Eurographics Symposium on Geometry Processing (2004)
R. Scopigno, D. Zorin, (Editors)

Geometric Texture Synthesis by Example

Pravin Bhat[1], Stephen Ingram[2], Greg Turk[2]

[1] Microsoft Corporation [2] Georgia Institute of Technology

Abstract

Patterns on real-world objects are often due to variations in geometry across the surface. Height fields and other common parametric methods cannot synthesize many forms of geometric surface texture such as thorns, scales, and bark. We present an example-based technique for synthesizing a variety of geometric textures on a model's surface. The applied textures can be from models specifically created for this purpose, or may be drawn from user-specified regions of an example model. We extend existing neighborhood-based texture synthesis algorithms to operate on volumetric models. Similar to image analogies [11], given a training pair of unfiltered and filtered source models and an unfiltered destination model (all volumetric grids), we synthesize a filtered fourth model that exhibits the desired geometric texture. The user defines vector fields to specify the direction of texture anisotropy on the source and destination models. The vector field defines a coordinate frame on the destination object's surface that is used to sample the voxel density values in the neighborhood near a given voxel, which then gives a feature vector that is matched to the neighborhoods in the source model. Destination voxels are visited in an order that is dictated by the vector field. We show geometric synthesis results on a variety of models using textures such as pits, grooves, thru-holes and thorns.

1. Introduction

The majority of three-dimensional objects in the world exhibit some form of texture, especially those found in nature. Examples include the animal surfaces like fur or feathers, and plant surfaces like needles, thorns or veins. Texture can be added to an object in two ways, with *color texture* and *geometry texture*. *Color texture* is added by setting an object's surface colors, and *geometry texture* is added by altering an object's surface geometry. The method that we present in this paper creates geometry texture using an example-based synthesis technique.

We present a method of sampling geometric texture directly from a model and synthesizing the geometry in three-dimensional space. This method is inspired primarily by the aforementioned two-dimensional texture-synthesis methods, especially [11], and the texture synthesis on surfaces method described in [22, 26, 27, 28, 21]. These methods compare pixel neighborhoods between source and target images, performing a closest-match search among a collection of neighborhoods in the input texture. Instead of using 2D images for texture input, we use volumetric models, building neighborhoods from surrounding voxel values. We have modified the method of voxel neighborhood construction to be relative to a voxel's local three-dimensional coordinate frame rather than from a globally defined frame. The model's normal field and a user-specified vector field define this coordinate frame.

By performing the search based on local voxel neighborhoods, we are less restricted than previous methods of geometric texture synthesis. This is because our geometric texture is synthesized by matching a voxel-neighborhood with geometric features in the search space, rather than to neighborhoods of scalar values across a surface.

2. Related Work

Work related to geometric texture synthesis includes procedural methods, artistic tools, surface editing, planar and surface-based texture synthesis. We review prior work in each of these areas in the next four sub-sections.

2.1. Procedural Methods

Procedural methods use functions to define object properties. The technique of solid texture employs a 3D function to produce texture properties for each relevant point in 3-space. This idea was introduced independently by Perlin and Peachey [17, 18] and defined the solid texture paradigm. When coupled with bump mapping, a technique that alters

