Size Preserving Pattern Mapping

Yair Kurzion[1], Torsten Möller[1,2], Roni Yagel[1]

[1]Department Of Computer And Information Science
[2]The Advanced Computing Center For The Arts And Design
The Ohio State University, Columbus, Ohio

Abstract.

We introduce a new approach for mapping texture on volumetric iso-surfaces and parametric surfaces. Our approach maps 2D images on surfaces while maintaining continuity and preserving the size of the mapped images on the models. Our approach is fully automatic. It eliminates the need for manual mapping of texture maps. We use the curvature of a surface at a point in order to continuously vary the scale of the mapped image. This makes our approach dependent only on local attributes of a point (position, normal and its derivatives) and independent of the global shape and topology of an object. Our method can map high resolution images on low resolution volumes, hence enhancing the visual appearance of rendered volume data. We describe a general framework useful for all surface types that have a C^1 continuous normal. We demonstrate the new method for painting volume data and for mapping cavities on volume data.

1 INTRODUCTION

Texture maps [7] are essential for enhancing the appearance of a model. They increase the detail of rendered models without increasing their geometric complexity. We often wish to paint a model with some generic texture. For example, in medical models, we wish to add generic skin texture to skin surfaces, and vein like formations inside walls of body cavities. We call these generic textures *patterns*. We wish to make these patterns continuous across the entire surface. We also wish to map patterns on a surface at a constant density the way they appear in nature. Adding such patterns to a model is traditionally a time consuming process. A modeler has to specify the mapping of a 2D pattern on every area of the surface, the smaller the area - the less size distortion. In this paper we present a method for mapping patterns on volumetric iso-surfaces and parametric surfaces with no user intervention. Our method takes a pattern image and a desired density and paints a model with the given pattern at the given density. We use the surface curvature at a point in order to vary the density of a projected pattern image. We use the mapped patterns to modify the color of surface points, thus painting the surface. We also use these mapped patterns to modify the intensity of volume sample points, thus adding cavities to a volume. Our method provides a general mapping scheme for all surfaces with a C^1 continuous normal.

1.1 Background

Adding texture on surfaces without distortion received a lot of attention in previous works. Previous methods fall into two categories: Direct painting methods [1,8,10], where a user manually paints the objects, and chart-based methods [2,11,16,17] where a user sub-divides the surface into sub-patches, parametrizes each sub-patch and then picks a mapping for each sub-patch. In both categories, a user has to manually specify the extent and contents of texture maps for each portion of the object. The amount of work required depends on the complexity of the model. Bier and Sloan [4] presented a two step method for mapping texture over surfaces. They map the texture on a simple surface first, and then project the texture from the simple surface onto the target model. Their method enables decreasing the amount of size distortion. However, it does not guarantee size preservation, and it is useful only if the intermediate surface is very simple.

Environment mapping [6] was introduced as a means for generating a cheap ray-tracing effect. It uses the surface normal at each point in order to lookup its color in a 2D texture map. For each point, the environment mapping algorithm reflects the viewing direction about the surface normal. It then intersects the reflected viewing direction with a box bounding the scene. Each face of the bounding box is painted by some texture map. The final color is given by the texture map color at the point of intersection between the reflected viewing vector and the box face.

Clearly, for smooth surfaces, environment mapping generates a continuous mapping of the texture map onto the surface. However, it is view dependent, and it does not preserve the density of the texture map across the surface. The higher the curvature of the surface, the higher the density of the texture image on the surface.

1.2 Our Method

One can easily eliminate the view dependency with a slight modification of the environment mapping technique. Instead of reflecting the viewing vector by the surface normal, we use the surface normal itself to map points onto the bounding box. At each point we intersect the surface normal with a bounding box, and use the intersection point to map into the texture image. We now have a very simple view independent scheme for continuous mapping of images onto smooth surfaces. This scheme is identical to the two-part mapping method [4] when picking a box as the intermediate surface, and using the target surface normal for mapping texture. Figure 1 shows two areas on a surface. The area marked A has zero curvature and so we map a wall segment of size P_0 on a segment surface of equal size C_0. The area marked B has high curvature, therefore we map a wall segment of size P_1 on a much smaller surface segment size C_1. If we paint the walls with a pattern of constant density then the projection on area B will be a lot denser than the projection on area A.

The idea behind our method is that instead of maintaining a constant image on the bounding box, we continuously modify the density of the image in order to match the surface curvature. If we want to paint the surface with pattern density D, we can express D_{wall}, the density on the wall that would generate a density D on the surface (C_i, P_i for i=0,1 as in Figure 1), as:

$$D_{wall} = D\frac{C_i}{P_i} \qquad (1)$$

D_{wall} can be any real number, so we have to develop a way to map images at any density on a given wall. Since D_{wall} is not necessarily an integer, the obvious mapping of D_{wall} instances side by side along the wall doesn't work - It generates discontinuities

on the edges and corners of the box. Our solution is to use the idea of homotopy in order to generate a bounding box image with the correct density. We vary the images on the bounding box in a continuous manner, thus making the final mapping continuous and size-preserving.

