Growing Fitted Textures

Gabriele Gorla
Electrical and Computer Engineering
University of Minnesota

Victoria Interrante
Computer Science and Engineering
University of Minnesota

Guillermo Sapiro
Electrical and Computer Engineering
University of Minnesota

Abstract

In this paper, we address the problem of how to seamlessly and without repetition artifacts or visible projective distortion cover the surface of a polygonally-defined model with a texture pattern derived from an acquired 2D image such that the dominant orientation of the pattern will everywhere follow the surface shape in an aesthetically pleasing way. Specifically, we propose an efficient, automatic method for synthesizing, from a small sample swatch, patches of perceptually similar texture in which the pattern orientation may locally follow a specified vector field, such as the principal directions of curvature, at a per-pixel level, and in which the continuity of large and small scale features of the pattern is generally preserved across adjacent patches. We demonstrate the results of our method with a variety of texture swatches applied to standard graphics datasets.

CR categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism — Texture.

Keywords: Texture synthesis, texture mapping, shape representation.

1. Introduction

Adding texture to the surface of a polygonal model can profoundly enhance its visual richness. In addition to contributing abundant detail and complexity at minimal computational expense, texture has the potential — depending upon the characteristics of the pattern, and how it is applied — to affect our perception of an object's geometry, masking faceting artifacts [8] or enhancing our appreciation of the curvature of the form [13]. Given a texture pattern and a surface model, the historical challenge has been to determine how to apply the pattern to the surface in an appropriate manner, minimizing the visual impact of seams and projective distortion while orienting the pattern so that it flows over the shape in a desirable way.

Many different approaches to this basic problem are possible. Our solution focuses on the case in which the desired texture is defined by a provided 2D image. The method that we propose has the advantages of being essentially automatic (requiring no manual intervention), reasonably efficient, fairly straightforward to implement, and applicable across a wide variety of texture types and models. In addition, the resulting textured objects can be easily displayed at interactive frame rates using a conventional renderer on a standard PC with texture mapping hardware.

Our technique consists of the following main steps:

- Partition the polygons of the model into contiguous patches, as nearly planar (to prevent distortion) and as nearly similarly sized (to reduce texture memory requirements) as reasonably possible.
- Compute a vector field over the object, or read a pre-defined field from a file.
- Synthesize the texture pattern over each patch, using an efficient, orientation-adaptive variation of the non-parametric sampling method proposed by Efros and Leung [7].

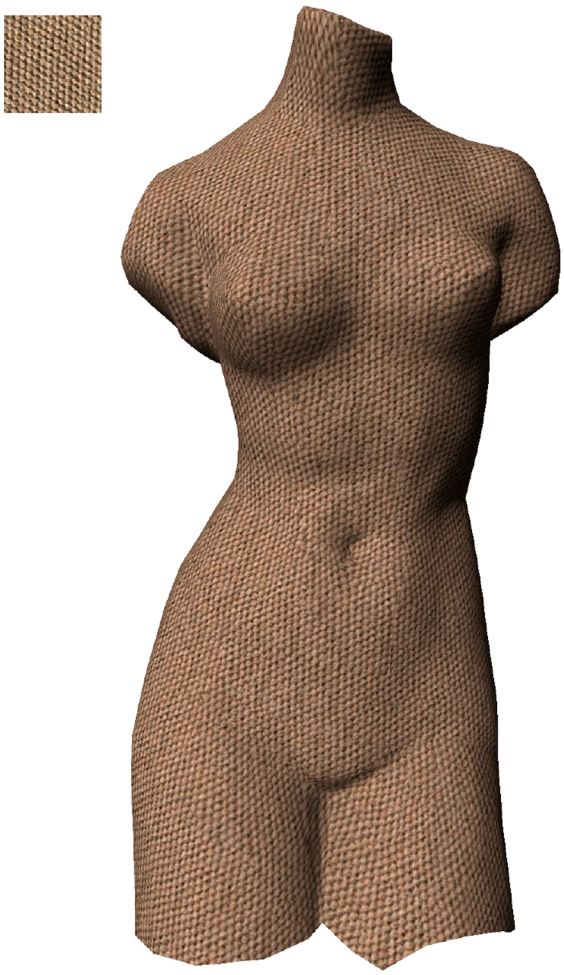

Figure 1: An example of a synthesized surface texture produced by our method. No manual intervention of any kind was employed. This texture was grown from an original 92x92 swatch [4] rotated to 63 orientations and cropped to 64x64 samples, to cover 291 surface patches at 128x128 resolution following a vector field locally defined by the projection of (0,1,0) onto the tangent plane at each point. The entire process required approximately 12 minutes.

2. Previous Work

A variety of methods have been previously proposed for texturing polygonal models with patterns that are as free as possible of seams and distortion artifacts.

One method is solid texturing [19,20,28], in which the texture pattern is defined over a 3D volume. Particularly good results have been achieved with this method for water, and for objects made of wood and stone. However there are significant challenges in synthesizing 3D textures modeled after sampled

materials [12,6], and current methods for creating custom-fitted 3D textures whose features follow a surface's shape [13] are severely limited in scope and applicability.

Methods for applying 2D image-based texture to arbitrary polygonal models for the most part must balance the inherent trade-off between seams and distortion (one cannot in general apply a 2D image to a non-developable surface without incurring one or the other), employing piecewise flattening in the case of arbitrary parametric [2] or polygonally-defined [16] models, or using surface re-parameterization [14], or pre-distortion of the texture [1,27] to achieve desired results in other particular cases. Conformal mapping [10] offers a global solution that preserves angles, but not lengths or areas.

