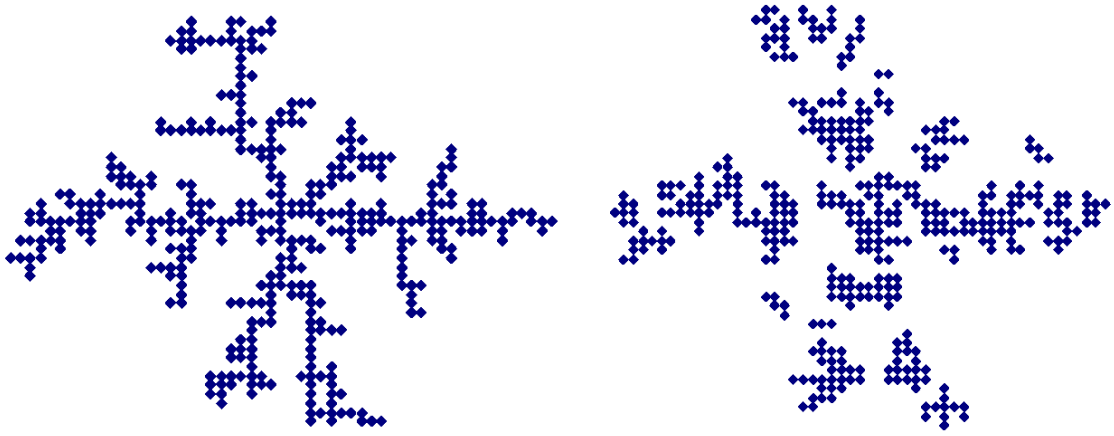

Figure 7.4 Aggregate composed of 400 square particles with equivalent faces before (left) and after (right) 200 coalescence cycles with an unstick chance of 5%.

The surface energy method is more complex, but allows the environment of the two particles to influence the chances of attachment. Beyond merely considering which faces are involved, the attachment probability is influenced by the nearest and next-nearest neighbors of each particle as well. This method works by assigning a crystallographic surface type to the aggregate at the point of collision (Figure 7.5). Each surface type is assigned a relative energy, and the surface energies before and after particle attachment are compared to determine a change in energy (ΔE) for attachment. The probability for attachment is determined by the ΔE value. Currently, attachment probabilities are assigned either for ranges of ΔE or simply as a sticking probability based on surface type (a hybrid between surface energy and sticking probability), but a logical extension is to allow the probability to vary continuously as a function of the energy change. This process can be viewed as multi-step attachment to the surface in which primary particles first form a reversible complex with the surface, which may then

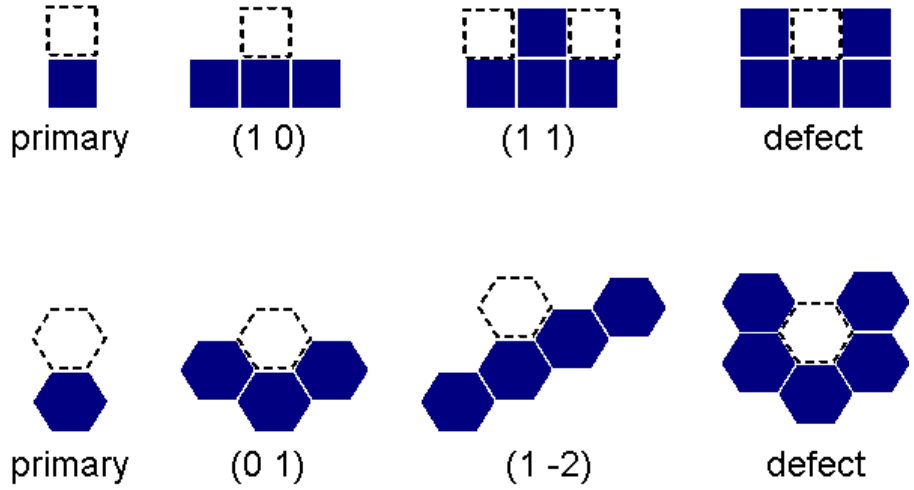

Figure 7.5 Overview of different surface types for square (top) and hexagonal (bottom) primary particles.

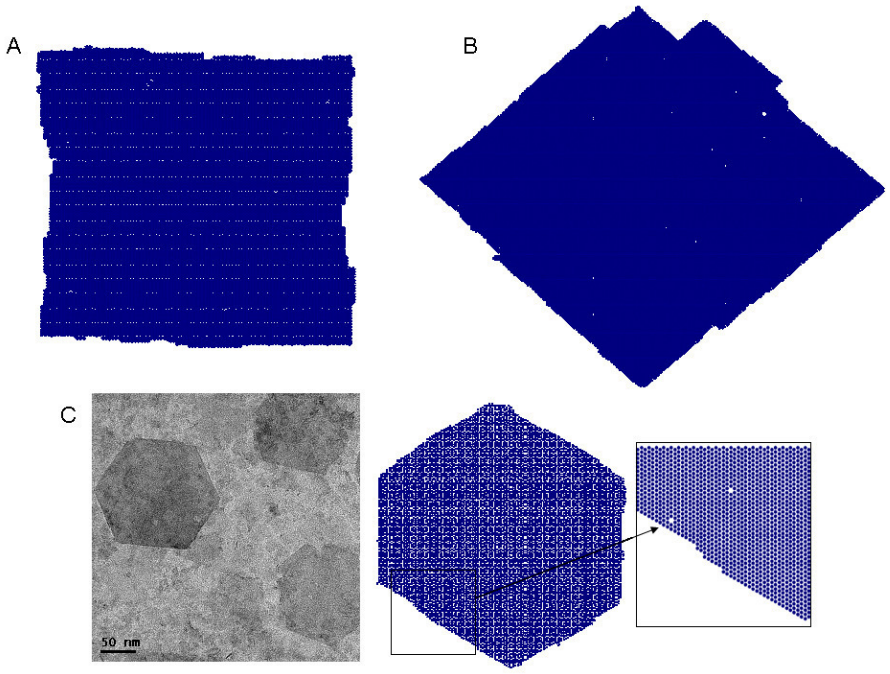

Figure 7.6 Aggregates formed from 10,000 primary particles using the surface type/sticking probability hybrid method. A) square particles: 0.5% chance to stick to (1 0) surface, 45% to (1 1), 95 % to defect. B) square particles: 45% to (1 0), 10% to (1 1), 95% to defect. C) comparison between TEM image of heterogenite particles (left) and a simulated aggregate from hexagonal particles: 0.5% to stick to (1 -2), 45% to (0 1), 95% to defect (right).

become a permanent attachment. The equilibrium of this complex is governed by the ΔE value and, in the simulation, the associated attachment probability. The use of the surface energy method allows for the formation of several aggregate types not observed under pure DLA or RLA conditions, such as large, geometrically shaped particle aggregates that also incorporate some surface and interior defects (Figure 7.6).

7.2.4 Aggregates

In order to accurately model many of the aggregation phenomena observed in colloidal system, it is necessary to construct secondary particles which, in turn, assemble to form larger tertiary structures. To this end, the simulation also allows aggregates to move freely, collide, and attach. Aggregate movement is only supported in the simultaneous movement mode, where movement speeds are chosen randomly from a Gaussian distribution based on the size of the aggregate (see 7.3.3). Aggregates behave similarly to primary particles: they are able to rotate freely and attachment checks are made based on collisions or proximity at the end of each movement turn. Currently, only the sticking probability method of attachment is fully implemented for aggregates (Figure 7.7). The surface energy method is partially in place; colliding surfaces can be identified for small or sharply faceted aggregates, and collisions with primary particles are fully functional, but large and irregular aggregates currently produce erratic behavior in this mode. With the completion of the surface energy method, secondary particle aggregates should be effectively interchangeable with primary particles in the simulation.

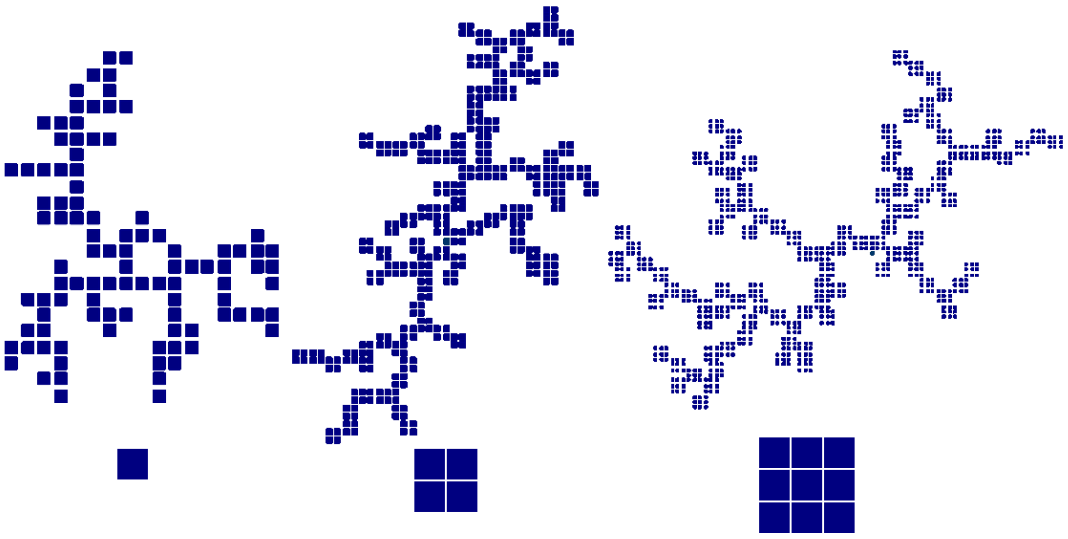

Figure 7.7 Fractal aggregates formed with a 90% stick chance from 100 1x1 primary particles (left), 2x2 secondary particles (center), and 3x3 secondary particles (right). Images are scaled to make basic unit (1x1, 2x2, or 3x3) approximately equal in size.

7.3 Implementation

7.3.1 C++ Language

The simulation program is implemented in the C++ language for several reasons. First, the low-level memory management abilities of the language allow the program to run very efficiently on systems with moderate amounts of RAM, while still allowing for runtime expansion if necessary for tasks such as unexpected resizing of the grid. Second, the pointer functionality present in C++ is extremely useful for efficient locating of many objects that must frequently interact with each other (such as the simulation grid, the particle array, and individual aggregate units). By tracking pointers rather than copying objects, both memory and processor usage are significantly reduced. Finally, support for objects, function overloading, and polymorphism makes it simple to integrate new functionality without significantly altering the existing base code. For example, adding

secondary particle (aggregate) functionality required substantial code additions, but the end result is that basic functions such as Move and Rotate can be called with either primary particle or aggregate targets, greatly simplifying the code used for the simulation itself.

7.3.2 Pseudorandom Number Generation

The utility of any stochastic simulation depends upon the quality of the randomness achieved. This simulation uses two different pseudorandom number generation methods. When a number is to be selected from a uniform floating point distribution, the Mersenne Twister[12] generator is used. This generator was designed for use in Monte Carlo type simulations and generates fast and high quality random numbers. When numbers are to be generated from a Gaussian distribution, the Ziggurat algorithm[13] is used with a feed of uniform values from the Mersenne Twister. The Ziggurat algorithm outputs random numbers with a Gaussian distribution of specified variance, and minimizes performance decreases by avoiding computationally expensive multiplication or division.

7.3.3 Object Interaction

The basic simulation functionality depends upon the interaction of three major objects: the GridArray, ParticleArray, and AggregateArray. These array objects govern the grid, primary particles, and secondary aggregates, and all functions related to these objects are called through the appropriate array. The three object arrays interact with each other via pointers, and they should be initialized in the order of GridArray,

ParticleArray, and finally AggregateArray. Each later array object notifies the earlier objects of relevant changes, so no pointer modification is required after initialization.

GridArray objects contain a two-dimensional array of spaces. Each space has an integer value that identifies the occupying particle or a value of −1 if the space is unoccupied. The GridArray also stores information about the maximum grid size, the x and y dimensions of the current region of interest, the type of boundary expansion (fixed, isotropic, anisotropic), and a pointer to the associated ParticleArray object. There are several public procedures callable on the GridArray. One allows the grid itself to be resized during the simulation. The remaining functions return information about the status of particles on the grid, such as whether a space is occupied, what positions are occupied by neighbors and secondary neighbors of a grid space, and whether a grid space is within the current region of interest.

A ParticleArray object governs the primary particles in the simulation. It contains an array of Particle objects, as well as the shape and number of particles present, the type of attachment checks (sticking probability or surface energy), and pointers to the GridArray and AggregateArray. The Particle objects represent the individual primary particles. Each retains small amounts of information, such as the location on the grid, orientation, attachment status, and a reference to the Aggregate, if any, to which the particle belongs. All particle manipulation functions are called through the ParticleArray, with a reference by number to the target particle. Movement, attachment checks, rotations, and addition or removal from an aggregate are all controlled through the ParticleArray's public procedures.

The AggregateArray is only created when mobile secondary-particle aggregates are used in the simulation. An AggregateArray object contains an array of Aggregate objects, as well as manipulation functions for the Aggregates. Most of the procedures that can be called with a Particle target will automatically reference the appropriate AggregateArray procedure if the particle is a member of an Aggregate. In this fashion, the AggregateArray member functions are rarely called directly. The AggregateArray also contains a pointer to the associated ParticleArray, and the Particle members of each Aggregate are referenced by number in that array.

7.3.4 Particle Movement

The most computationally complex feature of the simulation is the movement of particles and aggregates when used in simultaneous motion mode. The movement rate of the particles and aggregates are determined based on diffusion principles and the size of the aggregate. When only primary particles are present, all particles are treated as spheres with diameter of 1 grid space for the purpose of diffusion, and each movement step is assumed to have a variance of 1 space. Rotational or translational diffusion in 1 dimension can be expressed

$$\left\langle (\Delta x)^2 \right\rangle = 2D_x t \tag{7.1}$$

where x is the displacement in x, D_x is the diffusion coefficient, and t is the time elapsed. Diffusion coefficients for translational and rotational diffusion are given by

$$D_x = \frac{k_B T}{6\pi\eta r} \quad \text{for translation} \tag{7.2}$$

$$D_\theta = \frac{k_B T}{8\pi\eta r^3} \quad \text{for rotation} \tag{7.3}$$

where k_B is the Boltzmann constant, T is absolute temperature, η is the viscosity of the medium, and r is the radius of the particle. For a primary particle (r = 0.5 grid space), the time t_{trans} is defined as the time required to give a mean square translational displacement of $\left\langle (\Delta x)^2 \right\rangle$ = 1 space^2. The mean rotational displacement in the same amount of time can be determined by setting $t_{trans} = t_{rot}$ and solving for $\left\langle (\Delta\theta)^2 \right\rangle$. This yields

$$\frac{\left\langle (\Delta x)^2 \right\rangle}{k_B T}(6\pi\eta r) = \frac{\left\langle (\Delta\theta)^2 \right\rangle}{k_B T}(8\pi\eta r^3) \tag{7.4}$$

Substituting $\left\langle (\Delta x)^2 \right\rangle$ = 1 space^2 and r = 0.5 space in (7.4) and canceling like terms gives $\left\langle (\Delta\theta)^2 \right\rangle$ = 4/3 rad^2. At each movement step, the x and y components of each particle's motion is selected from a Gaussian distribution centered at zero with a variance of 1 space^2. When primary particles are rotated, a Gaussian distribution of rotation angles is binned into the allowed rotations based on the $\left\langle (\Delta\theta)^2 \right\rangle$ value. For square particles, this translates into a 50% chance of no rotation, 23% chance each for rotation 90° in either direction, and 4% chance of 180° rotation. For hexagonal particles, the chances become 34% chance of no rotation, 24% each of 60° rotation in either direction, 8% each of 120° rotation in either direction, and 4% chance of 180° rotation.

When aggregates are present in the simulation, the movement and rotation is adjusted based on the size of the aggregate. All movement is scaled based on primary

particles moving once per turn. Larger aggregates will move and rotate less frequently, and the movement rates are recalculated any time there is a change to the aggregate structure. To simply the calculations, aggregates are treated as ellipsoidal particles with a negligibly thin z-dimension. This should give good results for densely packed aggregates, but is less appropriate for highly branched, fractal aggregates.[14] The diffusion rate is based on $\left\langle (\Delta x)^2 \right\rangle$ = 1 space^2 for diffusion perpendicular to the widest direction. By substituting equation (7.2) into (7.1) and taking the ratio between the aggregate's largest direction and a primary particle, the number of turns, n_{trans}, between aggregate movements can be found

$$n_{trans} = \frac{r_{long}}{r_{primary}} \tag{7.5}$$

where r_{long} is the largest radius of the aggregate and $r_{primary}$ is the radius of a primary particle (0.5 space for all cases considered here). Translation along the narrower direction of the aggregate occurs at the same time but, because the cross-section is smaller, it will be expected to move a greater distance in the same time. By again taking a ratio of equations (7.1) between the short and long directions of the aggregate, the variance for movement in the short direction is given

$$\left\langle (\Delta x_{short})^2 \right\rangle = \frac{r_{long}}{r_{short}} \left\langle (\Delta x_{long})^2 \right\rangle \tag{7.6}$$

Finally, the time needed for a rotation of the aggregate to occur, n_{rot}, is determined by substituting equation (7.3) into (7.1) and again finding the ratio with a primary particle

$$n_{rot} = \frac{r^3{}_{agg}}{r^3{}_{primary}} \tag{7.7}$$

where r_{agg} is the geometric mean of the long and short radii of the aggregate.

To clarify this implementation, a summary of the simultaneous movement mode is presented. First, the number of turns for translation and rotation, n_{trans} and n_{rot}, are calculated according to (7.5) and (7.7) for each aggregate and stored. For each movement turn, every primary particle is moved in both the x and y directions with $\left\langle (\Delta x)^2 \right\rangle$ = 1 space^2. Each particle is also subjected to a random rotation with $\left\langle (\Delta \theta)^2 \right\rangle$ = 4/3 rad^2. Each aggregate has its turn counters decremented. If neither n_{trans} nor n_{rot}, reach zero, nothing else occurs for that aggregate during the turn. If n_{trans} = 0, the aggregate makes a translational movement. It is translated in the wide direction (perpendicular to the longest radius) with $\left\langle (\Delta x)^2 \right\rangle$ = 1 space^2 and in the narrow direction (perpendicular to the shortest radius) with $\left\langle (\Delta x)^2 \right\rangle = \frac{r_{long}}{r_{short}}$ space^2. If n_{rot} reaches zero that turn, the aggregate is subjected to rotation with $\left\langle (\Delta \theta)^2 \right\rangle$ = 4/3 rad^2, which occurs around the center of mass of the aggregate. After either a translation or rotation occurs, the related counter is reset. At the end of the turn, collisions are evaluated and new attachments are made as appropriate. If any new attachments occur, turn counters are recalculated for any aggregates that changed in size.

7.4 Extension

7.4.1Aggregate Surface Energy

One of the most important ways in which the current simulation could be enhanced is the full inclusion of the surface energy attachment method for multi-particle aggregates. There are two potential ways that this could be accomplished. The first is an extension of the surface energy/sticking probability hybrid method. The surface type present at any point neighboring an aggregate can currently be determined; the next step would be to chart all surface types that come into contact on both aggregates. From these data, an "average" surface type could be found for each aggregate, and sticking probabilities could be assigned based on which surfaces come into contact. E.g., for square particles, there would be a (1 0)-(1 0) chance, (1 0)-(1 1), (1 1)-(0 1), etc. This would allow aggregate attachment to be relatively scale invariant (although surface roughness would increase the probability of a misassigned surface type), and is perhaps the simplest effective method for joining larger secondary particles.

The other proposed method would extend the surface energy equilibrium (ΔE) method to larger aggregates. In this case, the total surface energy of both aggregates would be determined separately and compared to the surface energy of the new particle produced by attachment. This would allow for more complete control of attachment as compared to the hybrid method and would give a more realistic penalty for the formation of voids and other defects, but it has some difficulties in effective implementation. From a design perspective, this method is significantly slower, requiring at least double the number of surface type determinations as the hybrid method. From an accuracy

perspective, it requires several features of the simulation to be reconsidered. As aggregates get larger and larger, the fraction of total surface that is involved in the collision will become smaller, and therefore the magnitude of ΔE will decrease relative to the total surface energy involved. This means that attachment probabilities will not scale with size, which is undesirable in a simulation that does not explicitly define the size of the primary particles. A potential solution to this issue would be the determination of a normalized ΔE, adjusted for the size of the participating aggregates. With normalization, however, almost every collision will produce a negative ΔE by sheer virtue of the reduction in exposed surface. This could lead to uncontrolled growth to form ever larger particles. To counteract this effect, it would be necessary to introduce a size-based growth inhibition factor. The inclusion of equations from DLVO theory, albeit in simplified form, and a user-definable size parameter may present a solution to this issue.

7.4.2 Grid and Shape Abstraction

Ideally, the simulation method will be able to include additional particle shapes and the option to model growth in three dimensions. The best technique for incorporating these changes would be an abstraction of the GridArray and Particle objects. At present, the code explicitly refers to the four- and six-sided particle cases, and works on the assumption that the grid is two dimensional. By replacing these explicit statements with calls to virtual functions, the addition of new grid and particle shapes would be greatly simplified. In the case of the GridArray, the change would be relatively simple: the replacement of the GridArray type with an abstract class and the addition of

polymorphic 2DGrid and 3DGrid objects to handle the grid operations. Allowing expansion of the particle shapes may prove more challenging.

Ideally, the particle shape should be user definable. This would require wide-sweeping changes, however, especially in the case of shapes that do not tile. One way to mitigate this would be a switch to a coordinate grid system (see 7.4.3). The Particle object could then be similarly replaced with a virtual object and polymorphic calls to the desired shape of particle. The inclusion of user-designed particle shapes would also require information to be collected about the types of attachment. In the simplest form, a simulation instruction file would need to both define the face types present and provide sticking probabilities for every combination of them. To use the surface energy method, either the instructions would need to both enumerate the surface types and provide rules for identifying each surface, or the program would need the inclusion of crystallographic logic to make this determination automatic.

Essentially, the replacement of the GridArray with a more versatile object offers obvious benefits and no drawbacks aside from the effort of including it, but abstracting the Particle object offers more dubious gains. Simply adding additional common particle shapes or constructing more complex shapes as aggregates of primary units may offer a solution that gives nearly as many gains at a fraction of the complexity. In either case, however, additional logic to extend particle shapes and grids to three dimensions should be a relatively high priority.

7.4.3 Coordinate Grid

One potential enhancement of the simulation method is the conversion from the current grid system, in which one primary particle occupying one grid space, to a coordinate based system. This could be implemented by making a user-definable primary particle size, although resolution is perhaps a better term than size. The grid would still be divided into sections, and would therefore retain the speed advantage of searching for neighbors by grid location, but the particle position would no longer be limited to the exact grid space. This would give the advantage that primary particles could attach to each other in ways other than a perfect one-to-one face correspondence, allowing for new defect types in a simulated structure. Under this system, a particle could occupy portions of more than one grid space, and multiple particles could be found in the same space. The negative side to this change would be a decrease in the efficiency of the collision detection, because it would increase the number of grid spaces that must be searched for neighboring particles and would also require an additional check to see how close the particles actually are, as occupying an adjacent space would no longer be a sufficient condition for collision/proximity.

Another potential way to realize many of the same gains within the current system is by using aggregates as "primary" particles, such as the 3x3 squares shown in Figure 7.7. This allows the same misaligned attachment that is possible with the coordinate system, and also provides a means to create new particle shapes within the existing framework. In this way, the basic rectangular or hexagonal units could be considered as unit cells of the structure, rather than true primary particles. Using the current aggregate

movement routines, a random walk involving more complex shapes does not require substantially more computation time than one involving primary units. It does, however, require significantly more memory due to the need for proportionally higher numbers of primary units and grid spaces to accommodate the larger scale of the simulation. Given that, at present market conditions, memory is far less expensive than processor speed, the best solution may be to incorporate simpler ways of generating "primary" aggregates for use in simulations.

7.5 Acknowledgements

We thank the University of Minnesota and the National Science Foundation (Career-036385 and MRI EAR-0320641) for funding to support this work.

7.6 References

1. Cao, X.; Xu, G.; Li, Y.; Zhang, Z., "Aggregation of Poly (ethylene oxide)- Poly (propylene oxide) Block Copolymers in Aqueous Solution: DPD Simulation Study." *J. Phys. Chem. A* **2005**, *109*, 10418.

2. Rogel, E., "Simulation of interactions in asphaltene aggregates." *Energy Fuels* **2000**, *14*, 566.

3. Zeidan, M.; Jia, X.; Williams, R.; Wedlock, D., "Simulation of aggregation with applications to soot laden lubricating fluids." *Particle & Particle Systems Characterization* **2004**, *21*, 473.

4. Chang, F.; Civan, F., "Practical model for chemically induced formation damage." *Journal of Petroleum Science and Engineering* **1997**, *17*, 123.

5. Marrink, S.J.; Lindahl, E.; Edholm, O.; Mark, A.E., "Simulation of the Spontaneous Aggregation of Phospholipids into Bilayers." *Journal of the American Chemical Society* **2001**, *123*, 8638.

6. Eibeck, A.; Wagner, W., "Stochastic Particle Approximations for Smoluchoski's Coagulation Equation." *Annals of Applied Probability* **2001**, *11*, 1137.

7. Frenklach, M., "Dynamics of discrete distribution for Smoluchowski coagulation model." *Journal of colloid and interface science* **1985**, *108*, 237.

8. Kreer, M.; Penrose, O., "Proof of dynamical scaling in Smoluchowski's coagulation equation with constant kernel." *Journal of Statistical Physics* **1994**, *75*, 389.

9. Lyklema, J.; Van Leeuwen, H.; Minor, M., "DLVO-theory, a dynamic re-interpretation." *Advances in Colloid and Interface Science* **1999**, *83*, 33.

10. Drews, T.; Tsapatsis, M., "Model of the evolution of nanoparticles to crystals via an aggregative growth mechanism." *Microporous and Mesoporous Materials* **2007**, *101*, 97.

11. Rabani, E.; Reichman, D.; Geissler, P.; Brus, L., "Drying-mediated self-assembly of nanoparticles." *Nature* **2003**, *426*, 271.

12. Matsumoto, M.; Nishimura, T., "Mersenne Twister: A 623-Dimensionally// Equidistributed Uniform Pseudo-Random Number Generator." *ACM Transactions on Modeling and Computer Simulation* **1998**, *8*, 3.

13. Marsaglia, G.; Tsang, W., "The ziggurat method for generating random variables." *Journal of Statistical Software* **2000**, *5*, 1.

14. Kolwankar, K., "Brownian motion of fractal particles: Levy flights from white noise." *Arxiv preprint cond-mat/0511307* **2005**.

The following works were consulted in the development of the simulation algorithms:

15. Odriozola, G.; Leone, R.; Schmitt, A.; Moncho-Jordá, A.; Hidalgo-Álvarez, R., "Coupled aggregation and sedimentation processes: The sticking probability effect." *Physical Review E* **2003**, *67*, 31401.

16. Jullien, R.; Kolb, M., "Hierarchical model for chemically limited cluster-cluster aggregation." *Journal of Physics A: Mathematical and General* **1984**, *17*, L639.

17. Ludwig, J.; Vlachos, D.; Van Duin, A.; Goddard III, W., "Dynamics of the dissociation of hydrogen on stepped platinum surfaces using the ReaxFF reactive force field." *J. Phys. Chem. B* **2006**, *110*, 4274.

18. Wang, H., "Thermal rebound of nanometer particles in a diffusion battery." *Aerosol Science and Technology* **1993**, *18*, 180.

19. Fernández-Toledano, J.; Moncho-Jordá, A.; Martínez-López, F.; González, A.; Hidalgo-Álvarez, R., "Two-dimensional colloidal aggregation mediated by the range of repulsive interactions." *Physical Review E* **2007**, *75*, 41408.

20. Ribeiro, C.; Lee, E.; Longo, E.; Leite, E., "Oriented Attachment Mechanism in Anisotropic Nanocrystals: A "Polymerization" Approach." *ChemPhysChem* **2006**, *7*, 664.

21. Sayle, D.; Seal, S.; Wang, Z.; Mangili, B.; Price, D.; Karakoti, A.; Kuchibhatla, S.; Hao, Q.; Mo bus, G.; Xu, X., "Mapping nanostructure: a systematic enumeration of nanomaterials by assembling nanobuilding blocks at crystallographic positions." *ACS nano* **2008**, *2*, 1237.

22. Grassia, P.; Hinch, E.; Nitsche, L., "Computer simulations of Brownian motion of complex systems." *Journal of Fluid Mechanics* **2006**, *282*, 373.

23. Vermant, J.; Yang, H.; Fuller, G., "Rheooptical determination of aspect ratio and polydispersity of nonspherical particles." *AIChE Journal* **2001**, *47*, 790.

24. Cichocki, B.; Felderhof, B., "Long-time translation and rotational Brownian motion in two dimensions." *Journal of Statistical Physics* **1997**, *87*, 989.

25. Han, Y.; Alsayed, A.; Nobili, M.; Zhang, J.; Lubensky, T.; Yodh, A., "Brownian motion of an ellipsoid." *Science* **2006**, *314*, 626.

26. Fox, R.; Uhlenbeck, G., "Contributions to Non Equilibrium Thermodynamics. I. Theory of Hydrodynamical Fluctuations." *Physics of Fluids* **1970**, *13*, 1893.

Bibliography

Metallic Cobalt Particles (with or without Tungsten Carbide). In *IARC Monographs on the Evaluation of Carcinogenic Risks to Humans*; IARC: Lyons, France, 2006; Vol. 86.

Aloko, D.; Afolabi, E.A., "Titanium Dioxide as a Cathode Material in a Dry Cell." *Leonardo Electronic Journal of Practices and Technologies* **2007**, *11*, 97.

Ama, T.; Kawaguchi, H.; Yasui, T., "Preparation and Reaction of the Unsymmetrical-facial Isomers of the Bis(N-alkyliminodiacetato)cobaltate(III) Ions." *Chem. Lett.* **1981**, 323.

Antony, K. “Wear-resistant cobalt-base alloys”; International Conference on Cobalt: Metallurgy and Uses,, 1981.

Armstrong, R.D.; Briggs, G.W.D.; Charles, E.A., "Some effects of the addition of cobalt to the nickel hydroxide electrode." *J. Appl. Electrochem.* **1988**, *18*, 215.

