Preserving Arc Length in Texture Mapping

by

Brian Guenter and Scott Robertson

GIT-GVU-92-02
February 1992

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280
Preserving Arc Length in Texture Mapping

by

Brian Guenter¹

and

Scott Robertson²

Abstract:

Conventional mappings from the texture plane to three dimensional surfaces fail to maintain the arc length of the texture when it is mapped onto the surface. This leads to unnatural texture appearance. In addition, texture gradient information is distorted leading to poor perception of depth. We have developed an efficient algorithm which warps the texture in the texture plane such that the arc length of the texture on the three dimensional surface is approximately preserved everywhere on the surface. This leads to much more natural texture appearance and preserves texture gradient information. Texture images are preprocessed before the mapping from the texture plane to the three dimensional surface so no additional run time overhead is incurred compared to conventional two dimensional texture mapping methods.

Categories and Subject Descriptors: I.3.7 [Three-Dimensional Graphics and Realism]: Texture mapping

Additional Key Words and Phrases: arc length

¹. Assistant Professor  
   College of Computing  
   Georgia Institute of Technology  
   Atlanta GA 30332-0280  
   guenter@cc.gatech.edu

². College of Computing  
   Georgia Institute of Technology  
   Georgia Tech Box 36692  
   gt6692b@prism.gatech.edu
1. Introduction

One of the difficulties encountered in mapping planar texture patterns onto three dimensional surfaces is in establishing a mapping, \( f(u,v) : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \), which takes points \((u,v)\) in the texture plane to points \((x,y,z)\) on the three dimensional surface without distorting the scale of the texture. Many textures should approximately retain arc length when mapped onto a surface in order to look natural. Textures such as grass, planet surfaces, cloth weave patterns, skin pores, etc., all will look unnatural if the arc length of the texture on the surface varies as a function of position on the surface. In addition people make depth judgments based at least partly on texture gradients; anomalous changes in the scale of texture will diminish or confuse the perception of depth.

For parametric surfaces the simplest and most natural mapping function is the one provided by the surface parametrization, which is rarely arc length preserving. For surfaces whose parametrization is not arc length preserving it is usually not possible to create an arc length preserving mapping without introducing tears or overlaps into the texture. An example of the unnatural appearance which arises from using the natural parametric mapping function is shown in Plate 2. The texture cell shown in Plate 1 has been mapped onto a bicubic surface patch using the natural parametric mapping function.

Solid textures [Peachey 1985; Perlin 1985] do not solve the problem of preserving texture arc length since the solid texture intersects the surface of the object at different angles on different parts of the surface. This causes the scale and quality of the texture to vary as a function of the orientation of the surface. For textures such as wood this is the desired effect but for textures like grass, and the others mentioned above this would appear quite unnatural.

A statistical approach which locally maintains arc length of textures is based on the fact that second order statistics are sufficient to model homogeneous textures [Gagalowicz 1985]. Later work on third order statistics [Gagalowicz 3 1986] made it possible to model some types of nonhomogeneous textures as well. Both methods have a separate analysis and synthesis step. In the analysis step statistical properties of the texture are measured and stored in a table. In the synthesis step the texture is generated in such a manner that the error between the statistical properties of the synthetic image and the stored statistical properties of the original image is minimized. An extension of this analysis-synthesis technique to mapping textures onto 3D surfaces makes it possible to approximately maintain arc length [Gagalowicz 1 1986; Gagalowicz 2 1986]. A disadvantage of this technique is that the texture must be synthesized anew for each frame of an animation which can be time consuming and the texture must be relatively isotropic in order for the statistical characterization to yield acceptable results.

The issue of scaling scaling synthesized noise textures to approximately maintain arclength is addressed in [Wijk 1991] but the processing of textures which have been digitized is not addressed. In [Bennis 1991] the issue of maintaining arclength of existing textures is addressed but the issue of eliminating the discontinuities caused by the arclength mapping is not considered since it is not germane to the application they describe.

Our approximate arc length texture mapping scheme uses an entirely different approach. We create a texture image which has been warped by a function \( p(u,v) : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \) so that the composite function \( f(p(u,v)) \) approximately maintains arc length everywhere on the surface of the object.

The new algorithm proceeds in three steps. In the first step the texture space is discretized
into a finite number of sample points taken on a rectangular sampling lattice. The warping function, \( p(u, v) \), is computed for each lattice point. These lattice values are then interpolated to give the warping function for all the intermediate texture points. Finally the warped textures are blended together seamlessly into a single composite texture image using multiresolution image splining [Burt 1983]. The resulting texture can be used as any ordinary texture image would except that when it is mapped onto the three dimensional surface the texture will approximately maintain arc length.

