Interactive Modeling of Topologically Complex Geometric Detail

Jianbo Peng, Daniel Kristjansson, Denis Zorin

Media Research Laboratory, New York University

Abstract

Volume textures aligned with a surface can be used to add topologically complex geometric detail to objects in an efficient way, while retaining an underlying simple surface structure.

Adding a volume texture to a surface requires more than a conventional two-dimensional parameterization: a part of the space surrounding the surface has to be parameterized. Another problem with using volume textures for adding geometric detail is the difficulty in rendering implicitly represented surfaces, especially when they are changed interactively.

In this paper we present algorithms for constructing and rendering volume-textured surfaces. We demonstrate a number of interactive operations that these algorithms enable.

Keywords: modeling, volumetric texture, volumetric rendering.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Curve, surface, solid, and object representations; Geometric algorithms, languages, and systems; I.3.7 [Three-Dimensional Graphics and Realism]: Color, shading, shadowing, and texture.

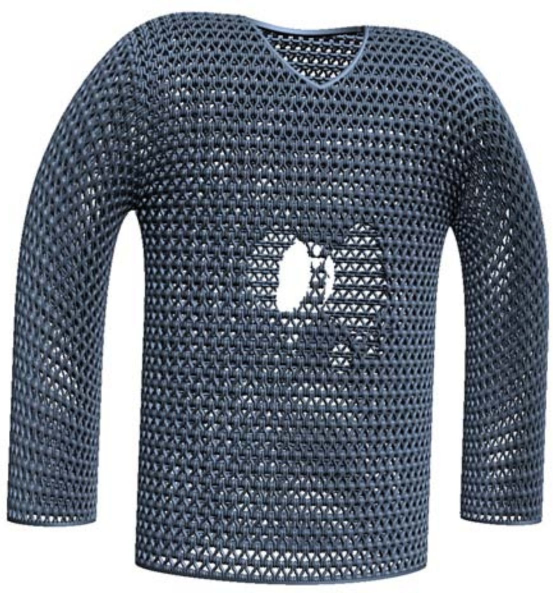

Figure 1: A surface with fine-scale detail added as volume texture.

1 Introduction

Common surface representations work well for objects of relatively simple topology and continuous geometric structure. However, for many types of objects, the local geometry can be highly complex. Examples include fur, bark, cracked surfaces, grilles, peeling paint, chain-link fences and others. In these cases, using meshes or patches to represent small-scale geometry is often prohibitively expensive. But if we ignore the small-scale structure, a complex surface often has a simple overall shape, well represented by a mesh or a smooth surface.

In this paper we describe a combined volume-surface representation for handling geometry of this type, extending the idea of volume textures. Volume textures aligned with the surface make it possible to represent geometrically and topologically complex details in implicit form, encoding the surface as an isosurface in a layer. This idea was explored by a number of researchers in the past (see Section 2).

This representation has important advantages:

- It uses simple and efficient data structures (textures) to represent highly irregular geometry.
- Small features of high topological complexity can be easily introduced and modified.
- Image processing techniques can be used to modify small-scale geometry without topological constraints.
- Hierarchical representations can be naturally constructed using filtering on volumetric textures.
- One can easily use procedural modeling and simulation to produce complex effects near the surface.

In this paper we focus on two algorithms central to the goal of using this approach in modeling applications. Specifically, in addition to surface parametrization required by 2D texturing, volume textures require parameterizing a region of space near a surface. Most of the previous work on volume textures used techniques such as normal displacement, which results in self-intersections near concave features. We describe an algorithm for computing volume layer parametrizations with a number of desirable properties, which can be used to update the parametrization interactively.

While isosurfaces are convenient for many types of operations, they are much more difficult to render than conventional meshes. We describe algorithms for volume texture rendering that enable interactive manipulation of volume-textured objects. Our algorithm for rendering volume-textured surfaces extends the approach of direct slice-based isosurface rendering for volumes. We take advantage of the programmable graphics hardware to reduce the geometry requirements of the slice-based methods, which is crucial for interactive rendering of volume textures.

Finally, we demonstrate a number of interactive modeling operations that are made possible by these algorithms.

Acknowledgements. We would like to thank the anonymous reviewers for many useful comments. This research was supported in part by NSF awards CCR-0093390, DMS-0138445, ACI-9978147, Sloan Foundation Fellowship, IBM University Partnership Award, NYU Center for Advanced Technology and a donation from the NVIDIA Corporation.

2 Previous work

Our work builds on research in several areas.

Volume textures. The idea of volume textures goes back to the work by Kajiya and Kay [Kajiya and Kay 1989]. Our work was motivated by the work of F. Neyret and co-workers (e.g. [Neyret 1995; Neyret 1998; Meyer and Neyret 1998]) as well as recent work on fur rendering [Lengyel 2000; Lengyel et al. 2001].

Our geometry is to some extent similar to the slab representation used for modeling weathered stone ([Dorsey et al. 1999]) and for volume sculpting in [Agarwala 1999]. ([Dorsey et al. 1999]) uses the fast marching method (e.g. [Sethian 1999]) to construct layers around a surface. Envelope construction [Cohen et al. 1996] provides another alternative. Our method is compared with both in Section 4.

Stable medial axes. Our construction is closely related to the work in vision and medical imaging on using various types of medial axis approximations to analyze shape and extract surfaces from volume data (e.g. [Pizer et al. 1994; Eberly et al. 1994]). In these papers a form of the medial axis of an object implicitly defined by a density function is first constructed without recovering the boundary of the object. Our generalized distance function (Section 4) is similar to some of the medialness functions used to construct stable medial axes. Our work is closest to [Siddiqi et al. 1999] which solves the Hamilton-Jacobi equations for the medialness function on a regular grid to recover a skeleton. [Yezzi and Prince 2002] uses a Laplacian equation solved on a regular grid to compute correspondences between nested surfaces. Our generalized distance function has the property of pruning away insignificant medial axis branches close to the surface (see Section 4). R-functions have been used to achieve similar effects ([Ricci 1973; Rockwood 1987; Pasko et al. 1995]), but with more complex computation based on a CSG representation of the surface.

Direct isosurface rendering. Indirect methods for rendering isosurfaces [Lorensen and Cline 1987; Chernyaev 1995] require significant preprocessing and result in large meshes. Our rendering technique extends the direct volume and isosurface rendering approach based on slicing and 3D textures (or collections of 2D textures), which has origins in [Cullip and Neumann 1993; Cabral et al. 1994; Wilson et al. 1994]. Our work is most closely related to isosurface rendering and displacement map rendering [Westermann and Ertl 1998; Schaufler 1998; Rezk-Salama et al. 2000; Dietrich 2000; Kautz and Seidel 2001]. For volumetric textures a slice-based approach is described in [Meyer and Neyret 1998]. We apply an extension of these techniques to a collection of relatively small distorted volumes. Our technique has the crucial advantage of allowing a more flexible choice of the number of slices per volumetric element, while minimizing visual artifacts associated with insufficient slice density. A more detailed comparison is given in Section 5.