Arslan, I.; Walmsley, J.C.; Rytter, E.; Bergene, E.; Midgley, P.A., "Toward Three-Dimensional Nanoengineering of Heterogeneous Catalysts." *Journal of the American Chemical Society* **2008**, *130*, 5716.

Baker, D.R. *Capillary Electrophoresis*; Wiley-Interscience: New York, 1995.

Banfield, J.F.; Zhang, H., "Nanocrystals in the Environment." *Reviews in Mineralogy and Geochemistry* **2001**, *44*, 1.

Barde, F.; Palacin, M.-R.; Beaudoin, B.; Delahaye-Vidal, A.; Tarascon, J.-M., "New Approaches for Synthesizing gamma III-CoOOH by Soft Chemistry." *Chem. Mater.* **2004**, *16*, 299.

Baxter, J.; Aydil, E., "Nanowire-based dye-sensitized solar cells." *Applied Physics Letters* **2005**, *86*, 053114.

Bligaard, T.; Honkala, K.; Logadottir, A.; Norskov, J.K.; Dahl, S.; Jacobsen, C.J.H., "On the Compensation Effect in Heterogeneous Catalysis." *The Journal of Physical Chemistry B* **2003**, *107*, 9325.

Bolt, G.H. *Interactions at the Soil Colloid-soil solution interface*; Kluwer Academic Publishers: Norwell, MA, 1991.

Booth, G.; McDuell, G.R.; Sears, J. *World of science*; Kogan Page Publishers, 1999.

Bosbach, D.; Charlet, L.; Bickmore, B.; Hochella, M.F., Jr., "The dissolution of hectorite: In-situ, real-time observations using atomic force microscopy." *American Mineralogist* **2000**, *85*, 1209.

Brantley, S.L.; Mellot, N.P., "Surface area and porosity of primary silicate minerals." *American Mineralogist* **2000**, *85*, 1767.

Brosse, E.; Magnier, C.; Vincent, B., "Modelling Fluid-Rock Interaction Induced by the Percolation of CO2-Enriched Solutions in Core Samples: the Role of Reactive Surface Area." *Oil & Gas Science and Technology* **2005**, *60*, 287.

Burgisser, C.S.; Stone, A.T., "Determination of EDTA, NTA, and Other Amino Carboxylic Acids and Their Co(II) and Co(III) Complexes by Capillary Electrophoresis." *Environ. Sci. Technol.* **1997**, *31*, 2656.

Butel, M.; Gautier, L.; Delmas, C., "Cobalt oxyhydroxides obtained by 'chimie douce' reactions: structure and electronic conductivity properties." *Solid State Ionics* **1999**, *122*, 271.

Cao, X.; Xu, G.; Li, Y.; Zhang, Z., "Aggregation of Poly (ethylene oxide)- Poly (propylene oxide) Block Copolymers in Aqueous Solution: DPD Simulation Study." *J. Phys. Chem. A* **2005**, *109*, 10418.

Chang, F.; Civan, F., "Practical model for chemically induced formation damage." *Journal of Petroleum Science and Engineering* **1997**, *17*, 123.

Chankvetadze, B. *Capillary electrophoresis in chiral analysis*; Wiley, 1997.

Chirvase, D.; Parisi, J.; Hummelen, J.; Dyakonov, V., "Influence of nanomorphology on the photovoltaic action of polymer–fullerene composites." *Nanotechnology* **2004**, *15*, 1317.

Chun, C.L.; Penn, R.L.; Arnold, W.A., "Kinetic and Microscopic Studies of Reductive Transformations of Organic Contaminants on Goethite." *Environmental Science & Technology* **2006**, *40*, 3299.

Chun, C.L.; Penn, R.L.; Arnold, W.A., "Kinetic and Microscopic Studies of Reductive Transformations of Organic Contaminants of Goethite." *Environ. Sci. Technol.* **2006**, *40*, 3299.

Cichocki, B.; Felderhof, B., "Long-time translation and rotational Brownian motion in two dimensions." *Journal of Statistical Physics* **1997**, *87*, 989.

Cooke, D.W., "The Stereochemistry of the Bis Complexes of Cobalt(III) with Iminodiacetic Acid and Methyliminodiacetic Acid." *Inorg. Chem.* **1966**, *5*, 1141.

Crook, P., "Cobalt and cobalt alloys." *ASM International, Metals Handbook, Tenth Edition* **1990**.

Dacey, J.R., "SURFACE DIFFUSION OF ADSORBED MOLECULES." *Industrial & Engineering Chemistry* **1965**, *57*, 26.

de Lunsford, D.; Flickinger, J.; Lindner, G.; Maitz, A., "Stereotactic radiosurgery of the brain using the first United States 201 cobalt-60 source gamma knife." *Neurosurgery* **1989**, *24*, 151.

Dogterom, R.; Oosterbeek, H.; Reynhout, M. Catalytic Process For The Conversion Of Co (II) Hydroxide In Co (III) Hydroxide U.S., 2005.

Drews, T.; Tsapatsis, M., "Model of the evolution of nanoparticles to crystals via an aggregative growth mechanism." *Microporous and Mesoporous Materials* **2007**, *101*, 97.

Dwyer, F.P.; Gyarfas, E.C.; Mellor, D.P., "The Resolution and Racemization of Potassium Ethylenediaminetetra-acetatocobaltate(III)." *J. Phys. Chem.* **1955**, *59*, 296.

Eibeck, A.; Wagner, W., "Stochastic Particle Approximations for Smoluchoski's Coagulation Equation." *Annals of Applied Probability* **2001**, *11*, 1137.

El-Batlouni, H.; El-Rassy, H.; Al-Ghoul, M., "Cosynthesis, Coexistence, and Self-Organization of [alpha]-and [beta]-Cobalt Hydroxide Based on Diffusion and Reaction in Organic Gels." *Journal of Physical Chemistry A* **2008**, *112*, 7755.

Fernández-Toledano, J.; Moncho-Jordá, A.; Martínez-López, F.; González, A.; Hidalgo-Álvarez, R., "Two-dimensional colloidal aggregation mediated by the range of repulsive interactions." *Physical Review E* **2007**, *75*, 41408.

Figlarz, M.; Guenot, J.; Tournemolle, J.-N., "Oxidation of cobalt (II) hydroxide to oxide hydroxide: solids evolution during reaction." *J. Mater. Sci.* **1974**, *9*, 772.

Figlarz, M.; Guenot, J.; Fievet-Vincent, F., "Morphological and topotactical aspects of the reactions Co(OH)2 -> CoOOH and CoOOH -> Co3O4." *J. Mater. Sci.* **1976**, *11*, 2276.

Fox, R.; Uhlenbeck, G., "Contributions to Non Equilibrium Thermodynamics. I. Theory of Hydrodynamical Fluctuations." *Physics of Fluids* **1970**, *13*, 1893.

Frenklach, M., "Dynamics of discrete distribution for Smoluchowski coagulation model." *Journal of colloid and interface science* **1985**, *108*, 237.

Freund, H.-J.; Libuda, J.; Bäumer, M.; Risse, T.; Carlsson, A., "Cluster, facets, and edges: Site-dependent selective chemistry on model catalysts." *The Chemical Record* **2003**, *3*, 181.

Furlanetto, G.; Formaro, L., "Precipitation of Spherical Co3O4 Particles." *Journal of Colloid and Interface Science* **1995**, *170*, 169.

Furrer, G.; Stumm, W., "The coordination chemistry of weathering: I. Dissolution kinetics of alumina and beryllium oxide." *Geochimica et Cosmochimica Acta* **1986**, *50*, 1847.

Galwey, A., "Compensation effect in heterogeneous catalysis." *Advances in Catalysis* **1977**, *26*, 247.

Galwey, A.K.; Brown, M.E., "Application of the Arrhenius equation to solid state kinetics: can this be justified?" *Thermochimica Acta* **2002**, *386*, 91.

Gao, L.; Zhang, Q., "Effects of amorphous contents and particle size on the photocatalytic properties of TiO2 nanoparticles." *Scripta materialia* **2001**, *44*, 1195.

Garn, P., "An examination of the kinetic compensation effect." *Journal of Thermal Analysis and Calorimetry* **1975**, *7*, 475.

Gas, B.; Stedry, M.; Kenndler, E., "Peak broadening in capillary zone electrophoresis." *Electrophoresis* **1997**, *18*, 2123.

Gilliland, E.R.; Baddour, R.F.; Perkinson, G.P.; Sladek, K.J., "Diffusion on Surfaces. I. Effect of Concentration on the Diffusivity of Physically Adsorbed Gases." *Industrial & Engineering Chemistry Fundamentals* **1974**, *13*, 95.

Grassia, P.; Hinch, E.; Nitsche, L., "Computer simulations of Brownian motion of complex systems." *Journal of Fluid Mechanics* **2006**, *282*, 373.

Han, Y.; Alsayed, A.; Nobili, M.; Zhang, J.; Lubensky, T.; Yodh, A., "Brownian motion of an ellipsoid." *Science* **2006**, *314*, 626.

Handler, R.; Beard, B.; Johnson, C.; Scherer, M., "Atom Exchange between Aqueous Fe (II) and Goethite: An Fe Isotope Tracer Study." *Environmental Science & Technology* **2009**, *43*, 1102.

He, Y.; Liu, S.; Kong, L.; Liu, Z., "A study on the sizes and concentrations of gold nanoparticles by spectra of absorption, resonance Rayleigh scattering and resonance non-linear scattering." *Spectrochimica Acta, Part A: Molecular and Biomolecular Spectroscopy* **2005**, *61*, 2861.

Helmholtz, H.Z., "About Electrical Interfaces." *Annal. Phys. Chem.* **1879**, *7*, 337.

Hodson, M., "Does reactive surface area depend on grain size? Results from pH 3, 25° C far-from-equilibrium flow-through dissolution experiments on anorthite and biotite." *Geochimica et Cosmochimica Acta* **2006**, *70*, 1655.

Holdren, G.R.; Speyer, P.M., "Reaction rate-surface area relationships during the early stages of weathering. II. Data on eight additional feldspars." *Geochimica et Cosmochimica Acta* **1987**, *51*, 2311.

Housecroft, C.E.; Sharpe, A.G. *Inorganic Chemistry*; Prentice Hall, 2001.

Hu, W.K.; Gao, X.P.; Geng, M.M.; Gong, Z.X.; Noreus, D., "Synthesis of CoOOH Nanorods and Application as Coating Materials of Nickel Hydroxide for High Temperature Ni-MH Cells." *J. Phys. Chem. B Lett.* **2005**, *109*, 5392.

Huang, F.; Zhang, H.; Banfield, J.F., "The Role of Oriented Attachment Crystal Growth in Hydrothermal Coarsening of Nanocrystalline ZnS." *Journal of PhysicalChemistry B* **2003**, *107*, 10470.

Ichiyanagi, Y.; Yamada, S., "The size-dependent magnetic properties of Co3O4 nanoparticles." *Polyhedron* **2005**, *24*, 2813.

Isley, S.L.; Anderson, E.R.; Penn, R.L., "Influence of Ionic Strength on Brookite Content in Sol-Gel Synthesized Titania before and after Hydrothermal Aging." *Electrochemical Society Transactions* **2006**, *3*, 37.

Jenkins, D.J.; Alabdulrahman, A.M.S.; Attard, G.A.; Griffin, K.G.; Johnston, P.; Wells, P.B., "Enantioselectivity and catalyst morphology: step and terrace site contributions to rate and enantiomeric excess in Pt-catalysed ethyl pyruvate hydrogenation." *Journal of Catalysis* **2005**, *234*, 230.

Jiang, H.; Rao, L.; Zhang, Z.; Rai, D., "Characterization and oxidation of chromium(III) by sodium hypochlorite in alkaline solutions." *Inorganica Chimica Acta* **2006**, *359*, 3237.

Jolivet, J.-P. *Metal Oxide Chemistry and Synthesis*; John Wiley & Sons Ltd.: West Sussex, England, 2000.

Jones, A.; Rule, J.; Moore, J.; White, S.; Sottos, N., "Catalyst morphology and dissolution kinetics of self-healing polymers." *Chem. Mater* **2006**, *18*, 1312.

Jullien, R.; Kolb, M., "Hierarchical model for chemically limited cluster-cluster aggregation." *Journal of Physics A: Mathematical and General* **1984**, *17*, L639.

Kamath, P.; Therese, A.; Helen, G.; Gopalakrishnan, J., "On the existence of hydrotalcite-like phases in the absence of trivalent cations." *Journal of Solid State Chemistry* **1997**, *128*, 38.

Kawaguchi, H.; Ama, T.; Yasui, T., "The Base-Catalyzed Isomerization of the (Iminodiacetato)(N-methyliminodiacetato)cobaltate(III) and Bis(iminodiacetato)cobaltate(III) Ions." *Bull. Chem. Soc. Jpn.* **1984**, *57*, 2422.

Keijser, T.H.d.; Langford, J.I.; Mittemeijer, E.J.; Vogels, A.B.P., "Use of the Voigt Function in a Single-Line Method for the Analysis of X-ray Diffraction Line Broadening." *J. Appl. Crystallogr.* **1982**, *15*, 308.

Kent, M. *Advanced biology*; Oxford University Press US, 2000.

Kolwankar, K., "Brownian motion of fractal particles: Levy flights from white noise." *Arxiv preprint cond-mat/0511307* **2005**.

Kreer, M.; Penrose, O., "Proof of dynamical scaling in Smoluchowski's coagulation equation with constant kernel." *Journal of Statistical Physics* **1994**, *75*, 389.

Landers, J.P. *Handbook of capillary electrophoresis*; CRC Press, 1997.

Lee, I.; Morales, R.; Albiter, M.A.; Zaera, F., "Synthesis of heterogeneous catalysts with well shaped platinum particles to control reaction selectivity." *Proceedings of the National Academy of Sciences* **2008**, *105*, 15241.

Lin, C.-E.; Wang, T.-Z.; Chiu, T.-C.; Hsueh, C.-C., "Determination of the Critical Micelle Concentration of Cationic Surfactants by Capillary Electrophoresis." *J. High Resol. Chromatogr.* **1999**, *22*, 265.

Liu, J.; Aruguete, D.M.; Jinschek, J.R.; Rimstidt, J.D.; Hochella, M.F., Jr., "The non-oxidative dissolution of galena nanocrystals: Insights into mineral dissolution rates as a function of grain size, shape, and aggregation state." *Geochim. Cosmochim. Acta* **2008**, *72*, 5984.

Liu, W., "Nanoparticles and their biological and environmental applications." *Journal of bioscience and bioengineering* **2006**, *102*, 1.

Liu, Z.; Ma, R.; Osada, M.; Takada, K.; Sasaki, T., "Selective and controlled synthesis of alpha-and beta-cobalt hydroxides in highly developed hexagonal platelets." *Journal of the American Chemical Society* **2005**, *127*, 13869.

Lucy, C.A.; McDonald, T.L., "Separation of Chloride Isotopes by Capillary Electrophoresis Based on the Isotope Effect on Ion Mobility." *Analytical Chemistry* **1995**, *67*, 1074.

Lucy, C.A.; Underhill, R.S., "Characterization of the Cationic Surfactant Induced Reversal of Electroosmotic Flow in Capillary Electrophoresis." *Anal. Chem.* **1996**, *68*, 300.

Ludwig, J.; Vlachos, D.; Van Duin, A.; Goddard III, W., "Dynamics of the dissociation of hydrogen on stepped platinum surfaces using the ReaxFF reactive force field." *J. Phys. Chem. B* **2006**, *110*, 4274.

Lyklema, J.; Van Leeuwen, H.; Minor, M., "DLVO-theory, a dynamic re-interpretation." *Advances in Colloid and Interface Science* **1999**, *83*, 33.

Ma, R.; Liu, Z.; Takada, K.; Fukuda, K.; Ebina, Y.; Bando, Y.; Sasaki, T., "Tetrahedral Co (II) Coordination in [alpha]-Type Cobalt Hydroxide: Rietveld Refinement and X-ray Absorption Spectroscopy." *Inorg. Chem* **2006**, *45*, 3964.

Mansilla, H.D.; Bravo, C.; Ferreyra, R.; Litter, M.I.; Jardim, W.F.; Lizama, C.; Freer, J.; Fernández, J., "Photocatalytic EDTA degradation on suspended and immobilized TiO2." *Journal of Photochemistry and Photobiology A: Chemistry* **2006**, *181*, 188.

Marrink, S.J.; Lindahl, E.; Edholm, O.; Mark, A.E., "Simulation of the Spontaneous Aggregation of Phospholipids into Bilayers." *Journal of the American Chemical Society* **2001**, *123*, 8638.

Marsaglia, G.; Tsang, W., "The ziggurat method for generating random variables." *Journal of Statistical Software* **2000**, *5*, 1.

Matsumoto, M.; Nishimura, T., "Mersenne Twister: A 623-Dimensionally// Equidistributed Uniform Pseudo-Random Number Generator." *ACM Transactions on Modeling and Computer Simulation* **1998**, *8*, 3.

McArdell, C.S.; Stone, A.T.; Tian, J., "Reaction of EDTA and Related Aminocarboxylate Chelating Agents with Co(III)OOH (Heterogenite) and Mn(III)OOH (Manganite)." *Environ. Sci. Technol.* **1998**, *32*, 2923.

Means, J.L.; Crerar, D.A.; Duguid, J.O., "Migration of radioactive wastes: radionuclide mobilization by complexing agents." *Science* **1978**, *200*, 1477.

Melanson, J.E.; Baryla, N.E.; Lucy, C.A., "Dynamic capillary coatings for electroosmotic flow control in capillary electrophoresis." *Trends in analytical chemistry* **2001**, *20*, 365.

Metz, V.; Raanan, H.; Pieper, H.; Bosbach, D.; Ganor, J., "Towards the establishment of a reliable proxy for the reactive surface area of smectite." *Geochimica et Cosmochimica Acta* **2005**, *69*, 2581.

Meusel, I.; Hoffmann, J.; Hartmann, J.; Libuda, J.; Freund, H., "Size Dependent Reaction Kinetics on Supported Model Catalysts: A Molecular Beam/IRAS Study of the CO Oxidation on Alumina-Supported Pd Particles." *J. Phys. Chem. B* **2001**, *105*, 3567.

Mock, J.J.; Barbic, M.; Smith, D.R.; Schultz, D.A.; Schultz, S., "Shape effects in plasmon resonance of individual colloidal silver nanoparticles." *Journal of Chemical Physics* **2002**, *116*, 6755.

Monning, C.A.; Kennedy, R.T., "Capillary Electrophoresis." *Analytical Chemistry* **1996**, 280R.

Myers, J.C.; Penn, R.L., "Evolving Surface Reactivity of Cobalt Oxyhydroxide Nanoparticles." *J. Phys. Chem. C* **2007**, *111*, 10597.

Nath, S.; Kaittanis, C.; Ramachandran, V.; Dalal, N.S.; Perez, J.M., "Synthesis, Magnetic Characterization, and Sensing Applications of Novel Dextran-Coated Iron Oxide Nanorods." *Chemistry of Materials* **2009**, *21*, 1761.

Odriozola, G.; Leone, R.; Schmitt, A.; Moncho-Jordá, A.; Hidalgo-Álvarez, R., "Coupled aggregation and sedimentation processes: The sticking probability effect." *Physical Review E* **2003**, *67*, 31401.

Park, N.-G.; van de Lagemaat, J.; Frank, A.J., "Comparison of Dye-Sensitized Rutile- and Anatase-Based TiO2 Solar Cells." *Journal of Physical Chemistry B* **2000**, *104*, 8989.

Park, S.-J.; Kim, S.; Lee, S.; Khim, Z.G.; Char, K.; Hyeon, T., "Synthesis and Magnetic Studies of Uniform Iron Nanorods and Nanospheres." *Journal of the American Chemical Society* **2000**, *122*, 8581.

Peeters, M.P.J., "An NMR Study of MeTMS Based Coatings Filled with Colloidal Silica." *Journal of Sol-Gel Science and Technology* **2000**, *19*, 131.

Penn, R.L., "Kinetics of Oriented Aggregation." *Journal of PhysicalChemistry B* **2004**, *108*, 12707.

Penn, R.L.; Stone, A.T.; Veblen, D.R., "Defects and Disorder: Probing the Surface Chemisry of Heterogenite (CoOOH) by Dissolution Using Hydroquinone and Iminodiacetic Acid." *J. Phys. Chem. B* **2001**, *105*, 4690.

Penn, R.L.; Tanaka, K.; Erbs, J., "Size dependent kinetics of oriented aggregation." *J. Cryst. Growth* **2007**, *309*, 97.

Piccolo, L.; Henry, C.R., "NO-CO Reaction Kinetics on Pd/MgO Model Catalysts: Morphology and Support Effects." *Journal of Molecular Catalysis A: Chemical* **2001**, *167*, 181.

Poizot, P.; Laruelle, S.; Grugeon, S.; Depont, L.; Tarascon, J.-M., "Nano-sized transition-metal oxides as negative-electrode materials for lithium-ion batteries." *Nature* **2000**, *407*, 496.

Pralong, V.; Delahaye-Vidal, A.; Beaudoin, B.; Gerand, B.; Tarascon, J.-M., "Oxidation mechanism of cobalt hydroxide to cobalt oxyhydroxide." *J. Mater. Chem.* **1999**, *9*, 955.

Pralong, V.; Delahaye-Vidal, A.; Beaudoin, B.; Leriche, J.-B.; Tarascon, J.-M., "Electrochemical behavior of cobalt hydroxide used as additive in the nickel hydroxide electrode." *J. Electrochem. Soc.* **2000**, *147*, 1306.

Rabani, E.; Reichman, D.; Geissler, P.; Brus, L., "Drying-mediated self-assembly of nanoparticles." *Nature* **2003**, *426*, 271.

Rao, L.; Zhang, Z.; Friese, J.I.; Ritherdon, B.; Clark, S.B.; Hess, N.J.; Rai, D., "Oligomerization of Chromium (III) and its impact on the oxidation of Chromium (III) by Hydrogen Peroxide in alkaline solutions." *Journal of the Chemical Society, Dalton Transactions: Inorganic Chemistry* **2002**, *2002*, 267.

Rasband, W.S. ImageJ; U. S. National Institutes of Health: Bethesda, Maryland, USA, http://rsb.info.nih.gov/ij/, 1997-2009.

Reijenga, J.C.; Aben, G.V.A.; Verheggen, T.P.E.M.; Everaerts, F.M., "Effect of Electroosmosis on Detection in Isotachophoresis." *J. Chromatogr.* **1983**, *260*, 241.

Ribeiro, C.; Lee, E.; Longo, E.; Leite, E., "Oriented Attachment Mechanism in Anisotropic Nanocrystals: A “Polymerization” Approach." *ChemPhysChem* **2006**, *7*, 664.

Roeffaers, M.B.J.; Sels, B.F.; Uji-i, H.; De Schryver, F.C.; Jacobs, P.A.; De Vos, D.E.; Hofkens, J., "Spatially Resolved Observations of Crystal-Face-Dependent Catalysis by Single Turnover Counting." *Nature* **2006**, *439*, 572.

Rogel, E., "Simulation of interactions in asphaltene aggregates." *Energy Fuels* **2000**, *14*, 566.

Sadtler, B.; Demchenko, D.O.; Zheng, H.; Hughes, S.M.; Merkle, M.G.; Dahmen, U.; Wang, L.-W.; Alivistados, A.P., "Selective Facet Reactivity during Cation Exchange in Cadmium Sulfide Nanorods." *J. Am. Chem. Soc.* **2009**, *131*, 5285.

Sayle, D.; Seal, S.; Wang, Z.; Mangili, B.; Price, D.; Karakoti, A.; Kuchibhatla, S.; Hao, Q.; Mo bus, G.; Xu, X., "Mapping nanostructure: a systematic enumeration of nanomaterials by assembling nanobuilding blocks at crystallographic positions." *ACS nano* **2008**, *2*, 1237.

Scheffler, I.E. *Mitochondria*; John Wiley and Sons, 1999.

Schubert, M.; Plzak, V.; Garche, J.; Behm, R., "Activity, selectivity, and long-term stability of different metal oxide supported gold catalysts for the preferential CO oxidation in H2-rich gas." *Catalysis Letters* **2001**, *76*, 143.

Schwarz, J.; Contescu, C. *Surfaces of nanoparticles and porous materials*; CRC, 1999.

Sellers, H.; Ulman, A.; Shnidman, Y.; Eilers, J.E., "Structure and binding of alkanethiolates on gold and silver surfaces: implications for self-assembled monolayers." *Journal of the American Chemical Society* **2002**, *115*, 9389.

Shedd, K.B. Cobalt. In *2006 Minerals Yearbook*; U.S. Geological Survey, 2006.

Singh, R.; Mendenhall, R. Heterogenite material for making submicron cobalt powders U.S., 2004.

Spinolo, G.; Ardizzone, S.; Trasatti, S., "Surface characterization of Co3O4 electrodes prepared by the sol-gel method." *Journal of Electroanalytical Chemistry* **1997**, *423*, 49.

St. Claire, R.L., "Capillary electrophoresis." *Analytical Chemistry* **1996**, *68*, 569.

Stone, A.T.; Ulrich, H.J., "Kinetics and reaction stoichiometry in the reductive dissolution of manganese(IV) dioxide and cobalt(III) oxide by hydroquinone." *J. Colloid Interface Sci.* **1989**, *132*, 509.

Stumm, W. *Chemistry of the Solid-Water Interface*; John Wiley & Sons, Inc.: New York, 1992.

Swartz, B.; Donegan, S.; Amos, S.; Reolon, P. "Processing considerations for cobalt recovery from Congolese copperbelt ores"; Hydrometallurgy Conference, 2009, The Southern African Institute of Mining and Metallurgy.

Terabe, S.; Yashima, T.; Tanaka, N.; Araki, M., "Separation of oxygen isotopic benzoic acids by capillary zone electrophoresis based on isotope effects on the dissociation of the carboxyl group." *Analytical Chemistry* **1988**, *60*, 1673.

Terabe, S.; Otsuka, K.; Ichikawa, K.; Tsuchiya, A.; Ando, T., "Electrokinetic separations with micellar solutions and open-tubular capillaries." *Anal. Chem.* **1984**, *56*, 111.

Tsuda, T., "Modification of Electroosmotic Flow with Cetyltrimethylammonium Bromide in Capillary Zone Electrophoresis." *J. high resolut. chromatogr.* **1987**, *10*, 622.

Van Cappellen, P., "Reactive Surface Area Control of the Dissolution Kinetics of Biogenic Silica in Deep-Sea Sediments." *Chemical Geology* **1996**, *132*, 125.

Vermant, J.; Yang, H.; Fuller, G., "Rheooptical determination of aspect ratio and polydispersity of nonspherical particles." *AIChE Journal* **2001**, *47*, 790.

Verziu, M.; Cojocaru, B.; Hu, J.; Richards, R.; Ciuculescu, C.; Filip, P.; Parvulescu, V., "Sunflower and rapeseed oil transesterification to biodiesel over different nanocrystalline MgO catalysts." *Green Chemistry* **2008**, *10*, 373.

Vollmer, S.; Witte, G.; Wöll, C., "Determination of site specific adsorption energies of CO on copper." *Catalysis Letters* **2001**, *77*, 97.

Wang, H., "Thermal rebound of nanometer particles in a diffusion battery." *Aerosol Science and Technology* **1993**, *18*, 180.

Wang, Z.; Huang, X.; Chen, L., "Characterization of Spontaneous Reactions of LiCoO with Electrolyte Solvent for Lithium-Ion Batteries." *Journal of the Electrochemical Society* **2004**, *151*, A1641.

Wang, Z.L.; Feng, X., "Polyhedral Shapes of CeO2 Nanoparticles." *J. Phys. Chem. B* **2003**, *107*, 13563.

Washton, N.; Brantley, S.; Mueller, K., "Probing the molecular-level control of aluminosilicate dissolution: A sensitive solid-state NMR proxy for reactive surface area." *Geochimica et Cosmochimica Acta* **2008**, *72*, 5949.

White, A.; Peterson, M. Chemical Modeling of Aqueous Systems II. In *ACS Symposium Series 416*; American Chemical Society: Washington DC., 1990; pp 461.

White, A.F.; Brantley, S.L., "The effect of time on the weathering of silicate minerals: why do weathering rates differ in the laboratory and field?" *Chemical Geology* **2003**, *202*, 479.

Wieland, E.; Wehrli, B.; Stumm, W., "The coordination chemistry of weathering: III. A generalization on the dissolution rates of minerals." *Geochimica et Cosmochimica Acta* **1988**, *52*, 1969.

Xie, X.; Shen, W., "Morphology control of cobalt oxide nanocrystals for promoting their catalytic performance." *Nanoscale* **2009**, *1*, 50.

Yang, X.; Loos, J.; Veenstra, S.; Verhees, W.; Wienk, M.; Kroon, J.; Michels, M.; Janssen, R., "Nanoscale morphology of high-performance polymer solar cells." *Nano Letters* **2005**, *5*, 579.

Yasui, T.; Kawaguchi, H.; Koine, N.; Ama, T., "Stereochemistry and Properties of unsym-fac-, sym-fac-, and mer- (Iminodiacetato)n(N-alkyliminodiacetato)2-ncobaltate(III) (n=0, 1, or 2) Complexes." *Bull. Chem. Soc. Jpn.* **1983**, *56*, 127.

Yeung, K.K.-C.; Lucy, C.A., "Improved resolution of inorganic anions in capillary electrophoresis by modification of the reversed electroosmotic flow and the anion mobility with mixed surfactants." *J. Chromatogr. A* **1998**, *804*, 319.

Yuan, A.; Cheng, S.; Zhang, J.; Cao, C., "The influence of calcium compounds on the behaviour of the nickel electrode." *Journal of Power Sources* **1998**, *76*, 36.

Zeidan, M.; Jia, X.; Williams, R.; Wedlock, D., "Simulation of aggregation with applications to soot laden lubricating fluids." *Particle & Particle Systems Characterization* **2004**, *21*, 473.

Zhang, L.; Lüttge, A., "Morphological evolution of dissolving feldspar particles with anisotropic surface kinetics and implications for dissolution rate normalization and grain size dependence: A kinetic modeling study." *Geochimica et Cosmochimica Acta* **2009**, *73*, 6757.

Zhang, W.; Elliott, D., "Applications of iron nanoparticles for groundwater remediation." *Remediation Journal* **2006**, *16*, 7.

Zhdanov, V., "Arrhenius parameters for rate processes on solid surfaces." *Surface Science Reports* **1991**, *12*, 185.