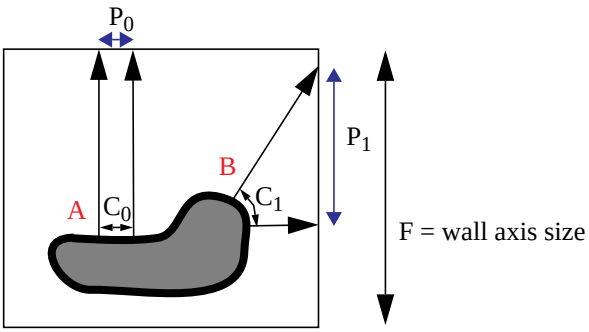


Figure 1: Modified environment mapping: color each surface point by the color where the normal hits the bounding box. The arrows marked P_0, P_1 show how the density of the mapping changes when the surface curvature does.

The next section introduces homotopy, and in particular, the two homotopies we use. Section 3 describes the simple case: pattern mapping on parametric surfaces, and Section 4 describes how to apply pattern maps Volume data.

2 HOMOTOPY

We use homotopy [5] in order to generate a continuous range of pattern images at any given pattern density. A *path* is a function from [0,1] to some space *W*. Two paths $p_0(s)$, $p_1(s)$ are *homotopic* if there exists a continuous function:

$$h:[0,1]\times[0,1]\rightarrow W \qquad (2)$$

where *W* is some space, and we have

$$h(s,0) = p_0(s), \quad h(s,1) = p_1(s), \qquad (3)$$
$$h(1,t) = x_1, \quad h(0,t) = x_0 \ \text{for all } t.$$

In our discussion, *W* is one axis of the 2D pattern image, and $x_0, x_1 \in \{0,1\}$ two endpoints of *W*. Without loss of generality, we assume that all our paths are along the X axis of a pattern image. A path is a traversal of the X axis of the image for a constant Y value. We map paths onto an axis of the bounding box wall. We express paths in image space by describing their behavior along an axis, using the parametrization [0,1]x[0,1] for the image. For example, the path (0, 0.75, 0) travels from the left edge of the pattern image (0), three quarters of the way to the right edge (0.75), and back to the left edge (0), all for a constant Y value. Figure 2 shows three examples of mapping paths on bounding box walls. The function *h* is a homotopy function. It produces a continuous transitions between two paths.

In the rest of this paper we use two specific homotopies. Figure 3 shows these two homotopies. Parts (b) and (d) show the mapping of the domain [0,1]x[0,1] (see Equation 2) into the X axis of a pattern image. The top and bottom edges depict the two original paths. The arrows represent full paths along the X axis. An arrow pointing right is the path (0,1), and an arrow pointing left is the path (1,0). The tip of the arrow is the right edge of the pattern image and the back of the arrow is the left edge of the pattern image.

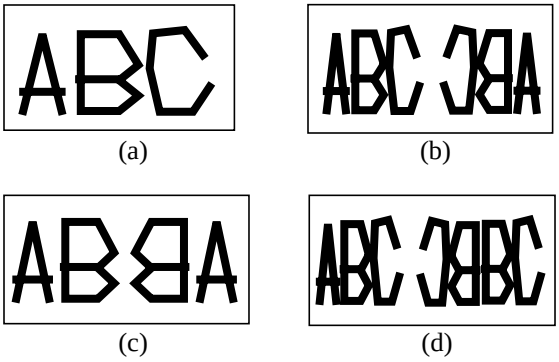


Figure 2: (a) A pattern image. (b) Mapping the path (0, 1, 0) on a box wall. (c) Mapping the path (0, 0.6, 0) on a box wall. (d) Mapping the path (0, 1, 0.6, 1) on a box wall.

Part (a) and (b) show how to continuously deform the path (0,1,0) into a stationary path at 0. We achieve this continuous transition by gradually taking paths from 0 to some t and back to 0, for values of t changing between 0 and 1. When t = 0, we are at the bottom of the homotopy. When t = 1, we are at the top. Part (a) shows this transition. Part (b) depicts the same homotopy on its domain [0,1]x[0,1]. It shows a continuous transition between the paths on the top and bottom of the [0,1]x[0,1] square. We call this homotopy 0:2. Figure 2c shows a typical intermediate path (for t = 0.6) between (0,1,0) and (0).

Similarly, Figures 3c, 3d show a homotopy between the path (0,1) and the path (0,1,0,1). We achieve this continuous transition by taking paths from 0 to 1, then back to some s, and back to 1. For s= 1, we have the bottom portion of the homotopy. For s= 0, we have its top. We call this homotopy 1:3. In order to use similar notation for both cases, we use t = (1-s). Now in both cases t=0 represents the bottom and t=1 the top. Figure 2d shows a mapping of a typical intermediate path (for t = 0.6) between (0,1,0,1) and (0,1) on a wall.

Since we map each path on a straight wall (Figure 2) we can count the number of instances of a pattern image mapped on a wall for a given t value. We preserve a constant mapping density across the wall: For the 0:2 homotopy, at a given t we have 2t instances of the full trip from left to right, at constant density. For the 1:3 homotopy we have liberty to choose the density of the transition of s1 and s2 (Figure 3d). We pick them to be the same as the density of s0. Therefore, for a given t, we have

$$\frac{3}{3-2t} = S \qquad (4)$$

instances.