Closer to our objectives, Neyret and Cani [18] proposed an excellent technique for achieving seamless and distortion-free mapping of 2D isotropic texture patterns on arbitrary objects via custom-defined triangular texture tiles that are continuous with one another across various of their boundaries. Unfortunately an extension of this method to anisotropic texture patterns is not obvious. Last year, Praun et al. [23] proposed "lapped textures", which provides capabilities that are the most similar to those towards which our method aspires, although the approach that we take is very different. The lapped texture method, in which copies of a sample texture swatch are repeatedly pasted onto overlapping patches across a surface after some subtle warping and reorientation to align the pattern with a user-defined vector field, produces very good results when used with texture patterns that contain enough high frequency detail and natural irregularity in feature element sizes and relative positions to perceptually mask artifacts due to the partial overlap or misalignment of feature elements across patch boundaries. It is best suited for use with vector fields that do not contain much high frequency variation. As currently formulated, the lapped texture approach appears less well-suited for more rigidly structured patterns, such as a checkerboard, or textures such as netting which are characterized by the global continuity of specific elongated elements.

The method that we describe in this paper provides capabilities beyond those offered by previous methods. It achieves nearly seamless and distortion-free texturing of arbitrary polygonally-defined models with a texture pattern derived from a provided sample, is suitable for use with anisotropic patterns, generally preserves larger scale texture pattern continuity across patch boundaries, does not require manual user intervention, and allows the orientation of the applied pattern to locally follow a specified vector field on close to a per-pixel basis.

Our method falls into the category of methods that achieve texture pattern continuity without distortion by in effect synthesizing the texture "in place" over the surface of the object. Previous methods in this category include direct painting [11], and reaction-diffusion texture synthesis [25], which yield excellent results for hand-crafted textures and textures modeled after organic processes. We want to achieve similarly good results with automatically synthesized textures that are perceptually equivalent to a given sample swatch.

Our work is perhaps most fundamentally motivated by the impressive advances in texture synthesis methods [12,5,22,29,7,26] which make it possible to create, for an increasingly wide range of patterns, unlimited quantities of a texture that is perceptually equivalent to a small provided sample.

Leveraging research in human texture perception, Heeger and Bergen [12] developed a highly successful method for synthesizing textures that capture the essential perceptual properties of a variety of homogeneous stochastic sample patterns. Their method works by modifying a random noise image so that its intensity histogram matches the histogram of the sample texture across each of the subbands in a steerable pyramid representation of each image. De Bonet [5] developed a related method based on swapping elements in the Laplacian pyramid representation of a self-tiling pattern where possible while preserving the joint-occurrence relationships of features across multiple resolutions. This method yields impressive results for an even wider variety of patterns, though some difficulties remain in preserving larger scale globally significant structure. Several other highly sophisticated statistically based approaches have been subsequently proposed for texture synthesis [cf. 22,29], however none of these methods, while quite inspiring, is exactly well-suited for our needs.

Rather we follow the statistical texture synthesis approach recently proposed by Efros and Leung [7], in which a new texture pattern is grown, pixel-by-pixel, by sampling into the provided template pattern and choosing randomly from among the pixels whose neighborhoods are close matches to the yet partially-defined neighborhood of the pixel to be filled in, in the pattern being synthesized. A serious drawback to an implementation based directly upon this method however is speed, as addressed last year by Wei and Levoy [26] who proposed a more efficient approach to the problem of finding the set of good matches from among all of the possible neighborhoods based on tree-structured vector quantization. Wei and Levoy in fact suggest as a direction for future work the possibility of extending their method to synthesize texture over surface meshes. However the speed advantage of their method is obtained via the imposition of constraints on the configuration of the causal neighborhood (*a priori* knowledge of the number and locations of the pixels at which the texture has already been synthesized) which creates significant complications for such an extension.

In the remainder of this paper we describe the method that we have developed and the details of its implementation, talk about some of the issues in automatically determining a good way to orient the texture over the surface, show representative results, and conclude with a discussion of the current limitations of our implementation and directions for future work.

3. Proposed Method

The proposed method is basically a two step process. First the surface is partitioned into many small almost flat patches, then the texture is grown over the planar projection of each individual patch taking into consideration the proper boundary conditions to maintain the continuity of the texture pattern across seams at the patch boundaries. During the synthesis of an anisotropic pattern, the texture is locally constrained into alignment with a specified vector field over the surface. For simplicity we store the resulting synthesized texture as a collection of separate small images for each patch, although other approaches are certainly possible.

3.1 Partitioning

The goal of the first stage is to partition the mesh into a minimum number of approximately planar patches. Obviously these are conflicting goals for any closed surface and a suitable tradeoff must be found. In order to keep the implementation and the texture memory management as simple as possible the patches should also be of approximately the same size.

Two input parameters define the maximum patch size (which influences the scale at which the texture appears over the surface) and the maximum projective distortion that the user is willing to tolerate. The initial partitioning is done with a greedy algorithm, after which an optimization step is performed to reduce the average projection error.