Zinder, B.; Furrer, G.; Stumm, W., "The coordination chemistry of weathering: II. Dissolution of Fe (III) oxides." *Geochimica et Cosmochimica Acta* **1986**, *50*, 1861.

A

Source Code: Particle Aggregation Simulation

List of included files:

Particles2.h	-	Header file for simulation objects
Particles2.cpp	-	Implementation for simulation objects
PAgS2.cpp	-	Main simulation program and file interface
Template.agg	-	example simulation instruction file

Required but not included:

MersenneTwister.h	-	Randon number generator file

Implementation by Richard J. Wagner, available at:
http://www-personal.umich.edu/~wagnerr/MersenneTwister.h

Begin File: Particles2.h

```
#ifndef PARTICLES_H
#define PARTICLES_H

#include <iostream>

using namespace std;
typedef unsigned char byte;

// forward declare classes, so pointers to those classes don't freak out when declared
class GridArray;                    // stores all grid information

class Particle;                     // individual particle object
class ParticleArray;                // array, references all Particles on a given GridArray

class Aggregate;                    // an aggregated unit, made of one or more Particles
class AggregateArray;               // array, references all Aggregates
class AggregatePattern;             // a pattern that can be used to save or load an Aggregate

// new enumerated types for specific applications:

// specify the reference corner for aggregate operations
enum Corner_t { UpLeft, UpRight, DownLeft, DownRight };

// error types for movement routines
enum MoveError_t { No_Error, Fixed_Particle, Not_Deployed, Redrop, Collision, Out_of_Range, Move_Count };


// declarations for other functions used by these objects
bool PercentCheck(double PerVal);   // prototype for the function to check % chances
void RotList(byte& BitList, int Times = 1, byte Sides = 4); // function for "rotating" bitlists
```

```
double RandGauss(double sigma);     // prototype for generating Gaussian randoms
inline int round(double x) {        // quick and easy function for rounding
   return int(x > 0.0 ? x + 0.5 : x - 0.5);
}

// tables for use with the Ziggarut algorithm
extern const double ytab[128];
extern const unsigned long ktab[128];
extern const double wtab[128];



/*
=============================================================================================
                                    Grid Declarations
=============================================================================================
*/

// GridArray: represents the simulation grid, used to interact with grid spaces
class GridArray {
private:
      // variables
      int** TheArray;         // 2D array of integers; refers to occupying particles by number
      int GridSize;                        // maximum dimension of the grid
      ParticleArray* PAPtr;         // pointer to the associated ParticleArray
      int ScopeRange;   // the size grid "scope" used to determine the usable range
      byte ScopeType;   // scope types: 0 - fixed, full grid
                        //              1 - expanding, isotropic grid
                        //              2 - expanding, anisotropic grid

      int MinX, MaxX, MinY, MaxY;   // variables for tracking scope boundaries
      int NearX, FarX, NearY, FarY; // track the position of the furthest particles from the center

      // functions
```

```
public:
      int ResizeGrid(int NewSize);  // reallocate to expand/contract TheArray
      int CountNeighbors(int x, int y, int A);  // return the count of occupied neighboring spaces.
BitList, if provided, will store the bitwise directions.
      int CountNeighbors(int x, int y, int A, byte& BitList);
      int CountNeighborsNot(int x, int y, int A);  // return the count of occupied neighboring spaces not
in aggregate A.  BitList, if provided, will store the bitwise directions.
      int CountNeighborsNot(int x, int y, int A, byte& BitList);
      bool OccupiedA(int x, int y, int A);      // returns true if the space is occupied by a particle
belonging to Aggregate A
      byte SecNeighbors(int x, int y, int A, byte Dir, byte RotOff = 0);      // return 2nd nearest
neighbors, optionally offset by a rotation

public:
      // functions
      GridArray(int Size);          // basic constructor, takes size as a parameter
      GridArray(int Size, byte SType, int SRange);// full constructor, initializes scope parameters
      ~GridArray();                       // destructor, frees all allocated memory

      int Size(void);                     // returns the size of the grid
      ParticleArray* GetPAPtr(void);      // points to the grid's ParticleArray
      void SetPAPtr(ParticleArray* PA);   // defines PA as the grid's ParticleArray
      int PRef(int x, int y); // returns the index of the Particle at (x,y)
      int ARef(int x, int y); // returns the index of an aggregate with a member at (x,y)
      bool Occupied(int x, int y);  // returns true if space (x,y) is occupied
      void MarkP(int P, int x, int y);    // mark particle P at (x,y)
      void ClearP(int x, int y);    // clear any marked particle from (x,y)
      void FullMap(ParticleArray* PA = NULL);   // checks all grid positions against the particle array.
Used to resync everything
      int Center(void); // returns the grid center
      bool CheckScope(const int& x, const int& y);    // returns whether the current space is in scope
      byte GetRandomEdge(int& x, int& y, int XWide = 1, int YWide = 1); // generates a random coordinate
pair on the edge of the scope.  Returns the edge selected (top = 1, clockwise).  XWide, YWide can be used
to specify the width of an object requiring extra space.
```

```
      void GetRandomSpace(int& x, int& y, int XWide = 1, int YWide = 1);      // generates a random (x,y)
somewhere inside the scope
      void UpdateScope(void); // used to initialize the scope for the first time.
      void UpdateScope(int x, int y);     // looks at new particle position, updates the scope if needed
      float SurfaceEnergy(int x, int y, int A, int A2=-2);  // find the "energy" associated with a surface
site of aggregate A (if A2 is specified, finds energy of a joined A+A2)
      byte GetScopeType(void);      // functions to check the scope settings
      int GetScopeRange(void);
      float ParticleEnergy(int x, int y); // find the the "energy" associated with an occupied space on the
grid
      float DynamicStickChance(int x, int y, bool unstick = false);     // chance of sticking at a given
surface space
};

/*
=============================================================================================
                                    Particle Declarations
=============================================================================================
*/

// an individual particle.  Almost all interaction must be performed through ParticleArray
class Particle {
private:
      // variables
      int x,y;                          // the (x,y) coordinates on the grid
      byte Orient;                      // stores the particle's orientation
      byte Flags;                       // holds a number of status flags
      byte Attached;                    // tells if the particle is attached to any neighbors
      int AggRef;                       // refers to the Aggregate (if any) the particle belongs to (-1
means no aggregate)

public:
      friend class ParticleArray;   // the ParticleArray object can access the internal variables
```

```
	// functions
	Particle(); // constructor, no parameters (no memory allocated, so no special destructor needed)
};

	/* The Particle Flags variable tracks the following different flag values:
		1 = Deployed.  True if the particle is currently active on the grid.
		2 = Fixed.  True if the particle is immobile, false if it can move
		4 = HadTurn.  True if the particle has already acted on a given turn.  False otherwise.
		8 = Meso.  True if the Attached variable refers to a meso crystal arrangement.
		16 =
		32 =
		64 =
		128 =
	*/


class ParticleArray {
private:
	// variables
	byte NumFaces;				// determines square or hex particles
	bool RandomOrientation;		// does the particle face a random direction at start?
	GridArray* GridPtr;			// pointer to the associated GridArray
	AggregateArray* AggPtr;		// pointer to array of particle aggregates
	Particle* TheArray;			// the array of particles
	int ParticleCount;			// the number of particles in the array
	byte StickType;				// the method of stick checking to use (0 = orient, 1 = crystal)
	float* StickValues;			// array of information used to determine sticking

public:
	static const int BaseStick = 2;		// testing only, will be moved into StickValues eventually.

	// functions
	int ResizeArray(int NewSize); // expand/contract the allocated array by BySize Particles
```

```
      bool StickCheckOrient(int P, int P2);     // performs an orientation-specific stick check between two
particles

public:
      ParticleArray(int Num);       // simple constructor, takes # of particles as an argument
      ParticleArray(int Num, unsigned char NumSides, bool RandSO, GridArray* Grid); // more complex
constructor, creates fully initialized object
      ~ParticleArray();             // destructor, frees all allocated memory
      void Output(int P, int& outX, int& outY); // output particle position info
      int Orient(int P, bool Bitwise = true);   // output the orientation of P (can be non-bitwise)
      void SetOrient(int P, byte NewO);   // set the orientation of a particle
      GridArray* GetGridPtr(void);  // returns a pointer to the GridArray
      void SetGridPtr(GridArray* Grd);    // sets the pointer to grid Grd
      void SetAggPtr(AggregateArray* AA); // assign an AggregateArray to the particles
      byte Faces(void);       // returns the number of faces the particles have
      int NumParticles(void); // returns the number of particles in the array
      bool AllowRot(void);    // returns true if rotation is allowed
      bool ShiftBy(int P, int XShift, int YShift);    // shifts particle P on the grid
      void SetDeployed(int P, bool Deployed);   // set the "deployed" flag for particle P
      bool GetDeployed(int P);      // returns status of the "deployed" flag
      void SetFixed(int P, bool Fixed);   // set the "fixed" flag status
      bool GetFixed(int P);   // return the "fixed" flag status
      void SetHadTurn(int P, bool Turn);  // set the "HadTurn" status flag
      bool GetHadTurn(int P); // return the "HadTurn" status
      void SetMeso(int P, bool Meso);     // set the "meso" flag
      bool GetMeso(int P);    // read the status of the "meso" flag
      MoveError_t RandMove(int P);  // randomly walk particle P one step.  Returns a MoveError_t describing
the results.
      void DropRandom(int P, bool Edge = true); // drop particle P at the edge of the grid
      bool Seed(int P, int SeedX, int SeedY);   // Move and fix particle P at (SeedX, SeedY)
      bool MoveTo(int P, int NewX, int NewY);   // move the particle directly to (x,y)
      byte CountNeighbors(int P);   // returns the number of particles neighboring particle P
      byte GetNeighbors(int P, bool SameAgg = true);  // returns a bitwise list of neighboring particle
positions.  Setting SameAgg = false stops return of neighbors that are members of the same aggregate as P
```

```
      int NextFree(void);     // returns the value of the next undeployed particle
      void Rotate(int P, char Times = -1);      // rotate the particle by counterclockwise by (Times) faces
      void RotAttach(int P, char Times);        // rotates the attachments of a particle
      void SetAggRef(int P, int A); // set AggRef for particle P to aggregate A
      int GetAggRef(int P);   // returns the AggRef value for Particle P
      void GhostTo(int P, int x, int y);  // change particle coordinates without interacting with the grid
or other particles
      void Kill(int P); // remove particle P from the grid and return it to the unassigned particles
      void ClearMark(int P);  // clears the particle's present location from the grid
      void MarkGrid(int P);   // marks the particle's current location on the grid
      byte GetAttached(int P);      // returns the bitwise attachment values for particle P
      void Attach(int P, byte Dir); // attachs particle P to neighboring particles in (bitwise)
direction(s) Dir
      void Unattach(int P, byte Dir);     // remove attachment between P and particle(s) in (bitwise)
direction(s) Dir
      byte CheckStick(int P); // performs a sticking check on particle P.  Uses the StickType variable to
determine check method.
      MoveError_t GaussMove(int P); // moves Particle P according to a Gaussian distribution
      MoveError_t VectorMove(int P, int XVec, int YVec);    // move the particle along vector <XVec, YVec>
      void GaussRot(int P);   // perform particle rotation on a Gaussian distribution
      void SetStickType(byte Type); // change the sticking type for the particles
      byte GetStickType(void);      // return the sticking type used by the particles
      void InitializeSticking(float P1, float P2=0, float P3=0, float P4=0, float P5=0,   // initialize the
sticking variables.  Orient square: (A-A,B-B,A-B); Orient hex: (stick chance);
                  float P6=0, float P7=0, float P8=0, float P9=0, float P10=0);                        //
Crystal square(single, (1 0), (0 1), (1 1), void); Crystal hex: (single, (1 -2), (0 1), void)
      float EnergyValue(string Surface);  // returns energy of the given surface
      float EnergyUnstickValue(string Surface); // returns the unstick energy of a given surface
      float GetStickValue(byte index);    // get one of the stick value parameters
      bool CheckUnstick(int P);     // checks if particle P unsticks.
};

/*
=============================================================================================
```

```
                                    Aggregate Declarations
================================================================================================
*/

class Aggregate {
private:
      // variables
      int* MemberParticles;   // array of pointers to all Particles in the Aggregate
      int MaxSize;                  // maximum # of Particles currently allocated for
      int UsedSize;                 // number of Particles in the array at present
      float SigmaX, SigmaY;   // standard deviations for Brownian movement
      int MoveCount, RotCount;      // delay variables for movement and rotation
      int MaxMoveCount, MaxRotCount;      // how many "steps" are necessary for each move/rotation

      // functions
      void Resize(int NewSize);     // change the maximum allocated size of the aggregate

public:
      // functions
      friend class AggregateArray;// give the array "container" access to the internals
      Aggregate(int Size = 25);     // constructor, MaxSize = Size at start
      ~Aggregate();                       // destructor, frees everything as usual
      Aggregate& operator=(const Aggregate& Agg);     // copy one aggregate to another

};


class AggregateArray {
private:
      // variables
      Aggregate* TheArray;          // array of Aggregate objects
      ParticleArray* PAPtr;         // pointer to the array that holds the particles used
      GridArray* GridPtr;                 // pointer to the grid they are placed on
      int MaxAggs;                        // maximum # of Aggregates currently allocated
```

```
	void ResizeArray(int NewSize);	// change the allocated size of the aggregate array
	void AddBranch(int A, int P); // used to recursively map aggregates on the grid
	bool StickCheckCrystal(int A, int A2);	// checks the sticking of two aggregates based on
crystallographic factors

public:
	// functions
	AggregateArray(int Size = 10, ParticleArray* PA = NULL);	// constructor, MaxAggs = Size at start
	~AggregateArray();			// destructor, does the usual
	void SetPAPtr(ParticleArray* PA);	// assign to a specific ParticleArray
	int AggSize(int A);	// returns the number of particles in Aggregate A
	void AddParticle(int A, int P);	// add particle P to aggregate A
	void RemParticleA(int A, int I);	// remove MemberParticle[I] from aggregate A
	void RemParticleP(int A, int P);	// remove ParticleArray[P] from aggregate A
	void FindEdges(int A, int& Top, int& Bottom, int& Left, int& Right);	// find the farthest edges of
aggregate A
	void FindCorner(int A, int& x, int& y, Corner_t corner = DownLeft);		// get coords of a
corner
	void MapAggregate(int A, int P);	// map aggregate A from all particles connected to P
	void MapAllAggregates(void);	// find and map all aggregates in the particle array
	void SavePattern(int A, AggregatePattern& AP);	// save the aggregate as pattern AP
	void LoadPattern(int A, const AggregatePattern& AP);	// construct an aggregate from pattern AP
	void Kill(int A); // remove an aggregate from the grid.	Also removes member particles.
	int NextFree(void);	// returns the index of the next unused aggregate.
	void Merge(int A1, int A2);	// merges aggregate A1 into A2
	void Clear(int A);	// reset Aggregate A to a blank status
	int PRef(int A, int I); // returns the particle reference for member I of aggregate A
	bool Rotate(int A, char Times = -1);	// rotate Aggregate A counterclockwise by (Times) faces
	MoveError_t MoveTo(int A, int x, int y, Corner_t corner = DownLeft);	// move Aggregate A to (x,y)
on the grid
	void DropRandom(int A, bool Edge = true); // place Aggregate A randomly along the scope boundaries
	int RandMove(int A);	// randomly move Aggregate A. Error codes: 1 - Particle(s) fixed;	2 - Empty
aggregate;	3 - Outside grid, redropped;	4 - Collision, no move.
	bool GetDeployed(int A);	// returns false if any particle in A is not deployed
```

```
      void SetDeployed(int A, bool Deployed);   // change the "deployed" flag for all particles in A
      bool GetFixed(int A);   // returns true if any particle in A is fixed.  Otherwise returns false.
      void SetFixed(int A, bool Fixed);   // change the "fixed" flag for all particles in A
      MoveError_t VectorMove(int A, int XVec, int YVec);    // move Aggregate A along vector <XVec, YVec>
      MoveError_t GaussMove(int A); // move the aggregate in a Gaussian/Brownian fashion
      void GaussRot(int A);   // rotate the aggregate using diffusion rules
      void GaussCalc(int A);  // calculate values for variables used in the Gauss... functions
      int AllocatedAggs(void);            // return the currently allocated number of aggregates
      int UsedAggs(void);           // return the count of aggregates that are being used
      bool CheckStick(int A); // checks for any possible attachments made by aggregate A.  Uses the
ParticleArray StickType variable to choose a sticking method.
      double SurfaceEnergy(int A, bool molar = false);      // returns the total surface energy of an
aggregate.  Molar(true) divides by number of surface sites.
      double SurfaceEnergyComb(int A, int A2, bool molar = false);
      bool HasNeighbors(int A);     // returns true if aggregate A has neighbors (particles or aggregates)
      bool AreNeighbors(int A, int A2);   // true if the two aggregates are neighbors
      void AttachAggs(int A, int A2);     // makes all neighbor attachments
};


class AggregatePattern {
private:
      // variables
      int Size;                           // # of particles comprising the pattern
      int* XOffsets;                      // position of each particle in x direction
      int* YOffsets;                      // ... and in y direction
      byte* Orient;                       // orientation of each particle in the pattern

public:
      friend class AggregateArray;  // Allow AggregateArray access to the private variables

      // functions
      AggregatePattern();                 // constructor, no parameters
      ~AggregatePattern();          // destructor
```

```
      AggregatePattern& operator=(const AggregatePattern& AP2);    // assignment, copies the pattern
};

#endif
```