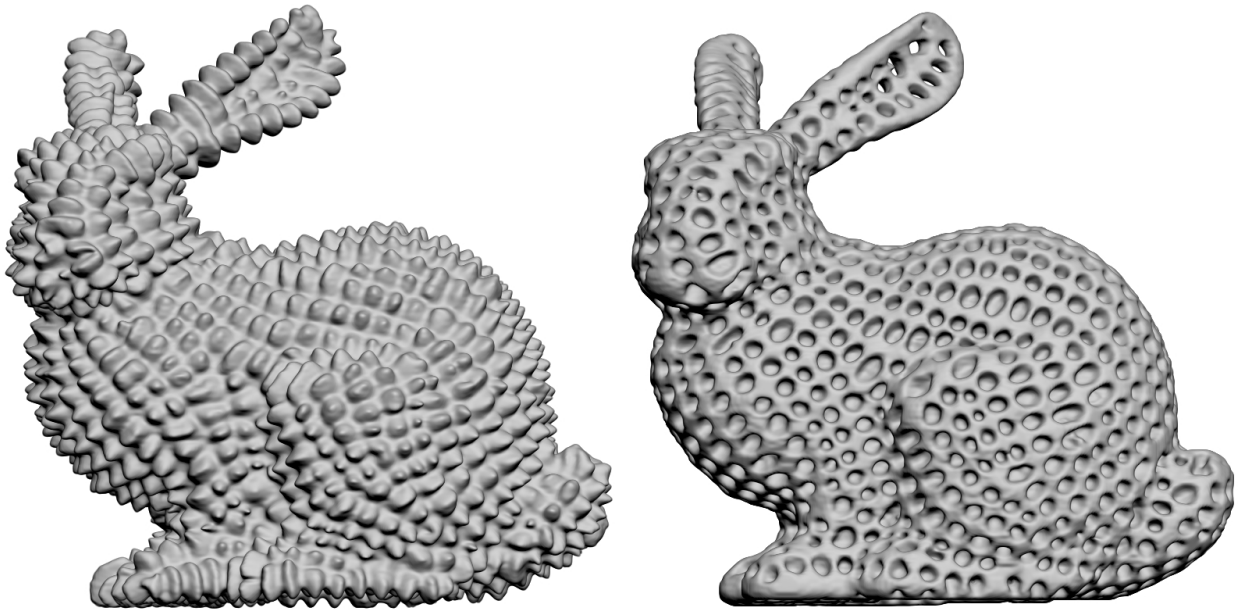

Figure 1: *Bunny with spikes (left) and pits (right).*

surface normals [4], or displacement mapping, which modifies actual surface geometry [5, 24], many geometric textures can accurately be synthesized. Perlin and Hoffert later devised a volumetric technique called Hypertexture [19] that procedurally alters a volumetric model's 3D geometry in a manner analogous to a solid texture. Fleischer *et al.* use a procedural approach to distributing geometric features such as scales and thorns on a 3D surface [8]. Legakis *et al.* use rule-based feature creation to place brick and stone patterns on architectural models [14].

2.2. Artistic Tools and Geometric Surface Editing

Artistic tools are subdivided into the genres of painting and sculpting. Many methods have been introduced to provide a user with interactive, manual control over a geometric surface and its properties. Hanrahan and Haeberli [9] presented an effective method for surface painting that provided an artist with several painting modes, including a "geometry painting" mode which involved surface height displacement.

There has been a recent boon of geometric surface sculpting tools, each with a distinct approach to surface editing. Biermann *et al.* introduced direct surface cut and paste operations with real-time interaction on multiresolution surfaces [3]. Zwicker *et al.* [29] created a system for interactive editing of point-based surfaces, eliminating the need for tessellation of 3D point-sampled geometries. Using level sets, Museth *et al.* [15] introduced a series of surface operations including smoothing, embossing, and cut-and-paste.

2.3. Example-Based Texture Synthesis

Example-based texture synthesis methods use 2D images of texture as input and synthesize new texture over an area of arbitrary size. A variety of schemes have been proposed to solve this problem. Efros and Leung [7] introduced an influential method of example-based synthesis relying on a non-parametric sampling of the input texture. In this method, new pixel values are decided after querying the sample texture and finding all similar pixel neighborhoods; a pixel value is then chosen randomly from one of these neighborhoods.

Wei and Levoy introduced a method similar to [7] that synthesized pixels in raster scan order, and used tree-structured vector quantization (TSVQ) [25] to accelerate the neighborhood search. To synthesize so-called "naturalistic textures," Ashikhmin [2] modified the Wei and Levoy method to preserve texture coherence. Instead of searching the entire space of neighborhoods in the sample image, the search was restricted to those neighborhoods mapped to the query neighborhood. Herztmann *et al.* combined both [25] and [2] in an algorithm called Image Analogies [11], which used training data "analogies" to mimic image-processing techniques like super-resolution and artistic filters. Hertzmann *et al.* applied a similar technique to modifying the style of line drawings to match a given style [12], and Kalnins *et al.* applied similar ideas for the creation of stylized lines for rendering 3D models as line drawings [13].

The method that we propose is a voxel-space extension of Image Analogies, therefore we provide a more detailed explanation of this algorithm in section 3.

2.4. Surface-Based Synthesis Methods

All of the texture synthesis methods described in the previous subsection were designed to produce a patch of texture in the plane. Mapping a planar region to a model surface introduces the problems of texture seams, texture stretch,

and specifying the texture orientation. Several methods have been proposed to synthesize textures directly on a model's surface. Wei and Levoy extended their synthesis method onto surfaces by building neighborhoods from a local parameterization of surface mesh vertices [26]. Ying [27] uses charts to map a model's surface to the plane and iterate over this map. Both [26] and [27] used displacement mapping to alter the geometry of a surface. Zhang uses texton masks to create spatially-varying textures on surfaces [28]. Tong *et al.* use example-based synthesis to create bidirectional texture functions on surfaces [21]. Soler *et al.* use a patch-based approach to perform texture synthesis on surfaces [20]. Our technique draws upon ideas from Turk's surface based synthesis [22]. This method synthesizes colors on a uniform mesh hierarchy, building pixel neighborhoods from surrounding mesh vertices. Synthesis is ordered according to a sweep distance determined by a user-defined vector field. The texture results are seamless and they are oriented according to the specified vector field.

3. Image Analogies Extended to Volumes

Our method of performing geometry synthesis based on voxel models is an extension of the Image Analogies algorithm [11]. Because this method is central to our own work, we review their method below. We then describe a naïve generalization of this approach to 3D, and discuss the limitations of this method. Section 4 then describes how to overcome these limitations.