The details of the arclength approximation and texture warping are described in section 2. The blending of the warped textures is described in section 3. Some simple techniques for minimizing the visibility of repetitive texture patterns are also described in this section. Images showing the results of the new algorithm are presented in section 4. Conclusions are stated in section 5.

2. Warping the texture

To compensate for the non-linear relationship between arc length along the surface and parametric distance we need to find a function which compensates at every point on the surface for the non-linearity. The arclength of a curve of constant \( v \) with value \( v_0 \) on a parametric surface \( R(u, v) \) is

\[
s_u = \int \frac{\partial R(u, v_0)}{\partial u}
\]

Taking derivatives on both sides

\[
\frac{ds_u}{du} = \frac{\partial R(u, v_0)}{\partial u}
\]

If we introduce a compensation function, \( p(u, v_0) \), then

\[
\frac{\partial R(p(u, v_0), v_0)}{\partial u} = \frac{\partial R(p(u, v_0), v_0)}{\partial p} \frac{\partial p(u, v_0)}{\partial u}
\]

If

\[
\frac{\partial R(p(u, v_0), v_0)}{\partial p} \frac{\partial p(u, v_0)}{\partial u} = 1
\]

then, from eq. 2,

\[
\frac{ds_u}{du} = \frac{\partial R(p(u, v_0), v_0)}{\partial p} \frac{\partial p(u, v_0)}{\partial u} = 1
\]

and a unit step in the \( u \) parametric space yields a unit step in arclength on the surface for every value of \( u \) along the constant \( v \) line. A similar set of equations can be derived for curves of constant \( u \).
The compensation function is the reciprocal of the partial of the surface along the u direction

\[
\frac{\partial p(u,v_0)}{\partial u} = \left(\frac{\partial R(p(u,v_0),v_0)}{\partial p}\right)^{-1}
\]

eq. 6

The exact compensation function may be very complex and not necessarily expressible in closed form. In the general case when the closed form doesn’t exist or can’t easily be computed eq. 6 can be integrated numerically. An alternative is to use a compensation function which approximately satisfies eq. 6 and which has a simple closed form integral.

We have chosen to use a patchwise approximate arclength compensation function computed in square blocks which tile the image. This makes it possible to use a very simple approximate compensation function in each block and still achieve good arclength compensation over the entire surface.

The arclength transformation and blending are more easily described by introducing a separate coordinate system corresponding to each stage of the calculation. The basic texture cell coordinate frame, whose coordinates are denoted \( u_b, v_b \), is the coordinate frame in which the original, unwarped texture cell is defined. The local texture image coordinate frame, denoted \( u_l, v_l \), is the frame in which the patchwise arclength compensation occurs. The composite texture image coordinate frame, denoted \( u_c, v_c \), is the coordinate frame in which the image blending is computed.

The result of the patchwise arclength compensation is a composite texture image. The composite texture image is a two dimensional array of texture samples all lying in the unit rectangle in the \( u_c, v_c \) plane. The composite texture image is made up of smaller texture images called local texture images. The local texture images are all of size \( 2^n + 1 \) by \( 2^n + 1 \) and they overlap by one pixel at their boundaries.
The composite texture image is constrained to be square and consequently it is of size $2^{n-m} + 1$ by $2^{n-m} + 1$. where

$$m = \frac{\text{log}_2 (\text{number of local texture images})}{2} \quad \text{eq. 7}$$

A separate arclength compensation function is applied to each local texture image.

The composite image arclength compensation function is computed in two steps:

1. The partials of the surface with respect to $u$ and $v$ are computed at points on a square grid in the $u_c, v_c$ plane. The rate at which to sample the partials is chosen by the user.

2. For each local texture image an approximate arclength compensation function is computed using the partial information computed at the boundaries and center of the local texture image. This is illustrated for the $u_1$ value in Figure 3. First the partials in $u$ are bilinearly interpolated to give an interpolated partial value. The bilinear interpolation also yields two intermediate partial values $e_0$ and $e_1$. The $u$ offset from the center of the basic texture cell is computed by assuming that the arclength compensation function is linear as shown in Figure 4. The process is repeated with the $v_1$ value to compute the $v$ offset from the center of the basic texture cell. This transformation is such that the center of the local texture image will always transform to the center of the basic texture cell.
Figure 3

The integral of the compensation function gives the $u_b, v_b$ coordinates in the basic texture cell which correspond to the pixel at coordinates $u_l, v_l$ in the local texture image. Bilinear interpolation of the four nearest neighbors of the point $u_b, v_b$ in the basic texture cell then gives the value to be stored in the pixel at coordinates $u_l, v_l$ in the local texture image.
3. Blending the images

The local arclength compensation applied to each local texture image in the composite texture image introduces discontinuities in the composite texture image at the boundaries where the local texture images abut. To make these discontinuities invisible the local texture images are blended using multiresolution image splining [Burt 1983]. As an illustration of how multi-resolution image splining works think about blending two images, A and B, along a vertical line. The simplest way is to choose a blend region of width \( p \) pixels and let the resulting image be a linear combination of images A and B in the blend region and entirely A or entirely B outside this region.

