Geometric Texture Synthesis and Transfer via Geometry Images

Y.-K. Lai[*] Tsinghua University Beijing, China

S.-M. Hu[†] Tsinghua University Beijing, China

D. X. Gu[‡] State University of New York at Stony Brook, USA

R. R. Martin[§] Cardiff University Wales, UK

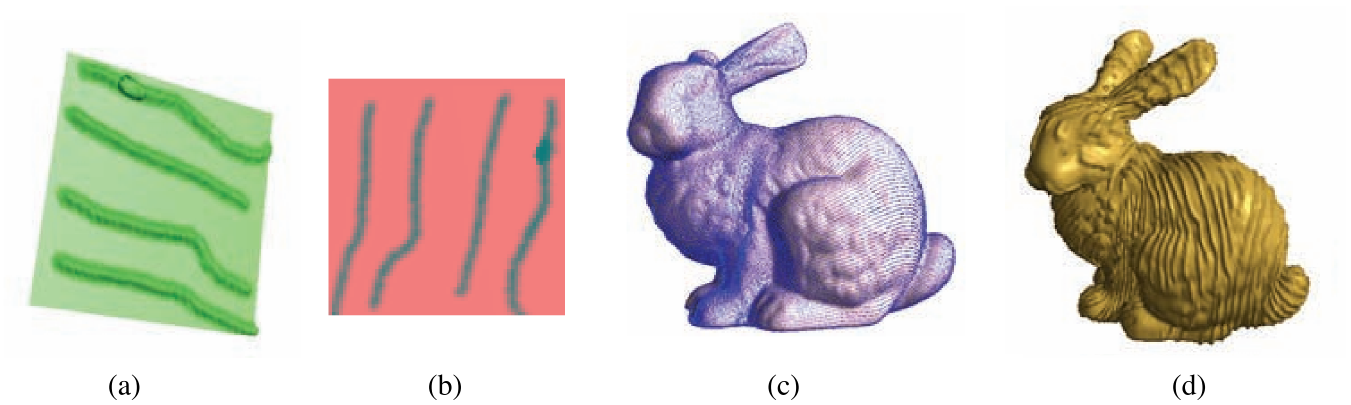

Figure 1: Steps in the algorithm. (a) shows a manually created texture, (b) is the texture converted to a geometry image, (c) shows the vector field giving texture orientation, and (d) shows the synthesized result.

Abstract

In this paper, we present an automatic method which can transfer *geometric textures* from one object to another, and can apply a manually designed geometric texture to a model. Our method is based on *geometry images* as introduced by Gu et al. The key ideas in this method involve geometric texture extraction, boundary consistent texture synthesis, discretized orientation and scaling, and reconstruction of synthesized geometry. Compared to other methods, our approach is efficient and easy-to-implement, and produces results of high quality.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems; I.3.5 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: texture transfer, texture synthesis, geometry images

1 Introduction

Triangle meshes are widely used for modeling geometry. They are easy to acquire, and can be used to represent surfaces of any complexity. However, it is more difficult to perform computations on meshes of this type which often have complicated topological domains compared to the relative simplicity of processing regular 2D images. Thus, in recent years, a variety of techniques has been developed for manipulation of such models; these are generally referred to as *digital geometry processing*. Triangle meshes are suitable for hardware rendering, but they are generally difficult to edit. There are different approaches to facilitate editing on meshes. One is multi-resolution editing, which constructs a hierarchy of mesh models using a wavelet-like structure. This allows the results of manipulation of low-resolution versions of models to be mapped back to the original high-resolution model. Another approach, called *cutting-and-pasting*, focuses on allowing the user to cut part of a model and paste it somewhere else (on the same or a different model). Our approach is different: we focus on geometric details or textures which we extract and transfer to another model, so that the new model has similar geometric details or patterns. By a geometric texture, we mean a small scale deformation vector field locally affecting a surface; we assume that the underlying geometric surface is smooth relative to the scale of details in the texture. Our approach allows the process to be done automatically, or it can be guided by the user to create interesting and visually pleasing results.

Our method processes an *input model* to give an *output model*: geometric details of interest are taken from a *sample model* and are applied to the input model to produce the output model. The basic steps are shown in Fig. 1.

If we use as a sample model an object with an existing interesting texture, we can use our technique for *texture transfer*. If we want simple regular patterns, it is also possible to manually design a sample model with any 3D surface editing tool, and transfer such patterns to other models in a *stylization* process. We could also use some surface patch covered with a special finish (e.g. fish scales or wood carving) as the sample model. In this case, the resulting effects are similar to non-photorealistic rendering and we call this *artificial texturing*.

[*]e-mail: laiyk@cg.cs.tsinghua.edu.cn
[†]e-mail:min@tsinghua.edu.cn
[‡]e-mail: gu@cs.sunysb.edu
[§]e-mail: ralph@cs.cf.ac.uk

This paper gives an approach for extracting geometric textures from sample models, and synthesizing similar textures on input models, in a way which hides seams across boundary cuts which are made when converting the input model to a geometry image. In order to further improve the synthesis results, especially for anisotropic textures, we have developed a discretized orientation and scaling approach to allow synthesis of textures with a desired density and orientation. Our method is based on the *geometry images* representation proposed by Gu *et al.* in [Gu et al. 2002]. We briefly summarize this idea and various related techniques in Section 2. An overview of our algorithm is given in Section 3 and details are provided in Section 4. Experimental results are presented in Section 5, and conclusions and discussions of future work are given in Section 6.

2 Related Work

Our method is based on the *geometry images* representation [Gu et al. 2002], proposed by Gu *et al.*. It allows a *regular grid* representation for meshes of *arbitrary topology*. Given a surface mesh, this is accomplished by first making cuts in the mesh to make it homeomorphic to a disk, then adding additional cuts to reduce distortion. This allows parameterization of the surface on a planar domain; geometric information (e.g. positions, normals) as well as other attributes are then sampled on a regular grid in this domain. Geometry images are useful in this work as they make operations on geometry efficient and easy to implement. However, they still have certain problems in areas which are addressed in this paper. Firstly, geometry images usually include cuts, and consistency across such cuts is required in model editing. Secondly, geometric textures may be anisotropic, and so users may wish to define an orientation field on the input model for more precise control. Traditional texture synthesis methods on surfaces are complicated and usually not very efficient, as they are usually based on some kind of *surface marching*. On the other hand, our method utilizes the regularity of geometry images, together with a preprocessing step, to synthesize texture at a combination of different orientations and scales, leading to very efficient generation of the output model. After the preprocessing steps of geometry image generation and texture preparation, the synthesis step is orders of magnitude faster than traditional methods based on surface synthesis, providing the possibility for interactive editing. Recently, geometry images have been further studied in [no et al. 2003; Losasso et al. 2003; Praun and Hoppe 2003; Sander et al. 2003].

Our work is also closely related to *image* texture synthesis methods, although here we are dealing with *geometric textures*, the common factor being the regular grid used to describe both 2D images, and geometry represented as a geometry image. Research on image textures and texture synthesis has a long history, and has received much attention as textures bring realism to computer graphics. A comprehensive survey of this field is outside the scope of this paper. Here we only focus on a few neighborhood matching-based texture synthesis approaches proposed recently which are most related to our work. Texture synthesis based on neighborhood matching was first introduced in [Efros and Leung 1999]; such methods can be generally categorized into three types: *pixel-based* approaches as in [Ashikmin 2001; Hertzmann et al. 2001; Wei and Levoy 2001a], *patch-based* approaches as in [Efros and Freeman 2001; Kwatra et al. 2003] and the combination of these two methods, e.g. [Nealen and Alexa 2003].

Pixel-based methods generate synthetic image pixels one by one usually in scan-line order. The basic idea of [Wei and Levoy 2001a] is to consider similarity measures between neighborhoods of pixels in the output image and the sample texture to decide what to synthesize next. A similar idea in [Ashikmin 2001] considers local consistency, which is especially important when synthesizing natural textures. A scheme which can be considered to be a combination of these ideas is given in [Hertzmann et al. 2001]. In most methods except those like the one in [Ashikmin 2001], searching through large data sets of high dimensions needs to be done frequently. Thus, different techniques have been used to speed up the process, including use of tree-structured vector quantization in [Wei and Levoy 2001a] and approximate nearest neighbors (ANN) in [Hertzmann et al. 2001].