Implicit surfaces and volume modeling. There is an extensive body of literature related to volume-based representations (see [Bloomenthal 1997] for a list of references); some recent important work includes [Frisken et al. 2000; Carr et al. 2001]. Interactive and procedural volume sculpting techniques [Wang and Kaufman 1995; Agarwala 1999; Cutler et al. 2002] can be applied to our surface representation. Most work on volume modeling focuses on volume data in pure form, i.e. objects are represented as level sets of a function defined by volume samples. In this paper we concentrate on techniques which blend parametric and surface representations.

Structured mesh generation. Constructing a collection of layers aligned with a surface is a common problem in structured mesh generation. Mesh generation is a large and complex field aiming to build meshes suitable for a variety of numerical algorithms for solving PDEs (see, e.g. surveys [Henshaw 1996; Bern and Plassmann 2000] and the book [Steinberg and Knupp 1993]). Such meshes often have to satisfy stringent requirements for the algorithms to achieve optimal or nearly-optimal convergence rates, especially for CFD problems, for which object-aligned grids are particularly important [Henshaw 1996].

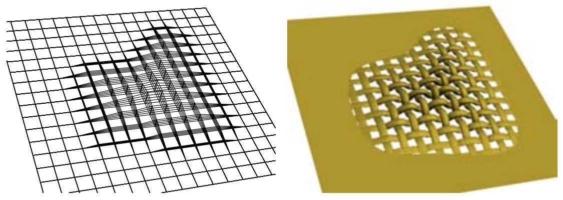
Figure 2: Surface with a volume layer attached.

Our goal is more modest: we aim to construct a shell aligned with the surface efficiently, maintaining nondegeneracy without explicitly minimizing a distortion measure. At the same time, the criteria used to formulate the PDEs in hyperbolic mesh generation methods (volume preservation and orthogonality), are not necessarily the best for our applications.

3 Representation

We refer to the initial surface for which we construct a shell as the *base surface*. We consider shells which are obtained by displacing points of the base surface along line segments defined at vertices, which we call *directors*. At each vertex of the surface, we store shell thickness, the number of shell layers stored and texture coordinates. The shell consists of slabs corresponding to the faces of the mesh or individual patches. Each slab is a deformation of a prism.

Shells can be *exterior* (e.g. for fur modeling), *interior* (e.g. for cracks) and *envelope* with layers located on both sides of the surface. Our technique works for all shell types.

The main additional storage is the 3D textures associated with the surface. The number of layers in the shell corresponds to the number of pixels in the texture in the direction perpendicular to the surface. The alpha channel of the texture defines the effective surface implicitly as the isosurface corresponding to a fixed alpha value. The remaining texture channels are used to store the gradient of α. The number of layers can vary across the surface. In an extreme case, as shown in Figure 2 the number of layers can go down to one. If there are no features on a portion of the surface we do not need textures for that region.

In our implementation we use multiresolution surfaces with subdivision connectivity. However, the basic techniques that we have developed can be applied to arbitrary meshes with 2D texture coordinates.

4 Constructing shells

In this section we describe our basic algorithm for constructing shells around surfaces. Intuitively, one can think about this process as growing thick skin on the surface; shells constructed by our method behave more or less like elastic compressible skin, which was our goal.

To make a shell useful for volumetric texturing, a number of properties are desirable:

- The layers should not intersect. This requirement is motivated by the ”skin” metaphor which we believe to be natural for manipulating this type of surface representation in many cases.
- The layers should have the same connectivity. This is crucial for defining a vertex's volumetric texture coordinates (s, t, r). They can be obtained as follows in this case: (s, t) are given by the base surface parametrization which is assumed to be known, and r is incremented proportionally along the displacement director from the base surface.

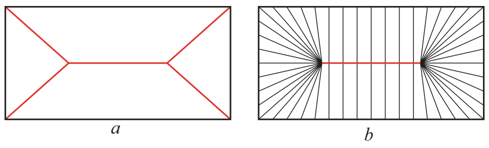

Figure 3: a. Medial axis of a box. b. The shell with target thickness exceeding one half of the box size constructed using the gradient along the medial axis. The shell director lines are shown.

- The shell should maintain prescribed thickness whenever possible. However, if thickness cannot be maintained due to geometric obstacles, a valid shell with locally decreased thickness should be produced. This corresponds to the intuitive idea of elastic "sponge-like" skin; note that volume preservation is somewhat undesirable as it is likely to result in fold formation.
- The shell should be close to the one obtained by normal displacement whenever possible.
- The shell at a point should depend only on the parts of the surface close to that point. This property is important for modeling applications and for efficient implementation.

Next, we describe our shell construction algorithm motivated by these requirements.

4.1 The basic algorithm

To describe our algorithm in detail, we need some formal notation. We assume that our base surface is a mesh or a higher order surface associated with a mesh (subdivision surface, spline surface etc.) without self-intersections. Formally, our goal of constructing a shell around the surface can be described as follows: given a surface M in $\mathbf{R}^3$, construct a one-to-one map $f(\mathbf{x}, t)$ from the direct product $M \times [0, 1]$ into $\mathbf{R}^3$. We focus on shells for which $f(\mathbf{x}, t)$ is linear, i.e. at each point, $f(\mathbf{x}, \cdot)$ is entirely defined by the direction of displacement and shell thickness.

Main ideas. Our algorithm is based on a simple idea: to construct a shell, we always need to move away from the surface. In the places where this is impossible (the simplest example is the center of a sphere) the shell cannot be extended further.

To understand how this idea can be made more formal, we consider the example shown in Figure 3 in more detail. Suppose we are building an *interior* shell, offsetting a surface M (in this case, a box) in the direction opposite to the outside normal. If the gradient of the point-to-surface distance function $d(\mathbf{x}, M)$ is defined, "moving away" from the surface more formally can be characterized as moving along the gradient of the distance function. This gradient points exactly along a normal direction to the surface whenever it is defined. In such cases we can propagate the shell away from the surface simply moving along the normal. However, the distance function is singular at some points of space which are called (*medial axis points*). Unfortunately the medial axis comes close to the surface at concavities and extends all the way to the object at sharp features, as shown in Figure 3. However, even on the medial axis it is often possible to move away from the surface. E.g., if we start from the corner of the box, we just move along the branch of the medial axis. While the complete gradient of the distance function is not defined, it is defined along the medial axis, i.e. the derivatives can be computed for any direction tangent to the medial axis. Define the *extended distance function gradient* by setting the value of the gradient at the medial axis to the gradient along the medial axis, whenever it is defined. The magnitude of this gradient is not necessarily one: the sharper the angle of the concavity, the smaller it is. For the horizontal part of the medial axis of the box, it is identically zero. We note that these are exactly the points where no further motion is possible, because shell parts extended from two sides of the box run into each other. This shows that the magnitude

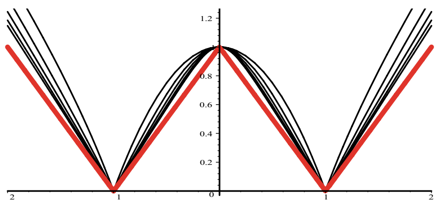

Figure 4: The red plot shows the standard distance function from a point on the line to the set of two points $\{-1, 1\}$. Other lines show the averaged distance functions for different values of p.

of the gradient of the distance function along the medial axis can be used as a measure of how easy it is to move a particle located at that point of space away from the surface.