The process for the initial partitioning can be summarized by the following pseudo code:

```
while (unassigned_triangles>0) {
   pick an arbitrary unassigned triangle T
   assign T to a new group G
   add to group G all connected triangles C that satisfy:
      - Normal(C) • Normal(T) >min_cosine_displacement
      - distance from the center of C to the farthest vertex
        of T is less than max_dist
}
```

The image on the left side of figure 2 shows a representative result after the first stage in the splitting. The green triangles are the reference triangles that define the plane onto which the patch will ultimately flattened. It is easy to notice that some of the patches obtained at this point are very small and/or contain triangles that are relatively far from being aligned with the reference plane. A refinement pass is used to reduce both the number of patches and the number of triangles that are oriented at a sharp angle to the plane into which they will ultimately be projected. Two simple experimental rules are iteratively applied until no significant improvement is observed:

- remove a patch if it is very small, and its triangles can be added to a neighboring patch without violating the distance constraint;
- reassign a triangle T from patch P1 to a neighboring patch P2 if T borders P2 and is more closely aligned with the reference plane of that patch than with its own.

The image on the right side of figure 2 shows the results after iterative refinement. While this refinement procedure is not theoretically optimal, and is not guaranteed to converge to a stable solution, it is simple and extremely fast (the greedy splitting step takes between 1/2–2s, depending upon the size of the model, and the refinement takes 2-10 seconds) and it produces satisfactory results in most cases. In the rare event that acceptable results are not achieved at this stage, the splitting process can be repeated using a tighter limit on the acceptable normal error (which will result in more, smaller patches), or by specifying a smaller maximum patch size directly. We have found that increasing the number of patches causes only a marginal increase in the computational expense. More significant is the issue that with very small patches there is comes an increased risk that the texture synthesis process will run into difficulties and "grow garbage", either due to the paucity of available contextual information along the shortened boundary, or to the near proximity of mutually incompatible pre-defined boundary conditions.

3.2 Parameterization

After the model has been partitioned into contiguous patches, the triangles comprising each patch are projected onto their common reference plane, and the texture coordinates are defined at each vertex according the coordinates of the projected vertices in the reference plane coordinate system. Adjacent triangles from the neighboring patches, which provide the boundary conditions for maintaining the continuity of the texture pattern during synthesis, are then rotated (rather than projected, to minimize the projective distortion of the reference texture) about their shared edges into the reference plane.

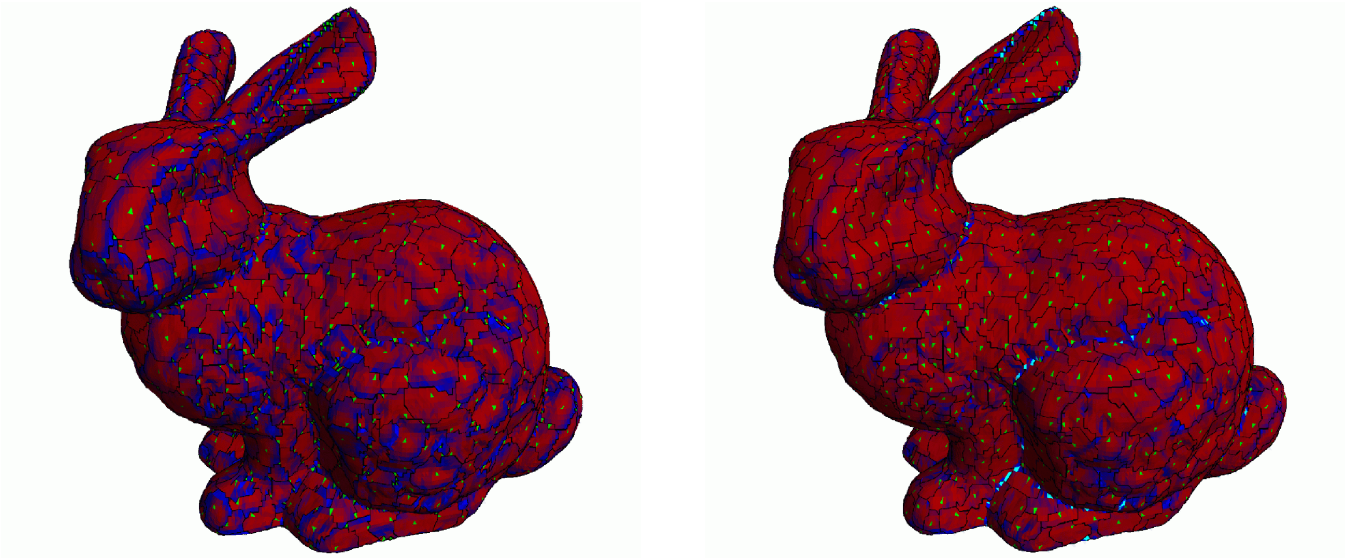

Figure 2: The partitioned bunny, after the first stage of our "greedy" splitting algorithm (left) and after iterative refinement (right), which decreased the number of patches from 988 to 699. Patch boundaries are outlined in black on each model. The green triangles are the seed triangles, which also define the reference plane for each patch. Triangles whose normal directions differ by less than 18 degrees from the direction of the normal to the reference plane, corresponding to a 5% error in the linear projection, are colored red. Triangles up to 25 degrees out of alignment with the reference plane for their patch, corresponding to a 10% error in the projection, are tinted blue. Triangles rotated more than 25 degrees from the reference plane (>10% error) are colored cyan.

3.3 Synthesis

At this point, we have created a 2D image for each patch, containing:

- an area, defined by the projection of the triangles of the patch onto the reference plane, which contains the pixels to be filled by the synthesized texture; and
- an area, defined by the rotation into the reference plane of the neighboring triangles from the adjacent patches, that will hold any previously synthesized texture and provide the boundary conditions necessary to avoid seams due to discontinuities between texture element features in adjacent patches.