End File: Particles2.h

Begin File: Particles2.cpp

```
/* This file contains the functions used for the Particle, GridArray, Aggregate,
AggregateArray, and AggregatePattern object types.  It also holds a few global functions
employed by those objects.

It's worth noting that, for purposes of setting pointers for interacting objects, the
order is: GridArray, ParticleArray, AggregateArray.  Each object notifies everything
earlier in the list when its pointers are changed.

*/

#include "Particles2.h"
#include <cstdlib>      // for the general pseudo-random number generator functions
#include "MersenneTwister.h"
#include <string>
#include <cmath>  // rounding function, square root, etc.

extern string mystring;

using namespace std;
MTRand RandGen;          // MT pseudo-random number generator variable.  Seeded by clock settings.

/*
=============================================================================================
                                                    GridArray Functions

=============================================================================================
*/

// basic constructor: creates a Size x Size array of Particle pointers to act as the simulation grid
GridArray::GridArray(int Size) {
```

```
      if ((float)Size/2 == Size/2)                    // always want odd grid dimensions, to have a real
center.
            GridSize = Size + 1;                            // if even, increase Size by one to make odd
      else GridSize = Size;                                 // if odd, use Size as is

      TheArray = new int*[GridSize];              // allocates the first set of pointers (x coordinate)
      for (int i = 0; i < GridSize; i++) {
            TheArray[i]=new int[GridSize];        // each x coord points to an array of GridSpace (y
coordinates)
            for (int j = 0; j < GridSize; j++)
                  TheArray[i][j] = -1;                      // -1 indicates an unoccupied space
      }

      PAPtr = NULL;                                         // currently no ParticleArray associated

      ScopeType = 0;    // default scope is fixed, use entire grid
      UpdateScope();    // construct the initial scope information
}


// full constructor: creates the array and initializes the scope parameters for the grid
GridArray::GridArray(int Size, byte SType, int SRange) {
      if ((float)Size/2 == Size/2)                    // always want odd grid dimensions, to have a real
center.
            GridSize = Size + 1;                            // if even, increase Size by one to make odd
      else GridSize = Size;                                 // if odd, use Size as is

      TheArray = new int*[GridSize];              // allocates the first set of pointers (x coordinate)
      for (int i = 0; i < GridSize; i++) {
            TheArray[i]=new int[GridSize];        // each x coord points to an array of GridSpace (y
coordinates)
            for (int j = 0; j < GridSize; j++)
                  TheArray[i][j] = -1;                      // -1 indicates an unoccupied space
      }
```

```
	PAPtr = NULL;                                          // currently no ParticleArray associated

	ScopeRange = SRange;     // set the scope range
	ScopeType = SType;       // set the scope type
	UpdateScope();     // call UpdateScope() to build the initial grid scope
}


// destructor: frees the memory used for the pointers.  Does not delete any particles
GridArray::~GridArray() {
	for (int i = 0; i < GridSize; i++) {            // clear the second dimension of the arry
		delete [] TheArray[i];
		TheArray[i] = NULL;
	}
	delete [] TheArray;                                        // clear the first dimension
	TheArray = NULL;

}


// Redefines the size of the allocated array.  Keeps all Particles in position, using the center as the
reference point
int GridArray::ResizeGrid(int NewSize) {

	// define a temporary array to hold the grid data
	int** TempArray = new int*[GridSize];     // first dimension
	for (int i = 0; i < GridSize; i++) {
		TempArray[i] = new int[GridSize];         // second dimension
		for (int j = 0; j < GridSize; j++)
			TempArray[i][j] = TheArray[i][j];   // copy over Particle reference
	}

	// clear out the original array data
```

```
	for (int i = 0; i < GridSize; i++) {
		delete [] TheArray[i];
		TheArray[i] = NULL;
	}
	delete [] TheArray;
	TheArray = NULL;

	// Check to ensure an odd NewSize (again, to make sure there's a true center space)
	if ((float)NewSize/2 == NewSize/2)
		NewSize++;                                          // if even, increase NewSize by one to
make odd

	// reallocate TheArray to the NewSize x NewSize
	TheArray = new int*[NewSize];
	for (int i = 0; i < NewSize; i++) {
		TheArray[i] = new int[NewSize];
		for (int j = 0; j < NewSize; j++)
			TheArray[i][j] = -1;    // initialize the new array to empty spaces
	}

	// Finally, copy the contents, keeping the center the same
	int SizeDiff = (NewSize - GridSize)/2;          // the change in each direction from the old grid to
the new
	for (int i = 0; i < GridSize; i++) {            // cycle through the old grid
		for (int j = 0; j < GridSize; j++)
			if (TempArray[i][j] != -1) {  // if there is a particle located at (i,j)
				// check to make sure the particle is still on the new grid
				if ((i + SizeDiff >= 0) && (j + SizeDiff >= 0) && (i + SizeDiff < NewSize) && (j +
SizeDiff < NewSize)) {
					int x, y;
					PAPtr->Output(TempArray[i][j], x, y);
					PAPtr->GhostTo(TempArray[i][j], x + SizeDiff, y + SizeDiff);      // change
the particle to the new position
```

```
                              TheArray[i+SizeDiff][j+SizeDiff] = TempArray[i][j];   // copy its new
position to the grid
                        }
                        else PAPtr->SetDeployed(TempArray[i][j],false); // if it moved off the grid, mark
it as not deployed
                  }
            delete [] TempArray[i];       // clear that portion of the TempArray memory
            TempArray[i] = NULL;
      }

      delete [] TempArray;    // clear the remainder of the TempArray memory
      TempArray = NULL;

      GridSize = NewSize;           // update the size of the grid

      UpdateScope();    // call the scope initialization to rebuild the scope with new coordinates

      return 0;   // 0 indicated no errors
}

// returns the current size of the grid
int GridArray::Size(void) {
      return GridSize;
}


// returns a pointer to the ParticleArray matched to this grid
ParticleArray* GridArray::GetPAPtr(void) {
      return PAPtr;
}


// sets the ParticleArray pointer to PA
void GridArray::SetPAPtr(ParticleArray* PA) {
```

```
	PAPtr = PA;
}


// get the ID of the Particle located at (x,y)
int GridArray::PRef(int x, int y) {
	if ((x < 0) || (x >= GridSize) || (y < 0) || (y >= GridSize)) return -1;	// out of bounds,
return empty space
	return TheArray[x][y];  // otherwise, give the particle reference number
}


// get the ID of the aggregate (if any) that has a member at (x,y)
int GridArray::ARef(int x, int y) {
	if (Occupied(x,y))
		return PAPtr->GetAggRef(TheArray[x][y]);
	else return -1;   // an empty space has no aggregate
}


// determine if space (x,y) is occupied
bool GridArray::Occupied(int x, int y) {
	// if the space is outside the defined grid, call it unoccupied (this stops problems with neighbor
determination)
	if ((x < 0) || (y < 0) || (x >= GridSize) || (y >= GridSize)) return false;

	if (TheArray[x][y] == -1) return false;   // -1 corresponds to an empty space

	else return true; // if not empty, return true

}


// mark particle P at (x,y) on the grid
```

```
void GridArray::MarkP(int P, int x, int y) {
      TheArray[x][y] = P;
}


// clear any marked particle from (x,y)
void GridArray::ClearP(int x, int y) {
      if ((x < 0) || (y < 0) || (x >= GridSize) || (y >= GridSize)) return;   // if outside the allocated
grid, do nothing
      TheArray[x][y] = -1;    // -1 indicates an empty grid space
}


// force the grid to resync with the particle array.  Uses PAPtr by default
void GridArray::FullMap(ParticleArray* PA) {
      for (int i = 0; i < GridSize; i++)  // begin by clearing the grid entirely
            for (int j = 0; j < GridSize; j++)
                  TheArray[i][j] = -1;    // -1 indicates an empty space

      if (PA == NULL) PA = PAPtr;   // if nothing specified, use the existing Particle Array
      if (PA == NULL) return; // if it's still null, exit here (just clears the grid)

      if (PA != PAPtr) PAPtr = PA;  // if it refers to a different array, make that the associated array

      int px, py; // store particle coords

      for (int i = 0; i < PAPtr->NumParticles(); i++) {
            PA->Output(i, px, py);  // get the particle coordinates
            if ((px >= 0) && (px < GridSize) && (py >= 0) && (py < GridSize)) // if it's in a valid grid
position
                  TheArray[px][py] = i;   // link that grid space to the particle
      }
}
```

```
// returns the center of a grid
int GridArray::Center(void) {
      return (GridSize/2);
}


// takes a coordinate, returns true if it is within the current grid scope
/* Maybe implemented ?: not sure if any other scope types are needed, or what they would be */
bool GridArray::CheckScope(const int& x, const int& y) {
      if ((x < MinX) || ( x > MaxX) || (y < MinY) || (y > MaxY)) // the point is outside of the defined
scope
            return false;
      else return true;
}


// this generates a random position on the edge of the grid scope.  Used to simplify DropRandom type
functions
byte GridArray::GetRandomEdge(int& x, int& y, int XWide, int YWide) {
      byte ChooseSide;

      ChooseSide = (byte)RandGen.randInt(3);    // randomly select a side of the grid to drop at

      if (ChooseSide == 0) {                                          // top of the grid scope
            x = MinX + RandGen.randInt(MaxX - MinX - XWide);      // random spot [MinX,MaxX] (adjusted for
the specified width)
            y = MaxY - YWide;                                 // y = MaxY, top edge of the scope (again,
adjusted for extra width)
      }
      else if (ChooseSide == 1) {                                     // right side of scope
            x = MaxX - XWide;
            y = MinY + RandGen.randInt(MaxY - MinY - YWide);
      }
```

```
	else if (ChooseSide == 2) {                                    // bottom of scope
		x = MinX + RandGen.randInt(MaxX - MinX - XWide);
		y = MinY;
	}
	else if (ChooseSide == 3) {                                    // left side
		x = MinX;
		y = MinY + RandGen.randInt(MaxY - MinY - YWide);
	}
	return ChooseSide+1;                                           // return which edge was selected
}


void GridArray::GetRandomSpace(int& x, int& y, int XWide, int YWide) {
	x = MinX + RandGen.randInt(MaxX - MinX - XWide);
	y = MinY + RandGen.randInt(MaxY - MinY - YWide);
}


// When called with no parameters, UpdateScope initializes the grid scope variables
// It is used when a new GridArray is created, or when the grid is resized.
void GridArray::UpdateScope(void) {
	if (ScopeType == 0) {
		MinX = 0;    // Type 0 uses the entire grid, never expands
		MinY = 0;
		MaxX = GridSize - 1;
		MaxY = GridSize - 1;
	}
	else {       // other scope types expand as particles grow outward
		int px, py;

		// set all Near/Far variables equal to the center to start
		NearX = Center();  FarX = NearX;  NearY = NearX;  FarY = NearX;
```

```
            if (PAPtr != NULL)      // if a ParticleArray is assigned, search it for the most distant
particles
                  for (int i = 0; i < PAPtr->NumParticles(); i++)
                        if (PAPtr->GetFixed(i)) {     // only look at fixed particles
                              PAPtr->Output(i, px, py);           // get particle coordinates
                              if (px < NearX) NearX = px;         // compare against the most distant
points
                              if (px > FarX) FarX = px;
                              if (py < NearY) NearY = py;
                              if (py > FarY) FarY = py;
                        }

            // call the regular update scope procedure for each corner
            UpdateScope(NearX, NearY);
            UpdateScope(NearX, FarY);
            UpdateScope(FarX, NearY);
            UpdateScope(FarX, FarY);
      }
}

// Otherwise, it updates the scope in response to new particle positions.
void GridArray::UpdateScope(int x, int y) {
      if (ScopeType == 0) return;         // Type 0 scope never needs updating

      if ((x < FarX) && (x > NearX) && (y < FarY) && (y > NearY))
            return;           // particle is not further away, no update needed

      else {      // if the particle is farthest from the center in some direction
            if (x > FarX) FarX = x;       // update the relevant Near/Far data
            if (x < NearX) NearX = x;
            if (y > FarY) FarY = y;
            if (y < NearY) NearY = y;

            // Isotropic scope, all sides grow farther out when a particle nears any side
```

```
            if (ScopeType == 1) {

                  // Set FarPoint equal to the greatest distance from the center of the grid
                  int FarPoint = FarX - Center();
                  if ((Center() - NearX) > FarPoint) FarPoint = Center() - NearX;
                  if ((FarY - Center()) > FarPoint) FarPoint = FarY - Center();
                  if ((Center() - NearY) > FarPoint) FarPoint = Center() - NearY;

                  // Grow the entire scope
                  MinX = Center() - FarPoint - ScopeRange;
                  MaxX = Center() + FarPoint + ScopeRange;
                  MinY = MinX;
                  MaxY = MaxX;
            }

            // Anisotropic scope, only the side nearest the particle grows out
            else if (ScopeType == 2) {
                  MinX = NearX - ScopeRange;
                  MaxX = FarX + ScopeRange;
                  MinY = NearY - ScopeRange;
                  MaxY = FarY + ScopeRange;
            }

            // if the scope expands beyond the allocated grid space, reallocate the grid to be larger
            if ((MaxX >= GridSize) || (MinX < 0) || (MaxY >= GridSize) || (MinY < 0))
                  ResizeGrid(GridSize + 2*ScopeRange);      // expand the grid by ScopeRange in all
directions
      }
}


// Count neighboring particles to a position that belong to Aggregate A (without the BitList)
int GridArray::CountNeighbors(int x, int y, int A) {
```

```
      if ((x < 0) || (y < 0) || (x >= MaxX) || (y >= MaxY)) return 0;         // no neighbors if outside
the range
      if (A < -1) return 0;   // do nothing if an invalid aggregate is specified

      int NCount = 0;
      // for square grids
      if (PAPtr->Faces() == 4) {
            if (Occupied(x,y+1) && (ARef(x,y+1) == A)) NCount++;              // increase the neighbor
count
            if (Occupied(x+1,y) && (ARef(x+1,y) == A)) NCount++;
            if (Occupied(x,y-1) && (ARef(x,y-1) == A)) NCount++;
            if (Occupied(x-1,y) && (ARef(x-1,y) == A)) NCount++;
      }

      // for hexagonal grids
      else if (PAPtr->Faces() == 6) {
            if (Occupied(x,y+1) && (ARef(x,y+1) == A)) NCount++;
            if (Occupied(x+1,y) && (ARef(x+1,y) == A)) NCount++;
            if (Occupied(x+1,y-1) && (ARef(x+1,y-1) == A)) NCount++;
            if (Occupied(x,y-1) && (ARef(x,y-1) == A)) NCount++;
            if (Occupied(x-1,y) && (ARef(x-1,y) == A)) NCount++;
            if (Occupied(x-1,y+1) && (ARef(x-1,y+1) == A)) NCount++;
      }

      return NCount;                                          // return the neighbor count
}


// Count neighboring particles to a position that belong to Aggregate A (and keep a bitlist of them)
int GridArray::CountNeighbors(int x, int y, int A, byte& BitList) {
      BitList = 0;                        // reset the list to no neighbors

      if ((x < 0) || (y < 0) || (x >= MaxX) || (y >= MaxY)) return 0;         // no neighbors if outside
the range
```

```
      if (A < -1) return 0;   // do nothing if an invalid aggregate is specified

      int NCount = 0;

      // for square grids
      if (PAPtr->Faces() == 4) {
            if (Occupied(x,y+1) && (ARef(x,y+1) == A)) {    // if a particle is above the space and belongs
to aggregate A
                  NCount++;               // increase the neighbor count
                  BitList |= 1;                 // mark it in the bit list
            }
            if (Occupied(x+1,y) && (ARef(x+1,y) == A)) {
                  NCount++;
                  BitList |= 2;
            }
            if (Occupied(x,y-1) && (ARef(x,y-1) == A)) {
                  NCount++;
                  BitList |= 4;
            }
            if (Occupied(x-1,y) && (ARef(x-1,y) == A)) {
                  NCount++;
                  BitList |= 8;
            }
      }
      // for hexagonal grids
      else if (PAPtr->Faces() == 6) {
            if (Occupied(x,y+1) && (ARef(x,y+1) == A)) {
                  NCount++;
                  BitList |=1;
            }
            if (Occupied(x+1,y) && (ARef(x+1,y) == A)) {
                  NCount++;
                  BitList |=2;
            }
```

```
            if (Occupied(x+1,y-1) && (ARef(x+1,y-1) == A)) {
                  NCount++;
                  BitList |=4;
            }
            if (Occupied(x,y-1) && (ARef(x,y-1) == A)) {
                  NCount++;
                  BitList |=8;
            }
            if (Occupied(x-1,y) && (ARef(x-1,y) == A)) {
                  NCount++;
                  BitList |=16;
            }
            if (Occupied(x-1,y+1) && (ARef(x-1,y+1) == A)) {
                  NCount++;
                  BitList |=32;
            }
      }

      return NCount;                                          // return the neighbor count
}

// Count neighboring particles to a position that don't belong to Aggregate A (without the BitList)
int GridArray::CountNeighborsNot(int x, int y, int A) {
      if ((x < 0) || (y < 0) || (x >= MaxX) || (y >= MaxY)) return 0;         // no neighbors if outside
the range
      if (A < -1) return 0;   // do nothing if an invalid aggregate is specified

      int NCount = 0;
      // for square grids
      if (PAPtr->Faces() == 4) {
            if (Occupied(x,y+1) && (ARef(x,y+1) != A)) NCount++;              // increase the neighbor
count
            if (Occupied(x+1,y) && (ARef(x+1,y) != A)) NCount++;
            if (Occupied(x,y-1) && (ARef(x,y-1) != A)) NCount++;
```

```
            if (Occupied(x-1,y) && (ARef(x-1,y) != A)) NCount++;
      }

      // for hexagonal grids
      else if (PAPtr->Faces() == 6) {
            if (Occupied(x,y+1) && (ARef(x,y+1) != A)) NCount++;
            if (Occupied(x+1,y) && (ARef(x+1,y) != A)) NCount++;
            if (Occupied(x+1,y-1) && (ARef(x+1,y-1) != A)) NCount++;
            if (Occupied(x,y-1) && (ARef(x,y-1) != A)) NCount++;
            if (Occupied(x-1,y) && (ARef(x-1,y) != A)) NCount++;
            if (Occupied(x-1,y+1) && (ARef(x-1,y+1) != A)) NCount++;
      }

      return NCount;                                          // return the neighbor count
}


// Count neighboring particles to a position that don't belong to Aggregate A (and keep a bitlist of them)
int GridArray::CountNeighborsNot(int x, int y, int A, byte& BitList) {
      BitList = 0;                        // reset the list to no neighbors

      if ((x < 0) || (y < 0) || (x >= MaxX) || (y >= MaxY)) return 0;         // no neighbors if outside
the range
      if (A < -1) return 0;   // do nothing if an invalid aggregate is specified

      int NCount = 0;

      // for square grids
      if (PAPtr->Faces() == 4) {
            if (Occupied(x,y+1) && (ARef(x,y+1) != A)) {    // if a particle is above the space and belongs
to aggregate A
                  NCount++;               // increase the neighbor count
                  BitList |= 1;                 // mark it in the bit list
            }
```

```
            if (Occupied(x+1,y) && (ARef(x+1,y) != A)) {
                  NCount++;
                  BitList |= 2;
            }
            if (Occupied(x,y-1) && (ARef(x,y-1) != A)) {
                  NCount++;
                  BitList |= 4;
            }
            if (Occupied(x-1,y) && (ARef(x-1,y) != A)) {
                  NCount++;
                  BitList |= 8;
            }
      }
      // for hexagonal grids
      else if (PAPtr->Faces() == 6) {
            if (Occupied(x,y+1) && (ARef(x,y+1) != A)) {
                  NCount++;
                  BitList |=1;
            }
            if (Occupied(x+1,y) && (ARef(x+1,y) != A)) {
                  NCount++;
                  BitList |=2;
            }
            if (Occupied(x+1,y-1) && (ARef(x+1,y-1) != A)) {
                  NCount++;
                  BitList |=4;
            }
            if (Occupied(x,y-1) && (ARef(x,y-1) != A)) {
                  NCount++;
                  BitList |=8;
            }
            if (Occupied(x-1,y) && (ARef(x-1,y) != A)) {
                  NCount++;
                  BitList |=16;
```

```
            }
            if (Occupied(x-1,y+1) && (ARef(x-1,y+1) != A)) {
                  NCount++;
                  BitList |=32;
            }
      }

      return NCount;                                          // return the neighbor count
}


// checks if (x,y) is occupied by a space belonging to aggregate A
bool GridArray::OccupiedA(int x, int y, int A) {
      if (A < -1)       // for invalid aggregates
            return 0;
      else
            return (Occupied(x,y) && (ARef(x,y) == A));
}


// find 2nd nearest neighbors of (x,y) in direction Dir.  Optionally corrects for a rotation offset
(RotOff)
byte GridArray::SecNeighbors(int x, int y, int A, byte Dir, byte RotOff) {
      int Nx (-1), Ny (-1);

      if (A < -1) return 0;   // do nothing if an invalid aggregate is specified

      if (RotOff != 0)
            RotList(Dir,-RotOff,PAPtr->Faces());

      // squares
      if (PAPtr->Faces() == 4) {
            if (Dir == 1) {         // (x,y+1)
                  Nx = x; Ny = y + 1;
```

```
        }
        else if (Dir == 2) {            // (x+1,y)
              Nx = x+1; Ny = y;
        }
        else if (Dir == 4) {            // (x,y-1)
              Nx = x; Ny = y - 1;
        }
        else if (Dir == 8) {            // (x-1,y)
              Nx = x-1; Ny = y;
        }
  }
  // hexagons
  else if (PAPtr->Faces() == 6) {
        if (Dir == 1) {           // (x,y+1)
              Nx = x; Ny = y + 1;
        }
        else if (Dir == 2) {      // (x+1, y)
              Nx = x+1; Ny = y;
        }
        else if (Dir == 4) {      // (x+1, y-1)
              Nx = x+1; Ny = y-1;
        }
        else if (Dir == 8) {      // etc...
              Nx = x; Ny = y-1;
        }
        else if (Dir == 16) {
              Nx = x-1; Ny = y;
        }
        else if (Dir == 32) {
              Nx = x-1; Ny = y+1;
        }
  }

  if ((Nx == -1) || (Ny == -1))
```

```
            return 0;         // something was invalid, return no neighbors

      byte NList;
      CountNeighbors(Nx,Ny,A,NList);      // get the list of neighbors for that position (belonging to
aggregate A)

      if (RotOff != 0)              // if there's a rotation offset
            RotList(NList,RotOff,PAPtr->Faces());     // rotate the new list to that offset

      return NList;     // return the result
}


// finds the energy associated with a surface site
float GridArray::SurfaceEnergy(int x, int y, int A, int A2) {
      if ((x < 0) || (y < 0) || (x >= MaxX) || (y >= MaxY)) return 0;         // do nothing if the space is
out of range

      // because we're interested in particles belonging to a certain aggrate, we used OccupiedA
      if (OccupiedA(x,y,A) || OccupiedA(x,y,A2)) return 0;                    // if it isn't empty, it
isn't a surface

      int SEnergy = 0;
      byte NList(0), NList2(0);     // bitmask lists of neighbors
      int NCount;             // map the neighbors belonging to A
      int RotOff (0);         // rotation offset (for checking all surfaces in the same orientation, less
code that way)

      NCount = CountNeighbors(x,y,A,NList) + CountNeighbors(x,y,A2,NList2);
      NList = NList | NList2;
      byte NLMin (NList), ROMin (0);            // store the minimum values (for standard position)

      if (NCount == 0) return 0;          // no neighbors = not a surface
      float Energy = 0;                   // used for spaces boardering more than one surface type
```

```
	// this part puts everything in "standard" orientation... the lowest sum of side numbers
	if (NCount < PAPtr->Faces()) {
		for (int i = 0; i < PAPtr->Faces(); i++) {
			RotList(NList,1,PAPtr->Faces());	// perform one rotation
			RotOff++;					// increment the rotation count
			if (NList < NLMin) {			// if the side values are the lowest yet
				NLMin = NList;				// update the minimum values
				ROMin = RotOff;
			}
		}
		NList = NLMin;			// set the NList and RotOff variables to the minimized values
		RotOff = ROMin;
	}

	// begin determining the surface structure
	// square particles
	if (PAPtr->Faces() == 4) {

		// positions with only one neighbor particle
		if (NCount == 1) {	// the only neighbor is in "1", see if it's a single or part of a
surface
			byte NL2 = SecNeighbors(x,y,A,1,RotOff)|SecNeighbors(x,y,A2,1,RotOff);	// get the
2nd nearest neighbors
			//cout << "AggA(" << A << ") SecN: " << (int)NL2 << endl;
			//cout << "AggA2(" << A2 << ") SecN: " << (int) SecNeighbors(x,y,A2,1,RotOff) << endl;
			if (((NL2 & 2) == 2) || ((NL2 & 8) == 8)) { // it's a surface
				if ((RotOff % 2) == 0) return PAPtr->EnergyValue("(0 1)");  // if even RotOff, it's
a (0 1) type
				else return PAPtr->EnergyValue("(1 0)");	// if odd, it's a (1 0) type
			}
			else return PAPtr->EnergyValue("single"); // not a part of a surface
		}
```

```
            // positions with two neighboring particles
            if (NCount == 2) {
                  if ((NList & 2) == 2)                        // one neighbor is above, if the other is right
                        return PAPtr->EnergyValue("(1 1)");                    // it's a step surface, (1 1)
type
                  else {            // the two particles are opposite, check to see if they are singles or
surfaces
                        byte NL2 = SecNeighbors(x,y,A,1,RotOff)| SecNeighbors(x,y,A2,1,RotOff);       //
secondary neighbors up
                        byte NL3 = SecNeighbors(x,y,A,4,RotOff)| SecNeighbors(x,y,A2,4,RotOff);       //
secondary neighbors down

                        if (((NL2 & 2) == 2) || ((NL2 & 8) == 8)) { // perform the single particle test on
the particle above
                              if ((RotOff % 2) == 0) Energy = PAPtr->EnergyValue("(0 1)");
                              else Energy = PAPtr->EnergyValue("(1 0)");
                        }
                        else Energy = PAPtr->EnergyValue("single");

                        if (((NL3 & 2) == 2) || ((NL3 & 8) == 8)) { // and the one below
                              if ((RotOff % 2) == 0) Energy += PAPtr->EnergyValue("(0 1)");     // if even
RotOff, it's a (0 1) type
                              else Energy += PAPtr->EnergyValue("(1 0)");           // if odd, it's a (1 0)
type
                        }
                        else Energy += PAPtr->EnergyValue("single");    // not a part of a surface
                        return Energy;
                  }
            }

            // positions with three or more particles
            if (NCount == 3)
                  return PAPtr->EnergyValue("void");        // with 3 neighbors, this must be a gap space
            if (NCount == 4)
```

```
                  return 10*PAPtr->EnergyValue("void");     // a true void.  Set energy at tenfold to
discourage formation of these
      }

      // hexagonal particles
      else if(PAPtr->Faces() == 6) {

            // positions with only one neighbor
            if (NCount == 1) {            // neighbor is in position "1" (because of RotOff)
                  byte NL2 = SecNeighbors(x,y,A,1,RotOff)|SecNeighbors(x,y,A2,1,RotOff);        // get the
2nd nearest neighbors
                  if ((NL2 & 34) == 34)   // if 2nd neighbors are in positions "2" and "32"
                        return PAPtr->EnergyValue("(0 1)"); // this is the convex part of a (0 1) surface
                  else
                        return PAPtr->EnergyValue("single");      // otherwise, a single surface
            }

            // positions with two neighbors
            if (NCount == 2) {
                  if( ((NList & 2) == 2) || ((NList & 32) == 32))       // (1 -2) type surface conditions
                        return PAPtr->EnergyValue("(1 -2)");
                  else {
                        byte OtherP = NList & ~1;
                        byte NL2 = SecNeighbors(x,y,A,1,RotOff)|SecNeighbors(x,y,A2,1,RotOff);        //
2nd neighbors of particle at "1"
                        byte NL3 = SecNeighbors(x,y,A,OtherP,RotOff)|SecNeighbors(x,y,A2,OtherP,RotOff);
      // 2nd neighbors of other particle

                        // check the state for the "1" particle
                        if ((NL2 & 34) == 34)
                              Energy = PAPtr->EnergyValue("(0 1)");
                        else
                              Energy = PAPtr->EnergyValue("single");
                        // and the other particle
```

```
                if ((NL3 & ((OtherP << 2) | (OtherP >> 2))) != 0)
                        Energy += PAPtr->EnergyValue("(0 1)");
                else
                        Energy += PAPtr->EnergyValue("single");
                return Energy;
        }
}

// positions with three neighbors
if (NCount == 3) {
        if (NList == 7)          // concave part of (0 1) surface
                return PAPtr->EnergyValue("(0 1)");
        else if (NList == 11) { // (1 -2) in "1" and "2", plus single in "8"
                Energy = PAPtr->EnergyValue("(1 -2)");
                byte NL2 = SecNeighbors(x,y,A,8,RotOff)|SecNeighbors(x,y,A2,8,RotOff);
                if ((NL2 & 20) == 20)
                        Energy += PAPtr->EnergyValue("(0 1)");
                else
                        Energy += PAPtr->EnergyValue("single");
        }
        else if (NList == 13) { // (1 -2) in "4" and "8", plus single in "1"
                Energy = PAPtr->EnergyValue("(1 -2)");
                byte NL2 = SecNeighbors(x,y,A,1,RotOff)|SecNeighbors(x,y,A2,1,RotOff);
                if ((NL2 & 34) == 34)
                        Energy += PAPtr->EnergyValue("(0 1)");
                else
                        Energy += PAPtr->EnergyValue("single");
        }
        else if (NList == 21) { // three singles at "1", "4", and "16"
                byte NL2 = SecNeighbors(x,y,A,1, RotOff)|SecNeighbors(x,y,A2,1,RotOff);
                byte NL3 = SecNeighbors(x,y,A,4, RotOff)|SecNeighbors(x,y,A2,4,RotOff);
                byte NL4 = SecNeighbors(x,y,A,16, RotOff)|SecNeighbors(x,y,A2,16,RotOff);
                if ((NL2 & 34) == 34) Energy = PAPtr->EnergyValue("(0 1)");
                else Energy = PAPtr->EnergyValue("single");
```

```
                        if ((NL3 & 10) == 10) Energy += PAPtr->EnergyValue("(0 1)");
                        else Energy += PAPtr->EnergyValue("single");

                        if ((NL4 & 40) == 40) Energy += PAPtr->EnergyValue("(0 1)");
                        else Energy += PAPtr->EnergyValue("single");
                  }
                  return Energy;
            }

            // positions with four neighbors
            if (NCount == 4) {
                  if (NList == 15)  // gap made by "1", "2", "4", and "8"
                        return PAPtr->EnergyValue("void");
                  else if (NList == 23) {       // (0 1) in "1", "2", and "4" plus single at "16"
                        Energy = PAPtr->EnergyValue("(0 1)");
                        byte NL2 = SecNeighbors(x,y,A,16,RotOff)|SecNeighbors(x,y,A2,16,RotOff);
                        if ((NL2 & 40) == 40) Energy += PAPtr->EnergyValue("(0 1)");
                        else Energy += PAPtr->EnergyValue("single");
                        return Energy;
                  }
                  else if (NList == 27)   // two (1 -2) at "1" & "2" and "8" & "16"
                        return 2*PAPtr->EnergyValue("(1 -2)");
            }

            // positions with five or six neighbors
            if (NCount == 5)
                  return PAPtr->EnergyValue("void");  // with five partners, this must be a gap
            if (NCount == 6)
                  return 10*PAPtr->EnergyValue("void");     // a true void.  Set energy at tenfold to
discourage this
      }
```

```
	return 0;	// If it gets here, something odd happened.  Return 0 and hope it doesn't cause any
problems
}

// pair of functions to display the scope settings for the grid
byte GridArray::GetScopeType(void) {
	return this->ScopeType;
}
int GridArray::GetScopeRange(void) {
	return this->ScopeRange;
}

// returns the "energy" value of a particle, based on the surface it's stuck to
float GridArray::ParticleEnergy(int x, int y) {
	int ParticleNum = this->PRef(x,y);
	if (ParticleNum < 0) return 0;			// there isn't a particle there
	this->ClearP(x,y);	// clear the grid marking temporarily

	float Energy = this->SurfaceEnergy(x,y,PAPtr->GetAggRef(ParticleNum));	// get the surface energy
without the particle

	this->MarkP(ParticleNum,x,y); // re-mark the particle on the grid
	return Energy;
}

// get the stick chance based on the surface available at (x,y)
float GridArray::DynamicStickChance(int x, int y, bool unstick) {
	if ((x < 0) || (y < 0) || (x >= MaxX) || (y >= MaxY)) return -1;		// do nothing if the space is
out of range

	int NCount;			// # of neighboring particles
	byte NList(0);			// bitmask list of neighbors
	int RotOff (0);			// rotation offset (for checking all surfaces in the same orientation, less
code that way)
```

```
// pointer to function garbage... Hope this works
float (ParticleArray::*Chance)(string) = NULL;

NCount = CountNeighborsNot(x, y, ARef(x,y));
if (NCount == 0) return -1;          // no neighbors = not a surface
byte NLMin (NList), ROMin (0);             // store the minimum values (for standard position)
int A;      // the aggregate which surface we're looking at
// find the aggregate... this assumes there's only one... good assumption in almost all cases
if (PAPtr->Faces() == 4) {
      if ((NList & 1) == 1)
            A = ARef(x,y+1);
      else if ((NList & 2) == 2)
            A = ARef(x+1,y);
      else if ((NList & 4) == 4)
            A = ARef(x,y-1);
      else if ((NList & 8) == 8)
            A = ARef(x-1,y);
}
else if (PAPtr->Faces() == 6) {
      if ((NList & 1) == 1)
            A = ARef(x,y+1);
      else if ((NList & 2) == 2)
            A = ARef(x+1,y);
      else if ((NList & 4) == 4)
            A = ARef(x+1,y-1);
      else if ((NList & 8) == 8)
            A = ARef(x,y-1);
      else if ((NList & 16) == 16)
            A = ARef(x-1,y);
      else if ((NList & 32) == 32)
            A = ARef(x-1,y+1);
}
```

```
	// this part puts everything in "standard" orientation... the lowest sum of side numbers
	if (NCount < PAPtr->Faces()) {