These observations suggest the following simple abstract algorithm for constructing the director of a shell: *to obtain the director of a shell of thickness h at point $\mathbf{x}$, first follow the extended gradient field $g(\mathbf{x}) = \nabla_{\mathbf{x}} d(\mathbf{x}, M)$ of the distance function, solving the ODE*

$$\frac{\partial F(\mathbf{x}, t)}{\partial t} = h g(\mathbf{x}) \tag{1}$$

where h is the desired thickness, and $F(\mathbf{x}, t)$ is position along the integral line of the gradient field passing through $\mathbf{x}$. Then define $f(\mathbf{x}, t)$ by linear interpolation between $\mathbf{x}$ and $F(\mathbf{x}, 1)$. Note that as long as the integral curve $F(\mathbf{x}, t)$ does not reach the medial axis, it remains a straight line with unit speed parametrization, as $\|g(\mathbf{x})\| = 1$. For our box example and sufficiently large h this yields a shell completely filling the box (Figure 3b). Unfortunately, it is difficult to solve Equation 1, as the field is discontinuous, and we would have to compute the medial axis and the gradient along it. To make the algorithm practical, we replace the distance function with a function we call *L_p-averaged distance function*.

Averaged distance functions. The basis of our definition is the following simple observation. We can rewrite the distance function from a point $\mathbf{x}$ to a surface M as

$$\begin{aligned} d(\mathbf{x}, M) &= \inf_{\mathbf{y} \in M} |\mathbf{x} - \mathbf{y}| = \left(\sup_{\mathbf{y} \in M} |\mathbf{x} - \mathbf{y}|^{-1} \right)^{-1} \\ &= \left(\| |\mathbf{x} - \mathbf{y}|^{-1} \|_{L_\infty(M)} \right)^{-1} \end{aligned} \tag{2}$$

This definition lends itself to a natural generalization:

$$\begin{aligned} d_p(\mathbf{x}, M) &= \left(\| A^{-1} |\mathbf{x} - \mathbf{y}|^{-1} \|_{L_p(M)} \right)^{-1} \\ &= A^{1/p} \left(\int_M |\mathbf{x} - \mathbf{y}|^{-p} d\mathbf{y} \right)^{-1/p} \end{aligned} \tag{3}$$

where A is the area of the surface M. This normalization by the area is introduced to ensure that the gradient of this distance function is nondimensional and close to magnitude 1 at infinity, which mimicks the properties of the gradient of the euclidean distance. Intuitively, one can expect the gradient direction field of this function to have similar properties to the gradient field of the euclidean distance function as p approaches ∞. Another intuitive interpretation of this distance function is as a potential field generated by charges on the surface raised to the power $-1/p$. In practice, we have observed that even for small values of p, the fields are quite similar. This is illustrated in Figure 4 and 5. The one-dimensional averaged distance functions are compared to the standard distance function in Figure 4, and fields of several values of p in a two-dimensional box are shown in Figure 5. However, unlike the case of the euclidean distance function, the gradient of this function is well defined away from the surface, as the integration and differentiation can be exchanged. Using the averaged $d_p(\mathbf{x}, M)$ yields the analog of Eq. 1

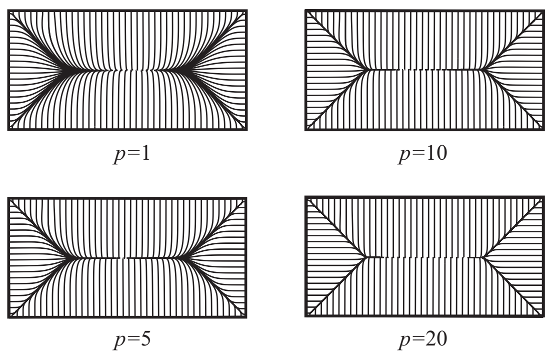

Figure 5: Field lines of the gradient field of the distance function for several values of p.

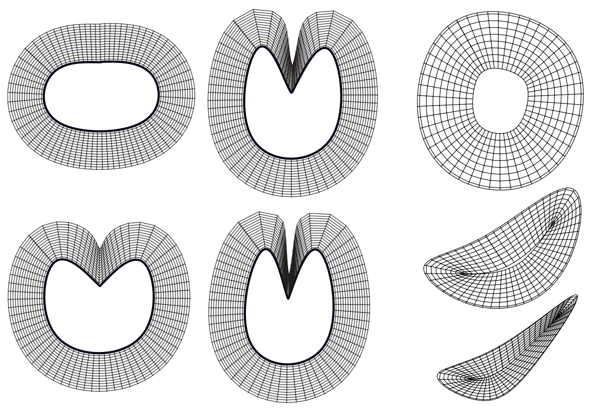

Figure 6: The four diagrams on the left show self-adjusting shell behavior of an exterior shell in the concave region. With an angle of up to 90 degrees, no compression in shell thickness is observed, but at greater angles the shell starts to compress. The three diagrams in the right column illustrate an interior shell. The first image shows the interior shell for which a prescribed thickness is achieved. As the object is deformed, the shell compresses to avoid folds (prescribed thickness remains the same).

in which the gradient has an explicit expression and *the medial axis does not have to be computed explicitly.*

It can be proved that for $p > 1$ in 3D (and $p > 0$ in 2D), the direction of the gradient g_p, at points on a smooth surface, coincides with the normal[1]. Furthermore, in all our experiments we have observed that the magnitude of the gradient remains close to one near the surface, and decays in the area close to the conventional medial axis. So our function defines a fuzzy medial axis, pruning away insignificant branches corresponding to concavities, and with the gradient field close to zero only in areas where the shell genuinely cannot be expanded (see Figure 6 for the results of our two-dimensional experiments on deforming curves).

Localization. The function is supported over the whole surface. However, it does not make sense to take into account portions of the surface which are much further away than double the target shell thickness; thus, we integrate only over the parts of the surface which fit inside a sphere of radius $2h$, making our calculation local. The extra distance beyond h is neccessary to ensure stability.

Boundaries. So far we have assumed that M does not have a boundary. Near the boundary, the averaged distance function is likely to yield shells with considerable distortion due to the fact that the distance field has to make a 180-degree turn. The standard distance function handles this case well, but the averaged function gradient field turns in the outward direction. This problem is solved by adding artificial faces at the boundary. A single additional vertex is added for each boundary vertex. The direction to the new vertex is obtained by using a tangent direction across the boundary, and the distance is taken to be equal to the shell thickness. It should be noted that such an extension is satisfactory if there are no other parts of the surface near the boundary. Otherwise, the extension can overlap a different surface part.