We now explain how these two homotopies help us pick different images for the walls of the environment mapping bounding box. When evaluating the pattern mapping of each point, we first compute the ratio between the size of a step on the surface to a corresponding step on the bounding box wall (In Figure 1 this ratio is P_1/C_1 or P_0/C_0). We calculate this ratio for the two major axes of the wall independently. From this ratio *R*, and the desired mapping density *D*, we calculate D_{wall} (Equation 1). Given the size of the bounding box axis *F* (see Figure 1) we calculate *S:* how many instances of the pattern we should fit onto the bounding box wall axis in order to achieve the desired density *D* on the surface:

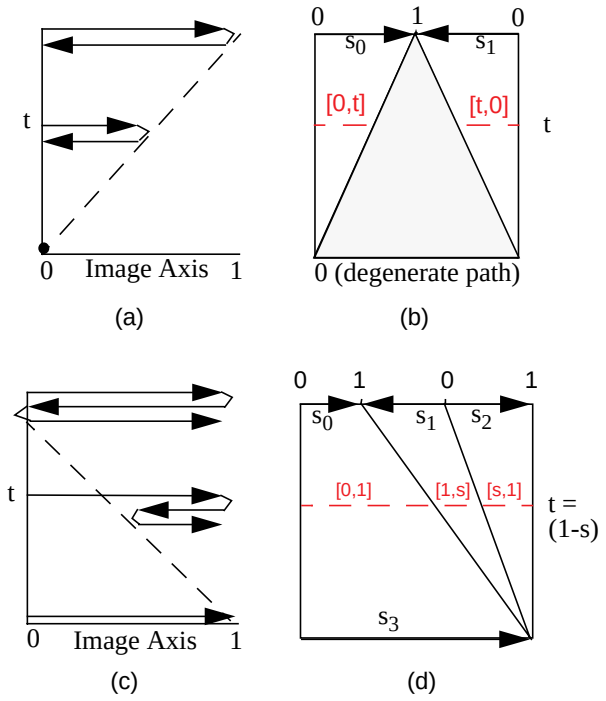


Figure 3: (a)(b) 0:2 Homotopy between the path (0,1,0), and the stationary path at point 0. (c)(d) 1:3 Homotopy between the path (0,1) to the path (0,1,0,1).

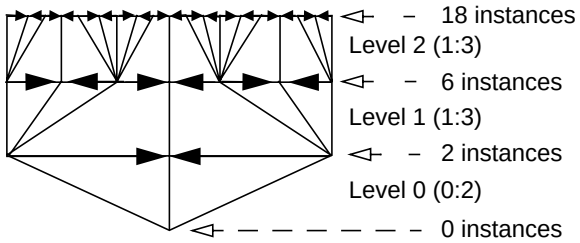
$$S = FD_{Wall} = F\frac{DC_i}{P_i} = F\frac{D}{R} \qquad (5)$$

It is easy to match an integer number of instances of a pattern image on the bounding box wall. We match 0, 2, 6, 18,..., $2 * 3^n$,... instances directly. For non-integer S values, we use the homotopies above in order to obtain a transition between two neighboring integer values. We use the 0:2 homotopy for values between 0 and 2, and the 1:3 homotopy for all other values. Figure 4a shows how we use the two homotopies for generating any number of instances of a pattern image axis along a bounding box wall. For example, matching 8.5 instances of the pattern map on a bounding box wall X axis falls into the homotopy between 6 and 18 instances. In Figure 4a this falls in level 2 so we'll be using a 1:3 homotopy. If the wall X axis spans from (-100) to 100, and our normal hit the wall at 50, then we lookup the value in level 2, at 0.75 away from the left end. Level 2 contains six (1:3) homotopies. Our point falls into the fifth one (0.75 * 6 = 4.5), at u = 0.5 on the homotopy horizontal axis. We now convert the overall number of instances 8.5 to the number of instances for this homotopy S = (8.5 / 6) = 1.41. Each of the six homotopies in level 2 generates 1.41 instances; total = 8.5 instances. Using Equation 4 we get $\dot{t}$ = 0.44. We pick the final index into the pattern image X axis to be the value at (u, t) = (0.5, 0.44) in Figure 3d. We calculate the Y index similarly and use the two indices to calculate the point color.

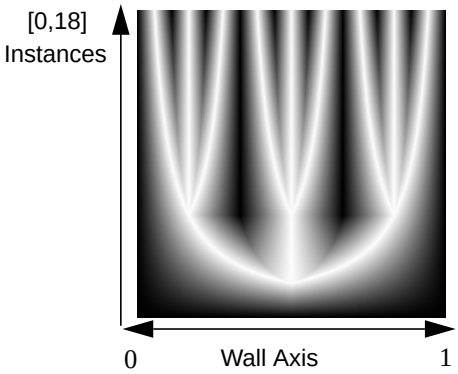
Figure 4b shows the behavior of this homotopy mapping in correct scale. The vertical axis holds the overall number of instances (range 0 to 18). The horizontal axis represents one axis of the pattern image. We encode the final result of the homotopy by grey levels. White represents 1 and black represents 0. We observe that linearly modifying the value of t does not linearly modify the resulting number of instances. From Equation 4 we get the non-linearity and compute t as a function of S.