The width of the blend region and the degree of overlap of the two images depends on the content of the image. If the image has only high spatial frequencies then a short blend region one or two pixels wide will give a nearly invisible blend. If the image has low spatial frequencies, however, the blend region must be wider in order to avoid introducing a sharp change in image intensity at the blend line. Since most texture images have energy at many spatial frequencies no single blend width is appropriate. The solution is to separate the images to be blended into a set of bandpass filtered images and then to blend each of the bandpass filtered images with the single best blend distance for that set of spatial frequencies. The final image is reconstructed by adding together the resulting blended bandpass images.
Multiresolution image splining provides a time and space efficient way of computing the set of bandpass filtered images and then of blending the images. First the original image is successively lowpass filtered and decimated by a factor of two in x and y. This continues until the final image is a single pixel in size. Since the separable lowpass filter originally used in [Burt 1983] has a roughly Gaussian shape the multiresolution image pyramid is called a Gaussian pyramid.

For simplicity each level of the Gaussian pyramid is constrained to be a square of size \(2^n + 1\) by \(2^n + 1\) pixels. If level \(G_i\) of the pyramid is of size \(2^n + 1\) by \(2^n + 1\) then level \(G_{i+1}\) is of size \(2^{n-1} + 1\) by \(2^{n-1} + 1\). Level \(G_0\) contains the original image. The set of bandpass images is constructed by subtracting successive levels of the Gaussian pyramid to yield difference of Gaussian images. Pyramid level \(G_{i+1}\) cannot be directly subtracted from pyramid level \(G_i\) because level \(G_{i+1}\) has resolution \(2^{n-1} + 1\) by \(2^{n-1} + 1\), and level \(G_i\) has resolution \(2^n + 1\) by \(2^n + 1\). By interpolating level \(G_{i-1}\) to twice its original spatial resolution, written \(Interp(G_{i-1})\), and then subtracting it from \(G_i\) we can compute a bandpass image \(L_i\)

\[
L_i = G_i - Interp(G_{i+1})
\]  

(eq. 8)

In an \(n + 1\) level pyramid \(L_n\) is equal to \(G_n\) because there is no higher level image to interpolate and subtract.

Since the difference of Gaussian image has a frequency response similar to that which would result from filtering the image with a Laplacian of Gaussian filter this new pyramid of bandpass filtered images is called a Laplacian pyramid. For a Laplacian pyramid of depth \(n + 1\) the original image can be exactly reconstructed by summing all the bandpass filtered image levels after they have been interpolated to the appropriate resolution:

\[
G_0 = L_0 + Interp(L_1 + Interp(L_2 + Interp(L_3 + \cdots (L_{n-1} + Interp(L_n)))))
\]  

(eq. 9)

The details of the construction of the Gaussian and Laplacian pyramids [Burt 1983] are somewhat involved and will not be repeated here.

Each of the local texture images overlaps its neighbors by one pixel. However a one pixel blend region is not sufficient to eliminate the visibility of the boundary discontinuities. Each local texture image is further extended by a one pixel border computed by taking the first difference in x and y. At the four corner points the average of the x and y first differences is used. Figure 7 shows the formulas for first difference extension for the three possible cases: a difference in x, a difference in y, and a difference at the corner.
After the first difference extension the region of overlap between neighboring local texture images increases to three pixels. A Laplacian pyramid is constructed for each of the expanded local texture images. At each level of the Laplacian pyramid the neighboring local texture images are blended along a three pixel wide blend line by taking a weighted sum of the neighboring local texture images which overlap each pixel. The resulting blended pyramids are then interpolated and summed to give the final blended image.

Points in the 1 pixel wide first difference border extension are given a weight of .25. Points on the original, unextended border are given a weight of .5. Points one pixel inside the unextended border are given a weight of .75 and all points further in the interior of the local texture image are given a weight of 1. An example of the weights for a local texture image - of size 5 by 5 before the 1 pixel extension - is shown in Figure 8.