		for (int i = 0; i < PAPtr->Faces(); i++) {
			RotList(NList,1,PAPtr->Faces());	// perform one rotation
			RotOff++;					// increment the rotation count
			if (NList < NLMin) {			// if the side values are the lowest yet
				NLMin = NList;				// update the minimum values
				ROMin = RotOff;
			}
		}
		NList = NLMin;			// set the NList and RotOff variables to the minimized values
		RotOff = ROMin;
	}

	if (unstick)
		Chance = &(ParticleArray::EnergyUnstickValue);//	EnergyValue = unsticking function
	else
		Chance = &(ParticleArray::EnergyValue);// EnergyValue = sticking function

	// begin determining the surface structure
	// square particles
	if (PAPtr->Faces() == 4) {

		// positions with only one neighbor particle
		if (NCount == 1) {	// the only neighbor is in "1", see if it's a single or part of a
surface
			byte NL2 = SecNeighbors(x,y,A,1,RotOff);		// get the 2nd nearest neighbors
			//cout << "AggA(" << A << ") SecN: " << (int)NL2 << endl;
			//cout << "AggA2(" << A2 << ") SecN: " << (int) SecNeighbors(x,y,A2,1,RotOff) << endl;
			if (((NL2 & 2) == 2) || ((NL2 & 8) == 8)) { // it's a surface
				if ((RotOff % 2) == 0) return (PAPtr->*Chance)("(0 1)");	// if even RotOff, it's
a (0 1) type
				else return (PAPtr->*Chance)("(1 0)");		// if odd, it's a (1 0) type
			}
```

```
                else return (PAPtr->*Chance)("single");    // not a part of a surface
            }

            // positions with two neighboring particles
            if (NCount == 2) {
                if ((NList & 2) == 2)                      // one neighbor is above, if the other is right
                    return (PAPtr->*Chance)("(1 1)");                    // it's a step surface, (1 1)
type
                else {            // the two particles are opposite, check to see if they are singles or
surfaces
                    byte NL2 = SecNeighbors(x,y,A,1,RotOff);        // secondary neighbors up
                    byte NL3 = SecNeighbors(x,y,A,4,RotOff);        // secondary neighbors down

                    float C1, C2;     // chances based on multiple particles

                    if (((NL2 & 2) == 2) || ((NL2 & 8) == 8)) { // perform the single particle test on
the particle above
                        if ((RotOff % 2) == 0) C1 = (PAPtr->*Chance)("(0 1)");
                        else C1 = (PAPtr->*Chance)("(1 0)");
                    }
                    else C1 = (PAPtr->*Chance)("single");

                    if (((NL3 & 2) == 2) || ((NL3 & 8) == 8)) { // and the one below
                        if ((RotOff % 2) == 0) C2 = (PAPtr->*Chance)("(0 1)");      // if even
RotOff, it's a (0 1) type
                        else C2 = (PAPtr->*Chance)("(1 0)");            // if odd, it's a (1 0) type
                    }
                    else C2 = (PAPtr->*Chance)("single");     // not a part of a surface
                    return (C1 > C2 ? C1 : C2);   // return the higher chance of the two cases
                }
            }

            // positions with three or more particles
            if (NCount == 3)
```

```
            return (PAPtr->*Chance)("void");          // with 3 neighbors, this must be a gap space
      if (NCount == 4)
            return (PAPtr->*Chance)("void");    // a true void.
}

// hexagonal particles
else if(PAPtr->Faces() == 6) {

      // positions with only one neighbor
      if (NCount == 1) {            // neighbor is in position "1" (because of RotOff)
            byte NL2 = SecNeighbors(x,y,A,1,RotOff);        // get the 2nd nearest neighbors
            if ((NL2 & 34) == 34)   // if 2nd neighbors are in positions "2" and "32"
                  return (PAPtr->*Chance)("(0 1)");   // this is the convex part of a (0 1) surface
            else
                  return (PAPtr->*Chance)("single");  // otherwise, a single surface
      }

      // positions with two neighbors
      if (NCount == 2) {
            if( ((NList & 2) == 2) || ((NList & 32) == 32))       // (1 -2) type surface conditions
                  return (PAPtr->*Chance)("(1 -2)");
            else {
                  byte OtherP = NList & ~1;
                  byte NL2 = SecNeighbors(x,y,A,1,RotOff);        // 2nd neighbors of particle at "1"
                  byte NL3 = SecNeighbors(x,y,A,OtherP,RotOff);   // 2nd neighbors of other particle

                  float C1, C2;     // two chance variables

                  // check the state for the "1" particle
                  if ((NL2 & 34) == 34)
                        C1 = (PAPtr->*Chance)("(0 1)");
                  else
                        C1 = (PAPtr->*Chance)("single");
                  // and the other particle
```

```
                if ((NL3 & ((OtherP << 2) | (OtherP >> 2))) != 0)
                        C2 = (PAPtr->*Chance)("(0 1)");
                else
                        C2 = (PAPtr->*Chance)("single");
                return (C1 > C2 ? C1 : C2);   // return the larger chance
        }
}

// positions with three neighbors
if (NCount == 3) {
        float C1, C2, C3;
        if (NList == 7)          // concave part of (0 1) surface
                return (PAPtr->*Chance)("(0 1)");
        else if (NList == 11) { // (1 -2) in "1" and "2", plus single in "8"
                C1 = (PAPtr->*Chance)("(1 -2)");
                byte NL2 = SecNeighbors(x,y,A,8,RotOff);
                if ((NL2 & 20) == 20)
                        C2 = (PAPtr->*Chance)("(0 1)");
                else
                        C2 = (PAPtr->*Chance)("single");
                return (C1 > C2 ? C1 : C2);
        }
        else if (NList == 13) { // (1 -2) in "4" and "8", plus single in "1"
                C1 = (PAPtr->*Chance)("(1 -2)");
                byte NL2 = SecNeighbors(x,y,A,1,RotOff);
                if ((NL2 & 34) == 34)
                        C2 = (PAPtr->*Chance)("(0 1)");
                else
                        C2 = (PAPtr->*Chance)("single");
                return (C1 > C2 ? C1 : C2);
        }
        else if (NList == 21) { // three singles at "1", "4", and "16"
                byte NL2 = SecNeighbors(x,y,A,1, RotOff);
                byte NL3 = SecNeighbors(x,y,A,4, RotOff);
```

```
                        byte NL4 = SecNeighbors(x,y,A,16, RotOff);
                        if ((NL2 & 34) == 34) C1 = (PAPtr->*Chance)("(0 1)");
                        else C1 = (PAPtr->*Chance)("single");

                        if ((NL3 & 10) == 10) C2 = (PAPtr->*Chance)("(0 1)");
                        else C2 = (PAPtr->*Chance)("single");

                        if ((NL4 & 40) == 40) C3 = (PAPtr->*Chance)("(0 1)");
                        else C3 = (PAPtr->*Chance)("single");
                        return (C1 > C2 ? (C1 > C3 ? C1 : C3) : (C2 > C3 ? C2 : C3));     // return the
greatest chance
                  }
            }

            // positions with four neighbors
            if (NCount == 4) {
                  float C1, C2;
                  if (NList == 15)  // gap made by "1", "2", "4", and "8"
                        return (PAPtr->*Chance)("void");
                  else if (NList == 23) {       // (0 1) in "1", "2", and "4" plus single at "16"
                        C1 = (PAPtr->*Chance)("(0 1)");
                        byte NL2 = SecNeighbors(x,y,A,16,RotOff);
                        if ((NL2 & 40) == 40) C2 = (PAPtr->*Chance)("(0 1)");
                        else C2 = (PAPtr->*Chance)("single");
                        return (C1 > C2 ? C1 : C2);
                  }
                  else if (NList == 27)   // two (1 -2) at "1" & "2" and "8" & "16"
                        return (PAPtr->*Chance)("(1 -2)");
            }

            // positions with five or six neighbors
            if (NCount == 5)
                  return (PAPtr->*Chance)("void");    // with five partners, this must be a gap
            if (NCount == 6)
```

```
                  return (PAPtr->*Chance)("void");    // a true void.  Set energy at tenfold to discourage
this
      }

      return 0;   // If it gets here, something odd happened.  Return 0 and hope it doesn't cause any
problems
}


// get the unstick chance of a particle, based on surface it's attached to
//float GridArray::DynamicUnstickChance(int x, int y) {

//}



/*
=============================================================================================
                                                    Particle Functions
=============================================================================================
*/

// constructor; not much to do here, just initializes the private variables
Particle::Particle() {
      x = -1;           // set (x,y) coords to nonsense until the particle is placed somewhere
      y = -1;

      Orient = 1; // default orientation
      Flags = 0;  // starts with no true flags
      Attached = 0;     // no attachments initially

      AggRef = -1;      // not a member of any Aggregate
}
```

```
/*
================================================================================================
                                                ParticleArray Functions
================================================================================================
*/

// simple constructor for a particle array.  Takes # of particles, uses defaults for all else
ParticleArray::ParticleArray(int Num) {
      if (Num <= 0) return;                                          // no zero particle arrays

      ParticleCount = Num;                                           // record the number of particles

      TheArray = new Particle[Num];                                 // allocate memory for Num particles

      GridPtr = NULL;                                                      // does not point to any grid
      AggPtr = NULL;                                                       // or any AggegateArray
      RandomOrientation = false;                                     // all particle start with orientation 1

      NumFaces = 4;                                                  // square particles are default

      StickType = 0;                                                 // orientation sticking by default
      StickValues = NULL;                                            // StickValues is initially a null
pointer
}


// full constructor for a particle array.  Initializes all values to given parameters
ParticleArray::ParticleArray(int Num, unsigned char NumSides, bool RandSO, GridArray* Grid) {
      if (Num <= 0) return;                                          // no zero particle arrays

      ParticleCount = Num;    //list the number of particles held

      TheArray = new Particle[Num]; // allocate the appropriately sized array
```

```
	GridPtr = Grid;				// point to the appropriate grid
	Grid->SetPAPtr(this);	// and have the grid point back here
	AggPtr = NULL;				// no aggregate array defined just yet

	RandomOrientation = RandSO;


	NumFaces = NumSides;
	StickType = 0;					// orientation sticking by default
	StickValues = NULL;				// StickValues is initially a null pointer
}


// destructor, frees the memory used by the particles in the array
ParticleArray::~ParticleArray() {
	delete [] TheArray;				// free memory from the particles
	TheArray = NULL;			// reset the pointer to nothing
	delete [] StickValues;		// free memory for sticking information
	StickValues = NULL;				// and the pointer reset
}


// function to resize the number of allocated particles
int ParticleArray::ResizeArray(int NewSize) {

	// copy the current particle information into a temporary array
	Particle* TempArray = new Particle[ParticleCount];
	for (int i = 0; i < ParticleCount; i++)
		TempArray[i] = TheArray[i];

	delete [] TheArray;		// free the current array

	TheArray = new Particle[NewSize];	// reallocate the array at the new size
```

```
	for (int i = 0; i < ParticleCount; i++)	// copy the particle data back to the new array
		TheArray[i] = TempArray[i];

	ParticleCount = NewSize;	// update the size

	return 0;	// zero for no errors
}


// get the (x,y) coords for particle P
void ParticleArray::Output(int P, int& outX, int& outY) {
	outX = TheArray[P].x;
	outY = TheArray[P].y;
}


// get the orientation of particle P.  Bitwise = false gives more human readible side info
int ParticleArray::Orient(int P, bool Bitwise) {

	if (Bitwise) return TheArray[P].Orient;	// just return the orient value for P

	else {	// convert to side 1, side 2, etc format for easier reading
		int Output (1);	// default orientation is 1
		int Temp (TheArray[P].Orient);		// binary orientation value for the particle
		while (Temp != 1) {		// "rotate" the particle until orientation is 1.

			Temp = Temp >> 1; // shift-right (divide Temp by 2)
			Output++;		// increment the side number
		}
		return Output;
	}
}
```

```
// sets the orientation of particle P
void ParticleArray::SetOrient(int P, byte NewO) {
      TheArray[P].Orient = NewO;
}


// returns a pointer to the GridArray the particles exist on
GridArray* ParticleArray::GetGridPtr(void) {
      return GridPtr;
}


// assign a grid to the particles
void ParticleArray::SetGridPtr(GridArray* GA) {
      GridPtr = GA;           // attach the particle to GA
      GA->SetPAPtr(this);     // and let GA know it has particles now
}


// assign an aggregate array to the particles
void ParticleArray::SetAggPtr(AggregateArray* AA) {
      AggPtr = AA;
}


// return the shape of the particles in the array
byte ParticleArray::Faces(void) {
      return NumFaces;
}


// returns the count of particles in the array
int ParticleArray::NumParticles(void) {
      return ParticleCount;
```

```
}


// returns true if rotation is allowed, false otherwise
bool ParticleArray::AllowRot(void) {
      return RandomOrientation;
}


// Shifts particle P by (XShift, YShift).
bool ParticleArray::ShiftBy(int P, int XShift, int YShift) {
      if (!GetDeployed(P)) return false;  // cannot shift a particle that isn't deployed

      if (!GridPtr->CheckScope(TheArray[P].x + XShift, TheArray[P].y + YShift))
            return false;     // if new position is outside the grid scope, mark false

      // change the particle position
      GridPtr->ClearP(TheArray[P].x, TheArray[P].y);
      TheArray[P].x += XShift;
      TheArray[P].y += YShift;
      GridPtr->MarkP(P, TheArray[P].x, TheArray[P].y);      // update the grid
      return true;      // particle successfully shifted
}


// Functions to set and read the various status values in the Flags variable
void ParticleArray::SetDeployed(int P, bool Deployed) {
      if (Deployed == true)
            TheArray[P].Flags |= 1;       // "Flags = Flags OR 1"  (turns on the 1 bit)
      else
            TheArray[P].Flags &= ~1;      // "Flags = Flags AND (NOT 1)"  (turns off the 1 bit)
}

bool ParticleArray::GetDeployed(int P) {
```

```
      if (TheArray[P].Flags & 1) return true;   // if "Flags AND 1" == 1  (true if 1 bit is on)
      else return false;
}

void ParticleArray::SetFixed(int P, bool Fixed) {
      if (Fixed == true)
            TheArray[P].Flags |= 2;
      else
            TheArray[P].Flags &= (~2);
}

bool ParticleArray::GetFixed(int P) {
      if (TheArray[P].Flags & 2) return true;
      else return false;
}

void ParticleArray::SetHadTurn(int P, bool Turn) {
      if (Turn == true)
            TheArray[P].Flags |= 4;
      else
            TheArray[P].Flags &= (~4);
}

bool ParticleArray::GetHadTurn(int P) {
      if (TheArray[P].Flags & 4) return true;
      else return false;
}

void ParticleArray::SetMeso(int P, bool Meso) {
      if (Meso == true)
            TheArray[P].Flags |= 8;
      else
            TheArray[P].Flags &= (~8);
}
```

```
bool ParticleArray::GetMeso(int P) {
      if (TheArray[P].Flags & 8) return true;
      else return false;
}


// moves the selected particle randomly on the grid
MoveError_t ParticleArray::RandMove(int P) {

      byte PosShift = 0;                                              // direction to shift the particle
      int OldX (TheArray[P].x), OldY (TheArray[P].y); // x,y coordinates before the move
      int NewX (OldX), NewY (OldY);                                   // x,y coordinate after the move

      if ((P < 0) || (P >= ParticleCount)) return Out_of_Range;   // invalid particle P
      if (GetFixed(P)) return Fixed_Particle;                     // cancel the move if the particle is fixed in
position
      if (!GetDeployed(P)) return Not_Deployed;       // can't move undeployed particles

      if (NumFaces == 4) {                                            // movement for square particles

            PosShift = (byte)RandGen.randInt(3);                  // random integer [0,3]

            if (PosShift == 0) NewY = OldY + 1;                         // move particle up
            else if (PosShift == 1) NewX = OldX + 1;              // move right
            else if (PosShift == 2) NewY = OldY - 1;              // move down
            else if (PosShift == 3) NewX = OldX - 1;              // move left
      }

      else if (NumFaces == 6) {                                       // movements for hexagonal particles

            PosShift = (byte)RandGen.randInt(5);                  // random integer [0,5]
            if (PosShift == 0) NewY = OldY + 1;                         // move up
            else if (PosShift == 1) NewX = OldX + 1;              // move up-right
```

```
            else if (PosShift == 2) {
                  NewX = OldX + 1;                                    // move down-right
                  NewY = OldY - 1;
            }
            else if (PosShift == 3) NewY = OldY - 1;          // move down
            else if (PosShift == 4) NewX = OldX - 1;          // move down-left
            else if (PosShift == 5) {
                  NewX = OldX - 1;                                    // move up-left
                  NewY = OldY + 1;
            }
      }

      // Check if particle is outside grid.  If so, drop at another entry point.
      if (!GridPtr->CheckScope(NewX, NewY)) {
            DropRandom(P);
            return Redrop;
      }

      // Check for collisions with other particles
      if (GridPtr->Occupied(NewX,NewY)) return Collision;

      // update particle positions
      TheArray[P].x = NewX;
      TheArray[P].y = NewY;

      // update the grid
      GridPtr->ClearP(OldX,OldY);
      GridPtr->MarkP(P,NewX,NewY);

      return No_Error;
}


// Randomly drop the particle at one of the grid boundaries
```

```
void ParticleArray::DropRandom(int P, bool Edge) {

	int NewX;	// the drop coords
	int NewY;

	if (GetFixed(P)) return;	// can't drop a fixed particle

	if (GetDeployed(P)) {
		SetDeployed(P, false);	// undeploy the particle
		GridPtr->ClearP(TheArray[P].x, TheArray[P].y);	// unmarked the current space
	}

	while (!GetDeployed(P)) {		// while loop, to ensure the drop is in an open space
		if (Edge) GridPtr->GetRandomEdge(NewX, NewY);	// generate a random space on the edge of the
grid scope
		else GridPtr->GetRandomSpace(NewX, NewY); // otherwise, get a random space anywhere in the
scope
		if (!GridPtr->Occupied(NewX,NewY))
			SetDeployed(P, true);	// if the selected space is empty, mark particle as deployed
	}

	TheArray[P].x = NewX;	// update the particle coordinates
	TheArray[P].y = NewY;
	if (RandomOrientation) Rotate(P);	// randomly rotate the particle if random start orientation is
enabled.
	GridPtr->MarkP(P, NewX, NewY);	// update the grid
}


// Seed the selected particle at (SeedX, SeedY).  If a coordinate is unspecified, it's randomized.
bool ParticleArray::Seed(int P, int SeedX, int SeedY) {

	if (GridPtr->Occupied(SeedX,SeedY)) return false;	// if the spot is occupied, it doesn't work
```

```
	if (!GetDeployed(P)) SetDeployed(P, true);	// if the particle wasn't deployed, it is now
	else GridPtr->ClearP(TheArray[P].x, TheArray[P].y);	// if it was, clear its old spot

	TheArray[P].x = SeedX;  // change the coordinates
	TheArray[P].y = SeedY;
	GridPtr->MarkP(P, SeedX, SeedY);	// update the grid
	SetFixed(P, true);	// seed particles are fixed in place
	return true;
}


// move a particle to coordinates (x,y) on the grid
bool ParticleArray::MoveTo(int P, int NewX, int NewY) {

	if (GridPtr->Occupied(NewX, NewY)) return false;	// check for occupied destination
	if (!GridPtr->CheckScope(NewX, NewY)) return false;	// check for position out of scope

	if (!GetDeployed(P)) SetDeployed(P, true);	// deploy the particle, if not already
	else GridPtr->ClearP(TheArray[P].x, TheArray[P].y);	// remove the old position

	TheArray[P].x = NewX;
	TheArray[P].y = NewY;
	GridPtr->MarkP(P, NewX, NewY);
	return true;
}


// Count the neighboring particles
byte ParticleArray::CountNeighbors(int P) {
	byte Neighbors = 0;
	int px = TheArray[P].x;
	int py = TheArray[P].y;

	// check each neighboring grid space and, if occupied, increment Neighbors
```

```
      if (GridPtr->Occupied(px + 1, py)) Neighbors++;
      if (GridPtr->Occupied(px, py + 1)) Neighbors++;
      if (GridPtr->Occupied(px - 1, py)) Neighbors++;
      if (GridPtr->Occupied(px, py - 1)) Neighbors++;

      if (NumFaces == 6) {    // check for the two extra hexagonal neighbors
            if (GridPtr->Occupied(px - 1, py + 1)) Neighbors++;
            if (GridPtr->Occupied(px + 1, py - 1)) Neighbors++;
      }

      return Neighbors; // return the count
}


// return a bitwise list of neighboring spaces with particles
byte ParticleArray::GetNeighbors(int P, bool SameAgg) {
      byte Neighbors = 0;
      int px = TheArray[P].x;
      int py = TheArray[P].y;
      int A = TheArray[P].AggRef;

      if (A == -1)      // A = -1 means "not in an aggregate."  By changing it to -2, we prevent two
            A = -2;           // unaggregated particles from being considered in the same aggregate (both -
1)

      if (NumFaces == 4) {    // side check for square particles
            if (GridPtr->Occupied(px, py + 1))        // there is a particle above P
                  if ((SameAgg == true) || (GridPtr->ARef(px,py+1) != A))           // if SameAgg counting
is on, OR they are in different aggregates
                        Neighbors |= 1;   // turn the 1 bit on, to indicate a neighbor on top
            if (GridPtr->Occupied(px + 1, py))
                  if ((SameAgg == true) || (GridPtr->ARef(px+1,py) != A))
                        Neighbors |= 2;   // 2 bit on, for neighbor on right
            if (GridPtr->Occupied(px, py - 1))
```

```
                  if ((SameAgg == true) || (GridPtr->ARef(px,py-1) != A))
                        Neighbors |= 4;   // 4 bit on, for neighbor on bottom
            if (GridPtr->Occupied(px - 1, py))
                  if ((SameAgg == true) || (GridPtr->ARef(px-1,py) != A))
                        Neighbors |= 8;   // 8 bit on, for neighbor on left
      }

      else if (NumFaces == 6) {
            if (GridPtr->Occupied(px, py + 1))
                  if ((SameAgg == true) || (GridPtr->ARef(px,py+1) != A))
                        Neighbors |= 1;   // 1 bit on, for neighbor on top
            if (GridPtr->Occupied(px + 1, py))
                  if ((SameAgg == true) || (GridPtr->ARef(px+1,py) != A))
                        Neighbors |= 2;   // 2 bit on, for neighbor on top-right
            if (GridPtr->Occupied(px + 1, py - 1))
                  if ((SameAgg == true) || (GridPtr->ARef(px+1,py-1) != A))
                        Neighbors |= 4;   // 4 bit on, for neighbor on bottom-right
            if (GridPtr->Occupied(px, py - 1))
                  if ((SameAgg == true) || (GridPtr->ARef(px,py-1) != A))
                        Neighbors |= 8;   // 8 bit on, for neighbor on bottom
            if (GridPtr->Occupied(px - 1, py))
                  if ((SameAgg == true) || (GridPtr->ARef(px-1,py) != A))
                        Neighbors |= 16;  // 16 bit on, for neighbor on bottom-left
            if (GridPtr->Occupied(px - 1, py + 1))
                  if ((SameAgg == true) || (GridPtr->ARef(px-1,py+1) != A))
                        Neighbors |= 32;  // 32 bit on, for neighbor on top-left
      }

      return Neighbors; // return the list
}


// function used by OrientStick to check for sticking between the two particles
bool ParticleArray::StickCheckOrient(int P, int P2) {
```

```
	if ((P == -1) || (P2 == -1)) return false;	// one of the particles is not a particle

	if (NumFaces == 4) {	// square particles
		int O1 = Orient(P,false);	// grab the orientations of each particle
		int O2 = Orient(P2,false);

		if ((O1 % 2) == (O2 % 2)) {	// if they face the same way, use A-A or B-B stick chance
			if ((O1 % 2) == 1) {
				if ((TheArray[P2].y == TheArray[P].y + 1) || (TheArray[P2].y == TheArray[P].y - 1))
				return PercentCheck(StickValues[0]);
				else return PercentCheck(StickValues[1]);
			}
			else {
				if ((TheArray[P2].x == TheArray[P].x + 1) || (TheArray[P2].x == TheArray[P].x - 1))
					return PercentCheck(StickValues[0]);
				else return PercentCheck(StickValues[1]);
			}
		}
		else	// if orientation is mismatched, use A-B
			return PercentCheck(StickValues[2]);
	}

	else if (NumFaces == 6)		// hexagonal particles
		return PercentCheck(StickValues[0]);	// hex has only one stick chance

	else	// if NumFaces is an undefined value for some reason
		return false;
}


// return the index of the first undeployed particle in the array
// if no undeployed particle is found, the array is expanded by 10 and a new particle is returned
int ParticleArray::NextFree(void) {
	for (int i = 0; i < ParticleCount; i++)	// look through the particles
```

```
            if (GetDeployed(i) == false) return i;    // return the value of the first undeployed one found

      // if no free particle is found, resize the particle array to make some
      int NextParticle = ParticleCount;   // record the location of the first new particle
      this->ResizeArray(ParticleCount + 10);    // expand the array by 10 particles
      return NextParticle;    // return the location of the (newly made) free particle
}


// rotate the particle by (Times) faces.  If (Times) unspecified, random rotation.
void ParticleArray::Rotate(int P, char Times) {
      if (Times < 0) Times = (char)RandGen.randInt(NumFaces - 1); // get a random number of rotations

      Times = Times % NumFaces;     // reduce to the simplest equivalent rotation (i.e. rotating a square 7
faces -> 3 faces)

      for (int i = 0; i < Times; i++) {
            TheArray[P].Orient = TheArray[P].Orient << 1;   // shift left by 1, eqivalent to cclockwise
rotate by 1 face
            if (TheArray[P].Orient == 1 << NumFaces) TheArray[P].Orient = 1;  // reset to 1 if it rotates
all the way around
      }
}


// Rotate the attachement directions of P (used with aggregate rotation, for example)
void ParticleArray::RotAttach(int P, char Times) {
      if ((P < 0) || (P > ParticleCount)) return;           // check index, do nothing if invalid
      if (Times < 0) return;                                      // check Times value as well

      RotList(TheArray[P].Attached, Times, NumFaces); // use the list rotate function to change the
attachment list
}
```

```
// change the AggRef variable of Particle P to Aggregate A
void ParticleArray::SetAggRef(int P, int A) {
      TheArray[P].AggRef = A;
}


// return the AggRef variable for Particle P
int ParticleArray::GetAggRef(int P) {
      return TheArray[P].AggRef;
}


// used to change particle coordinates with no interaction with the grid
void ParticleArray::GhostTo(int P, int x, int y) {
      TheArray[P].x = x;
      TheArray[P].y = y;
}


// remove a particle from the grid completely
void ParticleArray::Kill(int P) {
      if ((P >= ParticleCount) || (P < 0)) return;    // if the index is out of bounds, do nothing

      Unattach(P,~0);               // clear attachments (also breaks attachments *to* this particle)

      if (GridPtr->Occupied(TheArray[P].x, TheArray[P].y))  // if currently on the grid, remove it
            GridPtr->ClearP(TheArray[P].x, TheArray[P].y);

      TheArray[P].x = -1;           // reset to off grid coordinates
      TheArray[P].y = -1;
```

```
      TheArray[P].Orient = 1; // default orientation
      TheArray[P].Flags = 0;  // clear all flags

      if (TheArray[P].AggRef != -1) {     // if a member of an aggregate
            AggPtr->RemParticleP(TheArray[P].AggRef,P);     // remove it from the aggregate
            TheArray[P].AggRef = -1;      // clear the aggregate reference
      }
}


// This function clears the present location of particle P from the grid
void ParticleArray::ClearMark(int P) {
      if ((P < 0) || (P >= ParticleCount)) return;          // if the index is out of range, do nothing
      if (GetDeployed(P) == false) return;                  // if the particle isn't deployed yet, do
nothing

      if (GridPtr->PRef(TheArray[P].x, TheArray[P].y) == P) // make sure that the grid points to this
particle
            GridPtr->ClearP(TheArray[P].x, TheArray[P].y);                                  // clear that
grid coordinate
}


// This function marks the particle's present location on the grid
void ParticleArray::MarkGrid(int P) {
      if ((P < 0) || (P >= ParticleCount)) return;          // no invalid indices
      if (GetDeployed(P) == false) return;                  // can't mark a particle that isn't on the grid

      GridPtr->MarkP(P, TheArray[P].x, TheArray[P].y);      // mark the grid location
}


// Reads the value of the "attached" flags for particle P
byte ParticleArray::GetAttached(int P) {
```

```
      if ((P < 0) || (P >= ParticleCount)) return 0;  // nothing if index is out of range
      return TheArray[P].Attached;
}


// Attaches particle to neighbors in direction(s) Dir (specified bitwise)
void ParticleArray::Attach(int P, byte Dir) {
      if ((P < 0) || (P >= ParticleCount)) return;    // return if index is out of range
      if (Dir == 0) return;                                    // if there are no attachments to add

      int px, py;
      byte N;
      Output(P, px, py);                              // get coords of P
      N = GetNeighbors(P);                       // get the neighboring particles of P

      // this part updates the attachment flags of the neighboring particles
      if (NumFaces == 4) {
            if ((Dir & 1 & N) == 1) {     // check that Dir contains "up" and there is a neighbor there
                  TheArray[P].Attached |= 1;    // turn on P's "up" flag
                  TheArray[GridPtr->PRef(px,py+1)].Attached |= 4; // turn on the "up" particle's "down"
attachment flag
            }
            if ((Dir & 2 & N)== 2) {
                  TheArray[P].Attached |= 2;    // Attached = Attached OR 2 (sets the 2 bit to 1)
                  TheArray[GridPtr->PRef(px+1,py)].Attached |= 8;
            }
            if ((Dir & 4 & N) == 4) {
                  TheArray[P].Attached |= 4;
                  TheArray[GridPtr->PRef(px,py-1)].Attached |= 1;
            }
            if ((Dir & 8 & N) == 8) {
                  TheArray[P].Attached |= 8;
                  TheArray[GridPtr->PRef(px-1,py)].Attached |= 2;
            }
```

```
	}

	else if (NumFaces == 6) {
		if ((Dir & 1 & N) == 1) {
			TheArray[P].Attached |= 1;
			TheArray[GridPtr->PRef(px,py-1)].Attached |= 8;
		}
		if ((Dir & 2 & N) == 2) {
			TheArray[P].Attached |= 2;
			TheArray[GridPtr->PRef(px+1,py)].Attached |= 16;
		}
		if ((Dir & 4 & N) == 4) {
			TheArray[P].Attached |= 4;
			TheArray[GridPtr->PRef(px+1,py-1)].Attached |= 32;
		}
		if ((Dir & 8 & N) == 8) {
			TheArray[P].Attached |= 8;
			TheArray[GridPtr->PRef(px,py-1)].Attached |= 1;
		}
		if ((Dir & 16 & N) == 16) {
			TheArray[P].Attached |= 16;
			TheArray[GridPtr->PRef(px-1,py)].Attached |= 2;
		}
		if ((Dir & 32 & N) == 32) {
			TheArray[P].Attached |= 32;
			TheArray[GridPtr->PRef(px-1,py+1)].Attached |= 4;
		}
	}
}


// Removes the attachment between P and particles in direction(s) Dir
void ParticleArray::Unattach(int P, byte Dir) {
	if ((P < 0) || (P >= ParticleCount)) return;	// return if index is out of range
```

```
      if (Dir == 0) return;                                    // if there are no attachments to add

      int px, py;
      byte N;
      Output(P, px, py);                              // get coords of P
      N = GetNeighbors(P);                        // get the neighboring particles of P

      // this part updates the attachment flags of the neighboring particles
      if (NumFaces == 4) {
            if ((Dir & 1 & N) == 1) {           // check that Dir contains "up" and there is a neighbor
there
                  TheArray[P].Attached &= ~1;   // turn of P's "up" (1) flag
                  TheArray[GridPtr->PRef(px,py+1)].Attached &= ~4;      // turn off the "up" particle's
"down" attachment flag
            }
            if ((Dir & 2 & N) == 2) {
                  TheArray[P].Attached &= ~2;   // Attached = Attached AND NOT 2 (sets the 2 bit to 0)
                  TheArray[GridPtr->PRef(px+1,py)].Attached &= ~8;
            }
            if ((Dir & 4 & N) == 4) {
                  TheArray[P].Attached &= ~4;
                  TheArray[GridPtr->PRef(px,py-1)].Attached &= ~1;
            }
            if ((Dir & 8 & N) == 8) {
                  TheArray[P].Attached &= ~8;
                  TheArray[GridPtr->PRef(px-1,py)].Attached &= ~2;
            }
      }

      else if (NumFaces == 6) {
            if ((Dir & 1 & N) == 1) {
                  TheArray[P].Attached &= ~1;
                  TheArray[GridPtr->PRef(px,py-1)].Attached &= ~8;
            }
```

```
            if ((Dir & 2 & N) == 2) {
                  TheArray[P].Attached &= ~2;
                  TheArray[GridPtr->PRef(px+1,py)].Attached &= ~16;
            }
            if ((Dir & 4 & N) == 4) {
                  TheArray[P].Attached &= ~4;
                  TheArray[GridPtr->PRef(px+1,py-1)].Attached &= ~32;
            }
            if ((Dir & 8 & N) == 8) {
                  TheArray[P].Attached &= ~8;
                  TheArray[GridPtr->PRef(px,py-1)].Attached &= ~1;
            }
            if ((Dir & 16 & N) == 16) {
                  TheArray[P].Attached &= ~16;
                  TheArray[GridPtr->PRef(px-1,py)].Attached &= ~2;
            }
            if ((Dir & 32 & N) == 32) {
                  TheArray[P].Attached &= ~32;
                  TheArray[GridPtr->PRef(px-1,py+1)].Attached &= ~4;
            }
      }
}


// Performs a sticking check on particle P.  Returns bitwise list of successful stick directions.
/* not fully implemented */
byte ParticleArray::CheckStick(int P) {
      byte Neighbors = GetNeighbors(P, false);  // find the neighboring particles (exclude members of the
same aggregate)
      if (Neighbors == 0) return 0;       // no neighbors -> no stick checks
      int px = TheArray[P].x;                  // get particle coordinates
      int py = TheArray[P].y;
      byte Success (0);                        // stores a bitwise list of successful attaches
```

```
if (StickType == 0) {    // for orientation type sticking
      if (NumFaces == 4) {    // for square particles
            if ((Neighbors & 1) == 1)     // there's a neighbor to the top
                  if (StickCheckOrient(P, GridPtr->PRef(px, py + 1)))   // it passes the check
                        Success = Success | 1;        // turn on the 1 bit of the success list