It is important to note that the partitioning stage described in the immediately previous sections is completely independent from the synthesis algorithm. Any constrained synthesis method that can fill arbitrary regions with arbitrary boundary conditions could potentially be used, although none of the existing algorithms we reviewed appeared to provide both the flexibility and the speed required for this project. The work of Efros and Leung [7] had the greatest direct influence on the development of the method that we use. We follow their general approach, which has been shown to produce good results for a wide range of texture patterns that can be modeled by Markov random fields (i.e. textures whose characteristics are constant under translation over the image and in which the value at any given point can be fully characterized by the values at its closely neighboring points over a limited range).

3.3.1 Isotropic Textures

In order make tractable the problem of efficiently synthesizing enough texture to cover a standard model of arbitrary topology at a reasonable resolution, the first objective of our proposed method is to achieve results that are of the same caliber as those demonstrated by Efros and Leung but that require significantly less time, while preserving the flexible applicability of their approach. To do this, we use a new two-pass search strategy. The first pass, which is exhaustive, is done using a very small unweighted neighborhood (usually 5x5 for sample textures of sizes between 64x64 to 128x128). The best matches are saved in a list to be processed by the second pass. This first pass implies strong locality in the input textures (which holds true for many natural textures) and has the effect of rapidly eliminating most of the uninteresting part of the search space. The size of this preselect list ultimately determines the overall speed of the synthesis algorithm. We found that some textures produced excellent results with preselect lists of as few elements as the number contained in one scanline of the original image; these we considered easy to synthesize. Others required 4 or even 8 times more elements in the preselect list, and these we considered hard to synthesize. In the second pass, each of the pixels in the saved list is tested against the full size weighted neighborhood (this size is user selectable) and the error metrics are updated. Among the best 10 or 10% of matches (whichever is greater) a random pixel is chosen and used in the synthesized image. Fig. 3 shows a sample result of this synthesis algorithm in the 2D case. The speed of this method does not match the speed of Wei and Levoy's tree-structured vector quantization [26], but the results it produces are of consistently high quality and it is fast enough to make feasible our goal of growing of a fitted surface texture via the MRF sampling approach.

As basically formulated, the approach we have just described can be used to cover the surface of a model with an isotropic texture pattern in an orientation-insensitive way. In other words, if we assume that our sample texture pattern is perfectly rotationally symmetric, we can directly use this approach, in its most basic form, to seamlessly synthesize the texture pattern across all of the patches in the model without any special considerations apart from the boundary conditions. However, as we quickly found, there are very few textures can be applied with good results without regard to orientation. Even patterns which initially appeared to be isotropic in the original 2D sample revealed unexpected orientation dependencies due to such things as the subtle structuring that stems from nearly imperceptible correlations due to shading, as is evident in fig. 4.

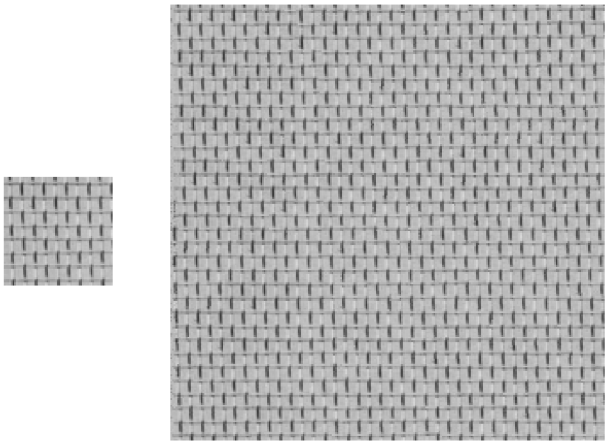

Figure 3: An brief example of the results of our two-pass texture synthesis method using pattern D06 from the Brodatz album [4]. The speed of the synthesis approach varies from pattern to pattern depending on the sizes of the match-defining neighborhoods and the length of the preselect list. In this case we used a first pass neighborhood of 5x5, a maximum preselect list length of 64, and a second pass neighborhood of 12x12 to synthesize the 256x256 patch on the right from the 64x64 pixel sample on the left in about 73s.

Figure 4: A texture ('wool.bw', from SGI) that originally appeared to be isotropic, reveals its anisotropic nature when synthesized over the Stanford bunny dataset via an approach in which the texture orientation is allowed to vary arbitrarily between each patch.

3.3.2 Directional Textures

For the vast majority of textures, it is required to control the orientation of the texture over the surface. In the case of directional textures, the texture should be able to follow an arbitrary direction field. The choice for the direction field is application dependent and the selection of method is user controlled. In the next section some examples will be discussed. We note that Praun et al. [23] also align the textures on a per patch basis, and distort slightly the parameterization to achieve the desired effect. In the case of sparse triangulation they reduce undersampling of the vector field by locally subdividing the mesh.

In our presented method the synthesis algorithm has been enhanced to allow per-pixel texture direction. The original texture is pre-rotated into a quantified number of orientations, and the synthesis algorithm searches only in the direction specified by the vector field. If the number of pre-rotated images is sufficiently high, the synthesized texture will follow the vector field smoothly (in all the examples in this paper 64 to 128 different orientations have been used). Fig. 5 shows a sample of one quadrant of pre-rotated brick texture. We use the PC's graphics hardware to perform the rotations and resampling, for efficiency and ease of implementation.