3.1. Review of Image Analogies

Images Analogies is an algorithm that learns analogies in image space. For example, the method might be given a photograph A and a blurred version of the same photo, called A'. The goal of the algorithm is to "learn" the relationship between A and A', and to apply that relationship to another image. Given a different photo B, the method produces a blurred version of B' based on the analogy with A and A'. The analogy is B is to B' as A is to A'. Other analogies that can be learned include embossing, super-resolution, painting styles, and pen-and-ink drawing techniques.

Input to the algorithm consists of a target image B and a training pair that includes a source image A and the filtered version of the source image A'. Given these inputs, the algorithm creates the filtered version of the target image B' such that the filter most resembles the transformation from A to A'. The algorithm achieves this through constrained texture synthesis, where images A and B serve as a guide to the textures in A' that B' selectively samples from. The learning in this algorithm is done by searching for similar pixel neighborhoods in the various images. The algorithm begins by creating a search space of feature vectors. Each feature vector consists of a full-square (*non-causal*) neighborhood of the given pixel in A and a half-square (*causal*) neighborhood of the corresponding pixel in A'. Synthesis is carried out in a raster-scan order, replacing every pixel in B' with the pixel in A' that was found to have the most similar feature vector. The reason for using a half-square neighborhood for pixels in A' and B' is that the pixels below the current pixel of B' have not yet been visited, and therefore they should not be used in neighborhood matching.

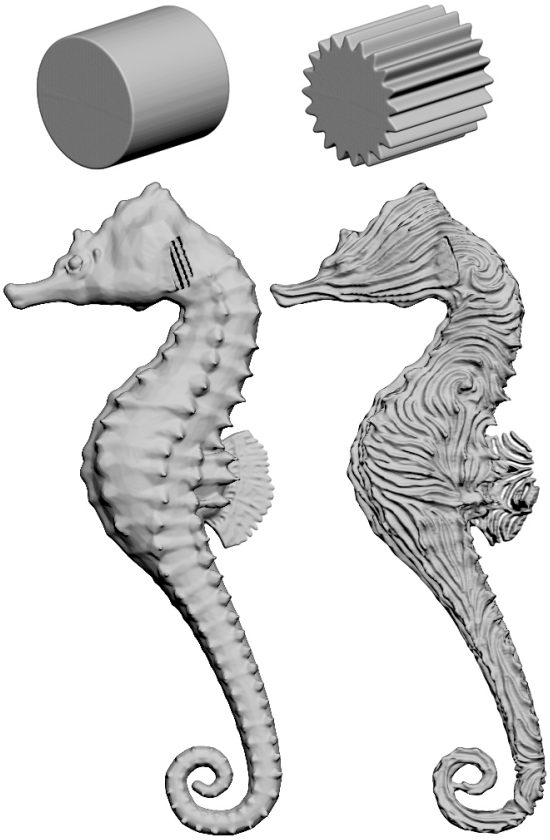

Figure 2: *A cylinder pair without and with grooves (top) are used to modify a seahorse model to have grooves (bottom).*

There are several additional issues that should be addressed to get high quality results for Image Analogies. This issues include image pyramids, fast search techniques, coherence between adjacent pixels, and histogram matching. For details about these issues, we refer the reader to the original Image Analogies paper [11].

3.2. Naïve Volume Analogies

The goal of our work is to produce 3D models instead of 2D images. To this end we work with voxel data, which is the most straightforward generalization of shape description when moving from 2D to 3D. We represent the interior of a 3D objects with a voxel density of one, and exterior voxels are given a density of zero. Voxels close to the object's surface have densities that may be intermediate between these extremes. Given this shape representation, the Image Analogies algorithm should be able to learn analogies in volumetric space by making the following changes:

- Voxels play the role of pixels, and voxel neighborhoods are collections of nearby voxels.

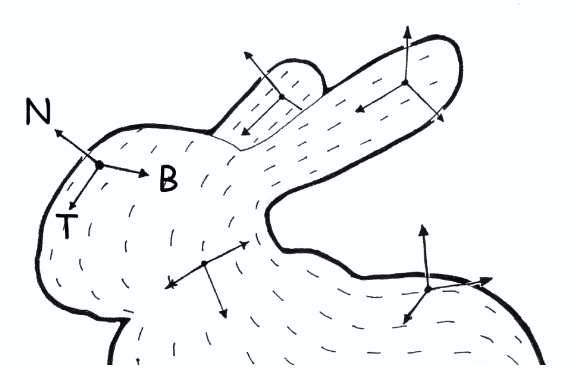

Figure 3: *Coordinate frames on a model.*

- The voxels are visited in order from the top volume slice to the bottom volume slice, and row by row within each slice.
- A non-causal neighborhood consists of a cube of voxels surrounding a given voxel.
- A causal neighborhood is made up of a half-cube of voxels from the slices above the given voxel, and possibly some voxels in the same slice as the current voxel.