Patch-based methods copy consecutive patches from the sample image, and stitch them together to generate the synthetic image. A dynamic programming approach is used in [Efros and Freeman 2001] to compute the minimum error boundary cut in order to hide the seam in the synthesized image. This method is limited in that the seam is processed sequentially, considering each row of pixels in turn. The *Graphcut* method proposed in [Kwatra et al. 2003], on the other hand, removes this limitation by using a graph cut technique, computing the minimum cut using the maxflow algorithm to find the least visible cutting path. This method can also be easily extended to higher dimensions.

Hybrid methods, e.g. in [Nealen and Alexa 2003], firstly use patch-based texture synthesis, possibly with an adaptive patch size, and in those overlapping areas where the synthesized results are worse than a specified threshold, pixel-based resynthesis is performed to further improve results.

Our current implementation is based on a pixel-based approach, but extending the principle to a patch-based or hybrid approach would not be difficult.

Various research aims to synthesize image textures directly on surfaces. The approach in [Turk 2001; Wei and Levoy 2001b] densely tessellates the surface with sample points and assigns synthesized attributes (e.g. color) in a per-point manner. Such approaches often require surface marching or similar operations, which are not very efficient. These methods can be considered as direct extensions of planar texture synthesis methods. A texture synthesis method based on a texture atlas was proposed in [Ying et al. 2001]. Praun *et al.* proposed a patch-based method that uses alpha-blending to hide cross-boundary seams [Praun et al. 2000]. Soler *et al.* [Soler et al. 2002] give a patch-based approach that covers the surface with image patches in a hierarchical way. Zelinka *et al.* [Zelinka and Garland 2003] compute a *jump map* during a preprocessing phase; this data structure allows rapid location of candidate matches, allowing textures to be directly synthesized on the surface. Region growing is used to decide the best vertex order for synthesis. This approach can synthesize textures on quite large models at interactive rates. The latter two methods do not synthesize new textures, but only assign texture coordinates to vertices of the original mesh. While they are suitable for stochastic image texture synthesis, they cannot easily be extended to perform geometric texture synthesis.

Various recent work considers the transfer of geometric textures or details. Sorkine *et al.* [Sorkine et al. 2004] propose an approach based on the properties of Laplacian coordinates, establishing the correspondence by parameterization and warping under a few specified constraints. Their approach deals with geometric detail transfer, rather than statistical textures. The similarity-based image synthesis method has been extended to geometric surfaces in [Zelinka and Garland 2004] using a polar sampling pattern called *geodesic fans*. This approach is similar to a generalized displacement mapping. Bhat *et al.* [Bhat et al. 2004] give an approach that synthesizes similar geometric textures from examples, which is similar to our method. Their method extends the idea of image analogies

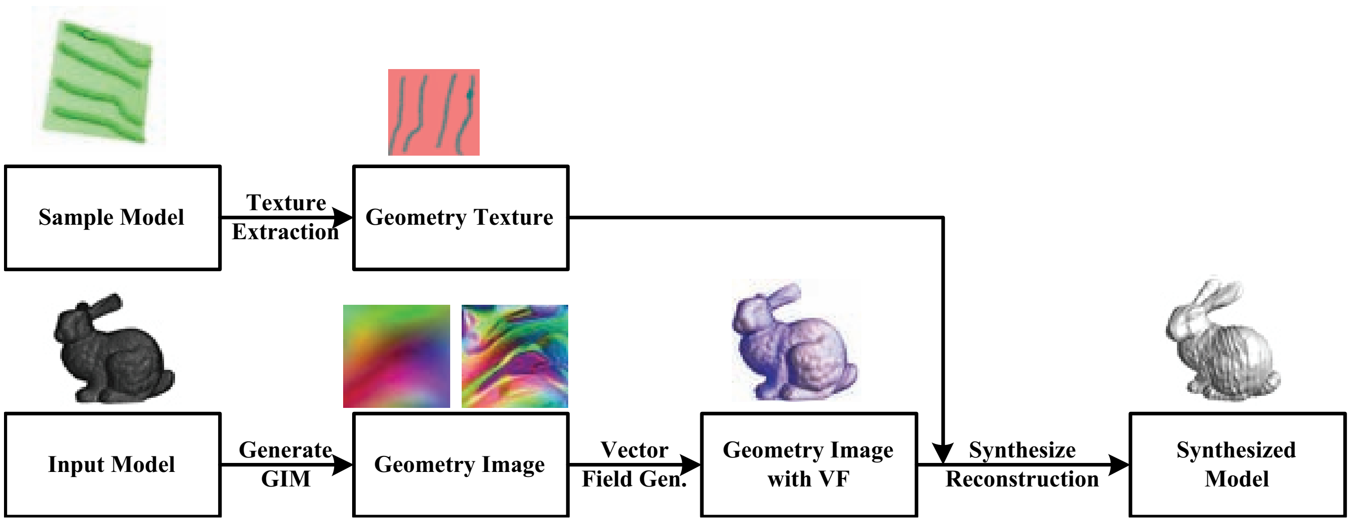

Figure 2: Algorithm overview

to surfaces; however, their approach is based on volume analogies and thus requires representation conversion if the input model is a mesh. As in the case of image analogies, their approach requires as input a pair of example models with and without texture, whereas our approach only needs a single model with the desired geometric texture.

Using traditional *image* texture synthesis methods to add geometric textures is often difficult because the topological structure of the sample model and the input model may be quite different. Because we use the *geometry image* representation, our method is much simpler as it works on a regular domain. It can efficiently achieve high quality results while providing user control. If we use an *image* as a texture source, we do not explicitly require texture extraction and reconstruction step. On the other hand, the use of regular, densely sampled geometry images allows us to efficiently and compactly carry out texture transfer for surfaces and textures with high-frequency geometric detail.

3 Algorithm Overview

Restating the problem, our aim is to take an input model and a textured sample model, and to transfer the texture from the sample model to the input model to produce an output model. Our method is outlined in Fig. 2.

For both the sample model and the input model, the first stage of processing is to generate a geometry image representation. The idea is basically the same as in [Gu et al. 2002], although we also use a different method of seam computation. If the model is of non-zero genus, we first cut it using a contraction scheme similar to that used in [Gu et al. 2002]. If the model is highly curved, containing some “extrema” or protrusions, we can use an iterative method similar to [Gu et al. 2002]. Alternatively, we can instead use a variation of the method in [Zhu et al. 2003] to add additional cuts. The whole cutting process makes each model homeomorphic to a disk. This disk is mapped topologically to a rectangular domain. The skeleton-based seam computation from [Zhu et al. 2003] has the advantage that it avoids the expensive iteration needed by [Gu et al. 2002]; also the resulting seams are shorter given the same extrema.

Next, we parameterize the model in a planar domain. We make some slight modifications to the original approach described in [Gu et al. 2002]. The parameterization method employed to generate geometry images aims to prevent large stretching to avoid undersampling, but at the same time a quasi-conformal mapping is also desirable. These two aims are contradictory for non-developable surfaces, so a compromise is necessary. For a given sampling resolution, the stretch minimization method proposed in [Sander et al. 2003] represents the geometry well. Other parameterization methods with similar attributes are also possible. However, scaling factors always vary across the model, as do local rotations. We need to devise some technique to efficiently compensate for such effects when synthesizing textures. This technique can further be used to advantage to provide user control of the alignment and scaling of geometric textures. These techniques are explained in detail in Section 4.