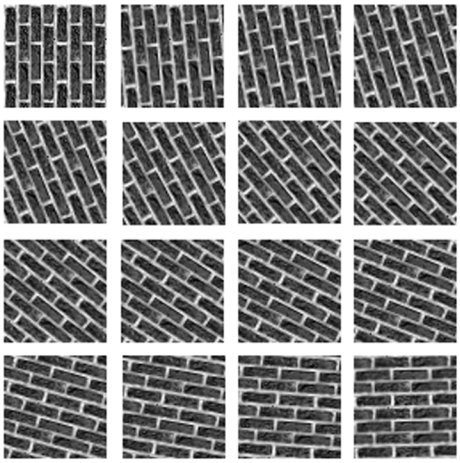

Figure 5: A quadrant of pre-rotated brick texture samples.

3.3.2.1 Constant Direction Field

We originally began this work with the intent to explore the possibility of applying textures along the principal directions of curvature. Despite the latent potential in that approach, it is not without its difficulties. We quickly discovered that aesthetic results could be achieved using other, much simpler, vector field definitions. The field of "up" directions, locally projected onto the tangent plane at each point, yields particularly nice results for a variety of textures and datasets, as shown in figures 6 and 7. It is worth mentioning in the context of these two images that we worked hard to challenge our texture synthesis method, testing its performance on texture patterns such as the crocodile skin, which contains potentially problematic set of features spanning a wide range of spatial frequencies (from 3–21 pixels in diameter) and the square glass blocks, which is a highly regular checkerboard-style pattern in which deviations in size, shape and/or positioning of any of the elements can be expected to stand out prominently.

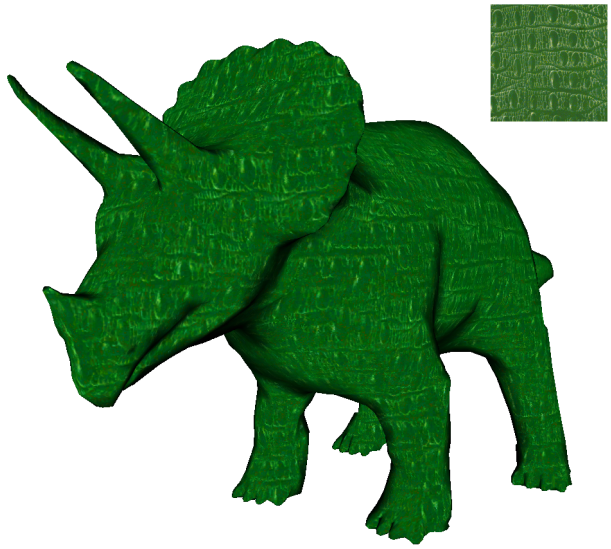

Figure 6: The Crocodile skin texture (D10) synthesized over the triceratops model following the direction field (0,1,0) locally projected onto the tangent plane.

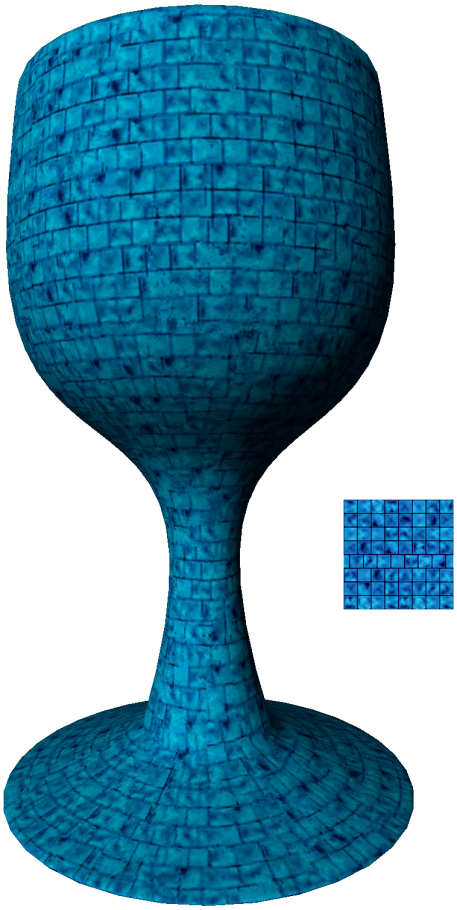

Fig. 7: Glass block texture applied to a simple model with a constant directional field. Note the general preservation of continuity in the texture pattern and the relative consistency of the bricks' shapes and sizes, despite scattered noticeable artifacts. The direction field is locally given by the projection of the central axis of the object onto the tangent plane of each patch. Some of the patches in this model were resynthesized in a postprocess, as described in section 7.

Figure 8: The orientation of a directed pattern over a curved surface can influence our perception of the surface's 3D shape. On the left, the bricks are oriented in the direction of greatest signed normal curvature; in the middle they are oriented in the direction of least signed normal curvature, and on the right they are oriented in the same constant "up" direction used for the models in figures 6-7.

3.3.2.2 Principal Direction Field

The constant direction field produces good results in some cases but not in all cases. Of greater intrinsic interest to our ongoing research is the possibility of applying an oriented texture pattern to the surface of an object such that it is everywhere aligned with the principal directions of curvature. Recent results in vision research support the idea that the principal directions play an important role in surface shape understanding. Specifically, Mammassian and Landy [17] have shown that observers' interpretations of line drawings are consistent with an inherent bias, among other things, towards perceiving lines on objects as being oriented in the principal directions, supporting an observation made by Stevens [24] nearly 20 years ago. Last year, Li and Zaidi [15] examined observers' ability to estimate the relative curvatures of surfaces textured with various patterns and found that shape perception depends critically upon the observation of changes in oriented energy along lines corresponding to the principal directions.