Our implementation of these modifications shows that the Image Analogies technique does indeed learn certain kinds of three dimensional filters (analogies). In our experience, these filters must be isotropic and must be uniformly applied across the training pair A and A'. An example of such a successful 3D dimensional analogy is volumetric blur. This naïve approach to volumetric analogies fails, however, when a given filter (such as a surface pattern) has a directional preference. The principal reason why the adaptation fails for such filters is because sampling a voxel neighborhood using the global XYZ axes assumes that the texture at every voxel is aligned with the world coordinate system. In reality this is seldom the case. Most textures on a surface have continually changing local orientation which may or may not be aligned with the coordinate axes. In the next section we describe an approach that handles this important issue.

4. Geometry Synthesis by Example

The key to synthesizing a geometric pattern on a surface is to recognize that many patterns have a preferential direction. Examples of such oriented patterns include scales on a lizard, bark on a tree, and grooves on corrugated cardboard. To properly place such patterns on an object we must use the direction of the input surface pattern, and we must also specify the direction of the pattern on the surface to be modified. As an example, consider putting grooves on a seahorse. The input analogy will be a cylinder A and a grooved cylinder A' (Figure 2, top row). We also need an input model B for the seahorse (Figure 2, bottom left). We wish to create a grooved seahorse B' (see Figure 2, bottom right). In addition to the voxels for A, A' and B, we also need the pattern orientation (the direction of the grooves) for A'. We specify this orientation as a vector field within the volume A. Finally, we specify the desired orientation of this pattern on the seahorse, which is another vector field that is defined on the surface of the seahorse. Both of the vector field are extended throughout their volumes, and we will discuss this issue in detail later.

Figure 3 shows a vector field on the surface of a bunny model. The direction **T** is a tangent vector that points in the direction that we wish the pattern to follow. Each point on the surface has such a tangent vector that has been specified by the user. When this tangent direction is taken together with the surface normal **N** and a second tangent **B** (perpendicular to **T** and **N**), this describes a coordinate frame at a given position on the surface. *All of our voxel neighborhoods will be locally aligned with such local coordinate frames.* We must also remember that voxels away from the surface must also have a coordinate frame, since all voxels in the volume B' must be visited.

Our geometry synthesis algorithm visits all of the voxels of B', creates a feature vector from voxels in B and B', and finds the best matching feature vector from A and A'. The feature vectors from B and B' consist of voxel densities near the voxel that is currently being visited. Voxels from B should be taken from a full-cube neighborhood that is aligned with the local coordinate frame. Figure 4 shows two such full-cube neighborhoods. The neighborhood from B', on the other hand, should only include those voxels near the current one that have already been visited. These will be the voxels "behind" the current voxel, that is, the opposite of the tangent direction T. Figure 5 shows two such half-cube neighborhoods that contain only voxels in the direction of $-\mathbf{T}$.

In the next section we will examine the details that are needed to make this approach concrete.

4.1. Volume Attributes

Before synthesis, we require the following attributes for a model:

Volume data : A density map that provides the volumetric representation of the given model.

Distance Field : Specifies each voxel's distance to the nearest surface voxel. The sign of the distance field signifies whether the given voxel is interior or exterior to the surface, with positive distance values denoting exterior voxels.

Local Coordinate Frame : Specifies a local coordinate axis for every voxel such that the axis is consistently aligned to the surface texture.

Sweep Distance Field : Specifies the order of visitation of voxels in a volume such that the local coordinate frame can always sample a causal (previously-visited) neighborhood.

4.2. Feature Vector Creation and Search

With the neighborhood description and sweep distance in hand, we can now examine the details of feature vector creation and synthesis. Here are the steps needed to make the training feature vectors and to search for closest matches during synthesis:

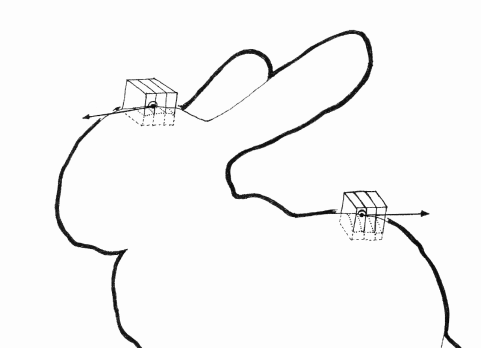

Figure 4: *Full-cube neighborhood (non-causal), aligned to the local coordinate frames.*