[1]An interesting observation that $p = 1$ in 3D corresponds to the the electric field potential which makes it clear that this value cannot be used: e.g. the potential is constant inside a hollow uniformly charged sphere.

4.2 Numerical and performance considerations

There are two main difficulties in using the averaged distance function to construct shells: we need to solve the ODE, which is stiff if the trajectory approaches the medial axis, and we need to compute the field gradient efficiently. While the ODE in most cases is well-behaved, it is stiff near the medial axis. The gradient, which is an integral over the surface, is also expensive to evaluate. We have evaluated several solution techniques (variants of explicit and implicit Euler and Runge-Kutta methods) and obtained the best performance and stability using an adaptive explicit Euler method. This algorithm is given below in somewhat simplified form, where Δ is the variable step size, $\mathbf{x}_0$ is the starting point on the surface, $\mathbf{x}$ is the current position along the trajectory, g is the gradient of the averaged distance function at the point, h is the prescribed thickness, and ϵ is the adaptivity threshold for the change in the direction.

```
$\mathbf{x} = \mathbf{x}_0$;    $t = 0$;
$g = Field(\mathbf{x}_0)$;
while $t < h$
   $\Delta = 2\Delta_0$;
   do
      $\Delta = \Delta/2$;
      $\mathbf{x}_{new} = \mathbf{x} + \Delta * g$;
      $g_{new} = Field(\mathbf{x}_{new})$;
   while the angle between $g$ and $g_{new}$ is above $\epsilon$;
   $\mathbf{x} = \mathbf{x}_{new}$;    $t += \Delta$;
   $g = g_{new}$;
end while
```

Computing integrals per face. The simplest method for calculating the integrals is to do pointwise summation over the surface. However, this approach does not work well in the case where the sampling is fixed and the surface has very sharp angles. This is easy to understand if the sample points are thought of as charges, considering the field as a surface charge density field. The approximate gradient field may "escape" between points when a surface region with high curvature is not sampled densely enough for numerical methods; this results in shell inversion. This "escape" problem can be avoided by integrating analytically over triangles of the surface mesh (quads can be split into triangles for this purpose). Fortunately, it is possible to integrate $1/r^3$ over a wedge, and a triangle can be represented as a complement of three wedges in a plane, obtained by extending each triangle side in one direction. For a single wedge, the integral can be computed explicitly. Without losing generality, we can assume that the non-negative x axis is the starting edge of the wedge, then the integral when $p = 3$ is

$$\int_{\angle} |\mathbf{x}-\mathbf{y}|^{-3} d\mathbf{y} = \frac{2}{w} \arctan\left(\frac{w}{(|\mathbf{x}|-u)\cot\frac{\beta}{2} - v}\right) \tag{4}$$

where (u, v, w) is the coordinates of the point $\mathbf{x}$, and β is the counter-clockwise angle of the wedge $\angle$. This formula is then differentiated on u, v, and w for the calculation of gradient. Using

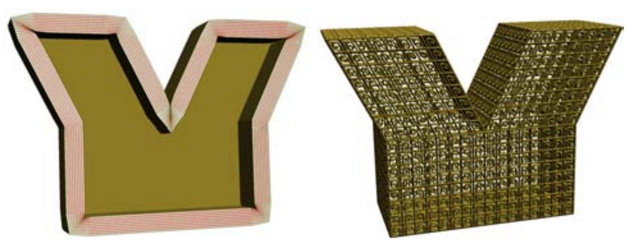

Figure 7: Left: cross-section of the shell for a shape with sharp corners; Right: same object with volume texture added.

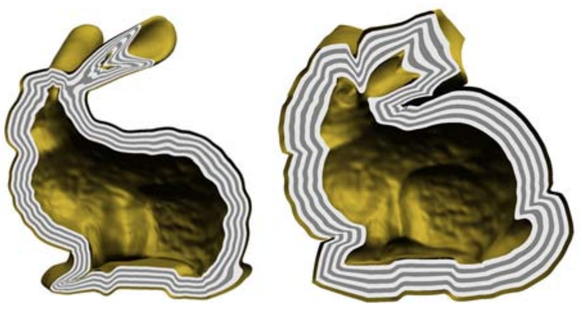

Figure 8: Cross-sections of interior and exterior shells of the bunny.

these formulas, the integral over the mesh can be evaluated *precisely* if desired. While computing the gradient in this way is more expensive, this eliminates the need for refinement, and in fact using a coarser resolution version of the mesh yields good results.

Accelerating integral computation. The expense of computing the gradient can be considerable for an interactive application since it involves a surface integral.

Although we only integrate over a small part of the surface, inside a ball near a given point, further acceleration helps. We use the Barnes and Hut algorithm [Barnes and Hut 1986] to compute the integral hierarchically. Although the calculation is already constant time, this algorithm is easy to implement, and provides a substantial speedup.

Examples. Several examples of external and internal shells and textures are shown in Figures 7,8 and 14-18. The timings for simpler objects were fractions of a second. For the bunny mesh in Figure 8, the external shell was generated in 1.8 sec on a 1GHz Pentium III, and the internal shell in 8 sec. The longer time for the internal mesh is due to refinement necessary to compute a valid shell inside the ears. The target thickness for the exterior shell was set at 10% of the bounding box size, and at 5% for the interior shell.

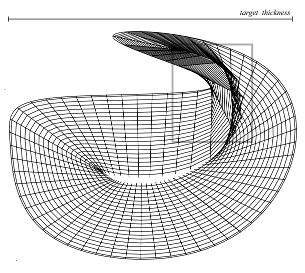

Figure 9: Folding for extreme shell thickness (prescribed thickness equal to the objects bounding box size, only 70% of the shell shown to show the fold clearly.)

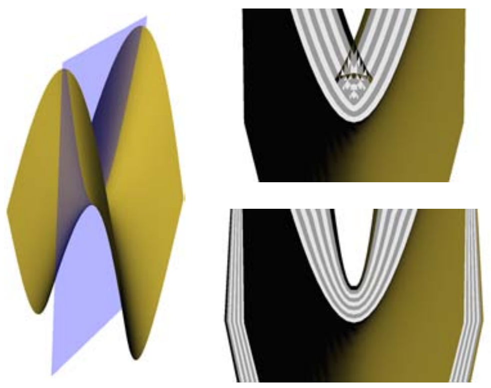

Figure 10: Comparison of the results of normal displacement method (upper right) and our method (lower right) for a saddle.

Limitations of the approach. The resulting shell is not guaranteed to be one-to-one; this is essentially inevitable, as we require directors to be straight. However, as shown in Figure 9, a rather large shell thickness needs to be prescribed with a special type of geometry for the failure to occur; for this figure, the requested shell thickness was close to the size of the bounding box of the whole object.

Comparison with alternatives. Shells created with normal displacement and with our method are compared in Figure 10. For a saddle as shown in the picture, the normal displacement method inevitably generates a self intersecting shell. It does not matter on which side of mesh the shell is expanded.