For sample models, although it is possible to use the full model as input, in practice, it is most likely that we will wish to use a small portion or patch of the model’s surface containing the desired geometric texture. This is possible because textures often have a short range stochastic structure, and so small patches suffice to be representative. Furthermore, it is less satisfactory to extract textures from highly curved surfaces, as distortion or other artifacts may result. For model stylization and artificial texturing applications, it is possible to extract the desired geometric details from planar or almost planar surfaces. Note also that sample models generally do not need to be converted exactly to geometry image representation. Instead, if we can map the texture to a planar domain, we can then cut out a rectangular portion of the domain to use as a source for texture synthesis. This can result in lower distortion than if we correctly compute a true geometry image for the sample patch; it also simplifies later processing as we can then conveniently neglect the distortion in the extracted texture for simplicity. We use a method based on that in [Lévy et al. 2002], but other parameterization approaches with natural boundaries could also be applied.

We next need to extract the geometric texture from the parameterized sample model. Intuitively, geometric details are *high-frequency* components of the corresponding geometry image. The sample model itself can be considered as a combination of a base mesh (representing the basic geometric shape) and geometric details which we call the geometric texture. Similar ideas are used in image editing [Oh et al. 2001], where images can be treated as a combination of low-frequency lighting variation and high-frequency image textures. We first use a smoothing process to approximate the base mesh, after which, based on the parameterization of the base mesh, we create a series of consistent local coordinate systems, and encode the geometric texture in the regular domain.

Using user defined orientation and scaling (if desired), as well as scaling of the parameterization, for efficiency we next precompute a set of texture samples with discretized orientation and scaling.

Using the extracted geometric textures, we then apply them to the input geometry model to create a textured geometry model. In order to keep the texture consistent across the cut boundary, we use a two-pass approach. First we synthesize the texture ignoring the boundary, after which we resynthesize the texture in regions near cut boundaries in a way which tries to enforce consistency for a local neighborhood, based on the already synthesized results in that neighborhood. This process can be repeated several times to further improve the synthesized results.

Finally, the synthesized textured geometry image is converted into an output surface mesh model. This may be remeshed for rendering efficiency if necessary.

Section 4 gives further details of the key steps above: texture extraction, boundary-consistent texture synthesis, handling orientation and scaling, and reconstruction.

4 Algorithm Details

4.1 Texture Extraction

The purpose of texture extraction is to extract geometric details from a sample model or patch, giving a representation suitable for synthesis and transfer. Various previous work deals with separation of details from a base mesh. For example, Biermann *et al.* [Biermann et al. 2002] studied this in the context of cut-and-paste editing, using a method based on multi-resolution subdivision surfaces; further such work can be found in the references in their paper. Kobbelt *et al.* give a mesh smoothing approach for extracting a base surface [Kobbelt et al. 1998]. Their method could be adapted for use here. As geometric textures are statistical in nature, a small patch would be sufficiently representative.

Depending on different application requirements, we suggest the use of one of three alternative approaches to texture extraction: namely, smoothing and differencing, planar parameter domain sampling, and height field sampling. These are efficient and easy-to-implement, as the computations are mostly based on a regular grid and so traditional image processing techniques can be used.

In order to take advantage of geometry images, we encode the extracted texture at each grid point of a regular grid as a vector in a well-defined local coordinate system. We call the resulting vector at each grid point a *geometry texel*. Such a representation can be considered to be a variant of a geometry image.

4.1.1 Smoothing and Differencing

This method extracts the texture as follows. Given a sample model or patch already in geometry image representation, we need to decouple the shape of the base mesh from the geometric texture details. We perform a smoothing operation to estimate the shape of the base mesh, filtering out the geometric textures, using a similar idea to that in [Oh et al. 2001]. In order to prevent sharp features in the base mesh from being filtered out as texture, we instead use *bilateral filters* proposed in [Tomasi and Manduchi 1998]. Note that because we have geometry images, we do *not* need to use bilateral filters adapted to meshes (see e.g. [Fleishman et al. 2003]) whose use can lead to topological problems.

Let us denote by $p_{i,j}$ the position of the grid point with coordinate (i,j), where $p=(x,y,z)$ is a vector in R^3. For the particular grid point at (i_0,j_0), we define the relative weight of any grid point at (i,j) with respect to (i_0,j_0) as:

$$w_{i,j}^{(i_0,j_0)} = \alpha_{i,j}^{(i_0,j_0)} \beta_{i,j}^{(i_0,j_0)}, \tag{1}$$

where

$$\alpha_{i,j}^{(i_0,j_0)} = \exp\left\{-\frac{(i-i_0)^2+(j-j_0)^2}{2\sigma_1^2}\right\} \tag{2}$$

and

$$\beta_{i,j}^{(i_0,j_0)} = \exp\left\{-\frac{||p_{i,j}-p_{i_0,j_0}||_2}{2\sigma_2^2}\right\}. \tag{3}$$

Here α is a weight measuring nearness in the grid while β is a weight measuring position vector similarity in 3D. $w_{i,j}^{(i_0,j_0)}$ combines these two. σ_1 controls how rapidly weights drop off as grid points get farther apart, while σ_2 has a similar effect for position vectors.

Then, the smoothed position of $p_{i,j}$, denoted by $\bar{p}_{i,j}$ is computed as follows (assuming a grid resolution of $N\times N$:

$$\bar{p}_{i,j} = \frac{\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} w_{u,v}^{(i,j)} p_{u,v}^{(i,j)}}{\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} w_{u,v}^{(i,j)}} \tag{4}$$

If the sample patch is not too highly curved, we assume that the bending of the base mesh can be neglected when extracting the texture. This holds true in many real situations as small geometric distortions are not very noticeable.

We construct a local coordinate system at each grid point. This comprises the smoothed mesh normal direction $\bar{n}$, a local x direction $\bar{x}$, and a third orthogonal vector $\bar{y}$. We can approximate the unit normal vector $\bar{n}_{i,j}$ at grid point (i,j) using a 1-ring neighborhood of triangles. We next fix a local x direction, $\bar{x}_{i,j}$ using

$$\bar{x}_{i,j} = \frac{\bar{p}_{i,j+1}-\bar{p}_{i,j-1}}{2} \tag{5}$$

(this should be made orthogonal to $\bar{n}_{i,j}$ by subtracting its component in the direction of $\bar{n}_{i,j}$, and then normalized). For $j=0$, we simply use $\bar{x}_{i,j}=\bar{p}_{i,1}-\bar{p}_{i,0}$, and $j=N-1$ is similar. We then compute the $\bar{y}$ direction using

$$\bar{y}_{i,j} = \bar{n}_{i,j}\times\bar{x}_{i,j}$$

.

To find the texture, the difference vector $p_{i,j}-\bar{p}_{i,j}$ is measured in this local frame as a 3-tuple $(dx_{i,j}, dy_{i,j}, dn_{i,j})$. The encoded texture is recorded as a regular grid of 3-tuples.

4.1.2 Planar Parameter Domain Sampling

For almost planar sample patches, which are common in applications, we may use other approaches to extract the geometric texture to reduce possible distortions. We can fit a plane to the patch using a robust regression method; we use the *RANSAC* approach [Fischler

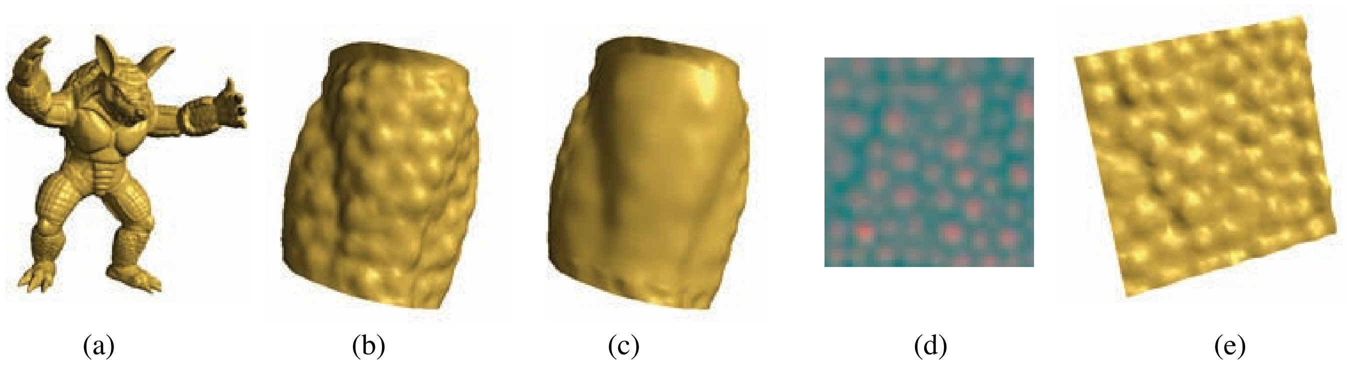

Figure 3: Geometric texture extraction using smoothing and differencing: (a) is the armadillo model, (b) gives a curved sample patch, (c) shows the smoothed surface, (d) presents the extracted geometric texture image, and (e) shows the reconstructed texture on a plane.

and Bolles 1981]. A set S of a few points is randomly chosen, and a plane H_S is fitted using least-squares. We then count the number $I(S)$ of data points that are within a predefined threshold distance δ of H_S. This process is repeated for a sufficiently large number of sets S, and we find the S with the largest number of $I(S)$. Then, a traditional least squares fit is done only using points in $I(S)$ to find the plane H. Note that we do not use vertices of the original mesh for sampling the mesh, as such points are generally not evenly distributed. Rather, we generate sample data points randomly according to the area.