We observe the following points in Figure 4:

- When S=0 (at the bottom edge of the graph in Figure 4b), we use only the zero point of the axis. This means that if a point requires zero instances of the image along some axis, we will use the pattern zero to paint this point.
- At the edges of the wall axis (the left and right edges of the graph), we have a constant (zero) parameter value. This is important for continuity between neighboring walls and corners of the bounding box. All normals hitting an edge connecting two bounding box walls always use zero for the axis perpendicular to the edge.
- In the graph we use mirrored and un-mirrored instances of the image interchangeably. We can see the changing when traveling across the graph for a fixed S value. Starting from the left, we begin with 0 (black), increase to 1 (white), and decrease back to zero repeating a few times. On the X axis of the pattern image, this sequence corresponds to scanning the image from left to right and the back to the left many times.



(a)



(b)

Figure 4: (a) Illustrate the behavior of the two homotopies on one axis of the pattern image. In Level 0, we use the homotopy from Figure 3a, and in all other levels, we use the homotopy from Figure 3c. Note that we only demonstrate this behavior for S in [0,18]. The same behavior continues for higher S values. (b) Show the non-linear mapping. The horizontal axis is an axis of the bounding box wall. The vertical axis varies the number of instances we fit on the bounding box wall. The shades show the choice of pattern map coordinates.

What remains to show is how to compute the ratio R for each one of the pattern image axes. We shall describe this computation for Bézier patches, and for volume data.

3 PATTERN MAPPING ON SURFACES

Bézier surfaces have a simple analytical expression for the surface normal. We use them as an introduction to the volume case. Let S(u,v) be a Bézier type surface defined by x = X(u, v), y = Y(u,v), z= Z(u,v). The surface normal is given by

$$N(u, v) = S_u(u, v) \times S_v(u, v) = (N_x, N_y, N_z) \quad (6)$$

Without loss of generality, assume that this normal hits the top wall of the bounding box and therefore we pick the surface point color from the image on this wall. The computation for other walls of the bounding box is similar. Assume this wall has plane equation z = maxZ. We express the point where a normal hits this plane as:

$$Hit(u, v) = S(u, v) + \frac{maxZ - Z(u, v)}{N_z} \cdot N(u, v) \quad (7)$$

At each point (u_0, v_0) we now look for two directions D_x, D_y in the (u,v) parameter space. For each direction we express

$$\begin{pmatrix} u_x \\ v_x \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + tD_x, \qquad \begin{pmatrix} u_y \\ v_y \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + sD_y \quad (8)$$

We pick D_x and D_y s.t. for D_x, the derivative

$$Hit_t(u_x, v_x) = (H_x, 0, 0) \quad (9)$$

and for D_y, the derivative

$$Hit_s(u_y, v_y) = (0, H_y, 0) \quad (10)$$

We now compute the hit ratios along u and v as:

$$R_x = \frac{H_x}{\|S_t(u_x, v_x)\|}, R_y = \frac{H_y}{\|S_s(u_y, v_y)\|} \quad (11)$$

R_x and R_y are the ratios we need for the homotopy calculations. From Equation 5 we get the number of instances of the pattern image on each one of the two wall axes (in this case X and Y). Plugging these numbers into the homotopy map we get a final parameters [x,y] in the range [0,1]x[0,1]. The final color of the point comes from the pattern image at location [x,y].

D_x and D_y are pre-images of (H_x, 0, 0) and (0, H_y, 0) under the partial derivatives of *Hit(u,v)*. We use a linear approximation to calculate them. We start by calculating Hit_u and Hit_v, the derivatives of the plane hit point along the major axes of the Bézier surface. These generate two vectors in the plane *z=maxZ*. We express (1, 0, 0) and (0, 1, 0) as linear combinations of these vectors, and use the resulting weights to calculate a scaled version of D_x, D_y in the (u,v) parameter space. The scaling does not matter because we only need the directions of D_x and D_y.

Figure 7 shows examples of mapping patterns on a bicubic Bézier patch. Note that since we use only a single image for all six walls of the bounding box, we must make sure that the bottom and left edges of the image can connect to each other and generate a continuous mosaic. The homotopy will always map bounding box edges to either the bottom or the left edges of the pattern image.

The only case where this process fails is when the derivative of *Hit* is zero. In geometric terms this means that the radius curvature at the given point is exactly the distance to the bounding box. We pick a threshold $\varepsilon > 0$ and clamp the derivative of *Hit* to this ε value. This means that we ignore very slight curvatures. We consider all the surfaces with $Hit < \varepsilon$ to be completely planar. This causes a very slight inaccuracy in the preservation of pattern sizes across surfaces. The larger the bounding box, and the smaller ε, the more accurately we preserve pattern sizes.

4 PATTERN MAPPING ON VOLUMES

Our measure of the ratios R_x, R_y (Equation 11) depends on the derivative of *Hit*, which depends on the normal of the surface. In order to avoid discontinuities in our pattern mapping approach, we need at least a C^2 continuous surface, which leads to C^1 continuous normals, and a C^0 continuous ratio R.