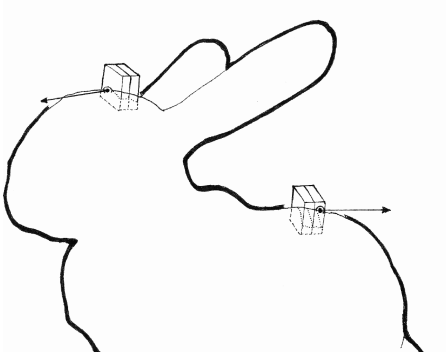

Figure 5: *Half-cube neighborhood (causal).*

- For each voxel in A' (pre-process):
  - Construct a feature vector by concatenating two sub-feature vectors:
    1. A causal neighborhood of the current voxel in A', sampled using the local coordinate frame.
    2. A full cubic neighborhood of the corresponding voxel in A sampled using the local coordinate frame.
  - Each such feature vector is added to a collection that serves as training data for the synthesis part of the algorithm. We store the training data in a data structure that is designed for fast search: approximate-nearest-neighbor search (ANN) [1].
- For each voxel in B', visited in increasing order of sweep distance (synthesis):
  - If the absolute distance field value of the current voxel is greater than the neighborhood size then set the output voxel to EMPTY or OPAQUE depending on the sign of the distance field value and continue on to the next voxel. Culling these distant voxels speeds up the method considerably.
  - Construct a feature vector as described above using the current neighborhoods in B' and B with the coordinate frame for B.
  - Perform a search to find the feature vector in the training data most similar to the feature vector of the current voxel. Distances between feature vectors are calculated as the sum of squared differences between corresponding voxel densities.
  - Replace the current voxel in B' by the voxel in A' found to have the most similar feature vector based on the search result.

4.3. Multiscale Synthesis

Like many related synthesis techniques, the quality of our results are significantly improved by using multiple passes at increasingly finer resolution. We create Gaussian pyramids of all our voxel data to allow for multiresolution synthesis such that A_1 forms the highest resolution of the pyramid and A_2, A_3 and so on form successively lower resolutions of the pyramid. All density-map Gaussian pyramid levels, X_{k+1}, are created by low-pass filtering and downsampling the density data in X_k, the pyramid level directly below.

The multiscale synthesis proceeds from the coarsest resolution to the finest, synthesizing B' at each level in the Gaussian pyramid. After a coarse level synthesis pass is complete, the coarse densities are upsampled and used to initialize the higher resolution volume. Each level in the pyramid is synthesized in the manner described above with a slight change in the construction of the feature vector. Specifically, a feature vector is constructed by concatenating four sub-feature vectors:

- A causal neighborhood of the current voxel in X'_k.
- A full cubic neighborhood of the corresponding voxel in X_k.
- A full cubic neighborhood of the low resolution voxel in X'_{k+1}.
- A full cubic neighborhood of the low resolution voxel in X_{k+1}.

For the results shown in this paper, we used a neighborhood of $5 \times 5 \times 5$ to sample the fine resolution level and a neighborhood of $3 \times 3 \times 3$ to sample the coarse level. We find that the quality of the synthesis is improved by conducting not one but two synthesis sweeps through the voxel data at each pyramid level.

5. Attribute Calculations

In this section we describe the tools used to create the volume attributes that are needed for geometry synthesis.

5.1. Voxelization and Distance Fields

Many of the objects that we wish to modify are in the form of a polygonal mesh, and therefore must be converted to a voxel representation. There are numerous published methods that describe how to convert a polygonal mesh to voxels, including [23, 10, 16]. We use the technique of Nooruddin and Turk [16] that performs ray stabbing and checks the inside/outside parity of each ray at voxel centers.

We create the distance field during the voxelization process using Danielsson's algorithm [6] adapted to work in 3D.

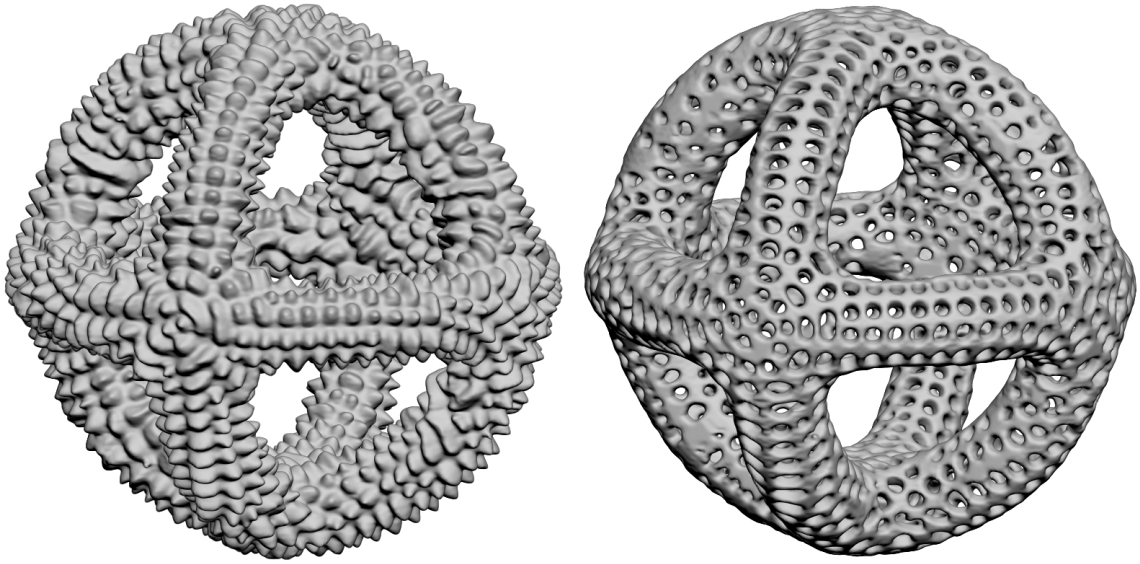

Figure 6: *Two versions of three intersecting tori. The model on the right has been hollowed out by the geometry synthesis.*