We are currently working with Dr. Jack Goldfeather to develop robust methods for computing smooth, accurate principal direction vector fields across arbitrary polygonally-defined objects [9]. A complementary approach being developed by Bertalmio *et al.* [3] has the potential to facilitate the anisotropic smoothing of these fields. Our present results in applying an anisotropic texture over the surface of an object such that its dominant orientation is everywhere aligned with the first and second principal directions are shown in figure 8, and contrasted there with the results obtained using a constant "up" direction. Despite the preliminary nature of these results, we believe that there is significant inherent potential for a method capable of synthesizing a variety of principal direction textures over arbitrary curving forms.

4. Implementation

The user-definable parameters of our system are illustrated in the GUI shown in figure 9. *Max Dist* constrains the sizes of the patches during splitting; *Max Norm* constrains the extent of the allowable non-planarity of patch triangles from the reference direction. *PDir Func* allows the user to choose among: orienting the texture to follow the direction of greatest or least signed or unsigned normal curvature, to follow the projection onto the local tangent plane of a constant direction specified in the boxes to the right, or to not follow any direction (*flat*). *Radius* defines the size of the neighborhood used in the second pass of the texture synthesis; *Linear/Gauss* allow the weights used in the matching to fall off with distance. *Sradius* defines the size of the neighborhood in the first pass of synthesis, while *Scanlines* limits the number of first-round preselected locations to be tested for a match on the second pass. There are also several viewing options, and several capabilities (*Disc* and *SLI*) which remain to be implemented. *Grow Direction* allows the user to control the direction of texture synthesis across a patch. For many textures this is left in *auto* mode, and the choice of direction is varied according to the distribution pattern of previously textured pixels

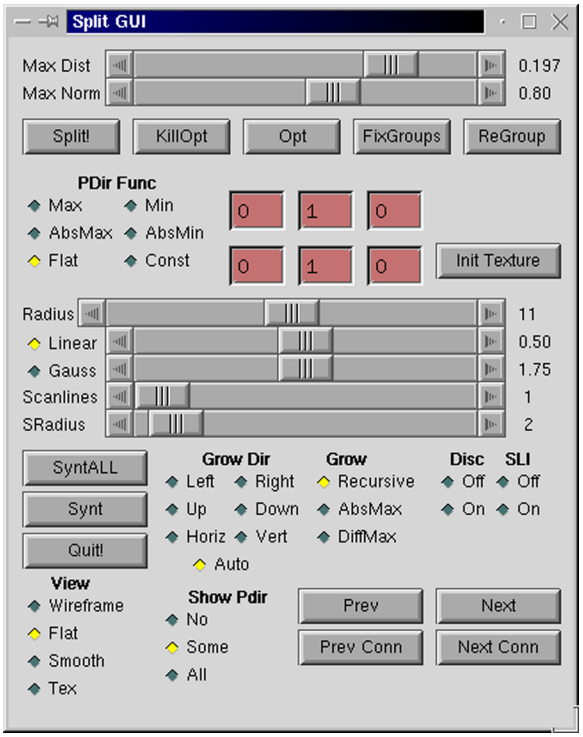

Figure 9: The capabilities of our system are shown in the graphical user interface.

in the boundary region. Specifically, it is started from the side containing the greatest number of previously filled pixels, in the hope that this will provide the most stable seed for the synthesis. Unfortunately, this is not always true (see the section on errors below). Certain strongly directional patterns seem to yield better results when the synthesis is performed in a particular pair of directions (left./right or top/down). For quickly varying direction fields, starting from areas in which the direction field is most calm seems to improve the quality of the synthesized patch.

We use one of two different methods to determine the direction in which the synthesis proceeds from patch-to-patch. The simple method that works well on many vector fields is to begin with a randomly chosen patch and proceed to any blank connected patch, filling in any holes at the end. For principal direction vector fields we get somewhat better results choosing the blank connected patch in which the difference between the two principal curvatures is greatest. This favors working first in areas over which the principal directions are clearly defined, providing a stable seed for the synthesis of the other patches.

The current implementation handles only grayscale and pseudo-color mapped images. While this is not an optimal choice, it allowed us to completely avoid the problem of choosing a color space and finding a good weighting function for the three channels. We achieved the colorized textures shown in this paper by the following method. An optimal color map is extracted from a true color image using a paint program such as The GIMP. The (YIQ) colormap is sorted by Y and the color indices in the picture are used directly as a greyscale image. While this method is not formally correct, as very different hues could end up to be nearly adjacent in the colormap because only the color *order* is defined by the Y values, we did not observe appreciable differences from a correct greyscale image formed from the Y values only. The disadvantage in using the y values directly is that it would force the use of floats very early in the program, making things more complicated. When the synthesis is finished, the sorted colormap is applied to the greyscale synthesized texture using the gray value again directly as an index into the colormap.

The most computationally expensive part of the algorithm is the texture synthesis, which accounts for 95 to 99% of the running time. Partitioning and optimizing the patches takes usually a total of 1-3 seconds on the simple meshes (Venus, triceratops) and 10-12 second on the 70,000 triangle bunny.

The synthesis time depends on the required number of pixels, (which determines the final resolution of the texture on the object) and the size of the pre-select list needed to correctly synthesize the texture. Growing the texture on the Venus model in Fig. 1 took slightly less than 12 min. All of these timings refer to a C++ implementation compiled with gcc[1] running on a standard linux (2.2.16) 933MHz Pentium III PC with a 32Mb GeForce 2 GTS.