4.1 A C^2 Continuous Reconstruction Filter

Volume data sets, are only given as a discrete function. In order to be able to query this function at an arbitrary point, we need to reconstruct a continuous function from this discrete data set. This can be achieved by computing a weighted sum of the sample points. The accuracy and smoothness of the reconstructed continuous function greatly depends on the filter weights, that we use. Besides accuracy and smoothness, we desire efficient filters, in order to keep the computational expense to a minimum. Once we have reconstructed a continuous function from the sampled data, we can find the analytical gradient by simply computing the derivative of the filter weights and applying those new weights for a new reconstruction process as suggested by Bentum et al. [3] Hence, in order to guarantee C^1 continuous normals, we require at least a C^2 continuous filter kernel.

The most obvious and most commonly used choices such as nearest neighbor interpolation or linear interpolation constitute only C^0 continuity. Keys [9] suggested Cardinal splines for the interpolation, which include filters of up to a 3EF accuracy, using the notation suggested by Möller et al. [13]. Unfortunately we found that these filters are not C^2 continuous. Mitchell and Netravali [12] introduced a larger group of cubic filters, also known as the BC-splines. In order to find C^2 continuous filters in this group, we required the basis functions of their first and second derivative to match at their endpoints accordingly. These restrictions forced the parameters to be B=1 and C=0. Given four samples S_0, S_1, S_2 and S_3, this resulted in the following C^2 continuous filter: For a point between S_1 and S_2, τ away from S_1,

$$Intensity(\tau) = S_0\left(-\frac{\tau^3}{6} + \frac{\tau^2}{2} - \frac{\tau}{2} + \frac{1}{6}\right) + S_1\left(\frac{\tau^3}{2} - \tau^2 + \frac{2}{3}\right) + S_2\left(-\frac{\tau^3}{2} + \frac{\tau^2}{2} + \frac{\tau}{2} + \frac{1}{6}\right) + S_3\left(\frac{\tau^3}{6}\right) \quad (12)$$

The normal is given as the derivative of Equation 12 (gradient vector), hence the normal is C^1 continuous as required:

$$Deriv(\tau) = S_0\left(-\frac{\tau^2}{2} + \tau - \frac{1}{2}\right) + S_1\left(\frac{3\tau^2}{2} - 2\tau\right) + S_2\left(-\frac{3\tau^2}{2} + \tau + \frac{1}{2}\right) + S_3\left(\frac{\tau^2}{2}\right) \quad (13)$$

Utilizing the analysis given in [13] we have found the accuracy of the reconstruction filter to be $T^2/6$, where T is the raster size of the volume, which is a 2EF filter. The error of the derivative filter is

also $T^2/6$, again a 2EF filter. We found these error bounds sufficient for our application. Möller et al [14] provide further discussion of these filters.

Given a 4x4x4 subset of a volume, we evaluate the normals at a point (*u, v, w*) anywhere in the center 2x2x2 grid as follows: We first use the *Intensity* filter to reconstruct four values in each major direction X, Y and Z. We then use the *Deriv* filter to calculate each component of the normal out of its corresponding four reconstructed values.

4.2 Using The C^1 Normal For Pattern Mapping

Now that we have a C^1 continuous normal *N(x, y, z)*, we can construct the pattern maps: Given a 4x4x4 cube of intensity values, we express the position inside the cube by

$$P(u, v, w) = (P_x, P_y, P_z) \quad (14)$$

Where u, v, and w are in the unit interval. We use the normal from sub-section 4.1:

$$N(u, v, w) = (N_x, N_y, N_z) \quad (15)$$

Without loss of generality, we assume that this normal hits the top wall of the bounding box and therefore we pick its color from the image on this wall. Assume this wall has plane equation z = maxZ. We express the point where a normal hits this plane as:

$$Hit(u, v, w) = P(u, v, w) + \left(\frac{maxZ - P_z}{N_z} \right) \cdot N(u, v, w) \quad (16)$$

Similar to Equations 8-11 we calculate the ratios along the two major axes of the wall. Since we have no parametrization of the surface, we use two arbitrary directions in the normal plane in order to calculate the directions D_x, D_y.

Figure 8 shows how mapping a high-resolution (32x32) pattern on a low resolution (4x4x4) volume can enhance it's visual appearance. Figure 9 shows the application of pattern maps on volume data. In order to eliminate volume noise we evaluate the pattern mapping variables out of a blurred and shrunk down version of the drawn volume.

4.3 Cavity Mapping

Pederson [15] presented a method for generating displacements along flow fields. We observe that the gradient vector direction in a volume is a flow field. Under the C^1 continuous normal filter we can use a 2D intensity image to generate a pattern of cavities in a model. These cavities can only extend through the portion of the volume where intensity varies from zero to one. We map a 2D image as before using the *Hit* derivative ratios. This time, we zero out the contents of a voxel if the volume intensity at the sample point is smaller than the pattern intensity. Intuitively, the pattern map chops an interval [0,*p*] from the intensity gradient of a volume. The value *p* is set by the pattern map. Figure 5 illustrates this process. Figure 10 shows examples of cavity mapping on volume data. We map both colors and cavities on volume models.