Level set methods, the fast marching method in particular, present the main alternative to our approach. However, the level set methods do not solve the problem of shell construction directly. The methods do not readily provide any mapping from the original surface to the advancing front, and the topology of the front may change. In fact this is an advantage for many applications but makes shell construction difficult. An additional step is required to establish the correspondence, as described in [Sethian 1994]. Another alternative is the envelope construction [Cohen et al. 1996] which preserves the topology of the original surface. We have explored this approach and found that the thickness of such envelopes is very low in the regions of concavities, and the shape of the surface of the envelope tends to be undesirable in such areas.

Finally we note that [Yezzi and Prince 2002] uses a conceptually similar (although numerically quite different) approach for constructing correspondences between surfaces, if we note that computing our integrals over the whole surface for $p = 1$ corresponds to solving the Laplace equation using Poisson formula. As we have pointed out, the value $p = 1$ does not work for constructing shells.

5 Rendering

In this section we present the algorithm for rendering a surface represented using a volume texture.

Slice-based direct isosurface rendering. Consider a single quadrilateral face of the surface. The part of the shell on the top of this surface is a hexahedron (Figure 11) with texture coordinates assigned to all vertices. Our surface inside the hexahedron is the isosurface of the function encoded in the alpha channnel of the texture. We assume that the other channels contain the texture gradient. Recall that the normal to an isosurface defined by $\alpha(x, y, z) = const$ is $\nabla\alpha$; therefore the gradient texture can be used for shading after normalization. It should be noted that the gradient in the light direction can be computed on the fly as suggested in [Westermann and Ertl 1998]. However, this approach works only if the gradient

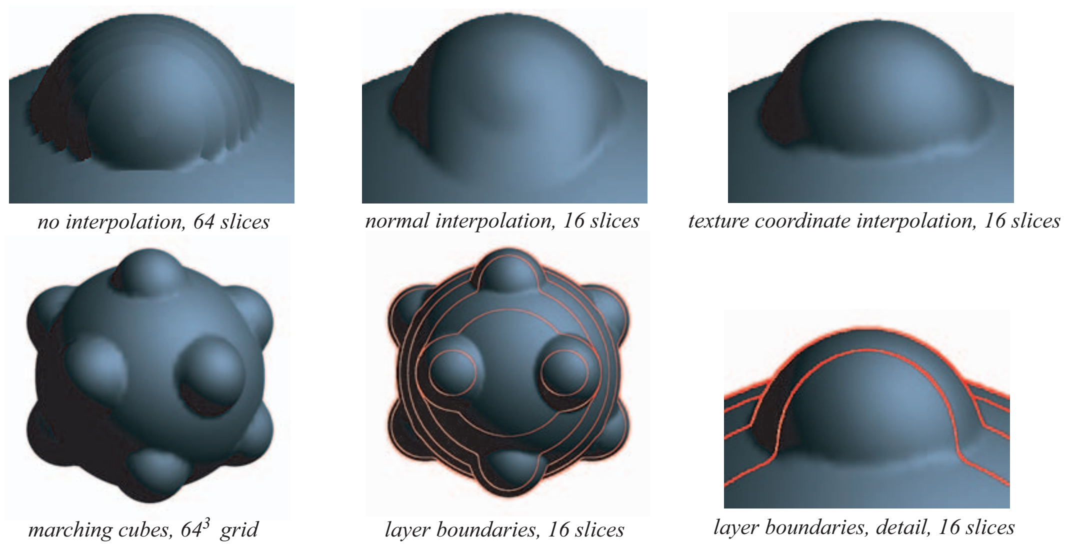
no interpolation, 64 slices
normal interpolation, 16 slices
texture coordinate interpolation, 16 slices
marching cubes, 64^3 grid
layer boundaries, 16 slices
layer boundaries, detail, 16 slices

Figure 12: Comparison of quality for rendering methods for a single volume texture. The texture is computed as the distance field for a large sphere with smaller spheres attached at vertices of a regular icosahedron. The number of slices in slice-based methods is chosen so that the total time required to render an 800x800 image is approximately the same for each method.

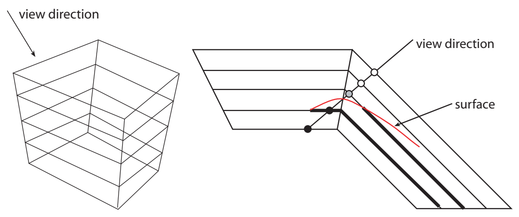

Figure 11: Left: Slices in a single texture hexahedron; slices are oriented in the direction closest to the view direction. Right: Our algorithm interpolates between the slice texture coordinates along the view direction of the last slice outside the surface and the first slice inside the surface.

already has unit magnitude (we need the unit normal for lighting), which places considerable restrictions on the way α can vary.

The idea of the slice-based direct volume rendering is to render polygons intersecting the 3D textured volume, assigning texture space vertex positions as texture coordinates. The alpha test is used to discard the part of each slice outside the object, and the gradient texture is used for shading. One of the problems with this approach is that there is a normal discontinuity at the boundary of slice images (Figure 12). This means that many slices are needed to achieve smooth shading. A large number of slices is affordable when a single volume is rendered. In fact, several slices per 3D texture layer are often used; this still amounts to rendering only few thousand slices, as the volume texture size rarely exceeds 512 in any direction.

The situation is different for volume textured surfaces; for a curved surface, a large number of polygons must be rendered for each surface layer, so rendering multiple slices per pixel is often not feasible.

Our algorithm addresses this problem by ensuring that the normal varies smoothly across slice boundaries, even if the distance between slices is considerable. This allows us to reduce the number of needed slices by a large factor.

The idea of the algorithm. The idea of the algorithm is illustrated in Figure 11 and is somewhat similar to the approach that was used in [Westermann and Ertl 1998] for volume rendering of unstructured meshes. Suppose that for each pixel we know the texture coordinates t_1, corresponding to the first slice along the view direction, for which $\alpha \geq 0.5$ (i.e. the point of the slice is inside the object) and texture coordinates t_2 for the *last* slice along the view direction for which $\alpha < 0.5$. In this case, we can approximate the point where $\alpha = 0.5$ by interpolating between two points, using α_1 and α_2 as weights. We interpolate the texture coordinates t_1 and t_2 using the same ratio and look up the normal using the interpolated coordinates. The interpolated coordinates change smoothly as we pass the slice image boundary, and the normal obtained in this way is considerably more accurate. This process can be regarded as bump-mapping for isosurfaces.

Conceptually, our algorithm proceeds in two passes; an actual implementation may be more efficient if the number of passes is increased as discussed below.