4.1.1 Limitations

The large amount of texture generated by the synthesis revealed to be a problem in some OpenGL implementations. When a great level of detail is desired even in close zoom conditions, the amount of texture required to cover the mesh can easily exceed the total size of the texture memory. All of the machines that do not allow storage of textures in main memory failed to display the more complex models (e.g. the crocodile skinned triceratops). Using a texture atlas similar to the one described in [23] could help reduce the texture memory usage at the cost of incorrect mipmapping. In the current implementation, every patch is stored as a single texture producing a texture memory waste of 25% to 50% depending on the model. Thanks to OpenGL texture compression extensions all the models presented in this paper can maintain interactive frame rates on a standard PC even if the amount of uncompressed texture exceeds 100Mb.

This method is currently designed to be applied to static models, and we have not thought about how to extend it to the case of deforming or animated objects. Modifications to make the texture synthesis process deterministic are an obvious and easy to satisfy this requirement, but it is not immediately clear how one could guarantee that independently synthesized patterns do not differ profoundly from frame to frame as the mesh defining the object is globally deformed.

To achieve good results with the shape-following textures, the direction field must be band-limited: for any given texture there is a maximum spatial frequency that can be followed in the synthesis process and still produce correct results. Nevertheless we found that the method did a pretty good job at singular points on the brick-textured lava lamp object.

Sometimes, as described in [7], the synthesis algorithm fails and produces some garbage patches. Depending on the texture, 0-5% of the patches contain synthesis errors. Areas as small as 5x5 pixels (in a 128x128 pixel patch) are easily noticed. The difficulties in automatically detecting such small areas prevented the implementation of an automatic correction. The current implementation allows the user to interactively select and re-synthesize the unaesthetic patches. Figs 7-11 in this paper show models in which texture was resynthesized on one or more patches. Figs 1, 4 and 6 show results that were obtained without any postprocessing.

5. Applications And Future Work

There are many promising applications for this system and many directions for future work. One of the most interesting of these is multi-texturing. On a per-pixel basis it is possible to change not only the direction of the synthesized texture but even the texture itself according to any arbitrary function. Figures 10 and 11 are made using the illumination equation and two and four different textures respectively, each one with 63 rotations. These models, like all of the others in this paper, can be displayed at interactive frames rates on our standard PC.

The multi-texturing methods described in this paper have the potential to be useful for important applications in scientific visualization, for example in encoding a scalar distribution using texture type variations across an arbitrary domain in 2D or 3D. Other direction fields, such as gradient descent, hold promise for different applications, such as non-photorealistic rendering of terrain models (esp. in the case when it is desired to see through the surface). The methods that we have proposed can also be used for the visualization of scientifically computed vector fields over surfaces. An intriguing possible use for an extension of this work is in defining texture mixtures.

6. Conclusions

In this paper we describe a fully automatic method for synthesizing, from a small provided sample, patches of a perceptually equivalent texture pattern that can be fit over contiguous sets of polygons in an object without appreciable seams or projective distortion, and in which the texture pattern can be locally oriented to follow any computed vector field over the surface. Our method is straightforward and computationally efficient, and demonstrates good results with a variety of different texture types and models.

[1] optimization flags -O2 -funroll-loops -fstrict-aliasing -march=i686

Fig. 10: A demonstration of multi-texturing, in which the search for matches is performed within an array of different texture types. The texture type index can be defined by any function. In this example, we used the illumination function. In a real application one would want to use something more meaningful, such as soil type.

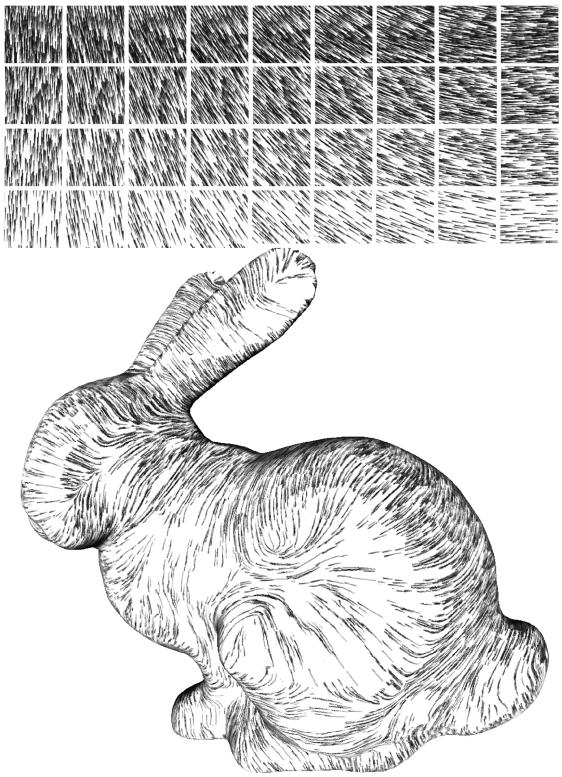

Fig. 11: Multiple textures containing lines of different widths applied to an automatically-defined smooth vector field approximating the first principal direction over the Stanford bunny. Indexing along the dimension of varying width was done as a function of the illumination.

7. Acknowledgments

This work was supported by a University of Minnesota Grant-in-Aid of Research, Artistry and Scholarship, NSF Presidential Early Career Award for Scientists and Engineers (PECASE) 9875368, NSF CAREER award CCR-9873670, ONR-N00014-97-2-0509, an Office of Naval Research Young Investigator Award, Presidential Early Career Award for Scientists and Engineers (PECASE) Navy/N00014-98-1-0631, and the NSF Learning and Intelligent Systems Program.