Next we project the boundary of the sample model patch onto this plane. Any point on H can be represented as $v_0 + xe_1 + ye_2$, where e_1 and e_2 are two normalized orthogonal vectors in the plane, and v_0 is a reference point in the plane. Each vertex v on the boundary of the patch, having position p_v, is projected onto the plane by calculating:

$$(x,y)_v = (p_v \cdot e_1, p_v \cdot e_2).$$

We then parameterize the patch with the boundary parameters fixed as the projected coordinate values. Then, the largest square in the parameter domain is detected, and we then regularly sample the texture inside this square. The parameterization step determines a piecewise linear mapping from the parameter domain to the patch. We use this mapping to find the corresponding point on the patch. A local coordinate system is trivial to find: we always use $(e_1, e_2, e_1 \times e_2)$. A difference vector between the sample model and the plane is measured in this coordinate system, thus giving a similar representation to the previous approach.

4.1.3 Height Field Sampling

In some cases, the texture may be sufficiently simple that each point above the base surface corresponds to a single point on the sample model, in which case a height field suffices to describe the texture. For example, it is reported in [Wang et al. 2003] that mesostructures on tree barks can be modelled as a height field. Height field sampling is done in a similar way to the previous approach, except that vertices can be projected directly on to the base plane without the need to perform a parameterization.

4.1.4 Observation and Example

If the sample model also contains traditional *image* texture information, as well as geometric texture, we can sample the image texture together with the geometric texture, giving a 6-tuple: (dx, dy, dn, r, g, b) where r, g, b are the red, green and blue components of the sample point respectively.

The whole process is illustrated in Fig. 3, which shows an example where a patch cut from the leg of the armadillo model is taken as the sample model.

4.2 Boundary-Consistent Texture Synthesis

If the input model is not homeomorphic to a disk, it is necessary to introduce boundaries using one or more cuts. The geometry image representation handles this issue and makes corresponding pairs of boundaries exactly meet. After adding geometric details, we need to again keep the boundary consistent. This is accomplished by resynthesis of textures near cut boundaries to hide the seams.

4.2.1 Initial Geometric Texture Synthesis

As in [Wei and Levoy 2001a] and other image texture synthesis methods, we need to synthesize a regular grid of geometric textures with a resolution equal to that of the geometry image of the input model. Two steps are used. Firstly, we fill each grid point with geometric texture samples randomly chosen from the sample model grid, which ensures that the initial random textures have the same statistics as the sample model. Next, we build Gaussian pyramids of the sample and the destination geometric textures; these pyramids do not go up to the highest level but just a few levels are used. We synthesize starting at low resolution and ending at high resolution. At the lowest resolution, we use a causal Γ-shaped template to update the current texel, by finding the nearest match in the sample to give the newly synthesized texel. At higher resolutions, we take a combined template using a Γ-shaped template for the current resolution together with full neighborhood information for lower resolution texels. In our implementation, we use *approximate nearest neighbor matching* [Mount 1998] to accelerate the matching process.

The above synthesis approach gives a synthetic geometry image without boundary consistency, which we address next.

4.2.2 Boundary Resynthesis

To get boundary consistency, the above method is modified slightly.

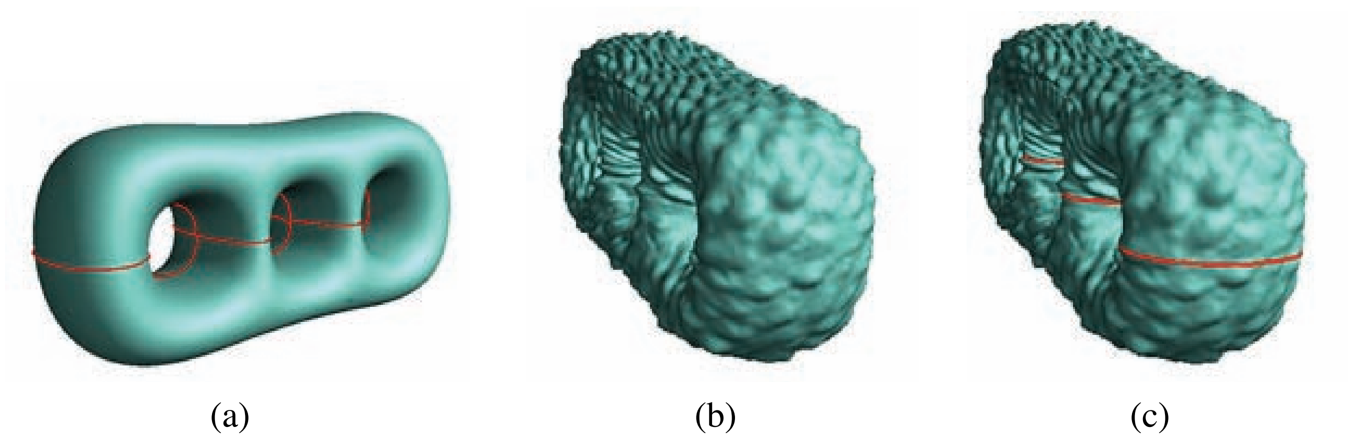

Figure 4: Synthesis results across the cuts. (a) gives a simple genus-3 model with a few cuts, (b) shows the synthesized result and (c) is the synthesized model with cuts highlighted.

We apply a resynthesis stage at each level of synthesis. To begin, we construct a *boundary fixing pyramid* similar to the Gaussian pyramid used in the previous texture synthesis stage. Such a hierarchy can easily be constructed provided that we use an initial parameterization for the input geometry image which fixes the cut-nodes at a low resolution grid spacing equal to that of the highest level of the pyramid. Subsequent sampling can then be done at a higher resolution. For example, the parameterization may place the cut-nodes on a 65×65 grid, while sampling may be done on a 257×257 grid. The boundary fixing pyramid is constructed by subsampling the boundary grid points; the chosen geometry image representation makes sure that after this down-sampling the boundary values still match.

Next, for each level of synthesis, to hide the seams across the boundary, we use a relatively small full neighborhood template with size e.g. 5×5. We resynthesize texture samples one by one in scanline order for those texture samples near the boundary of the grid, in a boundary strip of size in accordance with the characteristic size of the texture. We put the center of the template at each near-boundary position, and collect information from both sides of the appropriate boundary.

The resynthesis process can be repeatedly performed several times for better results. After each iteration, to make identified boundary texels have identical values, we randomly choose one and propagate its value to the other(s).

For other problem cases, e.g. texels in the corner of geometry images, or where two cuts meet, we use a heuristic approach. In practice such cases are rare and do not cause problems in the result.

Fig. 4 shows that our texture transfer method generally works well even across cuts in the surface of the input model; this is demonstrated using texture from the armadillo's leg shown in Fig. 3. The left hand illustration shows the cut made in a simple genus-3 model to convert it into a geometry image. The middle illustration shows the synthesized model while the right hand illustration shows the location of the cut. Occasionally, however, it may happen that there is large anisotropic distortion of the parameterization near the boundary, which is different on either side. Our approach cannot fully compensate for this, resulting in small remaining visual discontinuities, as can be seen in Fig. 4.