Danielsson's 2D algorithm is applied to each image slice in the volume, calculating the distance field for every voxel in that image using its six immediate neighbors, as the volume is swept from the topmost to bottommost image slice and then in the reverse direction.

5.2. Creating a Local Coordinate System

The surface vector field is created by having the user specify a few direction vectors on the surface that are then diffused across the surface using the vector field interpolation technique described by Turk [22]. This defines the local **T** axis everywhere on the surface. The normal vector **N** is calculated from the object's geometry, and the **B** axis is the cross-product of **T** and **N**.

We propagate this local coordinate frame from the surface voxels to the rest of the volume using a vector field interpolation method similar to the one described in [22]. We initialize the local coordinate frame of every voxel in the volume to zero vectors with the exception of surface voxels whose local coordinate frame we have already defined. We fix these non-zero local coordinate axes, and then diffuse the coordinate axes over the remaining volume. We repeat the fix-and-diffuse step until all voxels have a non-zero local coordinate frame. Propagating values out from a surface is a common operation in level set PDE techniques, and we could easily have used one of these methods instead.

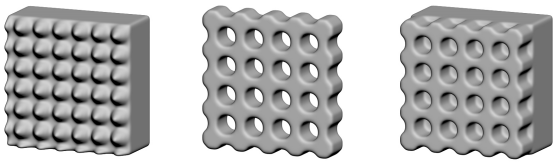

Figure 7: *Source geometry (A') for spikes, thru-holes and pits.*

5.3. Frame-Aligned Neighborhood Samples

Given a coordinate frame $(\mathbf{T},\mathbf{N},\mathbf{B})$ at a given voxel in B and B', we must be able to sample the voxel neighborhood in order to compare it to neighborhoods in A and A'. The basic idea is to move in the directions of $(\mathbf{T},\mathbf{N},\mathbf{B})$ instead of (x,y,z) to get neighboring voxel values. For non-causal (full-cube) neighborhoods in B, we travel in step-sizes of one voxel along each of the three local frame axes, both in the positive and negative directions. For causal (half-cube) neighborhoods in B', we travel in both positive and negative directions for **N** and **B**, but only in the $-\mathbf{T}$ direction. Stepping one voxel's length along an arbitrary direction will seldom take us to another voxel center, so we must interpolate voxel values. We use tri-linear interpolation of the eight nearest voxels.

Note that using a frame-aligned causal neighborhood for B' at a particular voxel does *not* guarentee that each voxel of the neighborhood has already been visited. Voxels on the leading edge of the neighborhood may or may not have been visited. Moreover, the user-defined vector field will have singularities such as a source. For this reason, we initialize the voxels of B' with the values from B at the coarsest pyramid level, and we use upsampling to initialize B' at the finer pyramid levels.

5.4. Sweep Distance Field Creation

Once we have a local coordinate system for our volumes we need to define an ordering to the voxels in B/B' such that the voxels visited in this order can always sample a causal neighborhood using their local coordinate frame. We do this by first defining a surface sweep distance field for B using

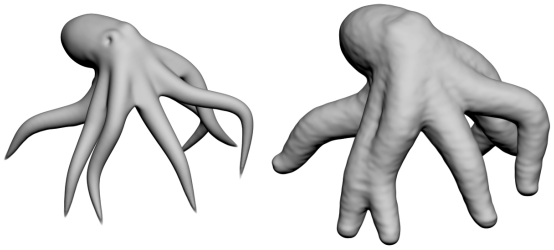

Figure 8: *Octopus (left) and dilated version (right).*

the technique described in [22]. This results in all surface voxels along the user-defined surface orientation vector to be assigned increasingly larger sweep distances. The sweep distance is then propagated through the volume along the local **N**-axis of the surface voxels. During synthesis we visit voxels in B/B' in increasing order of sweep distance.

6. Results

Two examples of geometric texture synthesis using a bunny are shown in Figure 1. The left bunny has been given spiky armor, while the right one has been pitted. Another example of geometric surface textures are the grooves on the seahorse and bunny (Figures 2 and 11). Notice how the grooves can closely match a complex, swirling vector field. Figure 7 shows the input geometric texture (A') for many of our examples. For these three textures, A is a thick slab of voxels. Figures 6 (right) and 10 demonstrate that when the A' geometry is thin (part of a hollow object) and is aligned atop a thick voxel slab A (which is solid), the synthesis can create an entirely hollow object.