Algorithm details. Suppose the top face of the hexahedron is a quad $P = [p_0, p_1, p_2, p_3]$ and the bottom face is $Q = [q_0, q_1, q_2, q_3]$. We assume that the normals of any two opposite faces are sufficiently close. Although nothing in the method relies on this assumption, for highly distorted hexahedra, the quality of the result will be low. We assume that the view direction is close to the normals of both faces P and Q. We render a set of M slices S_m where $S_0 = P$ and $S_{M-1} = Q$. The vertices p_j^m $j = 0 \ldots 3$ of the m-th slice are given by

$$p_j^m = \left(1 - \frac{m}{M}\right) p_j + \left(\frac{m}{M}\right) q_j$$

The texture coordinates are interpolated in a similar way. In addition to rendering the slices in the direction closest to the view direction, we also render the faces of the hexahedron to ensure correct texture coordinate interpolation as explained below. We assume that the top and bottom slices have alpha values below the threshold at all points; otherwise we render additional slices in front and back.

First pass. This pass is nearly identical to the simplest version of slice-based rendering. The difference is that the output of this pass is two textures. The 3D texture coordinates of the closest fragments with $\alpha \geq 0.5$ are stored in the texture T_1, while the z values of those fragments are stored in the texture Z_1.

The alpha test on this pass rejects the fragments with $\alpha < 0.5$ (outside the object) while the standard depth test is performed.

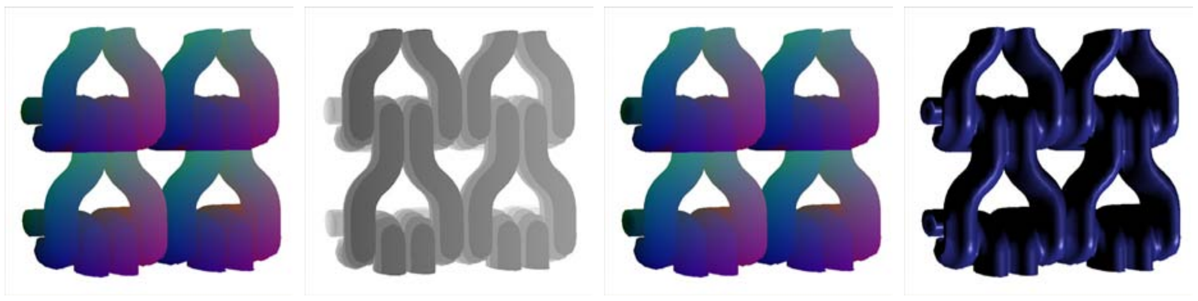

Figure 13: Steps of the rendering algorithm from left to right: Texture coordinates and depth from the first pass; Interpolated texture coordinates of the second pass; Final image.

Second pass. On this pass, we need to perform somewhat more complicated operations which require hardware programmability. The textures from the first pass are applied to the rendered fragments, using the viewport coordinates (x, y) as texture coordinates. Fragments are discarded this time if $\alpha \geq 0.5$, i.e. it is inside the object. For fragments with $\alpha < 0.5$, we compare the fragment z value with the depth texture value $Z_1(x, y)$ (the z value for the closest fragment inside the isosurface). A fragment is discarded if it is behind the closest fragment inside the isosurface.

If the fragment is not yet discarded, we look up α_1 and t_1, where $\alpha_1 = \alpha(T_1(x, y))$ and $t_1 = T_1(x, y)$, the α and 3D texture coordinate from the previous pass at the current location in the framebuffer. We interpolate the two 3D texture coordinates to give us t_3. $t_3 = ((\alpha_1 - 0.5)t_2 - (\alpha_2 - 0.5)t_1)/(\alpha_1 - \alpha_2)$ where α_2 and t_2 are the α and 3D texture coordinate from the current pass.

The resulting coordinate t_3 is used to look up the gradient for lighting. We note that the gradient of α is computed in texture coordinates and needs to be transformed to eye coordinates. Because the hexahedron is a trilinearly distorted cube, the texture-to-eye coordinates change from texel to texel. Instead of recomputing the transformation, we compute it at the vertices and use trilinear interpolation in the interior. This approach works unless the distortion is high.

Most importantly, the depth test on this pass is reversed. This means that the fragment which is actually rendered is the furthest fragment from the eye which is both outside the isosurface and passes the comparison with the depth value from the first pass.

Note that in the second step we assume that the texture coordinates from the first pass are taken from the same texture, so it makes sense to interpolate t_1 and t_2. This is ensured by rendering the boundaries between textures twice, with a small offset towards each of the textures.

Finally, we observe that for smoothly varying textures, or if the number of slices is equal to the texture dimension, normal interpolation can be used instead of texture coordinate interpolation.

Implementation. The algorithm was implemented using GeForceFX fragment programs.

The GeForceFX does not give you full depth when you bind a texture to the depth buffer; only the 8 most significant bits are present in the output. This precision is insufficient so we capture the depth values from the first pass in a seperate pass that writes the depth to 24 bits of a 32 bit RGBA buffer.

We also split the second pass into two passes: a pass to generate the 3D texture coordinates we need, followed by a final pass that uses the two 3D texture coordinates per pixel from the previous passes to look up the gradient, generate a normal and calculate the lighting. By using this extra pass we only need to perform the lighting and interpolation of texture coordinates for visible slices, and this final lighting pass also does not require geometry, so it is inexpensive in terms of AGP bus bandwidth.

6 Results

In this section we describe a variety of operations that we have implemented in our modeling system using algorithms described in the previous sections.

Deformations. When the base surface is deformed, the shell needs to be recomputed. We take advantage of the locality of the field defining the shell, and recompute only the part which is within the field influence distance from the modified surface part. This can be done at interactive rates (Figure 14). We note that if a volume deformer is used to modify the surface, the same volume deformation can be applied to the shell and no interactive recomputation is necessary; however, for significant deformations it is still better to recompute the shell.

Moving geometry along the surface. Image editing operations can be relatively easily applied to volumetric textures, which results in changes in the implicitly defined geometry (Figure 15). However, when these operations are implemented, geometric distortion of the 2D texture mapping should be taken into account. This problem is identical to the one addressed in [Biermann et al. 2002]. The target area, to which the texture is moved, and the source area are reparameterized on a common planar domain with a distortion-minimizing parametrization. The common parameterization is used to resample the source texture over the target geometry. The same approach can be used for volume textures.

Boolean operations and carving. One of the advantages of volume geometry representations is that boolean operations become relatively simple (Figure 14). In the case of volume textures, the situation is complicated by the fact that the transformation from world coordinates to texture coordinates is nonlinear. However, it is still relatively straightforward to compute a boolean operation between a regular nondistorted volume object and the volume-textured surface: this requires resampling the volume object over the shell grid, which is straightforward.

Applying a boolean operation to two volume-textured surfaces is much more difficult.

Animated Textures. Removing details from the underlying geometric representation and placing them into 3D textures makes some animations much easier to execute. One example of this is the boiling man (Figure 16). The texture is procedurally animated to show the bubbles. Bubbles can easily appear, separate from the surface and burst, as they are represented implicitly. Another example of texture animation is growing trees on the surface; in the video accompanying the paper, we added horses modeled as volumetric textures to a scene with a few dozen growing trees to show how these techniques might complement one another. The speed of our shell generation algorithm also enables us to animate the base mesh and the texture at the same time (Figure 16,17).