5 RESULTS

We measure how many pattern calculations we can perform per second. A pattern calculation includes calculating the normal, the hit point derivatives, and the homotopy result. We perform all our timing measurements on a 195MHz R10000 SGI Octane workstation. Table 1 shows a summary of our results. We picked a representative of each one of the modalities: one image of a Bézier patch, one of a painted volume and one of a volume with cavities. Timing are for 256x256 images.

Table 1: Timing Results

Figure	# Pattern Evals	Time w/ Patterns	Time w/o Patterns	Pattern Evals per Second
7 (left)	86181	32.8 sec	5.2 sec	3122.5
9 (right)	575158	635 sec	390 sec	2353.3
10 (right)	5964517	3249 sec	1045 sec	2706.5

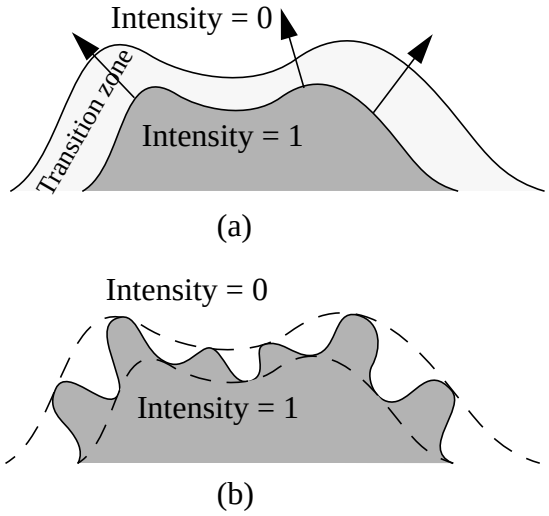


Figure 5: (a) A volume model with a thick transition zone between intensity 0 and 1. (b) Mapping new intensities into the transition area creates cavities in the model.

Rendering Bézier patches requires a single pattern map computation for each pixel sub-sample. Painting volumes requires a few pattern map computations per ray. We perform pattern computations for non-zero samples, and use early ray termination to minimize computations. The Bézier patch computation is slightly simpler because it does not require reconstructing the normal from sample points. Generating cavities in a volume requires the largest number of pattern map evaluations per ray. Since we change the contents of the volume, we also modify the volume iso-surfaces. Therefore, we have to re-calculate the normal and gradient at each sample point. We evaluate the cavity map for a small neighborhood around each sample, and use the central difference to calculate the normal and gradient values for shading and compositing. We attribute the difference between the right-most column of the two volume cases to cache coherency. In cavity mapping, we have to compute pattern mapping for a small neighborhood around a sample point. This neighborhood of samples is likely to be in cache after calculating the pattern mapping for the sample point itself.

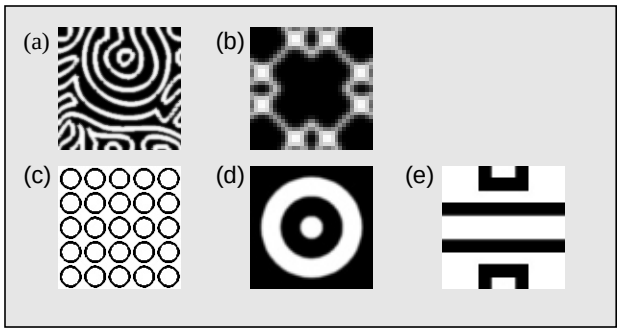


Figure 6: Sample pattern maps used to generate the images in this paper.

6 DISCUSSION

It is a common mistake to assume that varying the density of the pattern image on the bounding box walls introduces new sampling artifacts. We observe that we perform all our computations in continuous domains. Only at the last stage of the computation do we sample the pattern image and could introduce new aliasing problems. However, since our method maps pattern images on a surface preserving the size of patterns on the surface, we don't have areas with very dense mapping and we don't introduce any new aliasing problems.

6.1 Cons

- Our technique is only suitable for cases where uniform mapping of a pattern is called for. It does not consider special features and does not match any specific geometry with specific locations in the pattern map.
- Our technique is relatively expensive in CPU time. We calculate the entire mapping for each visible point on the surface every frame. When extracting surfaces from a volume (e.g. Marching Cubes) we can generate a texture map for each polygon using the homotopy calculations. Using these new texture maps for rendering the polygons could speed up rendering considerably.
- Our technique preserves the size of the pattern in two major axes. It does not avoid a shear effect. When the pre-images of the two perpendicular wall axes are not perpendicular, we introduce a shear in the final mapping.
- Our technique generates mirrors of the pattern map in one or two directions. These mirrors generate a C^0 continuous image when placed near the un-mirrored image. This means that if (for example) the pattern contains writing, the written portions will appear mirrored in some areas of the model.