4.3 Handling Orientation and Scaling

The sampled texture often has an implicit orientation and scaling which must be respected when the texture is applied to the input model. To control the orientation, the user defines an orientation at selected key points of the input surface, and from these an orientation field is interpolated across the surface. This controls the texture orientation during synthesis. Secondly, the parameterization process as well as the geometric shape of the input model cause local variations in scaling of coordinates which must also be taken into account when synthesis is performed. In this Section, we explain how this is done.

We use a set of texture samples with different orientations and scaling during the synthesis process. In practice, we use a set of 8 equally spaced orientations and 3 levels of scaling, which are found to be sufficient. This set of samples is built in advance before synthesis. For efficiency reasons, again for each orientation and scaling, we also build in advance an approximate nearest neighbor searching structure which is used to decide how to propagate the texture, based on the texture synthesized so far.

We need to estimate the scaling factor at each grid point of the input geometry image. For a grid point v_0, assuming its 4-connected neighbors are v_1, v_2, v_3, v_4, we can compute the local scaling factor as:

$$S(v_0) = \frac{1}{4}\sum_{i=1}^{4} ||p_{v_0} - p_{v_i}||_2 .$$

We can then compute the average scaling factor $\bar{S}$ for the whole input geometry image, and a *relative scaling factor* R for position v_0 as:

$$R(v_0) = \frac{S(v_0)}{\bar{S}}$$

During synthesis, we estimate the relative scaling factor near the local position to be synthesized. In a similar way to the idea in [Tonietto and Walter 2002], suppose the 3 scaled sample textures are T_k, $k = 1, 2, 3$, where T_2 has the original size, T_1 is a scaled up version, and T_3 is a scaled down version. We compute $\text{level}(v_0) = \log R(v_0) + 2$. If the level is outside the range of 1 to 3, we use a value of 1 or 3 as appropriate; otherwise if it falls between two levels, say, s and $s+1$, we then choose T_s with a probability

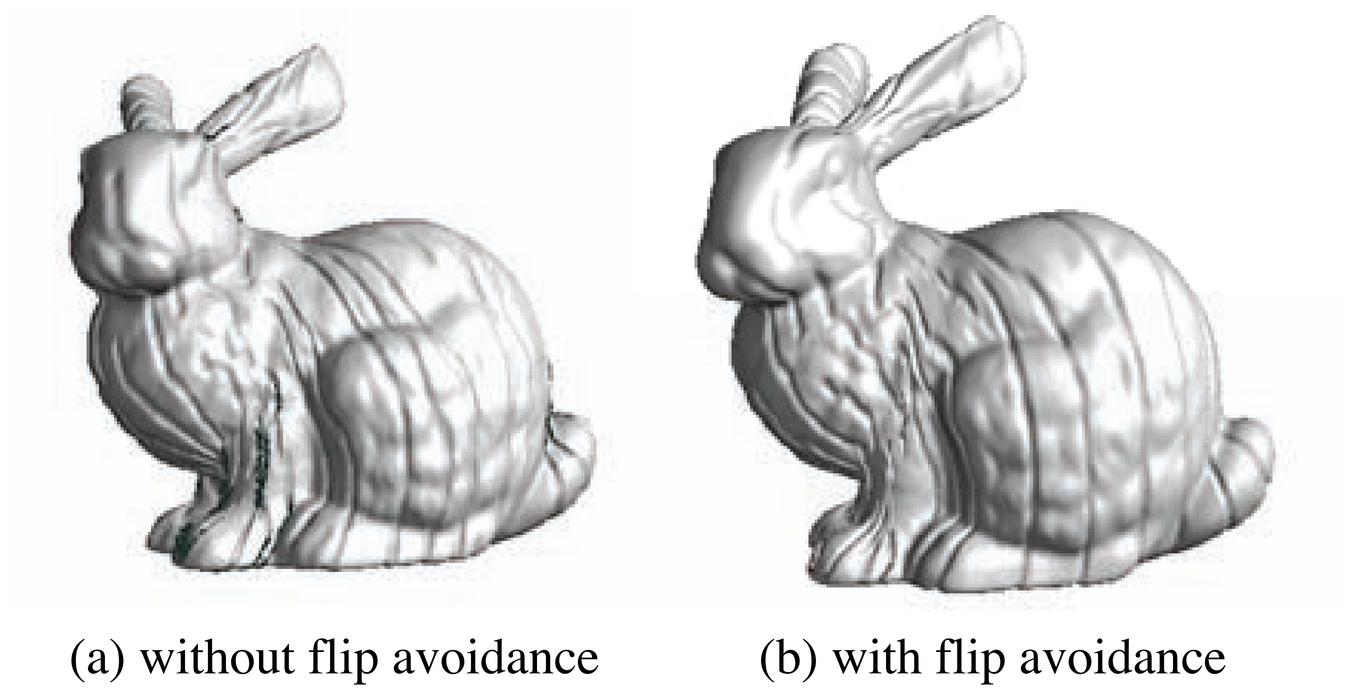

Figure 5: Results with and without flip avoidance

$s+1-\text{level}(v_0)$ and otherwise choose T_{s+1}. The idea can be generalized to use more levels of scaling, but at a greater computational cost. We find 3 levels acceptable in practice.

To define orientations, the user needs to define a few key tangent vectors on the mesh, and then a diffusion process is carried out to interpolate an orientation vector field on the mesh using similar ideas to those in [Turk 2001]. This is done as follows: vectors except for the preset ones are initialized to zero. For each vertex v_0 on the mesh, we consider its 1-ring neighborhood $v_1,\ldots,v_k$, map the corresponding vectors to the tangent plane at v_0—call them f_i—and iteratively update the vector field at v_0 using

$$f_0' = f_0 + t\sum_{i=1}^{k} W_i(f_i - f_0).$$

Here t is a step-size control; in practice, using a relatively small t (e.g. 0.1) ensures stability. W_i is a weight, proportional to the reciprocal of the edge length and normalized to have sum one. We iterate until the vector field converges.

For faster convergence, we construct a pyramid and compute the vector field from coarser to finer levels. The regular nature of geometry images greatly simplifies this process making it straightforward.

Next we map the vector field to the 2D parameter domain. Let us denote the surface point by $X(u,v)$ where $(u,v)\in R^2$ is the parameter value. As described before, we can easily estimate $\partial X/\partial u$ and $\partial X/\partial v$ by considering the 4-connected neighbors in the grid. We can then project the specified vector v onto the parameter domain, which reduces to finding x and y satisfying $xX_u + yX_v = v$; this vector then needs to be normalized. To get correct results at the boundary, the vector fields for vertices on either side of it should be identical when mapped to the parameter domain. On each iterative vector field update, we thus average the vectors for corresponding vertices. (Simply doing this at the end would cause the boundary vectors to be inconsistent with their neighbors' vectors.)

During the synthesis process, we apply a probability decision to choose the sample orientation which gives the best match, in the same way as for scaling.

4.4 Reconstruction

The last step is to add the synthesized geometric textures to the input model to get the output model. We need to construct a series of continuous local coordinate systems to apply the synthesized geometric texture to generate the real positions for each grid point. The local coordinate system is computed in the same way as for texture extraction, and again we use a system of $(\hat{x}_{i,j},\hat{y}_{i,j},\hat{n}_{i,j})$, where $\hat{x}$, $\hat{y}$ and $\hat{n}$ represent the two iso-curves and normal direction, all normalized.

For a grid point on the input mesh $\hat{p}_{i,j}$, we compute the new position vector after addition of texture detail as:

$$\hat{p'}_{i,j} = \hat{p}_{i,j} + \hat{dx}_{i,j}\hat{x}_{i,j} + \hat{dy}_{i,j}\hat{y}_{i,j} + \hat{dn}_{i,j}\hat{n}_{i,j}$$

where $\hat{dx}$, $\hat{dy}$, $\hat{dn}$ are synthesized geometric texel values at each grid point.