Our method is capable of performing geometric analogies other than texture. For example, Figure 8 shows dilation (expansion) of an octopus's surface. The input analogy was two spheres, with sphere A smaller than sphere A'.

Figure 9 shows the creation of thorns on a sphere. Note that a simple displacement in the normal direction could not have produced these curving thorns.

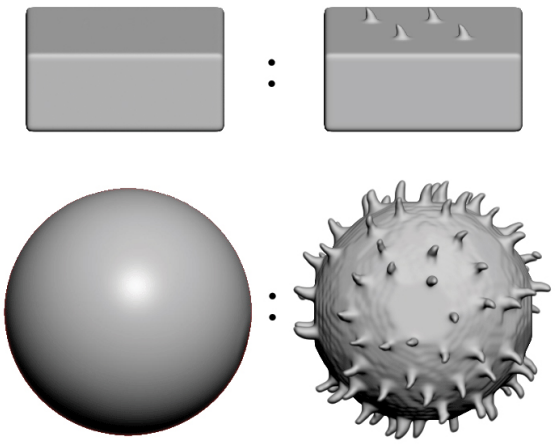

Figure 9: *Top row is input analogy, bottom row shows original smooth sphere and resulting sphere with thorns.*

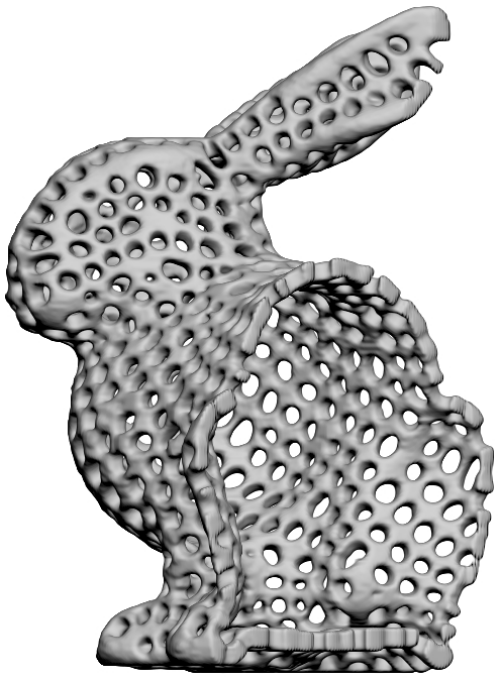

Figure 10: *Bunny with thru-holes, cut open to see the hollow interior.*

The sizes of the volumes we used were $102 \times 46 \times 227$ for the seahorse, $158 \times 200 \times 197$ for the bunny and $196 \times 200 \times 197$ for the triple-torus. Our synthesis times varied from slightly more than one hour for bunny grooves to nearly six hours for the triple-torus with holes. Geometry synthesis was calculated using a 3 GHz Pentium 4 processor. Synthesis times obviously scale up with larger volume sizes, but they also seem to be affected by the ease or difficulty of searching the ANN (approximate nearest neighbor) data structure to match a given feature vector. That is, geometry that is more radically altered tends to take longer to produce.

All of the rendered images in this paper are direct volume renderings of the volumetric results. If polygons are needed for a particular application, isosurface extraction methods can be used.

7. Conclusion and Future Work

We have presented a method that learns a geometric texture from one model and then copies this pattern to another model. To do this we have extended the Image Analogies algorithm into the volumetric domain. This approach makes actual geometric changes to a model, not just changes to appearance via surface normals or color. The method can induce extreme geometric modifications that cannot be achieved through offsets in the normal direction. The changes can even in some cases alter the topology of the surface (e.g. create a hollow surface or thru-holes).

There are several possibilities for future work. One is to modify the sampling technique to account for the surface

curvature. We believe that bending the voxel neighborhoods based on the local curvature may improve the results further since this might remove blind spots created by sharp changes in the surface curvature. The technique might also benefit from incorporating Ashikhmin style synthesis [2] to increase the coherence of the texture results.

8. Acknowledgements

We are grateful to the anonymous reviewers for their suggestions for improving our paper. We thank Gordon Kindlmann for his volume renderer, which was used to make our final images. This research was funded in part by NSF grant CCR-0204355 and a Microsoft Technical Scholarship.

References

[1] Arya S., Mount D. M., Netanyahu N. S., Silverman R., Wu A. Y., "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions," *Journal of the ACM*, 1998, pp.891–923. Source code available from http://www.cs.umd.edu/ mount/ANN.

[2] Ashikhmin, M. 2001. "Synthesizing natural textures," *ACM Symposium on Interactive 3D Graphics*, March 2001, pp. 217–226.

[3] Biermann H., Boier-Martin I. M., Bernardini F., Zorin D., "Cut-and-Paste Editing of Multiresolution Surfaces Surfaces," *Proceedings of SIGGRAPH 2002*, pp. 312–321.

[4] Blinn, J. F. "Simulation of Wrinkled Surfaces," *Computer Graphics (SIGGRAPH '78 Proceedings)*, 1978 pp. 286–292.