In general if the center pixel of a local texture image of size $2^n + 3$ by $2^n + 3$ after the first difference extension is given coordinates $(0,0)$ then for any pixel with coordinates $(x,y)$

- if $\max(|x|, |y|) = 2^n$  then weight = .25
- if $\max(|x|, |y|) = 2^n - 1$  then weight = .5
- if $\max(|x|, |y|) = 2^n - 2$  then weight = .75
- if $\max(|x|, |y|) < 2^n - 2$  then weight = 1

The final value for a given pixel is equal to the normalized weighted value of all the local texture images which overlap that pixel. The weights are normalized by recording each of the weights which fall in a pixel and then dividing by the sum of the weights.
<table>
<thead>
<tr>
<th>0.25</th>
<th>0.25</th>
<th>0.25</th>
<th>0.25</th>
<th>0.25</th>
<th>0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Figure 8: Blend weights for local texture image of size 5 by 5

3.1. Minimizing visibility of texture cell replication

In some cases the visibility of the periodicity of the individual texture cells is undesirable. For example, in simulating a planetary surface or the texture of rock it is best if the individual texture cells are not discernible. In these cases the visibility of the repetition of the basic texture cell can be significantly reduced by using as the basic texture cell a window onto a larger texture cell. The center point of this window is randomly displaced for each of the local texture images in the composite texture image. Relatively small displacements, on the order of 20% to 30% of the size of the basic texture cell essentially eliminate the visibility of pattern repetition.

Window onto larger texture image displaced by $\Delta x, \Delta y$ from the center of the basic texture cell
4. Results

Plate 1 shows a basic moon texture cell. The conventional mapping of this cell onto a bicubic
surface patch, using 16x16 tiles, is shown in Plate 2. Only the central 33 by 33 portion of the
moon image is used in this tiling.

Plate 3 shows the composite texture image, with 16x16 tiles, resulting from the arc length
mapping technique applied to the moon texture cell shown in Plate 1. Only the central 33 by
33 portion of the moon image, plus or minus an 8 pixel random displacement, is used in this
image. Plate 4 shows the moon texture cell mapped onto the same bicubic parametric surface
as in Plate 2, using 16x16 tiles, only this time using the arc length preserving method. The
resulting texture is much more uniform in scale than the parametric mapping.

Plate 5 shows the conventional mapping of the moon texture cell onto a vase-like bicubic
parametric surface, using 8x8 tiles. The change in texture scale as a function of position on
the surface is very noticeable. Plate 6 shows the arc length preserving mapping with the same
number of tiles. Notice that the mapping is still quite uniform texture scale. Plate 7 shows a
synthetic texture cell. Plate 8 shows the synthetic texture cell parametrically mapped onto the
bicubic vase surface, using 8x8 tiles, and plate 9 shows the results of the arc length mapping
method, using 8x8 tiles.

For the 16x16 tiled images only the central 33x33 pixel area, plus or minus an 8 pixel ran-
don offset, was used as the basic moon texture cell for the arc length preserving method. For
the 8x8 tiled images only the central 65x65 pixel area, plus or minus a 16 pixel random off-
s, was used.

The arclength compensation program reads in a 257 by 257 8-bit source image, creates a
513 by 513 8-bit tiled image (8x8 tiles or 16x16 tiles) by choosing random displaced win-
dows onto the source image, blends the tiled image, and writes it to disk. The program was
compiled using the standard 'cc' compiler (not gcc) with the '-O' optimizing option. Execution
times were measured using the UNIX 'time' command. On a Sparstation 1 user CPU time
was 49.78 seconds and system CPU time was 2.61 seconds. All of the arclength compensated
images illustrated here used a 513 by 513 arclength corrected composite texture image.

5. Conclusions

The arc length texture mapping method produces textures which have a much more natural
appearance than those generated by mappings which do not preserve arc length. Very little
computation is required; what is required is performed as a preprocessing step. There is no
additional run time overhead because the arc length compensation is incorporate in the
texture image itself before the mapping onto the surface.
Plate 1: Basic moon texture cell. For the 16x16 tiled images only the central 33x33 pixel area, plus or minus 25%, was used as the basic moon texture cell. For the 8x8 tiled images only the central 65x65 pixel area, plus or minus 25%, was used.

Plate 2: Conventional mapping of moon texture cell shown in Plate 1
Plate 3: Composite texture image

Plate 4: Moon texture cell with arc length mapping and random displacement
Plate 5: Moon texture cell with conventional mapping

Plate 6: Moon texture cell with arc length mapping and random displacement
Plate 7: Synthetic texture cell

Plate 8: Synthetic texture cell with conventional parametric mapping
Plate 9: Synthetic texture cell with arc length mapping and random displacement
References