            if ((Neighbors & 2) == 2)
                  if (StickCheckOrient(P, GridPtr->PRef(px + 1, py)))
                        Success = Success | 2;
            if ((Neighbors & 4) == 4)
                  if (StickCheckOrient(P, GridPtr->PRef(px, py - 1)))
                        Success = Success | 4;
            if ((Neighbors & 8) == 8)
                  if (StickCheckOrient(P, GridPtr->PRef(px - 1, py)))
                        Success = Success | 8;
      }

      else if (NumFaces == 6) {                // for hex particles
            if ((Neighbors & 1) == 1)
                  if (StickCheckOrient(P, GridPtr->PRef(px, py + 1)))
                        Success = Success | 1;
            if ((Neighbors & 2) == 2)
                  if (StickCheckOrient(P, GridPtr->PRef(px + 1, py)))
                        Success = Success | 2;
            if ((Neighbors & 4) == 4)
                  if (StickCheckOrient(P, GridPtr->PRef(px + 1, py - 1)))
                        Success = Success | 4;
            if ((Neighbors & 8) == 8)
                  if (StickCheckOrient(P, GridPtr->PRef(px, py - 1)))
                        Success = Success | 8;
            if ((Neighbors & 16) == 16)
                  if (StickCheckOrient(P, GridPtr->PRef(px - 1, py)))
                        Success = Success | 16;
            if ((Neighbors & 32) == 32)
```

```
                        if (StickCheckOrient(P, GridPtr->PRef(px - 1, py + 1)))
                              Success = Success | 32;
            }
            return Success;                                    // return the list of successful stick
directions
      }

      // crystallographic sticking is handled by aggregate, just return 0
      else if (StickType == 1) {
            return 0;
      }

      return 0;   // something went wrong.  Return 0 to avoid breaking anything else.
}


// Moves P according to a Gaussian distribution of movements
MoveError_t ParticleArray::GaussMove(int P) {

      int XMove (0), YMove (0);                                  // hold movement data in each direction

      if (GetFixed(P)) return Fixed_Particle    ;           // cancel the move if the particle is fixed in
position
      if (!GetDeployed(P)) return Not_Deployed;       // can't move undeployed particles

      XMove = round(RandGauss(1));                               // get x component from Gaussian distribution
      YMove = round(RandGauss(1));                               // and the y component

      if (AllowRot()) GaussRot(P);                               // gaussian movement also involves gaussian
rotation

      return VectorMove(P, XMove, YMove);                        // move the particle and return any error
generated
}
```

```
// Move the particle along a specified vector, checking for errors at each step
MoveError_t ParticleArray::VectorMove(int P, int XVec, int YVec) {
      int OldX (TheArray[P].x), OldY (TheArray[P].y); // x,y coordinates before the move
      int NewX (OldX), NewY (OldY);                          // x,y coordinate after the move
      bool Collide = false;                                        // becomes true if a collision occurs

      ClearMark(P);                                                      // begin by unmarking the grid
position

      // first check if the move is just +/- 1.  If so, no need to do the complicated stuff
      if ((abs(XVec) <= 1) && (abs(YVec) <= 1)) {
            if (GridPtr->Occupied(OldX + XVec, OldY + YVec))
                  Collide = true;
            else {
                  OldX += XVec;
                  OldY += YVec;
            }
      }

      else {      // the move wasn't simple, so here's the complicated stuff
            float Slope = (float)YVec / (float)XVec;  // find the |slope| of the particle's path
            bool SwitchXY = false;                                 // if x and y values were switched

            if (fabs(Slope) < 1) {                                 // if the particle moves more x than y
                  SwitchXY = true;                                 // activate the SwitchXY flag
                  int temp = XVec;                                 // and switch them
                  XVec = YVec;
                  YVec = temp;
                  Slope = 1/Slope;                                 // invert the slope
            }

            do {
```

```
                  if (abs(XVec * Slope) >= abs(YVec)) {      // if the particle is along the right path
                        if (XVec > 0) {                            // move it in the x direction,
                              XVec--;                                    // set XMove one closer to
zero,
                              if (SwitchXY) NewY++;                // and change the coordinates
                              else NewX++;                               // (change y if SwitchXY was
activated)
                        }
                        else if (XVec < 0) {                       // same thing, but this is for negative
XMove
                              XVec++;
                              if (SwitchXY) NewY--;
                              else NewX--;
                        }
                  }

                  if (YVec > 0) {                                  // y always moves.
                        YVec--;                                          // the rest of this is
analogous to the x case
                        if (SwitchXY) NewX++;
                        else NewY++;
                  }
                  else if (YVec < 0) {
                        YVec++;
                        if (SwitchXY) NewX--;
                        else NewY--;
                  }

                  if (GridPtr->Occupied(NewX, NewY)) {       // check that this location isn't occupied
                        Collide = true;                            // if it is, mark a collision
                        break;                                           // and abort the rest of the
loop
                  }
```

```
                  OldX = NewX;                                          // if all is clear, set this as the
new
                  OldY = NewY;                                          // last known "good" coordinates

            } while ((XVec != 0) || (YVec != 0));           // loop until all movement is complete
      }     // end of the movement procedures

      // check to make sure it hasn't moved outside the grid boundaries
      if (!GridPtr->CheckScope(OldX, OldY)) {
            DropRandom(P);                                          // drop it on the boundary
somewhere
            return Redrop;                                          // return the "Redrop" status
      }

      TheArray[P].x = OldX;                             // set P at the good (x,y) coords
      TheArray[P].y = OldY;
      MarkGrid(P);                                            // mark the new position on the grid

      if (Collide) return Collision;                    // if a collision occurred, return that
      else return No_Error;                             // otherwise, no errors
}


// Handle rotation for the Gaussian particle movement
void ParticleArray::GaussRot(int P) {
      int RandRot;
      byte Times;

      RandRot = RandGen.randExc(1);       // get a number [0,1)
      if (NumFaces == 4) {
            if (RandRot <= 0.5) Times = 0;              // 50% chance of no rotation
            else if (RandRot <= 0.73) Times = 1;        // 23% chance of 90 degree rotation
            else if (RandRot <= 0.96) Times = 3;        // 23% chance of -90 (270) degrees
            else Times = 2;                                     // 4% chance of 180 degrees
```

```
	}

	else if (NumFaces == 6) {
		if (RandRot <= 0.34) Times = 0;		// 34% chance of no rotation
		else if (RandRot <= 0.58) Times = 1;	// 24% chance of 60 degree rotation
		else if (RandRot <= 0.66) Times = 2;	// 8% chance of 120 degrees
		else if (RandRot <= 0.68) Times = 3;	// 4% chance of 180 degrees
		else if (RandRot <= 0.76) Times = 4;	// 8% chance of -120 (240) degrees
		else Times = 5;					// 24% chance of -60 (300) degrees
	}

	Rotate(P,Times);					// perform the particle rotation
}


// set or change the sticking type for the particles (orientation or cyrstallographic)
void ParticleArray::SetStickType(byte Type) {
	StickType = Type;
}


// returns the type of sticking used by the particles
byte ParticleArray::GetStickType(void) {
	return StickType;
}


// function used to initialize the sticking parameters for a given particle array.
// Takes 1-10 parameters based depending on the system used.
void ParticleArray::InitializeSticking(float P1, float P2, float P3, float P4, float P5, float P6,
                                       float P7, float P8, float P9, float P10) {
	delete [] StickValues;		// remove any previous stick value information
	StickValues = NULL;			// prevent multiple deletion, in case something goes wrong
```

```
if (StickType == 0) {   // orientation-based sticking
      if (NumFaces == 4) {    // square particles
            StickValues = new float[3];
            StickValues[0] = P1;    // A-A type sticking
            StickValues[1] = P2;    // B-B type sticking
            StickValues[2] = P3;    // A-B type sticking
      }
      else if (NumFaces == 6) {     // hexagonal particles
            StickValues = new float[1];
            StickValues[0] = P1;    // all faces equal... base stick chance
      }
}

else if (StickType == 1 || StickType == 2) {    // crystallography-based sticking
      if (NumFaces == 4) {    // square particles
            StickValues = (StickType == 1 ? new float[5] : new float[10]);
            StickValues[0] = P1;    // single surface energy
            StickValues[1] = P2;    // (1 0) surface energy
            StickValues[2] = P3;    // (0 1) surface energy
            StickValues[3] = P4;    // (1 1) surface energy
            StickValues[4] = P5;    // void energy
            if (StickType == 2) {
                  StickValues[5] = P6;    // single surface unstick
                  StickValues[6] = P7;    // (1 0) unstick
                  StickValues[7] = P8;    // (0 1) unstick
                  StickValues[8] = P9;    // (1 1) unstick
                  StickValues[9] = P10;   // void unstick
            }
      }
      else if (NumFaces == 6) {     // hexagonal particles
            StickValues = (StickType == 1 ? new float[4] : new float[8]);
            StickValues[0] = P1;    // single surface energy
            StickValues[1] = P2;    // (1 -2) surface energy
            StickValues[2] = P3;    // (0 1) surface energy
```

```
                  StickValues[3] = P4;    // void energy
                  if (StickType == 2) {
                        StickValues[4] = P5;    // single surface unstick
                        StickValues[5] = P6;    // (1 -2) unstick
                        StickValues[6] = P7;    // (0 1)
                        StickValues[7] = P8;    // void unstick
                  }
            }
      }
}

// used to access the StickValues for the particle array
float ParticleArray::EnergyValue(string Surface) {
      // if the array doesn't use crystallographic sticking, return -1 to indicate an error
      if (StickType != 1) return -1;

      if (Surface == "single")
            return StickValues[0];

      else if (Surface == "(1 0)")
            if (NumFaces == 4)
                  return StickValues[1];
            else if (NumFaces == 6)
                  return StickValues[2];
            else return -1;

      else if (Surface == "(0 1)")
            return StickValues[2];

      else if (Surface == "(1 1)")
            if (NumFaces == 4)
                  return StickValues[3];
            else return -1;
```

```
	else if (Surface == "(1 -2)")
		if (NumFaces == 6)
			return StickValues[1];
		else return -1;

	else if (Surface == "void")
		if (NumFaces == 4)
			return StickValues[4];
		else if (NumFaces == 6)
			return StickValues[3];
		else return -1;

	else {
		cout << "Invalid surface type request: " << Surface << endl;
		return -1;
	}
}

// used to access the unstick values of the particle array
float ParticleArray::EnergyUnstickValue(string Surface) {
// if the array doesn't use dynamic sticking, return -1 to indicate an error
	if (StickType != 2) return -1;

	if (Surface == "single") {
		if (NumFaces == 4)
			return StickValues[5];
		else if (NumFaces == 6)
			return StickValues[4];
		else return -1;
	}

	else if (Surface == "(1 0)") {
		if (NumFaces == 4)
			return StickValues[6];
```

```
            else if (NumFaces == 6)
                  return StickValues[6];
            else return -1;
      }

      else if (Surface == "(0 1)") {
            if (NumFaces == 4)
                  return StickValues[7];
            else return -1;
      }

      else if (Surface == "(1 1)") {
            if (NumFaces == 4)
                  return StickValues[8];
            else return -1;
      }

      else if (Surface == "(1 -2)") {
            if (NumFaces == 6)
                  return StickValues[5];
            else return -1;
      }

      else if (Surface == "void") {
            if (NumFaces == 4)
                  return StickValues[9];
            else if (NumFaces == 6)
                  return StickValues[7];
            else return -1;
      }

      else {
            cout << "Invalid surface type unstick request: " << Surface << endl;
            return -1;
```

```
	}
}

// see the value of one of the sticking parameters
float ParticleArray::GetStickValue(byte index) {
	if (index < 0 || index > 4) return 0;
	return this->StickValues[index];
};


// check if a particle successfully unsticks
/* Not fully implemented: needs a lot of work for different sticking types */
bool ParticleArray::CheckUnstick(int P) {
	if (CountNeighbors(P) == NumFaces) return false;
	float Chance = GridPtr->ParticleEnergy(TheArray[P].x, TheArray[P].y);
	return PercentCheck(Chance);
}



/*
=============================================================================================
                                                    Aggregate Functions
=============================================================================================
*/

// constructor for an Aggregate object.  Size = 25 by default
Aggregate::Aggregate(int Size) {

	MemberParticles = new int[Size];    // contains indices for all member particles
	for (int i = 0; i < Size; i++)
		MemberParticles[i] = -1;      // MemberParticles initially references nothing
	MaxSize = Size;         // limit it to the currently defined array length
	UsedSize = 0;           // no particles to start
```

```
}


// destructor, frees memory as usual
Aggregate::~Aggregate() {
	delete [] MemberParticles;	// free the allocated array
	MemberParticles = NULL;
}


// change the size of the allocated space in the aggregate
void Aggregate::Resize(int NewSize) {
	if (NewSize < 0) return;		// no negative size arrays

	if (NewSize < UsedSize) return;		// can't make the aggregate smaller than # of particles in it

	int* TempArray = new int[UsedSize]; // create a temporary array to hold particle info
	for (int i = 0; i < UsedSize; i++)
		TempArray[i] = MemberParticles[i];  // copy particle refs to TempArray

	delete [] MemberParticles;	// clear the allocated data
	MemberParticles = new int[NewSize]; // reallocate at the new size

	for (int i = 0; i < UsedSize; i++)  // copy data back to the
		MemberParticles[i] = TempArray[i];

	MaxSize = NewSize;
	delete [] TempArray;	// clear the temporary array memory
}


// copy data from one aggregate to another
Aggregate& Aggregate::operator=(const Aggregate& Agg) {
```

```
	MaxSize = Agg.MaxSize;  // make MaxSizes equal
	UsedSize = Agg.UsedSize;	// and UsedSizes

	delete [] MemberParticles;	// clear current member particle data
	MemberParticles = new int[MaxSize]; // reallocate to the appropriate size

	for (int i = 0; i < MaxSize; i++)
		MemberParticles[i] = Agg.MemberParticles[i];	// copy particle data over

	return *this;
}



/*
============================================================================================
                                                AggregateArray Functions
============================================================================================
*/

// constructor: for new AggregateArrays, Size = 10 by default
AggregateArray::AggregateArray(int Size, ParticleArray* PA) {
	TheArray = new Aggregate[Size];	// define the array of aggregates

	if (PA != NULL) // if a ParticleArray is provided
		SetPAPtr(PA);	// attach the aggregates to it
	else {
		PAPtr = NULL;	// otherwise, no ParticleArray or GridArray is assigned
		GridPtr = NULL;
	}

	MaxAggs = Size;	// the maximum number of aggregates with allocated storage
}
```

```
// destructor: clears the array
AggregateArray::~AggregateArray() {
      delete [] TheArray;     // free memory
      TheArray = NULL;  // for safety against duplicate deletion
}


// define the particle array associated the aggregates (gets the grid, too)
void AggregateArray::SetPAPtr(ParticleArray* PA) {
      if (PA == NULL) return; // do nothing if not ParticleArray is provided

      PAPtr = PA;       // assign PA as the particle array for these aggregates
      GridPtr = PA->GetGridPtr();   // and point to the same grid as the particles
      PA->SetAggPtr(this);    // and have the particles point to these aggregates
}


// returns the size of aggregate A
int AggregateArray::AggSize(int A) {
      return TheArray[A].UsedSize;
}


// add a new particle to aggregate A
void AggregateArray::AddParticle(int A, int P) {
      int AggSlot(0);

      // if Aggregate A is full, expand it by 10 and add the particle
      if (TheArray[A].UsedSize >= TheArray[A].MaxSize) TheArray[A].Resize(TheArray[A].MaxSize + 10);

      // if the particle is already a member of another aggregate, remove it
      int OldA = PAPtr->GetAggRef(P);
      if (OldA >= 0)
```

```
            RemParticleP(OldA,P);

      AggSlot = TheArray[A].UsedSize;           // get the next open space in the MemberParticles array
      TheArray[A].MemberParticles[AggSlot] = P; // set the array to reference that particle
      PAPtr->SetAggRef(P,A);  // set the particle to reference that aggregate
      TheArray[A].UsedSize++; // increment the count of particles in the aggregate
}


// remove a particle from the aggregate by referencing its AggregateArray index I
void AggregateArray::RemParticleA(int A, int I) {
      if (I >= TheArray[A].UsedSize) return;    // abort if the index is outside the range of particles

      // replace the indexed element with the last one, avoiding any gaps in the array
      TheArray[A].MemberParticles[I] = TheArray[A].MemberParticles[TheArray[A].UsedSize-1];
      TheArray[A].MemberParticles[TheArray[A].UsedSize] = -1;     // mark the last spot as empty
      TheArray[A].UsedSize--; // decrease the particle count for that aggregate
}


// remove a particle from the aggregate by referencing its ParticleArray index P
void AggregateArray::RemParticleP(int A, int P) {
      for (int i = 0; i < TheArray[A].UsedSize; i++)  // check the MemberParticles array
            if (PRef(A,i) == P)     // if element i refers to Particle P
                  RemParticleA(A,i);      // remove that element
}


// Locate the farthest particle dimensions in the aggregate
void AggregateArray::FindEdges(int A, int& Top, int& Bottom, int& Left, int& Right) {
      int BigX, SmallX, BigY, SmallY;
      int px, py;

      if (TheArray[A].UsedSize == 0) { // empty aggregate, nothing to do here
```

```
            Top = -1;  Bottom = -1;  Left = -1;  Right = -1;      // mark invalid edges
            return;     // and exit the function
      }

      PAPtr->Output(PRef(A,0), px, py);
      BigX = px;  SmallX = px;  BigY = py;  SmallY = py;    // start with coords of the first particle

      for (int i = 1; i < TheArray[A].UsedSize; i++) {      // look through the other particles
            PAPtr->Output(PRef(A,i), px, py);
            if (px > BigX) BigX = px;     // if it is the newest farthest particle in any direction
            if (px < SmallX) SmallX = px; // update the appropriate value
            if (py > BigY) BigY = py;
            if (py < SmallY) SmallY = py;
      }

      // copy the values into the return variables
      Top = BigY;  Bottom = SmallY;  Left = SmallX;  Right = BigX;
}


// Find the coordinates of a given corner of aggregate A
void AggregateArray::FindCorner(int A, int& x, int& y, Corner_t corner) {
      int Top, Bottom, Left, Right;

      if ((A < 0) || (A >= MaxAggs)) return;          // invalid index, do nothing

      FindEdges(A,Top, Bottom, Left, Right);          // find the edges of the aggregate

      if (corner == UpLeft) {
            x = Left;                                       // return the appropriate x and
            y = Top;                                        // y values for that corner
      }
      else if (corner == UpRight) {
            x = Right;
```

```
            y = Top;
      }
      else if (corner == DownLeft) {
            x = Left;
            y = Bottom;
      }
      else if (corner == DownRight) {
            x = Right;
            y = Bottom;
      }
}



// Create an aggregate from all fixed particles touching particle P
/* Not implemented */
void AggregateArray::MapAggregate(int A, int P) {
      if ((A < 0) || (A >= MaxAggs)) return;                // check for allowed indices
      if ((P < 0) || (P >= PAPtr->NumParticles())) return;

      Clear(A);                                                        // empty aggregate A of any
particles

      if (PAPtr->GetAttached(P) == 0) {               // if the particle has no listed attachments
            AddParticle(A,P);                                    // make it an aggregate by itself
            return;                                                        // and end the routine
      }

      else AddBranch(A,P);
}


// This function is private, used by MapAggregate() to recursively follow particle chains
```

```
void AggregateArray::AddBranch(int A, int P) {
      int px, py;
      unsigned char Neighbors, Attached;

      AddParticle(A, P);                                          // add this particle to the
aggregate
      PAPtr->Output(P, px, py);                                 // grab target coordinates
      Attached = PAPtr->GetAttached(P);                       // get list of attached particles
      Neighbors = PAPtr->GetNeighbors(P);                     // get list of neighboring particles

      // procedure for square particles
      if (PAPtr->Faces() == 4) {
            if ((Neighbors & Attached & 1) == 1)              // if there's a particle and an attachment at
(px,py+1)
                  if (GridPtr->ARef(px, py+1) != A)           // and if that neighbor isn't already a member
of A
                        AddBranch(A,GridPtr->PRef(px,py+1));      // run AddBranch centered on that
particle
            if ((Neighbors & Attached & 2) == 2)                        // repeat for a particle at
(px+1,py), etc.
                  if (GridPtr->ARef(px+1, py) != A)
                        AddBranch(A,GridPtr->PRef(px+1,py));
            if ((Neighbors & Attached & 4) == 4)
                  if (GridPtr->ARef(px, py-1) != A)
                        AddBranch(A,GridPtr->PRef(px,py-1));
            if ((Neighbors & Attached & 8) == 8)
                  if (GridPtr->ARef(px-1, py) != A)
                        AddBranch(A,GridPtr->PRef(px-1,py));
      }

      // same purpose, but for hex particles
      else if (PAPtr->Faces() == 6) {
            if ((Neighbors & Attached & 1) == 1)
                  if (GridPtr->ARef(px, py+1) != A)
```

```
                        AddBranch(A,GridPtr->PRef(px,py+1));
            if ((Neighbors & Attached & 2) == 2)
                  if (GridPtr->ARef(px+1, py) != A)
                        AddBranch(A,GridPtr->PRef(px+1,py));
            if ((Neighbors & Attached & 4) == 4)
                  if (GridPtr->ARef(px+1, py-1) != A)
                        AddBranch(A,GridPtr->PRef(px+1,py-1));
            if ((Neighbors & Attached & 8) == 8)
                  if (GridPtr->ARef(px, py-1) != A)
                        AddBranch(A,GridPtr->PRef(px,py-1));
            if ((Neighbors & Attached & 16) == 16)
                  if (GridPtr->ARef(px-1, py) != A)
                        AddBranch(A,GridPtr->PRef(px-1,py));
            if ((Neighbors & Attached & 32) == 32)
                  if (GridPtr->ARef(px-1, py+1) != A)
                        AddBranch(A,GridPtr->PRef(px-1,py+1));
      }
}



// Create a complete list of aggregates from the particles in the particle array
void AggregateArray::MapAllAggregates(void) {
      int AggNumber (0);                                                  // tracks which aggregate we're
working with

      for (int i = 0; i < MaxAggs; i++)
            Clear(i);                                                     // erase all existing aggregate
data

      // go through and assign all particles to aggregates
      for (int i = 0; i < PAPtr->NumParticles(); i++)
            if ((PAPtr->GetAggRef(i) == -1) && (PAPtr->GetDeployed(i))) {     // if the particle is
deployed and doesn't belong to an aggregate
```

```
                    AggNumber = NextFree();                         // find the next free aggregate
                    MapAggregate(AggNumber, i);                     // map the aggregate containing the
particle
              }
}



// Save the structure of the aggregate into an AggregatePattern
void AggregateArray::SavePattern(int A, AggregatePattern& AP) {
      int Top, Bottom, Left, Right;
      int px, py;

      AP.Size = TheArray[A].UsedSize;     // set the size of the pattern

      delete [] AP.XOffsets;  // clear any old arrays from the pattern object
      delete [] AP.YOffsets;
      delete [] AP.Orient;

      AP.XOffsets = new int[AP.Size];     // allocate arrays of the appropriate size
      AP.YOffsets = new int[AP.Size];
      AP.Orient = new byte[AP.Size];

      FindEdges(A, Top, Bottom, Left, Right);   // find the current edges of the aggregate A

      for (int i = 0; i < TheArray[A].UsedSize; i++) {      // for each particle making up the aggregate
            PAPtr->Output(PRef(A,i), px, py);   // get the particle coordinates
            AP.XOffsets[i] = px - Left;   // convert the (x,y) coords to offsets from the bottom-left
corner
            AP.YOffsets[i] = py - Bottom;
            AP.Orient[i] = PAPtr->Orient(PRef(A,i));
      }
}
```

```
// Produce a new aggregate from an AggregatePattern
// This constructs the aggregate with (0,0) as the bottom left corner, but does not notify the grid
void AggregateArray::LoadPattern(int A, const AggregatePattern& AP) {
      Clear(A);   // remove any pre-existing aggregate data from A

      for (int i = 0; i < AP.Size; i++) { // for each particle in the pattern
            int P = PAPtr->NextFree();    // find a free particle in the ParticleArray
            PAPtr->GhostTo(P, AP.XOffsets[i], AP.YOffsets[i]);    // move it to the appropriate position
            AddParticle(A, P);      // add the particle to Aggregate A
            PAPtr->SetDeployed(P, true);  // mark the particle as deployed
      }

      TheArray[A].UsedSize = AP.Size;
}


// Completely clear all data inside an aggregate.  Removes member particles as well.
void AggregateArray::Kill(int A) {
      cout << "Killing aggregate containing " << TheArray[A].UsedSize << " particles.\n";
      while (TheArray[A].UsedSize > 0) {
            cout << " UsedSize = " << TheArray[A].UsedSize << endl;
            cout << "Killing particle " << PRef(A,0) << endl;
            PAPtr->Kill(PRef(A,0)); // kill the particles comprising the array
      }
}


// Changes the number of allocated aggregates
void AggregateArray::ResizeArray(int NewSize) {
      if (NewSize < 0) return;      // no negative sizes

      // copy the current aggregate information into a temporary array
      Aggregate* TempArray = new Aggregate[MaxAggs];
```

```
	for (int i = 0; i < MaxAggs; i++)
		TempArray[i] = TheArray[i];

	delete [] TheArray;	// free the current array

	TheArray = new Aggregate[NewSize];  // reallocate the array at the new size

	int CopyAggs = (NewSize > MaxAggs) ? MaxAggs : NewSize;	// CopyAggs = the smaller of MaxAggs or
NewSize
	for (int i = 0; i < CopyAggs; i++) // copy the aggregate data back to the new array
		TheArray[i] = TempArray[i];

	MaxAggs = NewSize;	// update the size
}


// returns the index of the next unused aggregate in the array
// if none is found, expands the array and returns a new aggregate
int AggregateArray::NextFree(void) {
	for (int i = 0; i < MaxAggs; i++)	// search the aggregates
		if (TheArray[i].UsedSize == 0) return i;  // return the first one containing no particles

	// if no empty aggregate is found, expand the array to make new ones
	int NextAgg = MaxAggs;
	ResizeArray(MaxAggs + 10);	// expand the array by ten aggregates
	return NextAgg;	// return the index of the new aggregate
}


// Merges two aggregates into one
void AggregateArray::Merge(int A1, int A2) {
	if (A1 == A2) return;	// can't merge an aggregate with itself
	if ((A1 < 0) || (A2 < 0)) return;	// no invalid aggregate indices
	if ((A1 >= MaxAggs) || (A2 >= MaxAggs)) return; // outside the valid range
```

```
	for (int i = 0; i < TheArray[A1].UsedSize; i++) { // go through all particles in A1
		AddParticle(A2,TheArray[A1].MemberParticles[i]);	// add them to A2
		TheArray[A1].MemberParticles[i] = -1;	// remove the reference from A1
	}

	TheArray[A1].UsedSize = 0;	// Aggregate A1 is now empty
}


// Clears an existing Aggregate of all information
void AggregateArray::Clear(int A) {
	for (int i = 0; i < TheArray[A].UsedSize; i++) {
		if (PAPtr->GetAggRef(PRef(A,i)) == A)	// if the particle is still marked as part of the
aggregate
			PAPtr->SetAggRef(PRef(A,i), -1);	// then disassociate it
		TheArray[A].MemberParticles[i] = -1;	// remove reference to the particle
	}

	TheArray[A].UsedSize = 0;	// no particles are assigned to the aggregate
}


// return the particle reference for member I of aggregate A
int AggregateArray::PRef(int A, int I) {
	return TheArray[A].MemberParticles[I];
}



// Rotate the entire aggregate.  If Times is unspecified, the rotation is random
/* Not fully implemented: handle rotation outside in a satisfactory fashion
	Another potential spot for improvement: keep it within the properly transformed edges (fix 'wobble'
in rotations)*/
```

```
bool AggregateArray::Rotate(int A, char Times) {

      // single particle version
      if (AggSize(A) == 1) {
            PAPtr->Rotate(PRef(A,0),Times);
            return true;
      }

      bool Success = true;    // marked false if anything goes wrong
      bool OutsideGrid = false;     // tripped if the aggregate rotates outside the grid boundaries
      int CenterX, CenterY, Top, Bottom, Left, Right; // boundaries and center of the aggregate
      int OldDeltaX, OldDeltaY, DeltaX, DeltaY; // variables for use in the coordinate transforms
      int px, py; // particle coordinates

      int* XPositions;
      int* YPositions;
      XPositions = new int[TheArray[A].UsedSize];     // store the new x positions for each particle
      YPositions = new int[TheArray[A].UsedSize];     // store the new y positions for each particle

      if (Times < 0) // if negative (the default), a random value is used
            Times =     (char)RandGen.randInt(PAPtr->Faces()-1);  // get a random value for times

      Times = Times % PAPtr->Faces();     // mod Times by the number of faces (rotating a square 6 times is
the same as 2)

      FindEdges(A, Top, Bottom, Left, Right);
      CenterX = (Left + Right)/2;   // find the (approximate) center of the aggregate
      CenterY = (Top + Bottom)/2;

      for (int i = 0; i < TheArray[A].UsedSize; i++) {

            PAPtr->Output(PRef(A,i),px,py);     // get current coordinates of the member particle
            PAPtr->ClearMark(PRef(A,i));  // temporarily clear the grid locations, to avoid intra-aggregate
collision detection
```

```
            DeltaX = px - CenterX;  // find the offset from the center of the aggregate in each direction
            DeltaY = py - CenterY;

            // This is the actual rotation.  Perform these steps once for each face it is to be rotated
            for (int j = 0; j < Times; j++) {
                  OldDeltaX = DeltaX;           // store the current offset values
                  OldDeltaY = DeltaY;

                  // this part rotates by one face by transforming the x and y coordinates of the offsets
                  DeltaX = -1 * OldDeltaY;      // x' = -y for both axis types
                  if (PAPtr->Faces() == 4)
                        DeltaY = OldDeltaX;           // squares: y' = x
                  else if (PAPtr->Faces() == 6)
                        DeltaY = OldDeltaX + OldDeltaY;     // hexagons: y' = x + y
            }

            XPositions[i] = CenterX + DeltaX;
            YPositions[i] = CenterY + DeltaY;
      }

      // now check for collisions and grid boundaries
      for (int i = 0; i < TheArray[A].UsedSize; i++) {
            // grid boundaries; this part needs some serious redoing
            if (GridPtr->CheckScope(XPositions[i], YPositions[i]) == false) {
                  OutsideGrid = true;     // mark this flag, for when it actually does something
                  Success = false;  // rotation outside the grid causes failure
                  break;      // no need to check the rest as long as this is considered a fail
            }

            // check the grid for any collisions with other aggregates
            if (GridPtr->Occupied(XPositions[i], YPositions[i])) {      // if the new position is occupied
                  Success = false;  // mark failure
                  break;      // skip the rest of the check
            }
```

```
	}

	for (int i = 0; i < TheArray[A].UsedSize; i++) {
		if (Success) {                                          // if the rotation hasn't failed for any
reason
			int p = PRef(A,i);
			PAPtr->GhostTo(p, XPositions[i], YPositions[i]);      // update the actual particle
position
			PAPtr->Rotate(p, Times);      // change the orientation of the individual particles with
the rotation
			PAPtr->RotAttach(p, Times);   // change the "attached" flags to match the new
orientations
		}
		PAPtr->Output(PRef(A,i), px, py);         // grab the coordinates (new or old, depending on
success)
		GridPtr->MarkP(PRef(A,i), px, py);        // mark the grid position
	}

	delete [] XPositions;                                   // free memory used to store the temporary
positions
	delete [] YPositions;

	return Success;                                         // report whether it was successful or
not
}


// Moves the aggregates to a specific location (specify the corner to use as a reference)
MoveError_t AggregateArray::MoveTo(int A, int x, int y, Corner_t corner) {
	int Top, Bottom, Left, Right;                     // aggregate edges
	int PointX, PointY;                                     // coordinates of the selected corner
	int px, py, NewX, NewY;                           // particle coordinates