6.2 Pros

- Our technique provides an automated tool for texturing smooth surfaces of any topology. It does not require user intervention at any level. For a user supplied density, our method maps a given pattern on the entire model while preserving the requested pattern density.
- Our algorithm doesn't require any parametrization of the surface. In fact, we don't even have to specify the surface explicitly. It is enough that the surface exists implicitly (i.e. isosurfaces in volumes). Our method enables texturing of surfaces without ever extracting them. Since we only use local attributes at any point, we don't have to maintain any knowledge of the surface, its shape and topology.
- Our technique depends only on local information at each point. We need not have any knowledge of the general shape and form of the models. In a multiprocessing environment this algorithm is therefore very naturally parallelizable.
- The resolution of the pattern image does not depend on the resolution of the volume data. We can easily apply a high resolution pattern map on a low resolution volume. The final resolution of the pattern depends only on the resolution of the pattern map image. Figure 8 shows an example of this quality

7 CONCLUSIONS

This paper presented a new method for adding patterns to isosurfaces in volumes and to parametric surfaces. Our method is completely automatic and requires no user intervention. Our method is useful for all surface types with a C^1 continuous normal. It maps 2D images continuously on objects while preserving the size of mapped images on the object surface. In volumes, the final resolution of the pattern on the volume does not depend on the resolution of the volume. We can map very high resolution patterns on relatively low resolution volumes. We demonstrated our method for painting volume surfaces and for generating cavities in volume models.

Acknowledgments

This work was partially supported by National Science Foundation Grant CCR-9211288, and by the Advanced Research Projects Agency Contract DABT63-C-0056.

References

[1] Agrawala, M., Beers, A.C., Levoy, M., "3d painting on scanned surfaces", Proceedings of 1995 Symposium on Interactive 3D Graphics, pp. 145-150.

[2] Bennis, C., Vezien, J.M., Iglesias, G., Gagalowicz, A., "Piecewise Surface Flattening for non-distorted Texture Mapping", Proceedings of SIGGRAPH 91, pp. 237-246.

[3] Bentum, M.J., Malzbender, T., Lichtenbelt, B.B., "Frequency Analysis of Gradient Estimators in Volume Rendering", IEEE Transactions on Visualization and Computer Graphics, T-VCG 2(3), pp. 242-254, September 1996.

[4] Bier E. A. and Sloan K. S., "Two-Part Texture Mapping", IEEE Computer Graphics and Applications, September 1986, pp. 40-53.

[5] Greenberg, M., J., Harper, J, R., "Algebraic Topology", Addison Wesley Publishing Company, 1981.

[6] Greene, N., "Environment Mapping and Other Applications of World Projections", IEEE Computer Graphics and Applications, 6(11):21-29, November 1986.

[7] Foley, D.J., Van Dam, A., Feiner, K.S., Hughes, J.F., Phillips, R.L., "Introduction to Computer Graphics", Addison-Wesley Publishing Company, 1994.

[8] Hanrahan, P., Haeberli, P.E., "Direct WYSIWYG painting and texturing of 3D shapes", Proceedings of SIGGRAPH 90, pp. 215-223.

[9] Keys, R.G., "Cubic Convolution Interpolation for Digital Image Processing", IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-29(6), pp. 1153-1160, December 1981.

[10] Litwinowicz, P., Miller, G., "Efficient Techniques for Interactive Texture Placement", Proceedings of SIGGRAPH 94, pp. 119-122.

[11] Maillot, J., Yahia, H., Verroust, A., "Interactive Texture

Mapping", Proceedings of SIGGRAPH 93, pp.27-34.

[12] Mitchell, D.P., Netravali, A.N., "Reconstruction Filters in Computer Graphics", Proceedings of SIGGRAPH 88, Computer Graphics, 22(4), pp. 221-228.

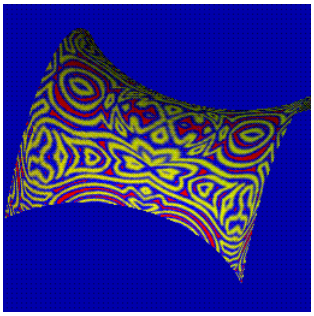
[13] Möller T., Machiraju R.K., Mueller K., Yagel R., "Evaluation and Design of Filters Using a Taylor Series Expansion", IEEE Transactions on Visualization and Computer Graphics, ITVCG 3(2): 184-199, June 1997.

[14] Möller T., Mueller K., Kurzion Y., Machiraju R., Yagel R., "Design of Accurate and Smooth Filters for Function and Derivative Reconstruction", Proceedings of the 1998 Symposium on Volume Visualization, October 1998. to appear.

[15] Pederson, H.K., "Displacement Mapping Using Flow Fields", Proceedings of SIGGRAPH 94, pp. 279-286.

[16] Pederson, H.K., "Decorating Implicit Surfaces", Proceedings of SIGGRAPH 95, pp. 291-300

[17] Shirman, L., Kamen, Y., "Fast and Accurate Texture Placement", IEEE Computer Graphics and Applications, 17(1):60-66, January-February 1997



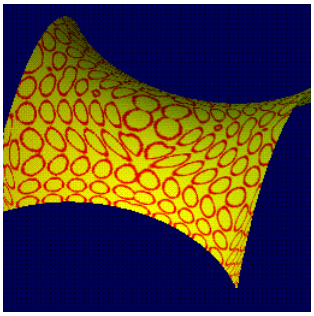


Figure 7: Mapping the patterns in Figure 6a and 6c onto a Bézier patch.