Rendering Performance. The performance of the rendering algorithm is quite good, especially for large textures. The turbine blade shown in the video uses 128 slices through a 512x512x512 texture (compressed to 134 MB) and exhibits real-time performance. With 512 slices shown near the end of the clip, the quality is slightly greater, and the rendering time is still acceptable for interactive tasks.

On the other hand, when we try to stress geometric complexity, we run into performance limitations. For example, with the shirt shown in the video, we are limited to about 16 slices while still obtaining close to real-time performance (17fps with either normal or texture coordinate interpolation). The video was created using a Quadro 3000 card clocked at the standard 400/850 Mhz (core/memory).

7 Conclusions and Future Work

In this paper we have presented methods for constructing and rendering surfaces with small-scale geometry represented by volumetric textures. These methods enable a variety of interactive modeling operations. The shell construction algorithm might be useful for a variety of applications using volume textures, such as fur rendering, weathering, etc. The rendering algorithm is likely to be useful for standard volume rendering applications to improve the isosurface rendering quality.

Future Work. As we have mentioned, the shell construction algorithm that we have proposed has considerable potential for acceleration. Ideally, normal displacement should be used whenever possible, with our algorithm applied only in complex areas.

It is easy to construct hierarchies for volume textures by low-pass filtering and subsampling. However, it is well known that low-pass filtering is not the best way to simplify implicit geometry, as undesirable topological artifacts may appear. Exploring techniques for topologically correct subsampling is a topic of future work.

As demonstrated in [Neyret 1995] and [Lengyel 2000], volume textures provide a good foundation for a heterogeneous hierarchy, when explicit geometry is converted to a form of reflection function or a normal distribution at coarser levels. Exploring interactive techniques using such hierarchies is another promising direction for future work.

References

AGARWALA, A. 1999. *Volumetric Surface Sculpting*. Master's thesis, MIT.

BARNES, J., AND HUT, P. 1986. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature 324*, 446.

BERN, M., AND PLASSMANN, P. 2000. Mesh generation. In *Handbook of computational geometry*. North-Holland, Amsterdam, 291–332.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste editing of multiresolution surfaces. In *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21-3, 312–321.

BLOOMENTHAL, J., Ed. 1997. *Introduction to implicit surfaces*. Morgan Kaufmann.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of SIGGRAPH 94*, 91–98.

CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001*, 67–76.

CHERNYAEV, E. 1995. Marching cubes 33 : Construction of topologically correct isosurfaces. Tech. Rep. CN/95-17, CERN.

COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., JR., F. P. B., AND WRIGHT, W. 1996. Simplification envelopes. In *Proceedings of SIGGRAPH 96*, 119–128.

CULLIP, T. J., AND NEUMANN, U. 1993. Accelerating volume reconstruction with 3d texturing hardware. Tech. Rep. TR93-027, University of North Carolina.

CUTLER, B., DORSEY, J., MCMILLAN, L., MÜLLER, M., AND JAGNOW, R. 2002. A procedural approach to authoring solid models. In *ACM Transactions on Graphics (SIGGRAPH 2001)*, vol. 21-3, 302–311.

DIETRICH, S. 2000. Elevation maps. Tech. rep., NVIDIA Corporation.

DORSEY, J., EDELMAN, A., LEGAKIS, J., JENSEN, H. W., AND PEDERSEN, H. K. 1999. Modeling and rendering of weathered stone. In *Proceedings of ACM SIGGRAPH 99*, 225–234.

EBERLY, D., GARDNER, R., MORSE, B., PIZER, S., AND SCHARLACH, C. 1994. Ridges for image analysis. *Journal of Mathematical Imaging and Vision 4*, 351–371.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, 249–254.

HENSHAW, W. D. 1996. Automatic grid generation. In *Acta numerica, 1996*, vol. 5 of *Acta Numer.* Cambridge Univ. Press, Cambridge, 121–148.

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering fur with three dimensional textures. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques (SIGGRAPH 89)*, 271–280.

KAUTZ, J., AND SEIDEL, H.-P. 2001. Hardware accelerated displacement mapping for image based rendering. In *Graphics Interface 2001*, 61–70.

LENGYEL, J. E., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2001. Real-time fur over arbitrary surfaces. In *2001 ACM Symposium on Interactive 3D Graphics*, 227–232.

LENGYEL, J. 2000. Real-time hair. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, Eurographics, 243–256.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (SIGGRAPH 87)*, vol. 21, 163–169.

MEYER, A., AND NEYRET, F. 1998. Interactive volumetric textures. In *Eurographics Rendering Workshop 1998*, Springer Wein, New York City, NY, G. Drettakis and N. Max, Eds., Eurographics, 157–168.

NEYRET, F. 1995. A general and multiscale model for volumetric textures. In *Graphics Interface '95*, Canadian Human-Computer Communications Society, W. A. Davis and P. Prusinkiewicz, Eds., Canadian Information Processing Society, 83–91.

NEYRET, F. 1998. Modeling animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan.–Mar.), 55–70.

PASKO, A., ADZHIEV, V., SOURIN, A., AND SAVCHENKO, V. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8, 429–446.

PIZER, S., BURBECK, C., COGGINS, J., FRITSCH, D., AND MORSE, B. 1994. Object shape before boundary shape: Scale-space medial axes. *Journal of Mathematical Imaging and Vision 4*, 303–313.

REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G., AND ERTL, T. 2000. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 109–118.

RICCI, A. 1973. A constructive geometry for computer graphics. *The Computer Journal 16*, 2, 157–160.

ROCKWOOD, A. 1987. *Blending surfaces in solid geometric modeling*. PhD thesis, Cambridge University.

SCHAUFLER, G. 1998. Per-object image warping with layered impostors. In *9th Eurographics Rendering Workshop*, 145–156.

SETHIAN, J. A. 1994. Curvature flow and entropy conditions applied to grid generation. *J. Comput. Phys. 115*, 2, 440–454.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*. Cambridge University Press. ISBN 0521645573.

SIDDIQI, K., BOUIX, S., TANNENBAUM, A., AND ZUCKER, S. W. 1999. The hamilton-jacobi skeleton. In *ICCV (2)*, 828–834.

STEINBERG, S., AND KNUPP, P. M. 1993. *Fundamentals of Grid Generation*. CRC Press.

WANG, S. W., AND KAUFMAN, A. E. 1995. Volume sculpting. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM Press, 151–156.

WESTERMANN, R., AND ERTL, T. 1998. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of SIGGRAPH 98*, 169–178.

WILSON, O., GELDER, A. V., AND WILHELMS, J. 1994. Direct volume rendering via 3d textures. Tech. Rep. UCSC-CRL-94-19, UCSC.

YEZZI, A., AND PRINCE, J. L. 2002. A PDE approach for thickness, correspondence, and gridding of annular tissues. In *ECCV (4)*, Springer, vol. 2353 of *Lecture Notes in Computer Science*, 575–589.