	FindEdges(A, Top, Bottom, Left, Right);         // find the aggregate boundaries
```

```
	if (corner == UpRight) {                                        // upper right corner
		PointX = Right;
		PointY = Top;
	}
	else if (corner == DownRight) {                                 // lower right corner
		PointX = Right;
		PointY = Bottom;
	}
	else if (corner == DownLeft) {                                  // lower left corner
		PointX = Left;
		PointY = Bottom;
	}
	else if (corner == UpLeft) {                                // upper left corner
		PointX = Left;
		PointY = Top;
	}

	// check that no destination spaces are occupied
	for (int i = 0; i < TheArray[A].UsedSize; i++) {
		PAPtr->Output(PRef(A,i), px, py);
		NewX = px + x - PointX;                             // calculate the new position of the particle
		NewY = py + y - PointY;

		if (!GridPtr->CheckScope(NewX, NewY)) return Redrop;  // return "Redrop" if it would leave the
grid

		if (GridPtr->Occupied(NewX, NewY))          // if the space is occupied
			if (GridPtr->ARef(NewX, NewY) != A) // and the occupying particle does not belong to A
				return Collision;                   // abort the move, return "Collision"

	}

	// all checks are passed, now move the particles
	SetDeployed(A, true);                                       // mark all particles as deployed
```

```
	for (int i = 0; i < TheArray[A].UsedSize; i++) {
		PAPtr->ClearMark(PRef(A,i));		// clear the grid marking for that particle
		PAPtr->Output(PRef(A,i), px, py);	// get the current coordinates
		NewX = px + x - PointX;			// calculate new coordinates
		NewY = py + y - PointY;
		PAPtr->GhostTo(PRef(A,i), NewX, NewY);	// move the particle
		PAPtr->MarkGrid(PRef(A,i));		// mark the new position on the grid
	}

	return No_Error;
}


// Randomly drops the entire aggregate A along the scope boundaries
void AggregateArray::DropRandom(int A, bool Edge) {
	int NewX, NewY;					// stores the new coordinates
	int Top, Bottom, Left, Right;			// the aggregate boundaries
	MoveError_t Result;

	FindEdges(A, Top, Bottom, Left, Right);		// find the edges of the aggregate

	do {
		if (Edge) GridPtr->GetRandomEdge(NewX, NewY, Right-Left, Top-Bottom);	// find an edge site
		else GridPtr->GetRandomSpace(NewX, NewY, Right-Left, Top-Bottom); // or a space anywhere in the
scope
		Result = MoveTo(A, NewX, NewY);		// try to move the aggregate, get result
	} while (Result != No_Error);			// if the move fails, get a new position and try
again

	SetDeployed(A, true);				// mark the aggregate as deployed

	if (PAPtr->AllowRot()) Rotate(A);		// if rotation is allowed, give it a new random
orientation
}
```

```
// Randomly moves all particles in the aggregate one step
int AggregateArray::RandMove(int A) {
      int Top, Bottom, Left, Right;
      byte PosShift(0);
      int px, py;
      int XShift(0), YShift(0);

      if ((A >= MaxAggs) || (A < 0)) return Out_of_Range;   // abort for out of range A
      if (GetDeployed(A) == false) return Not_Deployed;           // abort if not all particles are marked
deployed

      FindEdges(A, Top, Bottom, Left, Right);         // get the aggregate edges

      if (PAPtr->Faces() == 4) {                            // movement for square grids

            PosShift = (byte)RandGen.randInt(3);      // random integer [0,3]

            if (PosShift == 0) YShift = +1;                 // move the aggregate up
            else if (PosShift == 1) XShift = +1;      // move right
            else if (PosShift == 2) YShift = -1;      // move down
            else if (PosShift == 3) XShift = -1;      // move left
      }

      else if (PAPtr->Faces() == 6) {

            PosShift = (byte)RandGen.randInt(5);      // random integer [0,5]
            if (PosShift == 0) YShift = +1;                 // move up
            else if (PosShift == 1) XShift = +1;      // move up-right
            else if (PosShift == 2) {                       // move down-right
                  XShift = +1;
                  YShift = -1;
            }
```

```
            else if (PosShift == 3) YShift = -1;      // move down
            else if (PosShift == 4) XShift = -1;      // move down-left
            else if (PosShift == 5) {                       // move up-left
                  XShift = -1;
                  YShift = +1;
            }
      }

      if (PosShift == 0) cout << "Shift up\n";
      if (PosShift == 1) cout << "Shift right\n";
      if (PosShift == 2) cout << "Shift down\n";
      if (PosShift == 3) cout << "Shift left\n";

      // Check if the move will put part of the aggregate outside the scope
      if (!(GridPtr->CheckScope(Right+XShift, Top+YShift)) || !(GridPtr->CheckScope(Left+XShift,
Bottom+YShift))) {
            DropRandom(A);                                          // drop the aggregate somewhere else on
the boundary
            return Redrop;                                                // return error code 3 - moved
outside the grid
      }

      // check for collisions with other particles
      for (int i = 0; i < TheArray[A].UsedSize; i++) {
            PAPtr->Output(PRef(A,i), px, py);         // get current particle coordinates
            if (GridPtr->Occupied(px+XShift, py+YShift))    // if the new location is occupied
                  if (GridPtr->ARef(px+XShift, py+YShift) != A)   // and the particle there is not a part
of A
                        return Collision;                               // error code 4 - destination
occupied
      }

      // all checks passed, move the particles that make the aggregate
      for (int i = 0; i < TheArray[A].UsedSize; i++) {
```

```
            PAPtr->ClearMark(PRef(A,i));              // clear the particle's grid marking
            PAPtr->Output(PRef(A,i), px, py);         // get current particle (x,y)
            PAPtr->GhostTo(PRef(A,i), px+XShift, py+YShift);      // move to the new location
            PAPtr->MarkGrid(PRef(A,i));   // mark the new location on the grid
      }

      return No_Error;                                // if it made it this far, there are no errors
}


// Check the "deployed" flag for the particles in A.  Returns false if any are false
bool AggregateArray::GetDeployed(int A) {
      if (TheArray[A].UsedSize == 0) return false;    // false for empty aggregates
      if ((A >= MaxAggs) || (A < 0)) return false;    // false for out of range A

      for (int i = 0; i < TheArray[A].UsedSize; i++)  // check each particle in A
            if (PAPtr->GetDeployed(PRef(A,i)) == false)     // if "deployed" is false for any
                  return false;                                     // return false for the function

      return true;                                                  // otherwise return true
}


// Change the "deployed" flag for all member particles of aggregate A.
void AggregateArray::SetDeployed(int A, bool Deployed) {
      if ((A >= MaxAggs) || (A < 0)) return;                // do nothing for out of range A

      for (int i = 0; i < TheArray[A].UsedSize; i++)
            PAPtr->SetDeployed(PRef(A,i), Deployed);  // set the flag for each particle
}


// Check the "fixed" flag for the aggregate.  Returns true if any are fixed
bool AggregateArray::GetFixed(int A) {
```

```
      if (TheArray[A].UsedSize == 0) return false;    // false for empty aggregates
      if ((A >= MaxAggs) || (A < 0)) return false;    // false for out of range A

      for (int i = 0; i < TheArray[A].UsedSize; i++)  // check each particle in A
            if (PAPtr->GetFixed(PRef(A,i)) == true    )     // if "fixed" is true for any
                  return false;                                   // return true for the function

      return false;                                               // otherwise return false
}


// Change the "fixed" flag for all member particles of aggregate A.
void AggregateArray::SetFixed(int A, bool Fixed) {
      if ((A >= MaxAggs) || (A < 0)) return;                // do nothing for out of range A

      for (int i = 0; i < TheArray[A].UsedSize; i++)
            PAPtr->SetFixed(PRef(A,i), Fixed);              // set the flag for each particle
}


// Moves A along the specified vector, checking for collisions, grid boundaries
MoveError_t AggregateArray::VectorMove(int A, int XVec, int YVec) {


      int OldX, OldY;                                               // x,y coordinates before the move
      FindCorner(A, OldX, OldY);                                // find lower left corner of A

      int NewX(OldX), NewY(OldY);                               // x,y coordinate after the move
      MoveError_t Result = No_Error;                            // tracks potential errors from each
individual move

      float Slope = (float)YVec / (float)XVec;  // find the |slope| of the particle's path
      bool SwitchXY = false;                                    // if x and y values were switched
```

```
      if (fabs(Slope) < 1) {                                  // if the particle moves more x than y
            SwitchXY = true;                                  // activate the SwitchXY flag
            int temp = XVec;                                  // and switch them
            XVec = YVec;
            YVec = temp;
            Slope = 1/Slope;                                  // invert the slope
      }

      do {
            if (abs(XVec * Slope) >= abs(YVec)) {     // if the particle is along the right path
                  if (XVec > 0) {                                 // move it in the x direction,
                        XVec--;                                         // set XMove one closer to zero,
                        if (SwitchXY) NewY++;               // and change the coordinates
                        else NewX++;                                    // (change y if SwitchXY was activated)
                  }
                  else if (XVec < 0) {                      // same thing, but this is for negative XMove
                        XVec++;
                        if (SwitchXY) NewY--;
                        else NewX--;
                  }
            }

            if (YVec > 0) {                                         // y always moves.
                  YVec--;                                           // the rest of this is analogous to
the x case
                  if (SwitchXY) NewX++;
                  else NewY++;
            }
            else if (YVec < 0) {
                  YVec++;
                  if (SwitchXY) NewX--;
                  else NewY--;
            }
```

```
            Result = MoveTo(A, NewX, NewY);

            if (Result == Collision) return Collision;
            if (Result == Redrop) {                          // if A moved outside the grid
                  DropRandom(A);                                   // redrop it somewhere else
                  return Redrop;                                   // and return a "Redrop" result
            }

      } while ((XVec != 0) || (YVec != 0));           // loop until all movement is complete

      return No_Error;                          // if it makes it this far, there were no errors
}


// Moves A according to a Gaussian distribution based on aggregate size and shape
MoveError_t AggregateArray::GaussMove(int A) {

      int XMove (0), YMove (0);                                    // hold movement data in each direction

      if (GetFixed(A)) return Fixed_Particle    ;           // cancel the move if the particle is fixed in
position
      if (!GetDeployed(A)) return Not_Deployed;       // can't move undeployed particles

      // if it's a single particle, speed things up
      if (AggSize(A) == 1)
            return PAPtr->GaussMove(PRef(A,0));             // use the particle version

      TheArray[A].MoveCount--;                                     // decrease the turn counter for A
      if (TheArray[A].MoveCount > 0)                               // if it isn't A's turn to move
            return No_Error;                                       // no error, it just doesn't move

      else                                                               // otherwise, reset its counter and
move it
            TheArray[A].MoveCount = TheArray[A].MaxMoveCount;
```

```
	XMove = round(RandGauss(TheArray[A].SigmaX));	// get x component using st. dev. SigmaX
	YMove = round(RandGauss(TheArray[A].SigmaY));	// and the y component, using SigmaY

	if (PAPtr->AllowRot())					// if rotation is enabled
		GaussRot(A);					// use Gaussian rotation
	return VectorMove(A, XMove, YMove);		// move the particle and return any error
generated
}


// Rotate aggregate A using Gaussian probabilities
void AggregateArray::GaussRot(int A) {
	int RandRot;
	byte Times;

	TheArray[A].RotCount--;				// decrease the rotation count
	if (TheArray[A].RotCount > 0)		// if it isn't time to rotate
		return;						// ...do nothing
	else							// otherwise, reset the counter and carry on
		TheArray[A].RotCount = TheArray[A].MaxRotCount;

	RandRot = RandGen.randExc(1);		// get a number [0,1)
	if (PAPtr->Faces() == 4) {
		if (RandRot <= 0.5) Times = 0;		// 50% chance of no rotation
		else if (RandRot <= 0.73) Times = 1;	// 23% chance of 90 degree rotation
		else if (RandRot <= 0.96) Times = 3;	// 23% chance of -90 (270) degrees
		else Times = 2;					// 4% chance of 180 degrees
	}

	else if (PAPtr->Faces() == 6) {
		if (RandRot <= 0.34) Times = 0;		// 34% chance of no rotation
		else if (RandRot <= 0.58) Times = 1;	// 24% chance of 60 degree rotation
		else if (RandRot <= 0.66) Times = 2;	// 8% chance of 120 degrees
```

```
            else if (RandRot <= 0.68) Times = 3;      // 4% chance of 180 degrees
            else if (RandRot <= 0.76) Times = 4;      // 8% chance of -120 (240) degrees
            else Times = 5;                                 // 24% chance of -60 (300) degrees
      }

      Rotate(A,Times);                                      // perform the particle rotation
      if ((Times % PAPtr->Faces()) != 0) GaussCalc(A);      // if rotation occurred, recalculate Gauss
stuff
}


// Calculate values of the Gaussian movement variables for aggregate A
void AggregateArray::GaussCalc(int A) {
      int Top, Bottom, Left, Right;
      int dx, dy;
      double RotCountRoot

      // single particle version
      if (AggSize(A) == 1) {        // everything is defined as 1 for single particles
            TheArray[A].MaxMoveCount = 1;
            TheArray[A].SigmaX = 1;
            TheArray[A].SigmaY = 1;
            TheArray[A].MaxRotCount = 1;
            return;
      }


      FindEdges(A, Top, Bottom, Left, Right);         // find the dimensions of the aggregate
      dx = Right - Left + 1;                                // x diameter
      dy = Top - Bottom +1;                                 // y diameter

      TheArray[A].MaxMoveCount = dx > dy ? dx : dy;                     // MoveCount = the larger of the
two diameters
```

```
	TheArray[A].SigmaX = sqrt((float)TheArray[A].MaxMoveCount/(float)dy);	// standard deviation of x
movement
	TheArray[A].SigmaY = sqrt((float)TheArray[A].MaxMoveCount/(float)dx);	// st. dev. of y movement
	RotCountRoot = (sqrt((float)dx*(float)dy))/2;	// treat as spehere with geometric mean of diameters
	TheArray[A].MaxRotCount = (int)ceil(RotCountRoot*RotCountRoot*RotCountRoot);	// RotationCount = mean
diameter^3 rounded up
}


// how many aggregates are currently allocated
int AggregateArray::AllocatedAggs(void) {
	return MaxAggs;
}


// how many aggregates are acutally being used
int AggregateArray::UsedAggs(void) {
	int UsedCount = 0;
	for (int i = 0; i < MaxAggs; i++)
		if (AggSize(i) > 0)
			UsedCount++;
	return UsedCount;
}


// not implemented: checks all possible sticking of the aggregate (uses PAPtr->GetStickType)
// returns true if any new stick occurs, false if there are none
bool AggregateArray::CheckStick(int A) {
	bool Result = false;

	// for orientation stickings - check for individual particles with neighbors, check for each stick
	if (PAPtr->GetStickType() == 0) {

		// if the aggregate is a singlet, use that routine
```

```
	/*
	if (AggSize(A) == 1) {
		byte AttachDir = PAPtr->CheckStick(PRef(A,0));
		if (AttachDir) {  // if sticking is successful
			PAPtr->Attach(PRef(A,0),AttachDir); // attach the particles
			return true;      // and return true
		}
		else return false;      // otherwise, return false
	}*/

	// single particle sticking using the BaseStickChance
	/*Not fully implemented - temporary work around */
	if (AggSize(A) == 1) {
		return PercentCheck(PAPtr->BaseStick);
	}


	for (int i = 0; i < AggSize(A); i++) {          // look at each particle in A
		int p = PRef(A,i);
		if (PAPtr->GetNeighbors(p,false) != 0) {  // see if it has neighbors
			byte NewSticks = PAPtr->CheckStick(p);    // check for any new sticks
			if (NewSticks != 0) {                     // if a stick is successful:
				PAPtr->Attach(p, NewSticks);  // make the relevant attachments
				Result = true;                            // ...and mark the result as true
			}
		}     // end if neighbors
	}     // end for loop
}     // end StickType = orient

// for crystallographic stickings
else if (PAPtr->GetStickType() == 1) {
	for (int A2 = 0; A2 < UsedAggs(); A2++)         // check all aggregates
		if ((A2 != A) && (AreNeighbors(A,A2))) {  // if the aggregate neighbors A
			if (StickCheckCrystal(A,A2)) {      // perform a crystallographic stick check
```

```
                                    Result = true;                                  // if it happens, mark the result
as true
                                    AttachAggs(A,A2);                               // mark the necessary attachments
                              }
                        }
            }
            return Result;
}

#include <fstream>
// Check to see if two aggregates attach using crystallographic methods
// not implemented: currently a mess, a lot of tweaking is needed here.
bool AggregateArray::StickCheckCrystal(int A, int A2) {

      double SurfE1, SurfE2, SurfET;            // variables to stores the surface energy changes

      SurfE1 = SurfaceEnergy(A);                // energy of the first aggregate isolated
      SurfE2 = SurfaceEnergy(A2);               // second aggregate energy
      SurfET = SurfaceEnergyComb(A,A2);   // energy of the combined aggregate

      double DeltaH = SurfET - (SurfE1 + SurfE2);           // find the overall energy chance

      // single particle check.  CHANGE THIS LATER!!
      if (AggSize(A) == 1) {
            if (AggSize(A2) == 1) return true;
            int px, py;
            PAPtr->Output(PRef(A,0),px,py);
            if (RandGen.rand53() < (GridPtr->SurfaceEnergy(px,py,A2))/100)
                  return true;
            else return false;
      }

      /* Constants for the energy equation */
      long double k = 1;
```

```
	double PreExp = .1;
	/* Change these to make some sense */

	/* This will be the part that changes stick chance with aggregate size */
	double SizeEffect = 1;
	/* For now, it has no effect */

	// calculate the sticking probability
	double StickChance = SizeEffect * PreExp * exp(-DeltaH/k);

	// DEBUG info
	//cout << "Stick check attempted: \n";
	//cout << "\t DeltaH = " << DeltaH << endl;
	//cout << "\t StickChance = " << StickChance*100 << "%\n";
	ofstream File;
	int px, py;
	static bool FileOpened(false);

	if (!FileOpened) {
		File.open("DeltaH.txt",ios::out);
		File << "DeltaH\tx\ty\tA\tSurfE2\tSurfET\tStick %\n";
		FileOpened = true;
	} else File.open("DeltaH.txt",ios::out | ios::app);
	PAPtr->Output(PRef(A,0),px,py);
	File << DeltaH << "\t" << px << "\t" << py << "\t" << A << "\t" << SurfE2 << "\t" << SurfET << " \t"
<< StickChance*100 << endl;
	File.close();

	if (RandGen.rand53() < StickChance) { // if the roll is less than the stick chance
		//cout << "Stick successful: ";
		//cin >> mystring;
		//cout << endl;
		return true;                  // the stick is successful
	}
```

```
	else
		return false;			// otherwise, it isn't.
}


// this returns the total surface energy of aggregate A.
// Molar = true means results are divided by the number of surface sites.
double AggregateArray::SurfaceEnergy(int A, bool molar) {
	int Top, Bottom, Left, Right;
	double TotalEnergy (0);
	int SiteCount (0);

	// single units are not considered to have surface energy
	//if (AggSize(A) == 1)
	//	return 0;

	FindEdges(A,Top,Bottom,Left,Right);

	for (int i = Left-1; i <= Right+1; i++) {
		for (int j = Bottom-1; j <= Top+1; j++) {
			double Temp = GridPtr->SurfaceEnergy(i,j,A);
			if (Temp != 0) {
				SiteCount++;		// if it's a valid site, increase the count
				TotalEnergy += Temp;	// add the energy value to total energy
				//cout << "Energy: " << Temp << " at (" << i << ", " << j << ")\n";
				//cout << "\tTotal energy = " << TotalEnergy << endl;
			}

		}
	}

	//cout << "DEBUG: Aggregate " << A << " has " << SiteCount << " surface sites.\n";
	//cout << "DEBUG: Total energy = " << TotalEnergy << "; Molar energy = " << TotalEnergy/SiteCount <<
endl;
```

```
	//cout << "\n\n";

	if (molar) // if we're requesting molar energy, divide the energy by the # of surface sites
		TotalEnergy /= SiteCount;

	return TotalEnergy;
}


double AggregateArray::SurfaceEnergyComb(int A, int A2, bool molar) {
	int Top, Bottom, Left, Right;
	int Top2, Bottom2, Left2, Right2;
	double TotalEnergy (0);
	int SiteCount (0);

	// if combining two singles, give it a base energy of zero
	//if ((AggSize(A) == 1) && (AggSize(A2) == 1))
	//	return 0;

	FindEdges(A,Top,Bottom,Left,Right);
	FindEdges(A2,Top2,Bottom2,Left2,Right2);

	Top = (Top > Top2) ? Top: Top2;
	Bottom = (Bottom < Bottom2) ? Bottom : Bottom2;
	Left = (Left < Left2) ? Left : Left2;
	Right = (Right > Right2) ? Right: Right2;

	for (int i = Left-1; i <= Right+1; i++) {
		for (int j = Bottom-1; j <= Top+1; j++) {
			float Temp = GridPtr->SurfaceEnergy(i,j,A,A2);
			if (Temp != 0) {
				SiteCount++;			// if it's a valid site, increase the count
				TotalEnergy += Temp;	// add the energy value to total energy
				//cout << "Energy: " << Temp << " at (" << i << ", " << j << ")\n";
```

```
                        //cout << "\tTotal energy = " << TotalEnergy << endl;
                  }

            }
      }

      //cout << "DEBUG: Aggregate " << A << " has " << SiteCount << " surface sites.\n";
      //cout << "DEBUG: Total energy = " << TotalEnergy << "; Molar energy = " << TotalEnergy/SiteCount <<
endl;
      //cout << "\n\n";

      if (molar) // if we're requesting molar energy, divide the energy by the # of surface sites
            TotalEnergy /= SiteCount;

      return TotalEnergy;
}


// this checks for any neighbors to the array
bool AggregateArray::HasNeighbors(int A) {
      for (int i = 0; i < AggSize(A); i++)      // check all particles in the array
            if (PAPtr->GetNeighbors(PRef(A,i),false) != 0)  // if they have neighbors
                  return true;      // then the aggregate has neighbors

      return false;     // if it gets here, nothing has a neighbor so return false
}


// Returns true if aggregates A and A2 are neighbors
bool AggregateArray::AreNeighbors(int A, int A2) {
      if ((A < 0) || (A2 < 0) || (A == A2)) return false;         // inappropriate conditions
      for (int i = 0; i < AggSize(A); i++) {    // check all particles in the aggregate
            if (PAPtr->GetNeighbors(PRef(A,i), false) != 0) {     // if the particle has neighbors
                  int px, py;
```

```
                  PAPtr->Output(PRef(A,i),px,py);
                  if (GridPtr->CountNeighbors(px,py,A2) != 0)          // check to see if any belong to A2
                        return true;            // if so, it's true
            }
      }
      return false;     // if nothing is found, it's false
}


// this updates the particle-particle attach flags between two aggregates
// Result: merged by attachment, but not by aggregate object
void AggregateArray::AttachAggs(int A, int A2) {
      for (int i = 0; i < TheArray[A].UsedSize; i++) {      // for each particle in the aggregate
            int px, py;
            byte NList;
            int P = PRef(A,i);                  // store the particle index
            PAPtr->Output(P,px,py);                                    // get the particle's coordinates
            GridPtr->CountNeighbors(px,py,A2,NList);        // get a list of neighbors belonging to A2
            if (NList != 0)
                  PAPtr->Attach(P,NList);                                    // attach P to all A2 member
neighbors
      }
}

/*
=============================================================================================
                                          AggregatePattern Functions
=============================================================================================
*/

// simple constructor, initialize all variables
AggregatePattern::AggregatePattern() {
      Size = 0;   // nothing stored yet
      XOffsets = NULL;  // doesn't allocate any memory yet, but sets the pointers to
```

```
	YOffsets = NULL;  // NULL, so it doesn't cause any harm if delete is used on them
	Orient = NULL;
}


// destructor, also simple
AggregatePattern::~AggregatePattern() {
	delete [] XOffsets;	// free all memory.  If the pattern is destructed without ever having
	delete [] YOffsets;	// memory allocated, this does nothing because the pointers are set
	delete [] Orient; // to NULL, not any (potentially) conflicted memory address
}


// assignment operator: used to copy aggregate patterns
AggregatePattern& AggregatePattern::operator =(const AggregatePattern& AP2) {
	if (this == &AP2) return *this;			// check for self-assignment

	Size = AP2.Size;  // match the array sizes
	delete [] XOffsets;	// clear out any pre-existing pattern data
	delete [] YOffsets;
	delete [] Orient;

	XOffsets = new int[Size];	// reallocate the arrays at the proper size
	YOffsets = new int[Size];
	Orient = new byte[Size];

	for (int i = 0; i < Size; i++) {	// copy the data over from AP2
		XOffsets[i] = AP2.XOffsets[i];
		YOffsets[i] = AP2.YOffsets[i];
		Orient[i] = AP2.Orient[i];
	}

	return *this;	// return the new pattern
}
```

```
/*
================================================================================================
                                                Universal Functions
================================================================================================
*/

// Uses Mersenne Twister floating point random to perform probability checks.  Returns true if the value <
PerVal.
bool PercentCheck(double PerVal) {
      if (PerVal == 0) return false;                                  // no need to generate for the zero and
100% cases
      else if (PerVal == 100) return true;

      double test = RandGen.randExc();                          // get a random number [0,1)
      if ((test * 100) < PerVal) return true;                   // if the number < specified % chance, return
true
      else return false;                                                  // otherwise, it fails the check,
return false
}


// function to "rotate" a bitlist counterclockwise, using particle logic (used for surface energy,
aggregate rotation)
void RotList(byte& BitList, int Times, byte Sides) {
      if (Times < 0)
            Times = (Times % Sides) + Sides;    // convert negative (clockwise) rotations into the
equivalent positive

      if (Times >= Sides) Times = Times % Sides;      // no need for extra equivalent rotations

      for (int i = 0; i < Times; i++) {         // repeat once for each desired rotation
            if ((BitList & 1) == 1)                   // if the lowest face is occupied
```

```
                BitList |= (1 << Sides);            // make the faces loop as it rotates
            BitList >>= 1;                                  // shift the values right one place
      }
}

// Uses the MT to generate a gaussian random number with the Ziggarut algorithm
double RandGauss(double sigma) {
      unsigned long  U, sign, i, j;
      double  x, y;
      const double PARAM_R = 3.44428647676;

      while (true) {
            U = RandGen.randInt();  // get random number from uniform distribution
            i = U & 0x0000007F;           /* 7 bit to choose the step */
            sign = U & 0x00000080;  /* 1 bit for the sign */
            j = U>>8;               /* 24 bit for the x-value */

            x = j*wtab[i];
            if (j < ktab[i])  break;

            if (i<127) {
                  double  y0, y1;
                  y0 = ytab[i];
                  y1 = ytab[i+1];
                  y = y1+(y0-y1)*RandGen.randExc();
            }
            else {
                  x = PARAM_R - log(1.0-RandGen.randExc())/PARAM_R;
                  y = exp(-PARAM_R*(x-0.5*PARAM_R))*RandGen.randExc();
            }

            if (y < exp(-0.5*x*x))  break;
      }
```

```
  return  sign ? sigma*x : -sigma*x;
}

// Tables for use in the Ziggarut function
// tabulated values for the height of the Ziggurat levels
static const double ytab[128] = {
  1, 0.963598623011, 0.936280813353, 0.913041104253,
  0.892278506696, 0.873239356919, 0.855496407634, 0.838778928349,
  0.822902083699, 0.807732738234, 0.793171045519, 0.779139726505,
  0.765577436082, 0.752434456248, 0.739669787677, 0.727249120285,
  0.715143377413, 0.703327646455, 0.691780377035, 0.68048276891,
  0.669418297233, 0.65857233912, 0.647931876189, 0.637485254896,
  0.62722199145, 0.617132611532, 0.607208517467, 0.597441877296,
  0.587825531465, 0.578352913803, 0.569017984198, 0.559815170911,
  0.550739320877, 0.541785656682, 0.532949739145, 0.524227434628,
  0.515614886373, 0.507108489253, 0.498704867478, 0.490400854812,
  0.482193476986, 0.47407993601, 0.466057596125, 0.458123971214,
  0.450276713467, 0.442513603171, 0.434832539473, 0.427231532022,
  0.419708693379, 0.41226223212, 0.404890446548, 0.397591718955,
  0.390364510382, 0.383207355816, 0.376118859788, 0.369097692334,
  0.362142585282, 0.355252328834, 0.348425768415, 0.341661801776,
  0.334959376311, 0.328317486588, 0.321735172063, 0.31521151497,
  0.308745638367, 0.302336704338, 0.29598391232, 0.289686497571,
  0.283443729739, 0.27725491156, 0.271119377649, 0.265036493387,
  0.259005653912, 0.253026283183, 0.247097833139, 0.241219782932,
  0.235391638239, 0.229612930649, 0.223883217122, 0.218202079518,
  0.212569124201, 0.206983981709, 0.201446306496, 0.195955776745,
  0.190512094256, 0.185114984406, 0.179764196185, 0.174459502324,
  0.169200699492, 0.1639876086, 0.158820075195, 0.153697969964,
  0.148621189348, 0.143589656295, 0.138603321143, 0.133662162669,
  0.128766189309, 0.123915440582, 0.119109988745, 0.114349940703,
  0.10963544023, 0.104966670533, 0.100343857232, 0.0957672718266,
  0.0912372357329, 0.0867541250127, 0.082318375932, 0.0779304915295,
  0.0735910494266, 0.0693007111742, 0.065060233529, 0.0608704821745,
```

```
  0.056732448584, 0.05264727098, 0.0486162607163, 0.0446409359769,
  0.0407230655415, 0.0368647267386, 0.0330683839378, 0.0293369977411,
  0.0256741818288, 0.0220844372634, 0.0185735200577, 0.0151490552854,
  0.0118216532614, 0.00860719483079, 0.00553245272614, 0.00265435214565
};

// tabulated values for 2^24 times x[i]/x[i+1],
// used to accept for U*x[i+1]<=x[i] without any floating point operations
static const unsigned long ktab[128] = {
  0, 12590644, 14272653, 14988939,
  15384584, 15635009, 15807561, 15933577,
  16029594, 16105155, 16166147, 16216399,
  16258508, 16294295, 16325078, 16351831,
  16375291, 16396026, 16414479, 16431002,
  16445880, 16459343, 16471578, 16482744,
  16492970, 16502368, 16511031, 16519039,
  16526459, 16533352, 16539769, 16545755,
  16551348, 16556584, 16561493, 16566101,
  16570433, 16574511, 16578353, 16581977,
  16585398, 16588629, 16591685, 16594575,
  16597311, 16599901, 16602354, 16604679,
  16606881, 16608968, 16610945, 16612818,
  16614592, 16616272, 16617861, 16619363,
  16620782, 16622121, 16623383, 16624570,
  16625685, 16626730, 16627708, 16628619,
  16629465, 16630248, 16630969, 16631628,
  16632228, 16632768, 16633248, 16633671,
  16634034, 16634340, 16634586, 16634774,
  16634903, 16634972, 16634980, 16634926,
  16634810, 16634628, 16634381, 16634066,
  16633680, 16633222, 16632688, 16632075,
  16631380, 16630598, 16629726, 16628757,
  16627686, 16626507, 16625212, 16623794,
  16622243, 16620548, 16618698, 16616679,
```

```
  16614476, 16612071, 16609444, 16606571,
  16603425, 16599973, 16596178, 16591995,
  16587369, 16582237, 16576520, 16570120,
  16562917, 16554758, 16545450, 16534739,
  16522287, 16507638, 16490152, 16468907,
  16442518, 16408804, 16364095, 16301683,
  16207738, 16047994, 15704248, 15472926
};

// tabulated values of 2^{-24}*x[i]
static const double wtab[128] = {
  1.62318314817e-08, 2.16291505214e-08, 2.54246305087e-08, 2.84579525938e-08,
  3.10340022482e-08, 3.33011726243e-08, 3.53439060345e-08, 3.72152672658e-08,
  3.8950989572e-08, 4.05763964764e-08, 4.21101548915e-08, 4.35664624904e-08,
  4.49563968336e-08, 4.62887864029e-08, 4.75707945735e-08, 4.88083237257e-08,
  5.00063025384e-08, 5.11688950428e-08, 5.22996558616e-08, 5.34016475624e-08,
  5.44775307871e-08, 5.55296344581e-08, 5.65600111659e-08, 5.75704813695e-08,
  5.85626690412e-08, 5.95380306862e-08, 6.04978791776e-08, 6.14434034901e-08,
  6.23756851626e-08, 6.32957121259e-08, 6.42043903937e-08, 6.51025540077e-08,
  6.59909735447e-08, 6.68703634341e-08, 6.77413882848e-08, 6.8604668381e-08,
  6.94607844804e-08, 7.03102820203e-08, 7.11536748229e-08, 7.1991448372e-08,
  7.2824062723e-08, 7.36519550992e-08, 7.44755422158e-08, 7.52952223703e-08,
  7.61113773308e-08, 7.69243740467e-08, 7.77345662086e-08, 7.85422956743e-08,
  7.93478937793e-08, 8.01516825471e-08, 8.09539758128e-08, 8.17550802699e-08,
  8.25552964535e-08, 8.33549196661e-08, 8.41542408569e-08, 8.49535474601e-08,
  8.57531242006e-08, 8.65532538723e-08, 8.73542180955e-08, 8.8156298059e-08,
  8.89597752521e-08, 8.97649321908e-08, 9.05720531451e-08, 9.138142487e-08,
  9.21933373471e-08, 9.30080845407e-08, 9.38259651738e-08, 9.46472835298e-08,
  9.54723502847e-08, 9.63014833769e-08, 9.71350089201e-08, 9.79732621669e-08,
  9.88165885297e-08, 9.96653446693e-08, 1.00519899658e-07, 1.0138063623e-07,
  1.02247952126e-07, 1.03122261554e-07, 1.04003996769e-07, 1.04893609795e-07,
  1.05791574313e-07, 1.06698387725e-07, 1.07614573423e-07, 1.08540683296e-07,
  1.09477300508e-07, 1.1042504257e-07, 1.11384564771e-07, 1.12356564007e-07,
  1.13341783071e-07, 1.14341015475e-07, 1.15355110887e-07, 1.16384981291e-07,
```

```
  1.17431607977e-07, 1.18496049514e-07, 1.19579450872e-07, 1.20683053909e-07,
  1.21808209468e-07, 1.2295639141e-07, 1.24129212952e-07, 1.25328445797e-07,
  1.26556042658e-07, 1.27814163916e-07, 1.29105209375e-07, 1.30431856341e-07,
  1.31797105598e-07, 1.3320433736e-07, 1.34657379914e-07, 1.36160594606e-07,
  1.37718982103e-07, 1.39338316679e-07, 1.41025317971e-07, 1.42787873535e-07,
  1.44635331499e-07, 1.4657889173e-07, 1.48632138436e-07, 1.50811780719e-07,
  1.53138707402e-07, 1.55639532047e-07, 1.58348931426e-07, 1.61313325908e-07,
  1.64596952856e-07, 1.68292495203e-07, 1.72541128694e-07, 1.77574279496e-07,
  1.83813550477e-07, 1.92166040885e-07, 2.05295471952e-07, 2.22600839893e-07
};
```