8. References

[1] Nur Arad and Gershon Elber. Isomeric Texture Mapping for Free-form Surfaces, *Computer Graphics Forum, 1997*, **16**(5): 247– 256.

[2] Chakib Bennis, Jean-Marc Vézien and Gérard Iglésias. Piecewise Surface Flattening For Non-Distorted Texture Mapping, *Proceedings of SIGGRAPH 91*, pp. 237 – 246.

[3] Marcelo Bertalmio, Guillermo Sapiro, Li-Tien Cheng and Stanley Osher. A Framework for Solving Surface Differential Equations for Computer Graphics Applications, UCLA-CAM report 00-43, December 2000 (www.math.ucla.edu).

[4] Phil Brodatz. Textures: A Photographic Album for Artists and Designers, Dover Publications, 1966.

[5] Jeremy S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images, *Proceedings of SIGGRAPH 97*, pp. 361 - 368

[6] J.-M. Dischler, D. Ghazanfarpour and R. Freydier. Anisotropic Solid Texture Synthesis Using Orthogonal 2D Views, Computer Graphics Forum, **17**(3), September 1998.

[7] Alexei A. Efros and Thomas K. Leung. Texture Synthesis by Non-Parametric Sampling, *Proceedings of the International Conference on Computer Vision*, vol. 2, 1999, pp. 1033 -1038.

[8] James A. Ferwerda, Sumanta N. Pattanaik, Peter Shirley and Donald P. Greenberg. A Model of Visual Masking for Computer Graphics, *Proceedings of SIGGRAPH 97*, pp. 143 – 152.

[9] Jack Goldfeather. "Understanding Errors in Approximating Principal Direction Vectors", TR01-006, Dept. of Computer Science and Engineering, Univ. of Minnesota, Jan. 2001.

[10] Steven Haker, Sigurd Angenent, Allen Tannenbaum, Ron Kikinis, Guillermo Sapiro and Michael Halle. Conformal Surface Parameterization for Texture Mapping, *IEEE Transactions on Visualization and Computer Graphics*, **6**(2), April/June 2000, pp. 181-189.

[11]Pat Hanrahan and Paul Haeberli. Direct WYSIWYG Painting and Texturing on 3D Shapes, *Proceedings of SIGGRAPH 90*, pp. 215 – 223.

[12]David J. Heeger and James R. Bergen. Pyramid-Based Texture Analysis/Synthesis, *Proceedings of SIGGRAPH 95*, pp. 229 – 238.

[13] Victoria Interrante. Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution, *Proceedings of SIGGRAPH 97,* pp. 109 – 116.

[14] Bruno Lévy and Jean-Laurent Mallet. Non-distorted Texture Mapping for Sheared Triangulated Meshes, *Proceedings of SIGGRAPH 98*, pp. 343 – 352.

[15] Andrea Li and Qasim Zaidi. Perception of Three-Dimensional Shape From Texture is Based on Patterns of Oriented Energy, *Vision Research*, **40**(2), January 2000, pp. 217 – 242.

[16] Jérôme Maillot, Hussein Yahia and Anne Verroust. Interactive texture mapping, *Proceedings of SIGGRAPH 93*, pp. 27 – 34

[17] Pascal Mamassian and Michael S. Landy. Observer Biases in the 3D Interpretation of Line Drawings, *Vision Research*, **38**(18), September 1998, pp. 2817 – 2832.

[18] Fabrice Neyret and Marie-Paul Cani. Pattern-Based Texturing Revisited, *Proceedings of SIGGRAPH 99*, pp. 235-242.

[19] Darwyn R. Peachey. Solid texturing of Complex Surfaces, *Proceedings of SIGGRAPH 85*, pp. 279– 286.

[20] Ken Perlin. An Image Synthesizer, *Proceedings of SIGGRAPH 85*, pp. 287 – 296.

[21] Dan Piponi and George Borshukov. Seamless Texture Mapping of Subdivision Surfaces by Model Pelting and Texture Blending, *Proceedings of SIGGRAPH 2000*, pp. 471 – 478.

[22] Javier Portilla and Eero P. Simoncelli. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients, *International Journal of Computer Vision*, **40**(1), October 2000, pp. 49-70.

[23] Emil Praun, Adam Finkelstein and Hughes Hoppe. Lapped Textures, *Proceedings of SIGGRAPH 2000*, pp. 465-470.

[24]Kent A. Stevens. The Visual Interpretation of Surface Contours, *Artificial Intelligence*, **17**(1-3), August 1981, pp. 47-73.

[25] Greg Turk. Generating Textures for Arbitrary Surfaces Using Reaction-Diffusion, *Proceedings of SIGGRAPH 91*, pp. 289 – 298.

[26] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis Using Tree-structured Vector Quantization, *Proceedings of SIGGRAPH 2000*, pp. 479 – 488.

[27]Andrew Witkin and Michael Kass. Reaction-Diffusion Textures, *Proceedings of SIGGRAPH 91*, pp. 299– 308.

[28] Steven Worley. A Cellular Texture Basis Function, *Proceedings of SIGGRAPH 96*, pp. 291– 294.

[29] Song Chun Zhu, Ying Nian Wu and David Mumford. Filters, Random Fields and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling, *International Journal of Computer Vision*, **27**(2), March 1998, pp. 107-126.