[5] Cook R. L., "Shade Trees," *Computer Graphics (SIGGRAPH '84 Proceedings)*, July 1984, pp. 223–231.

[6] Danielsson P., "Euclidean Distance Mapping," *Computer Graphics and Image Processing*, vol. 14, 1980, pp. 227–248.

[7] Efros A. and Leung T., "Texture Synthesis by Non-parametric Sampling," *In International Conference on Computer Vision*, Vol. 2, September 1999, pp. 1033–1038.

[8] Fleischer K., Laidlaw D., Currin B., Barr A., "Cellular Texture Generation," *Proceedings Computer Graphics (ACM SIGGRAPH)*, August 1995, pp. 239–248.

[9] Hanrahan P. and Haeberli P., "Direct WYSIWYG Painting and Texturing on 3d Shapes," in *Proceedings Computer Graphics (ACM SIGGRAPH)*, August 1990, vol. 24, pp. 215–223.

Figure 11: *Bunny with grooves.*

[10] Huang, J., Yagel, R., Filippov, V., and Kurzion, Y., "An Accurate Method for Voxelizing Polygonal Meshes," *Proceedings of the Symposium on Volume Visualization*, pp. 119-126, October 1989.

[11] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H., "Image analogies," *Proceedings of SIGGRAPH 2001*, August 2001, pp. 327–340.

[12] Hertzmann A., Oliver N., Curless B., Seitz S., "Curve analogies," *Eurographics Workshop on Rendering*, June 2002, pp. 233–245.

[13] Kalnins, Markosian, Meier, Kowalski, Lee, Davidson, Webb, Hughes, Finkelstein, "WYSIWYS NPR: Drawing Strokes Directly on 3D Models, *Proceedings of SIGGRAPH 2002*, July 2002, pp. 755–762.

[14] Legakis J., Dorsey J., Gortler S., "Feature-Based Cellular Texturing for Architectural Models," *Proceedings of SIGGRAPH 2001*, August 2001, pp. 309–316.

[15] Museth K., Breen D. E., Whitaker R. T., Barr A. H., "Level Set Surface Editing Operators," *Proceedings of SIGGRAPH 2002*, 2002, pp. 330–338.

[16] Nooruddin F. S. and Turk G., "Simplification and Repair of Polygonal Models Using Volumetric Techniques," *IEEE Transactions on Visualization and Computer Graphics*, April-June 2003, pp. 191–205.

[17] Peachey D. R., "Solid Texturing of Complex Surfaces," *Computer Graphics, (SIGGRAPH '85)*, July 1985, pp. 279–286.

[18] Perlin K., "An image synthesizer," *Computer Graphics (SIGGRAPH '85 Proceedings)*, 1985, pp. 287–296.

[19] Perlin K. and Hoffert E. M., "Hypertexture," *Computer Graphics (SIGGRAPH 1989)*, 1989, pp. 253–262.

[20] Soler C., Cani M.-P., Angelidis A., "Hierarchical Pattern Mapping," *Proceedings of SIGGRAPH 2002*, July 2002, pp. 673–680.

[21] Tong X., Zhang J., Liu L., Wang X., Guo B., Shum H.-Y., "Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces," *Proceedings of SIGGRAPH 2002*, July 2002, pp. 665–672.

[22] Turk G., "Texture synthesis on surfaces," *Proceedings of SIGGRAPH 2001* August 2001, pp. 347–354.

[23] Wang, S. and Kaufman, A.E., "Volume Sampled Voxelization of Geometric Primitives," *IEEE Visualization '93*, pp. 78-84, October 1993.

[24] Wang L., Wang X., Tong X., Liu L., Guo B., Shum H.-Y., "View Dependent Displacement Mapping," *Proceedings of SIGGRAPH 2003*, August 2003, pp. 334–339.

[25] Wei L.-Y. and Levoy M., "Fast Texture Synthesis Using Tree-structured Vector Quantization," *Proceedings of SIGGRAPH 2000*, July 2000, pp. 479–488.

[26] Wei L.-Y. And Levoy M., "Texture synthesis over arbitrary manifold surfaces," *Proceedings of SIGGRAPH 2001*, August 2001, pp. 355–360.

[27] Ying L., Hertzmann A., Biermann H., AND Zorin D., "Texture and shape synthesis on surfaces," *Proceedings of 12th Eurographics Workshop on Rendering*, June 2001, pp. 301–312.

[28] Zhang, J., Zhou, K., Velho, L., Guo, B. and Shum, H.-Y., "Synthesis of Progressively-Variant Textures on Arbitrary Surfaces," *Proceedings of SIGGRAPH 2003*, pp. 295–302.

[29] Zwicker M., Pauly M., Knoll O., Gross M., "Pointshop 3D: An Interactive System for Point-Based Surface Editing," *Proceedings of SIGGRAPH 2002*, pp. 322–329