End File: Particles2.cpp

Begin File: PAgS2.cpp

```
/*
      Particle Aggregation Simulator
      Jason C. Myers

============================================================================================
Notes:
  + Sides of particles are labelled clockwise, using binary values to represent each side.
    Directional vectors use the same notation.  The GetNeighbors() function returns a byte
that flags each side as occupied (true) or empty (false).  So a square particle might
return 00001001, meaning that side 8 and side 1 have neighbors.

          ___1__                              __1__
        |       |                       32  /     \  2
      8 |       | 2                        /       \
        |_______|                          \       /
            4                           16  \_____/  4
                                              8

  + Sticking probabilities are also tracked bitwise

  + It's also interesting to note that the tiling of hexagons produces a grid with the
      other orientation of hexagon.  So, with hex particles as shown above, the hexagonal
      scope is rotated 30 degrees, with points up and down and flats left and right.
      Fortunately, the scope doesn't need to be referenced much.


*/


/*
============================================================================================
```

```
                                                        Headers/Resources
================================================================================================
*/

#include <iostream>                     // input/output stream functions
#include <fstream>                      // file manipulation functions
#include <string>                   // string functions, used mostly for parsing the input/output files
#include <sstream>                      // stringsteams.  Also used for file parsing, type conversion,
etc.
#include "particles2.h"             // the Particle, GridSpace, and GridArray objects
using namespace std;

/*
================================================================================================
                                                        Global Variables
                                                        Function Prototypes
================================================================================================
*/

// Function declarations

int StartUp(ParticleArray* &PArray, GridArray* &GArray, string RFileName);          // Initialize
simulation, returns number of particles to simulate
int ShutDown(ParticleArray* &PArray, GridArray* &GArray);                                  // Terminate the
simulation
int WriteData(ParticleArray*, GridArray* GArray, string WFileName);                        // Write particle
data to a file
string ParseLine(ifstream& ReadFile);           // parses data from the input file
stringstream ParseTab(const string& InStr, int index = 1);  // returns a stringstream value[index] from tab
delimited data
int ReadData(ParticleArray*& PArray, GridArray *& GArray, string RFileName);

// functions to perform the actual simulations
void ScatterSim(ParticleArray* &PArray, GridArray* &GArray);
```

```
void AssembleSim(ParticleArray* &PArray, GridArray* &GArray, bool Seed = true);
void ScatterAssemble(int ScatterNum, ParticleArray* &PArray, GridArray* &GArray);
void AssembleSimUnstick(ParticleArray* &PArray, GridArray* &GArray, bool Seed = true);
void ScatterSimUnstick(ParticleArray* &PArray, GridArray* &GArray);



// global variables (there shouldn't be any of these when all is said and done)
string mystring;                                        // Used to grab most user input
string ParamFile;                                       // file name
string OutFile;                                               // file name for output
bool batchmode = false;                                 // is the program in batch mode

/*
=============================================================================================
                                                        Global Code and Functions
=============================================================================================
*/



// Main executable code
int main (int argc, char *argv[]) {

      ParticleArray* Particles;                                 // holds the particles
      GridArray* Grid;                                          // the grid

      if (argc > 1) {
            cout << "Using filename: " << argv[1] << endl;
            ParamFile = argv[1];
            if (argc > 2) {
                  cout << "Outputting to file: " << argv[2] << endl;
                  OutFile = argv[2];
                  batchmode = true;
```

```
            }
            else OutFile = ParamFile;
      }

      else {
            // This part is all about getting input
            cout << "Enter filename with parameters: ";
            cin >> ParamFile;                                    // read parameter file from the input
stream
            OutFile = ParamFile;
      }

      StartUp(Particles, Grid, ParamFile + ".agg");   // read input, initialize memory/objects

      //ScatterSim(Particles,Grid);
      AssembleSim(Particles,Grid);
      //ScatterAssemble(100,Particles,Grid);

      /* Snapshots
            if ((i % 1000) == 0) {
                  stringstream tempstring2;
                  tempstring2  << ParamFile << "_snap" << i << ".out";
                  tempstring2 >> tempstring;
                  cout << "Writing file: " << tempstring << endl;
                  WriteData(Particles, Grid, tempstring);
            }*/

      cout << "Press any key to write data\n";
      WriteData(Particles, Grid, OutFile + ".out");
      if (!batchmode) cin >> mystring;

ProgEnd:
      cout << "\nJob complete.  Shutting down.\n";
      ShutDown(Particles, Grid);                                 // free allocated memory
```

```
	cout << "\nShutdown Complete. \n";

	if (!batchmode) cin >> mystring;
	return 0;
};

void ScatterSim(ParticleArray* &Particles, GridArray* &Grid) {
	// This is the part that actually performs the simulation
	string tempstring;
	AggregateArray Aggs(2000);                                    // hold the aggregates
	Aggs.SetPAPtr(Particles);                                     // Connect the aggregates to the grid

	int px, py, po;

	for (int i = 0; i < Particles->NumParticles(); i++) {
		Particles->DropRandom(i,false);                  // randomly scatter particles on the grid
	}


	int JoinCount = 0;                                            // monitor the number of attachments

	bool NewAdd = true;

	WriteData(Particles, Grid, "default_snap0.out");

	while (JoinCount < Particles->NumParticles()*.9) {
		if (NewAdd) {
			cout << "Mapping now\n";
			Aggs.MapAllAggregates();                      // sort everything by aggregate
			for (int i = 0; i < Aggs.UsedAggs(); i++)
				Aggs.GaussCalc(i);
			cout << "Aggregates count: " << Aggs.UsedAggs() << "\nJoinCount: " << JoinCount << endl;
			NewAdd = false;
			Particles->Output(0,px,py);
```

```
                  //cout << "Particle 0 at (" << px << ", " << py << ")\n";
                  //cout << "Any key to continue:";
                  cout << "\n\n";
            }

            for (int i = 0; i < Aggs.UsedAggs(); i++) {
                  if (Aggs.AggSize(i) > 0) {
                        Aggs.GaussMove(i);
                        if (Aggs.HasNeighbors(i)) {
                              NewAdd = Aggs.CheckStick(i);
                              if (NewAdd) {
                                    JoinCount++;
                                    Aggs.MapAllAggregates();
                                    /*          // snapshots
                                    if ((JoinCount % 200) == 0) {
                                          stringstream tempstring2;
                                          tempstring2  << ParamFile << "_snap" << JoinCount << ".out";
                                          tempstring2 >> tempstring;
                                          cout << "Writing file: " << tempstring << endl;
                                          WriteData(Particles, Grid, tempstring);
                                    }      */ // end snapshot part
                              }     //  if (NewAdd)
                        }     //  if HasNeighbors
                  }     //  if AggSize > 0
            } //  for loop (each aggregate)

      } // while loop (JoinCount)
}

void AssembleSim(ParticleArray* &Particles, GridArray* &Grid, bool Seed) {
      int px, py, po;
      AggregateArray Aggs(2);
      Aggs.SetPAPtr(Particles);
      string tempstring;
```

```
if (Seed) {
      Particles->Seed(0, Grid->Center(), Grid->Center());   // set the first particle in the center
      Aggs.AddParticle(0,0);
      Particles->Output(0,px,py);
      po = Particles->Orient(0,false);
      cout << "Particle 0: (" << px << ", " << py << "), " << po << endl;
}
else Aggs.MapAllAggregates();

for (int i = 1; i < Particles->NumParticles(); i++) {
      if (Particles->GetFixed(i)) continue;
      Particles->DropRandom(i);
      Aggs.Clear(1);
      Aggs.AddParticle(1,i);
      Aggs.GaussCalc(1);
      while (Particles->GetAggRef(i) == 1) {
            Aggs.GaussMove(1);
            if (Aggs.HasNeighbors(1)) {
                  Particles->Output(i,px,py);
                  //cout << "Stick check at (" << px << ", " << py << ")\n";
                  if (Aggs.CheckStick(1))
                        Aggs.AddParticle(0,i);
            }
      }
      Particles->Output(i,px,py);
      po = Particles->Orient(i, false);
      Grid->UpdateScope(px, py);

      cout << "Particle " << i << ": (" << px << ", " << py << "), " << po << endl;

      // Snapshots
      /*
      if ((i % 200) == 0) {
```

```
                  stringstream tempstring2;
                  tempstring2  << ParamFile << "_snap" << i << ".out";
                  tempstring2 >> tempstring;
                  cout << "Writing file: " << tempstring << endl;
                  WriteData(Particles, Grid, tempstring);
            } */
      }
}


// scatter a given number of particle seeds, then run assembly
void ScatterAssemble(int ScatterNum, ParticleArray* &PArray, GridArray* &GArray) {
      if (ScatterNum > PArray->NumParticles()) return;
      for (int i = 0; i < ScatterNum; i++) {
            PArray->DropRandom(i, false);
            PArray->SetFixed(i,true);
            int px, py;
            PArray->Output(i,px,py);
            cout << i << " at (" << px << ", " << py << ")\n";
      };
      AssembleSim(PArray, GArray, false);
}


void AssembleSimUnstick(ParticleArray* &Particles, GridArray* &Grid, bool Seed) {
      int px, py, po;
      AggregateArray Aggs(2);
      Aggs.SetPAPtr(Particles);
      string tempstring;

      if (Seed) {
            Particles->Seed(0, Grid->Center(), Grid->Center());   // set the first particle in the center
            Aggs.AddParticle(0,0);
            Particles->Output(0,px,py);
```

```
		po = Particles->Orient(0,false);
		cout << "Particle 0: (" << px << ", " << py << "), " << po << endl;
	}
	else Aggs.MapAllAggregates();

	for (int i = 1; i < Particles->NumParticles(); i++) {
		if (Particles->GetFixed(i)) continue;
		Particles->DropRandom(i);
		Aggs.Clear(1);
		Aggs.AddParticle(1,i);
		Aggs.GaussCalc(1);

		while (Particles->GetAggRef(i) == 1) {
			Aggs.GaussMove(1);
			if (Aggs.HasNeighbors(1)) {
				Particles->Output(i,px,py);
				//cout << "Stick check at (" << px << ", " << py << ")\n";
				if (Aggs.CheckStick(1))
					Aggs.AddParticle(0,i);
			}
		}
		Particles->Output(i,px,py);
		po = Particles->Orient(i, false);
		Grid->UpdateScope(px, py);

		cout << "Particle " << i << ": (" << px << ", " << py << "), " << po << endl;

		// Snapshots
		/*
		if ((i % 200) == 0) {
			stringstream tempstring2;
			tempstring2  << ParamFile << "_snap" << i << ".out";
			tempstring2 >> tempstring;
			cout << "Writing file: " << tempstring << endl;
```

```
                WriteData(Particles, Grid, tempstring);
        } */

        // check for unsticking
        for (int j = 0; j < i; j++) {
                if (Particles->CheckUnstick(j)) {
                        Particles->Output(j,px,py);
                        //cout << "Particle " << j << " UNSTUCK from (" << px << ", " << py << ")\n";
                        Particles->Unattach(j,~0);    // break all attachments
                        //cout << "Agg 1 has " << Aggs.AggSize(1) << " particles\n";
                        cout << "Mapping now\n";
                        Aggs.MapAllAggregates();                        // sort everything by aggregate
                        for (int a = 0; a < Aggs.UsedAggs(); a++)
                                Aggs.GaussCalc(a);

                        bool AggsFree = false;
                        do {
                                for (int a = 0; a < Aggs.UsedAggs(); a++) {
                                        if (!Aggs.GetFixed(a))  // move all non-fixed aggregates
                                                Aggs.GaussMove(a);
                                        if (Aggs.HasNeighbors(a)) {   // if the aggregate has a neighbor
                                                if(PercentCheck(Particles->BaseStick)) {
                                                        for (int b = 0; b < Aggs.UsedAggs(); b++)
                                                                if (Aggs.AreNeighbors(a,b))   // check for
neighboring aggregates
                                                                        Aggs.AttachAggs(a,b);   // attach them
                                                        Aggs.MapAllAggregates();      // remap the aggregates
                                                }
                                        }     // end if HasNeighbors
                                }     // end for each aggregate

                                AggsFree = false;
                                for (int a = 0; a < Aggs.UsedAggs(); a++)
                                        AggsFree = AggsFree || (!Aggs.GetFixed(a));
```

```
                        } while (AggsFree);
                  }     // end if(CheckUnstick)
            }     // end of unsticking portion
      }     // end of for each particle loop
}


void ScatterSimUnstick(ParticleArray* &Particles, GridArray* &Grid) {
      // This is the part that actually performs the simulation
      string tempstring;
      AggregateArray Aggs(2000);                                    // hold the aggregates
      Aggs.SetPAPtr(Particles);                                     // Connect the aggregates to the grid

      int px, py, po;

      for (int i = 0; i < Particles->NumParticles(); i++) {
            Particles->DropRandom(i,false);                 // randomly scatter particles on the grid
      }


      int JoinCount = 0;                                            // monitor the number of attachments

      bool NewAdd = true;

      WriteData(Particles, Grid, "default_snap0.out");

      while (JoinCount < Particles->NumParticles()*.9) {
            if (NewAdd) {
                  cout << "Mapping now\n";
                  Aggs.MapAllAggregates();                          // sort everything by aggregate
                  for (int i = 0; i < Aggs.UsedAggs(); i++)
                        Aggs.GaussCalc(i);
                  cout << "Aggregates count: " << Aggs.UsedAggs() << "\nJoinCount: " << JoinCount << endl;
                  NewAdd = false;
```

```
                Particles->Output(0,px,py);
                //cout << "Particle 0 at (" << px << ", " << py << ")\n";
                //cout << "Any key to continue:";
                cout << "\n\n";
          }

          for (int i = 0; i < Aggs.UsedAggs(); i++) {
                if (Aggs.AggSize(i) > 0) {
                      Aggs.GaussMove(i);
                      if (Aggs.HasNeighbors(i)) {
                            NewAdd = Aggs.CheckStick(i);
                            if (NewAdd) {
                                  JoinCount++;
                                  Aggs.MapAllAggregates();
                                  /*          // snapshots
                                  if ((JoinCount % 200) == 0) {
                                        stringstream tempstring2;
                                        tempstring2  << ParamFile << "_snap" << JoinCount << ".out";
                                        tempstring2 >> tempstring;
                                        cout << "Writing file: " << tempstring << endl;
                                        WriteData(Particles, Grid, tempstring);
                                  }      */ // end snapshot part
                            }     //  if (NewAdd)
                      }     //  if HasNeighbors
                }     //  if AggSize > 0
          } //  for loop (each aggregate)

     } // while loop (JoinCount)
}


// (eventually) Read data from the experiment file, initialize dynamic arrays
int StartUp(ParticleArray* &PArray, GridArray* &GArray, string RFileName) {
```

```
      ifstream ReadFile;                                                    // File handle for the
parameter file
      string TempString;                                                    // Used to store stuff
read from the file

      // these variables store data to be read from the file before storing in Particle/GridArray objects
      int NumParticles (400);                                             // number of particles to
simulate
      int GridSize (600);                                                   // maximum size of the
grid
      int NumFaces (4);                                                   // number of faces for
particles
      float DefaultStickChance (70);                                      // used if not replaced by
specific Sticking probabilities
      int RotateChance (0);                                               // chance to rotate when
moving
      bool RandomStartOrientation (0);                                  // true or false only.  If false,
all start in SeedOrientation
      int UseScope (0);                                                   // Scope function: 0 = fixed,
1 = isotropic, 2 = anisotropic
      int ScopeRange (400);                                               // Range of the scope in each
direction
      float DefaultUnstickChance (0);                                     // chance of particles
leaving the aggregate
      int StickType (0);                                                    // the type of particle
sticking
      float StickP1 (0), StickP2 (0), StickP3 (0),
            StickP4 (0), StickP5 (0);                                     // parameters for
initializing sticking

      // As usual, you're gonna need error checking
      ReadFile.open(RFileName.c_str());                                 // open the file for reading
      if (!ReadFile.is_open()) {
```

```
            cout << "File not found.  Using default parameters" << endl;      // if it's not there, use the
default file
            ReadFile.clear();
            ReadFile.open("default.agg");
      }

      cout << "Geting file parameters\n\n\n";
      // get the particle count
      TempString = ParseLine(ReadFile);
      stringstream(TempString) >> NumParticles;

      // read the shape of the particles
      TempString = ParseLine(ReadFile);
      stringstream(TempString) >> NumFaces;

      // check if rotation is allowed
      TempString = ParseLine(ReadFile);
      if (TempString == "true") RandomStartOrientation = true;
      else if (TempString == "false") RandomStartOrientation = false;

      // get the grid size
      TempString = ParseLine(ReadFile);
      stringstream(TempString) >> GridSize;

      // get the scope details
      TempString = ParseLine(ReadFile);
      stringstream(TempString) >> UseScope;
      TempString = ParseLine(ReadFile);
      stringstream(TempString) >> ScopeRange;

      // get sticking type (not implemented fully)
      TempString = ParseLine(ReadFile);
      if (TempString == "orient") StickType = 0;
      else if (TempString == "crystal") StickType = 1;
```

```
	// initialize the Grid and Particles
	GArray = new GridArray(GridSize, UseScope, ScopeRange);		// declare the grid for the
simulation
	PArray = new ParticleArray(NumParticles, NumFaces, RandomStartOrientation, GArray); // Assign memory
for the particle array
	PArray->SetStickType(StickType);	// set the particle array sticking type

	// assign the sticking characteristics
	if (StickType == 0) {
		if (NumFaces == 4) {
			// orientation squares or crystallographic hexagons: the 3 parameter states
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP1;
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP2;
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP3;
		}
		else if (NumFaces == 6) {
			// orientation hexagons: only 1 parameter
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP1;
		}
	}
	else if (StickType == 1) {
		if (NumFaces == 4) {
			// crystallographic squares: takes 5 parameters
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP1;
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP2;
			TempString = ParseLine(ReadFile);
			stringstream(TempString) >> StickP3;
```

```
                TempString = ParseLine(ReadFile);
                stringstream(TempString) >> StickP4;
                TempString = ParseLine(ReadFile);
                stringstream(TempString) >> StickP5;
        }
        else if (NumFaces == 6) {
                // crystallographic hexagons: takes 4 parameters
                TempString = ParseLine(ReadFile);
                stringstream(TempString) >> StickP1;
                TempString = ParseLine(ReadFile);
                stringstream(TempString) >> StickP2;
                TempString = ParseLine(ReadFile);
                stringstream(TempString) >> StickP3;
                TempString = ParseLine(ReadFile);
                stringstream(TempString) >> StickP4;
        }
}
// use the parameters read to initialize the sticking variables
PArray->InitializeSticking(StickP1, StickP2, StickP3, StickP4, StickP5);

// close the file now that reading is done
ReadFile.close();

// Debugging stuff.
cout << "NumParticles: " << NumParticles << endl;
cout << "GridSize: " << GridSize << endl;
cout << "NumFaces: " << (int) NumFaces << endl;
if (StickType == 0) {
        cout << "StickType: Orientation" << endl;
        if (NumFaces == 4) {
                cout << "\tA-A Stick Chance: " << StickP1 << endl;
                cout << "\tB-B Stick Chance: " << StickP2 << endl;
                cout << "\tA-B Stick Chance: " << StickP3 << endl;
        }
```

```
            else if (NumFaces == 6)
                  cout << "Stick Chance: " << StickP1 << endl;
      }
      else if (StickType == 1) {
            cout << "StickType: " << "Crystallographic" << endl;
            if (NumFaces == 4) {
                  cout <<"\tSingle: " << StickP1 << endl;
                  cout << "\t(1 0) Suface: " << StickP2 << endl;
                  cout << "\t(0 1) Suface: " << StickP3 << endl;
                  cout << "\t(1 1) Suface: " << StickP4 << endl;
                  cout << "\tVoid: " << StickP5 << endl;
            }
            else if (NumFaces == 6) {
                  cout << "\tSingle: " << StickP1 << endl;
                  cout << "\t(1 -2) Surface: " << StickP2 << endl;
                  cout << "\t(0 1) Surface: " << StickP3 << endl;
                  cout << "\tVoid: " << StickP4 << endl;
            }
      }

      cout << "Rotation: " << ((RandomStartOrientation == 0) ? "Forbidden" : "Allowed") << endl;
      cout << "ScopeType: ";
      if (UseScope == 0) cout << "Fixed\n";
      if (UseScope == 1) cout << "Equal expansion\n";
      if (UseScope == 2) cout << "Unequal expansion\n";
      cout << "ScopeRange: " << ScopeRange << endl;
      if (!batchmode) cin >> mystring;

      return 0;                                                    // returns 0 if there are no errors

      // Add in error checking for this part, since it will involve both memory and file allocation
};
```

```
// Free memory, shutdown
int ShutDown(ParticleArray* &PArray, GridArray* &GArray) {

      delete GArray;                                                    // free memory from the grid
      GArray = NULL;                                                    // reset pointer to empty

      delete PArray;                                                    // same for the particles
      PArray = NULL;

      return 0;                                                         // 0 for all is well
};


// Writes particle data to the output file when simulation is complete.  Can also be used
// to take "snapshots" of the structure with time
int WriteData(ParticleArray* PArray, GridArray* GArray, string WFileName) {
      ofstream SaveFile;                                                        // file to store
particle positions
      int px, py;                                                               // holds particle x, y
positions

      SaveFile.open(WFileName.c_str());                                 // open output file for writing

      // output standard position information for square particles
      if (PArray->Faces() == 4) {
            SaveFile << "Particle # \t x \t y \t orient \t AggRef" << endl;   // header for particle
positions

            for (int i = 0; i < PArray->NumParticles(); i++) {    // output all fixed particle positions to
file
                  PArray->Output(i,px,py);
                  if (px > -1)
                        SaveFile << i << "\t" << px << "\t" << py << "\t" << ( ((PArray->Orient(i,false) %
2) == 0) ? "2" : "1")  <<"\t" << PArray->GetAggRef(i) << endl;
```

```
            };
      }

      // output for hexagonal files.  This involves an additional coordinate transformation
      else if (PArray->Faces() == 6) {
            SaveFile << "Particle # \t x \t y \t orientation \tAggRef \t \t hex x \t hex y" << endl;  //
header for particle positions

            for (int i = 0; i < PArray->NumParticles(); i++) {                              // output all
fixed particle positions to file
                  PArray->Output(i,px,py);
                  if (px > -1)
                        SaveFile << i << "\t" << (0.8660254*px) << "\t" << (py + .5*px) << "\t" << PArray-
>Orient(i,false)
                        << "\t" << PArray->GetAggRef(i) << "\t \t" << px << "\t" << py << endl;
            };
      }

      // now we output the simulation parameters to a nice table at the bottom
      SaveFile << "\n\n";
      SaveFile << "*** SIMULATION PARAMETERS: ***\n";
      SaveFile << "NumParticles: " << PArray->NumParticles() << endl;
      SaveFile << "GridSize: " << GArray->Size() << endl;
      SaveFile << "NumFaces: " << (int) PArray->Faces() << "\n\n";
      if(PArray->GetStickType() == 0) {
            SaveFile << "StickType: " << "Orientation" << endl;
            if (PArray->Faces() == 4) {
                  SaveFile << "A-A Stick Chance: " << PArray->GetStickValue(0) << endl;
                  SaveFile << "B-B Stick Chance: " << PArray->GetStickValue(1) << endl;
                  SaveFile << "A-B Stick Chance: " << PArray->GetStickValue(2) << endl;
            }
            else if (PArray->Faces() == 6)
                  SaveFile << "Stick Chance: " << PArray->GetStickValue(0) << endl;
      }
```

```
	else if (PArray->GetStickType() == 1) {
		SaveFile << "StickType: Crystallographic" << endl;
		if (PArray->Faces() == 4) {
			SaveFile <<"Single: " << PArray->GetStickValue(0) << endl;
			SaveFile << "(1 0) Suface: " << PArray->GetStickValue(1) << endl;
			SaveFile << "(0 1) Suface: " << PArray->GetStickValue(2) << endl;
			SaveFile << "(1 1) Suface: " << PArray->GetStickValue(3) << endl;
			SaveFile << "Void: " << PArray->GetStickValue(4) << endl;
		}
		else if (PArray->Faces() == 6) {
			SaveFile << "Single: " << PArray->GetStickValue(0) << endl;
			SaveFile << "(1 -2) Surface: " << PArray->GetStickValue(1) << endl;
			SaveFile << "(0 1) Surface: " << PArray->GetStickValue(2) << endl;
			SaveFile << "Void: " << PArray->GetStickValue(3) << endl;
		}
	}
	SaveFile << endl;
	SaveFile << "Rotation: " << (PArray->AllowRot() ? "Allowed" : "Forbidden") << endl;
	SaveFile << "ScopeType: ";
	if (GArray->GetScopeType() == 0) SaveFile << "Full grid\n";
	else {
		if (GArray->GetScopeType() == 1) SaveFile << "Equal expanding\n";
		if (GArray->GetScopeType() == 2) SaveFile << "Unequal expanding\n";
		SaveFile << "ScopeRange: " << GArray->GetScopeRange() << endl;
	}

	// all done, close the file
	SaveFile.close();
	return 0;                                                    // as usual, return of
0 equals good
}

// reads the contents of a particle output file into a particle array
// Not fully implemented: could use a lot of clean up work
```

```
int ReadData(ParticleArray*& PArray, GridArray*& GArray, string RFileName) {
      ifstream ReadFile;
      string TS = "";
      string TS2 = "";
      stringstream SS;
      int PNum = -1;
      int px, py, po;

      ReadFile.open(RFileName.c_str());
      getline(ReadFile,TS);   // this is the header line, discard it
      //ParseTab(TS,7) >> TS; // check for the hex column info
      //if (TS = "\n")
      //    FaceCount = 4;
      //else FaceCount = 6;

      getline(ReadFile,TS);   // read the first real line

      while ((!ReadFile.eof()) && (TS != "")) {
            ParseTab(TS,1) >> PNum;
            if (PArray->Faces() == 4) {               // if square particles
                  ParseTab(TS,2) >> px;
                  ParseTab(TS,3) >> py;
            }
            else if (PArray->Faces() == 6) {          // if hex, read the hex columns
                  ParseTab(TS,7) >> px;
                  ParseTab(TS,8) >> py;
            }
            ParseTab(TS,4) >> po;

            PArray->GhostTo(PNum,px,py);
            GArray->MarkP(PNum,px,py);
            PArray->SetOrient(PNum,(byte)po);

            getline(ReadFile,TS);         // read the next line
```

```
	}
	return 1;
}


// functions to parse input file data.  Removes comments, blank lines, etc.
string ParseLine(ifstream& ReadFile) {
	string StrIn = "";
	string TempString = "";

	while ((TempString == "") && (! ReadFile.eof())) {
		getline(ReadFile, StrIn);
		for (unsigned int i = 0; i < StrIn.length(); i++) {
			if (StrIn[i] == '#')	// a comment marker has been hit, end parsing this line.
				break;
			else if ((StrIn[i] != ' ') && (StrIn[i] != '\t'))	// ignore white space, otherwise
move on
				TempString += StrIn[i];	// add valid characters to TempString;
		}
	}	// end of while.  Repeats if line was blank (as long as the end of the file has not been
reached)

	return TempString;				// return the string value
}

stringstream ParseTab(const string& InStr, int index) {
	int SIndex = 0;
	string TS2 = "";
	for (int i = 0; i < index; i++) {	// repeat until the desired value is reached
		TS2 = "";
		while (InStr[SIndex] != '\t'){	// sort through the data until a tab is reached
			if (InStr[SIndex] == '\n') {	// if eoline reached, return \n character
				TS2 = "\n";
				return stringstream(TS2);
```

```
                    }

                    TS2 += InStr[SIndex];
                    SIndex++;
                }
        }
        return stringstream(TS2);
}
```

End File: PAgS2.cpp

Begin File: Template.agg

```
# Particle Variables
      # number of particles to use in the simulation
      # number of faces per particle (4 or 6)
      # allow rotation ("true" or "false")

# Grid Variables
      # size of defined grid
      # scopetype (0 = fixed, 1 = isotropic, 2 = anisotropic)
      # scoperange (if scopetype not = 0)

# Sticking Probability Info
      # type of sticking ("orient" or "crystal")
```

```
      # sticking type parameters (see below)
      # ...
      # ...

####################################
# Orientation based sticking check:
# All three variables used for square, only A-A used for hex
#
# orient
# A-A sticking chance
# B-B sticking chance
# A-B sticking chance
####################################
# Crystallographic based sticking check:
# Five definitions for square, only four for hex
#
# (square)
# crystal
# single unit energy
# (1 0) surface energy
# (0 1) surface energy
# (1 1) surface energy
# void energy
#
# (hex)
# crystal
# single unit energy
# (1 -2) surface energy
# (0 1) surface energy
# void energy
####################################
```

End File: Template.agg