4.4.1 Flip Avoidance

In practice, if we simply modify the positions as above, in highly curved areas, the vertices may be modified in an inconsistent manner, resulting in flipped faces. This can happen if the surface is

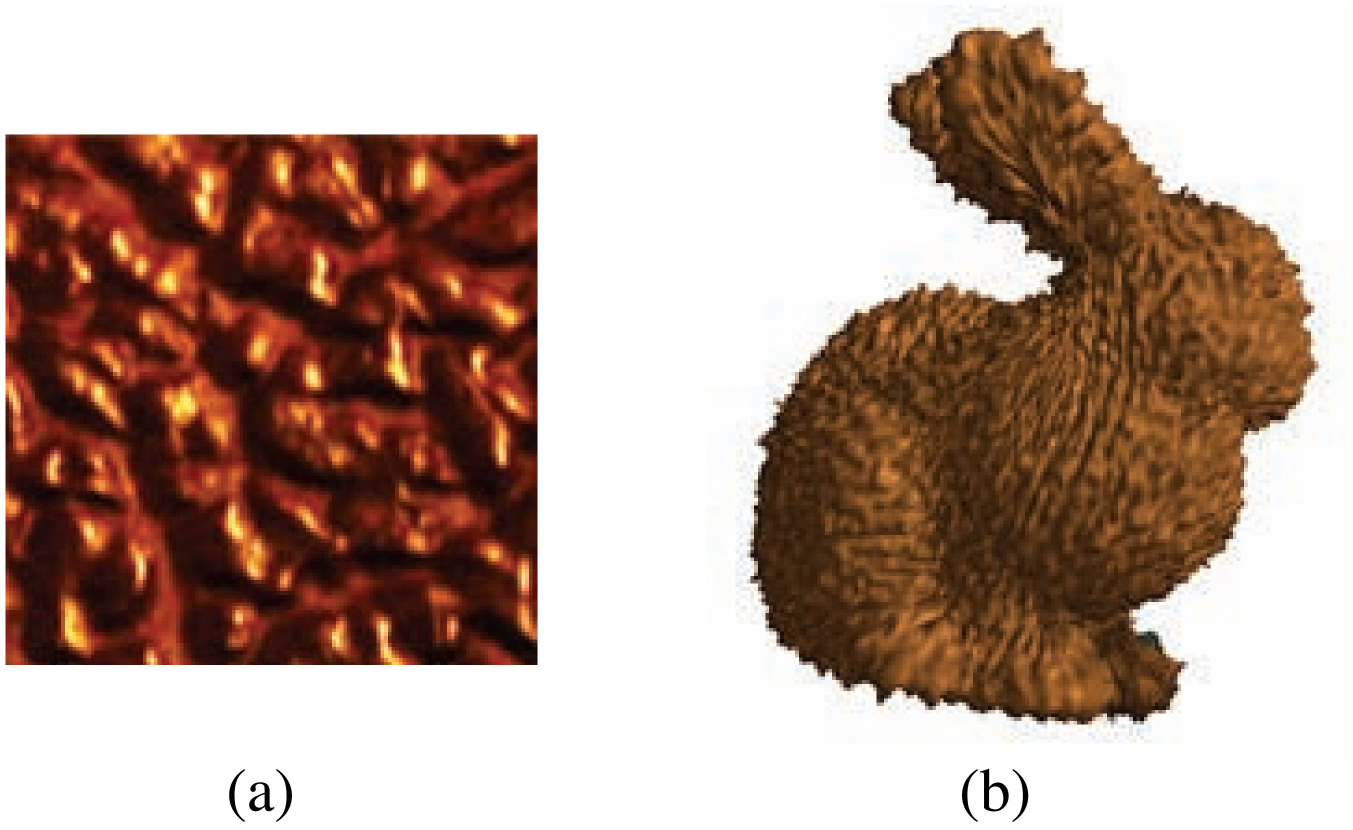

Figure 6: Geometric texture transfer using an existing image as height field. (a) an image of a piece of leather (b) transferred to bunny model.

highly curved and the applied texture is relatively large. We can avoid flipping by reducing the amount of modification of the surface. Thus, in practice we add the texture in a series of increasing steps, rather than all at once. If flipping is detected (the normal change is larger than a threshold, e.g. 150°), the last step is unwound for the related vertices, and further growing there is inhibited. This works well in practice, as shown in Fig. 5.

5 Experimental Results

Geometric textures have a variety of different applications, as noted in the introduction. We present here a few experimental results which demonstrate the effectiveness of our methods in different circumstances.

The first example involves model stylization. We transfer a simple manually designed geometric texture consisting of a few lines to a destination model and cover it with such geometric details. See Fig. 1.

The second example shows how an existing *2D image* can be used to produce a height field for model stylization: *intensity* in the image is converted to *height* above a plane. We used an image of a piece of leather to create a height field which was then applied to the bunny model—see Fig. 6. Such an approach provides a plentiful source of geometric textures for application to models.

The third set of examples shows the ability to transfer geometric textures from one model to another. See Fig. 7. Two different textures have been taken from the lower and upper leg of the armadillo model. The first has been applied to a tyrannosaur model and a gargoyle model; the second has also been applied to the tyrannosaur model. This shows the effects of (a) applying the same texture to different models, and (b) applying different textures to the same model. Note that models which originally contain geometric details need to be smoothed before geometric texture synthesis. The neck on Fig. 7(c) contains denser detail than other regions as there is a large anisotropic distortion in this region which cannot be fully compensated for by our current method. The blurring seen on the nose in Fig. 7(e) is due to its representation by a relatively small area in parameter space. The parameterization of geometry images has isotropic scaling, and we use a scaled-up or scaled-down version to synthesize regions with different scalings, leading to the blurring seen.

The fourth set of examples in Fig. 8 shows the gargoyle model with transferred geometric texture with globally modified density. We achieve this by simply offsetting the scaling by a small amount, to globally increase or decrease the geometric texture density. The final set of examples in Fig. 9 shows the effect of different vector fields on the texture transfer process. Several key vectors were manually selected for each example, and interpolation was used to derive the vector field at each grid point. It can be seen that vector field can be used to guide the synthesized texture as desired.

We implemented our algorithm on a PC with a Pentium IV 2.4GHz CPU using C++. The models used for the tests described in this paper were converted to 257×257 geometry images. A three-level Gaussian pyramid was used, and the sample models were sampled