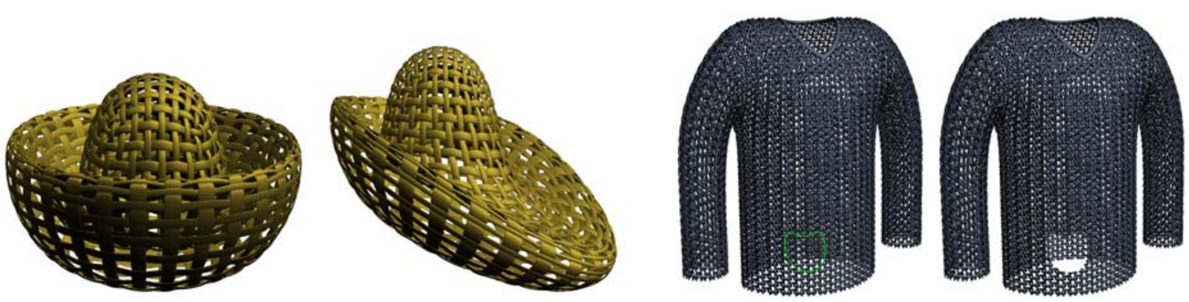
Figure 14: Editing operations: deforming a volume-textured surface and cutting a hole on the chain-mail shirt.

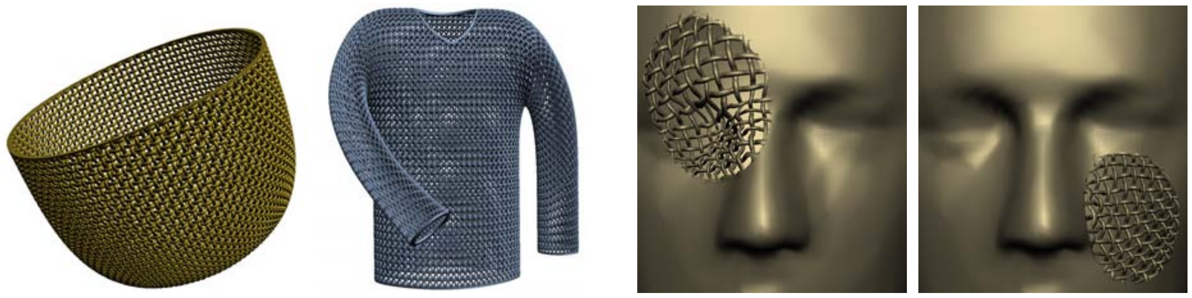
Figure 15: The first two are simple objects with small-scale geometry added. The last two show the operation of moving a geometric texture on a surface.

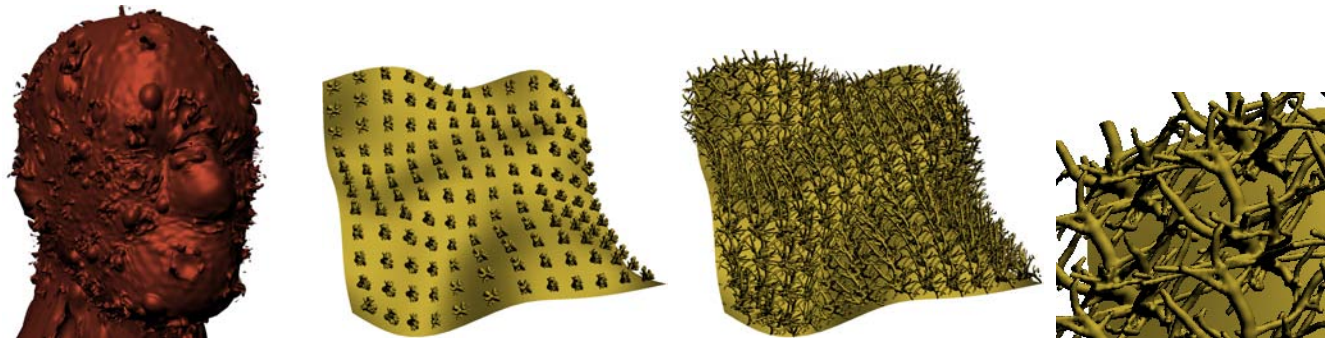
Figure 16: An animated bubbling texture applied on a deforming head and two stages of a growing bush texture with a zoomed-in view.

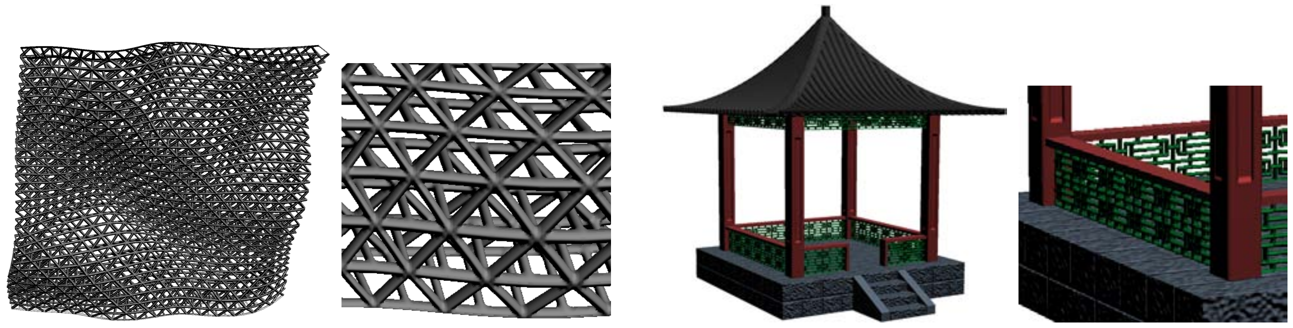
Figure 17: A structural texture on a deforming plane and a kiosk. The second and the forth pictures are showing zoomed-in details. Volume textures are used on the kiosk roof and walls.

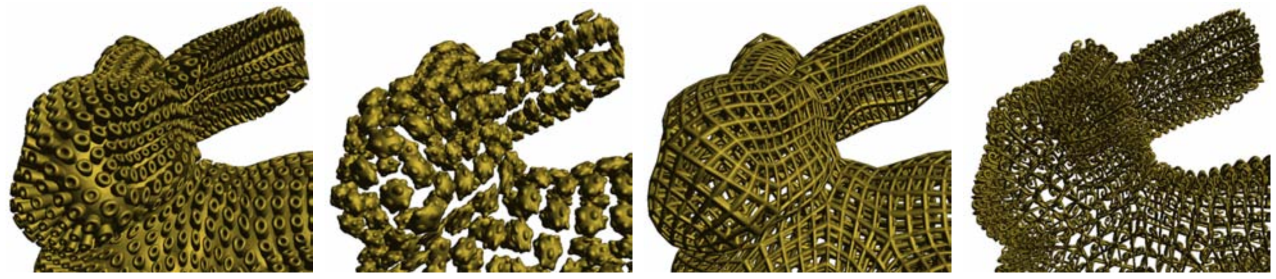
Figure 18: Several different volumetric textures applied on the bunny. Only the heads are shown to view the geometric details.