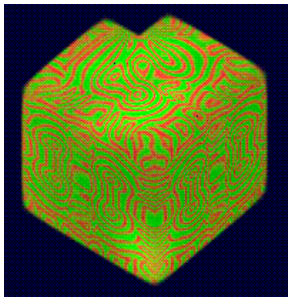
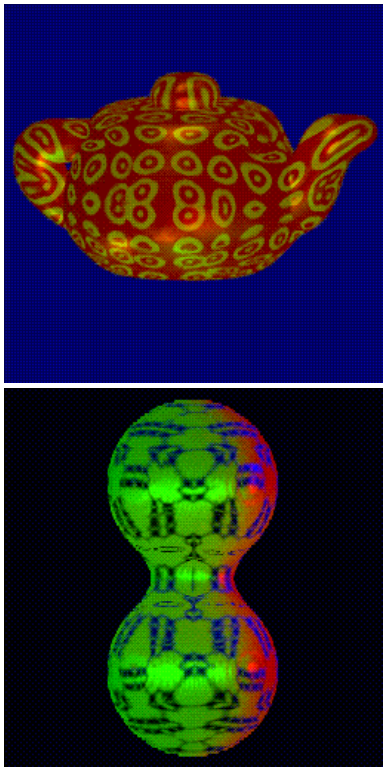


Figure 8: Mapping a high resolution pattern on a low resolution volume.



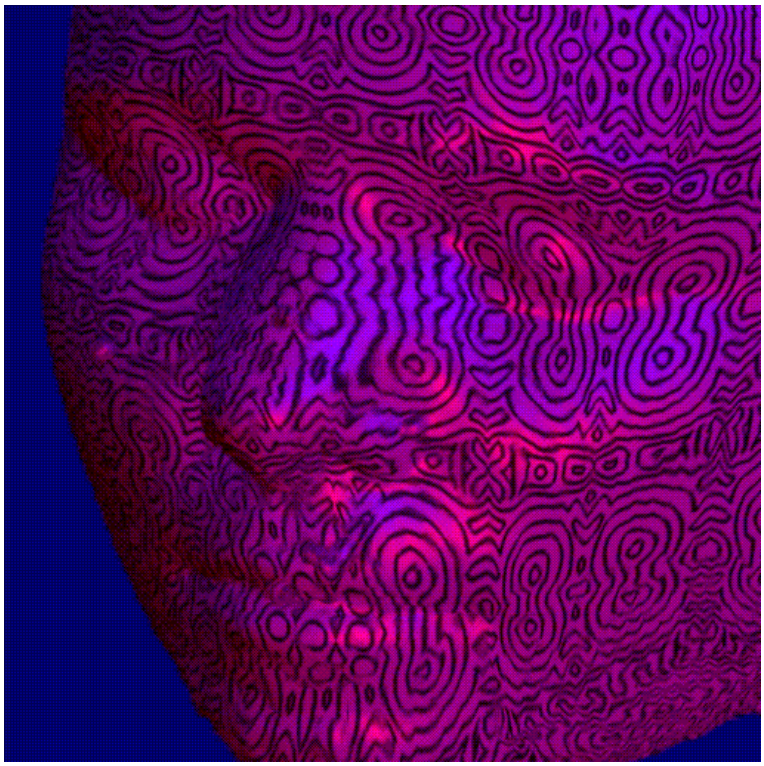
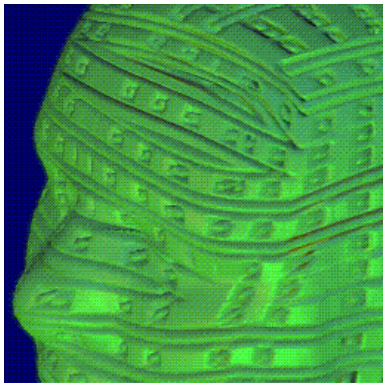
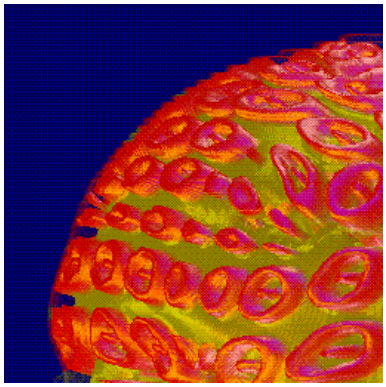


Figure 9: (top left) Figure 6d mapped on a volumetric teapot. (bottom left) Figure 6b mapped on a volumetric dipole. (right) Figure 6a mapped on an MRI scan of a head.





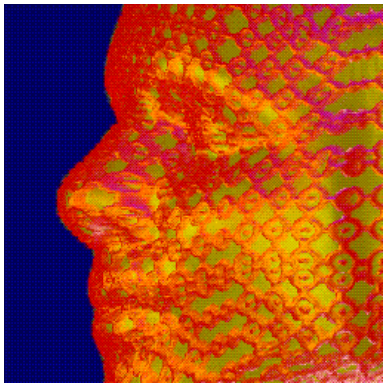


Figure 10: Examples of cavity mapping on volume data. Mapping patterns from Figure 6.