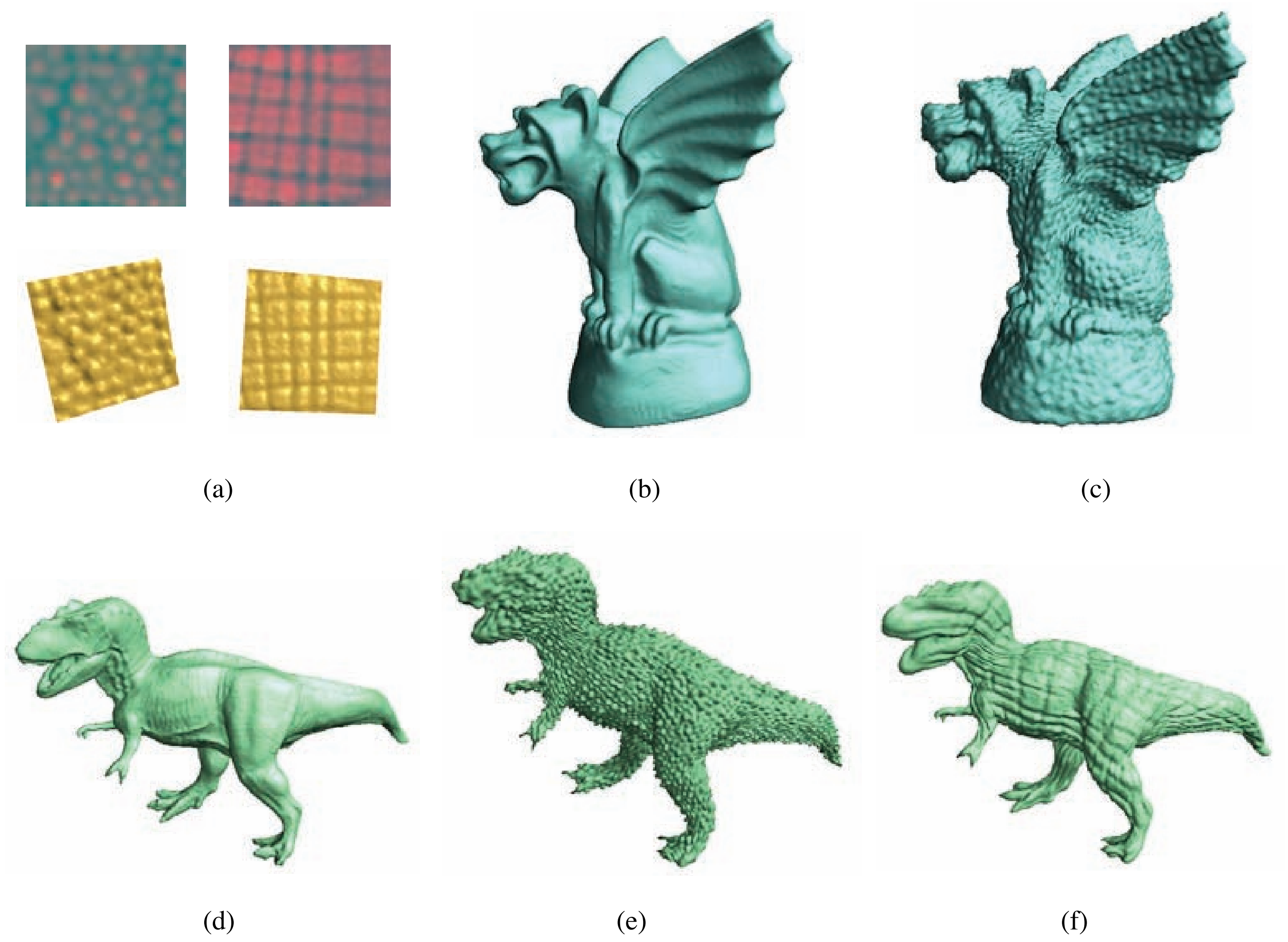

Figure 7: Geometric texture transfer. (a) shows two patches containing geometric textures cut from the armadillo's leg, (b) shows a gargoyle model, (c) shows the gargoyle model with synthesized texture added, (d) shows a tyrannosaur model, and (e) and (f) show tyrannosaur models with two different geometric textures.

either at 64×64 or 128×128 resolution. For texture extraction, we cut two patches from the armadillo model, each containing about 8,000 vertices. The extraction step took about 10 seconds, most of which was used in parameterization. The time taken to reconstruct the model was around 0.6 seconds. The synthesis time, including training time for the approximate nearest neighbor structure (ANN) was dominated by nearest neighbor queries. Using appropriate approximation error bounds for the ANN, this step took 2 to 10 seconds, depending on the resolution of the sample patch, to produce synthesized results without visible degradation of quality.

6 Conclusions

In this paper, we have presented an efficient algorithm to transfer geometric textures from one model to another, which requires very little user intervention while providing opportunities for user control, if desired. Our method is much faster than most previously reported, especially after the preprocessing stage of constructing query structures. The synthesis and reconstruction stages can be done in a few seconds. The geometry image representation has been found to be suitable for representing surfaces having geometric textures. However, there are still some limitations in our current work. Geometry images usually have relatively large distortions, which may include anisotropic scaling, and these cannot be fully compensated for by our approach. Our method could be extended to use a conformal geometry image representation, and we would expect better results due to the avoidance of anisotropic scaling and usually lower distortion; we intend to explore this in future. Our method is based on use of parameterization of the mesh representation, and so the topology can not be changed after synthesizing geometric texture on a surface.

Geometry images are discrete samples of a continuous surface, and are limited by the resolution chosen. While higher resolution geometry images can be used, this is inefficient if only parts of the final textured object have high levels of detail. Instead of using a regular grid for the geometry image, it might be worth investigating the use of a *geometry quadtree* to overcome this problem. Gaussian pyramid synthesis algorithms are suitable for implementing this approach as the synthesis is actually done from coarse to fine levels. Difficulties lie in organizing the quadtree so that it remains relatively regular, and achieving continuity between different levels of

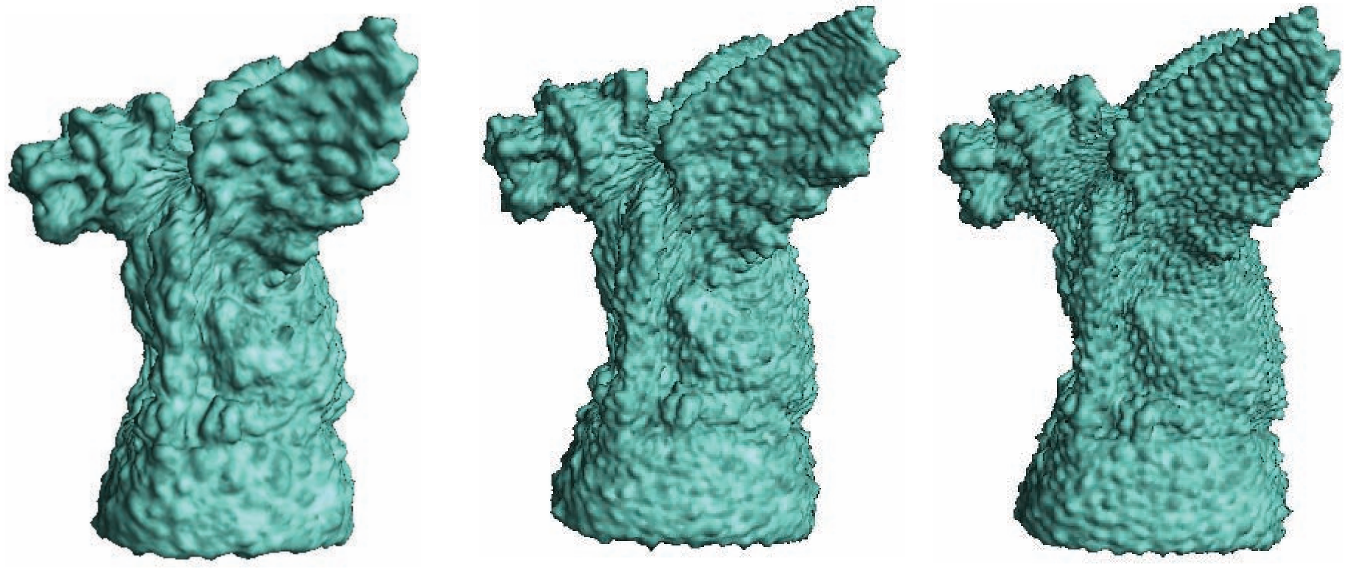

Figure 8: Geometric texture transfer with different density.

detail. The disadvantage is that the fully regular structure of geometry images will be lost.

Further issues remain to be explored. Firstly, the idea in this paper could be implemented as an interactive geometric painting tool for model editing—a cloning-like tool for geometric detailing would be very useful. Secondly, it is possible to extend the idea to different synthesis methods, especially patch-based approaches. Thirdly, spherical geometry images [Praun and Hoppe 2003] contain more regular boundary connectivity which will possibly provide smoother results across the boundary of an object. Following the global conformal parameterization work of [Gu and Yau 2003; Gu et al. 2004], it is also possible to implement our algorithm on conformal geometry images: a multiple-patch representation, in which each patch is a regularly sampled rectangular domain. The distortion could be lower due to the introduction of multiple patches, and the nature of conformality. These can be exploited in future work.

Acknowledgements

We thank the anonymous reviewers for their valuable comments. The work was supported by the Natural Science Foundation of China (Project Number 60225016, 60321002, 60333010).

We gratefully thank Stanford University and Cyberware for use of models in this research.

References

ASHIKMIN, M. 2001. Synthesizing natural textures. In *ACM Symposium on Interactive 3D Graphics 2001*, 217–226.

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *Proceedings of Eurographics Symposium on Geometry Processing*, 43–46.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste editing of multiresolution surfaces. In *Proceedings of ACM SIGGRAPH 2002*, 312–321.

EFROS, A., AND FREEMAN, W. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, 341–346.

EFROS, A., AND LEUNG, T. 1999. Texture synthesis by non-parametric sampling. In *Proceedings of IEEE Intl. Conf. on Computer Vision*, vol. 2, 1033–1038.

FISCHLER, M. A., AND BOLLES, R. C. 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the ACM 24*, 381–395.

FLEISHMAN, S., DRORI, I., AND COHEN-OR, D. 2003. Bilateral mesh denoising. In *Proceedings of SIGGRAPH 2003*.

GU, X., AND YAU, S. 2003. Global conformal surface parameterization. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2003*, 127–137.

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry images. In *Proceedings of SIGGRAPH 2002*, 355–361.

GU, X., WANG, Y., AND YAU, S. T. 2004. Computing conformal invariants: Period matrices. *Communications in Information and Systems 3*, 153–170.

HERTZMANN, A., JACOBS, C., OLIVER, N., CURLESS, B., AND SALESIN, D. 2001. Image analogies. In *Proceedings of SIGGRAPH 2001*, 327–340.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of SIGGRAPH 1998*, 105–114.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video texture synthesis using graph cuts. In *Proceedings of SIGGRAPH 2003*, 227–286.

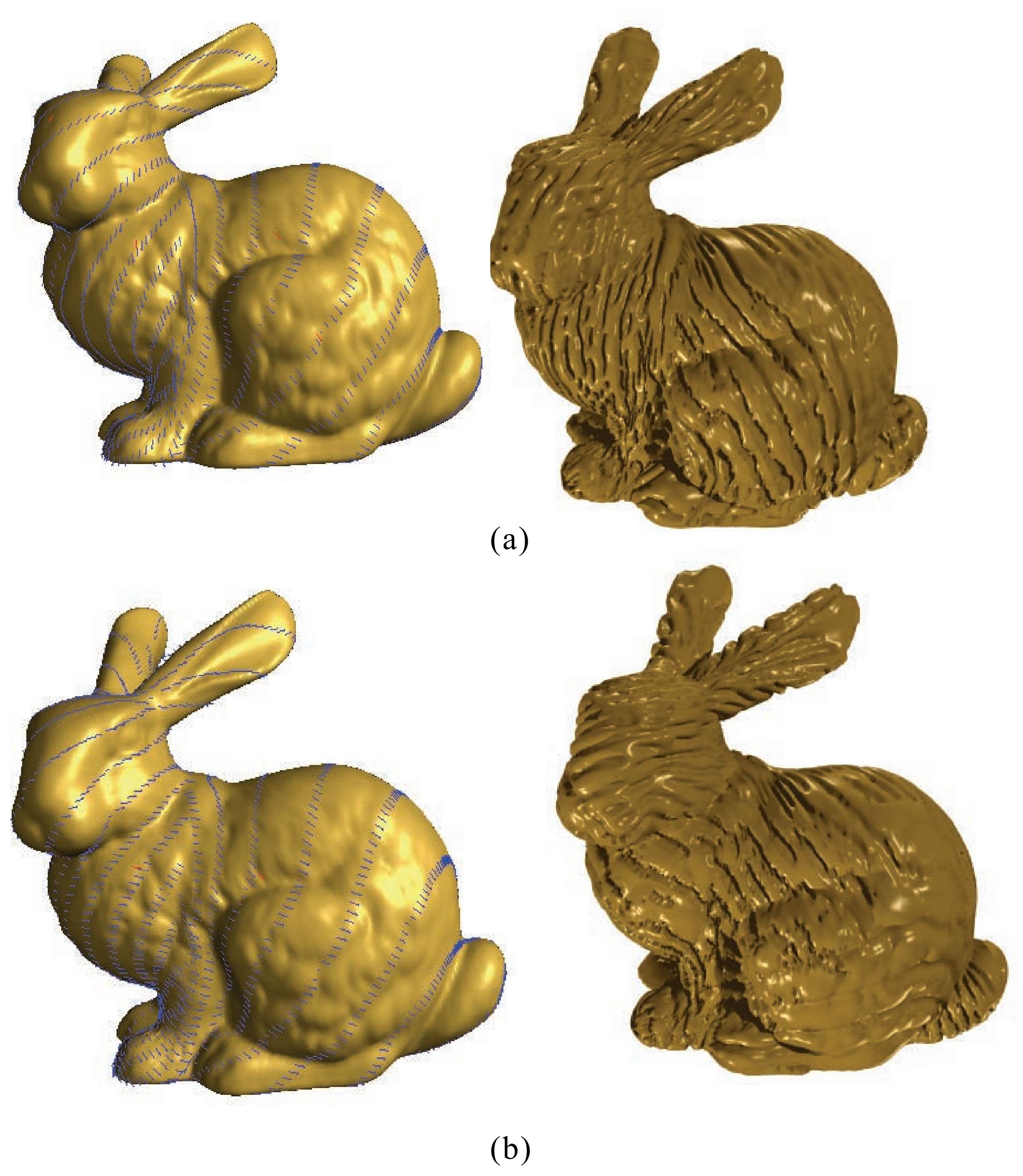

Figure 9: Geometric texture transfer using different vector fields.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of SIGGRAPH 2002*, 362–371.

LOSASSO, F., HOPPE, H., SCHAEFER, S., AND WARREN, J. 2003. Smooth geometry images. In *Eurographics Symposium on Geometry Processing 2003*, 138–145.

MOUNT, D. 1998. Ann library, version 0.1.

NEALEN, A., AND ALEXA, M. 2003. Hybrid texture synthesis. In *Proceedings of the 13th Eurographics Workshop on Rendering*, 97–105.

NO, H. B., SANDER, P., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: A new representation for 3d animations. In *ACM Symposium on Computer Animation 2003*, 136–146.

OH, B., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-based modeling and photo editing. In *Proceedings of SIGGRAPH 2001*.

PRAUN, E., AND HOPPE, H. 2003. Spherical parameterization and remeshing. In *Proceedings of SIGGRAPH 2003*, 340–349.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of SIGGRAPH 2000*, 465–470.

SANDER, P., WOOD, Z., GORTLER, S., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry Processing 2003*, 146–155.

SOLER, C., CANI, M., AND ANGELIDIS, A. 2002. Hierarchical pattern mapping. In *Proceedings of SIGGRAPH 2002*, 673–680.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proceedings of Eurographics Symposium on Geometry Processing*, 179–188.

TOMASI, L., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *Proceedings of IEEE International Conference on Computer Vision 1998*, 839–846.

TONIETTO, L., AND WALTER, M. 2002. Towards local control for image-based texture synthesis. In *Proceedings of Brazilian Symposium on Computer Graphics and Image Processing XV*, 252–258.

TURK, G. 2001. Texture synthesis on surfaces. In *Proceedings of SIGGRAPH 2001*, 347–354.

WANG, X., WANG, L., LIU, L., HU, S., AND GUO, B. 2003. Interactive modeling of tree bark. In *Proceedings of 11th Pacific Conference on Computer Graphics and Applications 2003*, 83–90.

WEI, L., AND LEVOY, M. 2001. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH 2001*, 479–488.

WEI, L., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of SIGGRAPH 2001*, 355–360.

YING, L., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and shape synthesis on surfaces. In *Proceedings of 12th Eurographics Workshop on Rendering*, 301–312.

ZELINKA, S., AND GARLAND, M. 2003. Interactive texture synthesis on surfaces using jump maps. In *Proceedings of the Fourteenth Eurographics Symposium on Rendering Techniques*, 90–96.

ZELINKA, S., AND GARLAND, M. 2004. Similarity-based surface modelling using geodesic fans. In *Proceedings of Eurographics Symposium on Geometry Processing*, 209–218.

ZHU, X., HU, S., AND MARTIN, R. R. 2003. Skeleton-based seam computation for triangulated surface paramterization. In *Proceedings of Mathematics of Surfaces X*, Springer, vol. LNCS 2768, 1–13.