CREATING ISOTROPIC TOROIDAL TEXTURE PATTERNS

Sanjiv K. Bhatia
Southern Illinois University – Edwardsville
and FlightSafety International

Abstract

A number of applications require texture patterns that can seamlessly tile against each other to create larger patterns. A simple example of such an application is background textures for documents on world-wide web. Another example is to assemble large terrain models for flight simulators. These patterns are painstakingly generated by hand to ensure the seamless fit, and still may suffer from artifacts. This paper describes the algorithm and implementation of a tool to create such texture patterns, called isotropic toroidal texture patterns, from any given image, without human intervention.

Introduction

A toroidal texture pattern is one that can be laid side-by-side or one-above-another, such that the texture on the left side must fit the texture on the right seamlessly, and the top edge must fit the bottom edge without any seam being visible. Such textures are useful in creating image backgrounds for web applications. A special class of toroidal patterns is *isotropic toroidal patterns*, that contain a single uniform texture type, such as grass, water, forest, or urban area. They are also useful in applications such as digital terrain modeling where a set of isotropic toroidal patterns can be laid against each other in left-to-right and top-to-bottom manner to create seemingly large terrain. Such patterns form the basis for terrain models in flight simulation applications. The toroidal topology of the textures ensures that the tiles or texture fit seamlessly into each other when a terrain model is constructed.

Presented at the IMAGE 2003 Conference
Scottsdale, Arizona 14–18 July 2003.

This paper describes the algorithm and implementation of a tool to create toroidal texture patterns from any given image. The algorithm was originally developed for isotropic toroidal patterns but has generated excellent results for the larger class of toroidal patterns as well.

The input to the algorithm is an SGI IRIS RGB image. It generates a texture pattern by following the line of minimum resistance in L value in the `L*a*b*` space. We used the `L*a*b*` space primarily for convenience, and can get equally good results with other measures of comparison between pixels, such as Euclidean distance in RGB space.

The only requirement for input image is that it should be at least as large as the texture being generated. The algorithm performs better when the input image is somewhat larger than the texture being generated. The algorithm performs equally well with square or rectangular texture sizes, both for input and output images.

The algorithm starts by extracting a base pattern of desired size from the center of the input image, described in next section. It then creates minimum resistance cut-lines on two sides of the extracted pattern, taken as left and bottom edges for our implementation and illustration. Then, it fills in the areas to the left and top of cut-line, blending the areas around cut-line to ensure that no artifacts are visible.

Extracting the Base Pattern

The height and width of the input image is measured and based on the output size, the exact center of the input is selected to be the base on which we'll build our texture. An input image is shown in Fig. 1. The extracted base pattern is highlighted on the input in Fig. 2 and shown in Fig. 3.

Once the base texture is identified, we identify

Figure 1: Input image

Figure 2: Input image with the base for texture outlined

the areas on the west and south edges that can be cropped and filled with new patterns to create a toroidal topology. To achieve this, we fill the cropped area along the west edge with pattern just east of the east edge. This east edge here is taken to be the east edge of the cropped tile, and the area just east of the east edge is from the uncropped tile. Similarly, the cropped area in the south edge is filled with the pattern just north of the north edge. In the next section, we will show an algorithm to create a "cut line" that will be used to crop out the portion of the texture along an edge.

Creating the Cut Line

Figure 3: Base for the texture

The texture base is used to create two "cut lines" – one on the west side and the other on the south side – inside the extracted picture. The cut line should be created to match the edges of the texture so that it has a toroidal topology. The cut line starts at a corner in the texture base and moves to the other corner along the edge that is to be cropped. In the following, we describe the creation of the cut line along the western edge of texture base.

The cut line along western edge is created by starting at the southwestern corner of the base, and following along the pixels that differ the least in terms of their L value in the `L*a*b*` space. The input image is given in RGB space. The conversion from RGB to `L*a*b*` is performed using CCIR Recommendation 709 at CIE Standard Illuminant D65 (Aldus, 1992; Foley *et al*, 1990) as follows:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

Equation 1 is used to compute the tristimulus values for reference white at D65 illumination, by substituting the value of R, G, and B to be 255 each. The tristimulus values are denoted by X_n, Y_n, and Z_n, and are defined to be 242.36628, 255.00, and 277.63228, respectively. The formulas for 1976 CIE `L*a*b*` are taken from Aldus (1992).

$$
\begin{aligned}
L* &= 116 \cdot \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - 16 && \text{for } \frac{Y}{Y_n} > 0.008856 \\
L* &= 903.3 \cdot \frac{Y}{Y_n} && \text{for } \frac{Y}{Y_n} \leq 0.008856 \\
a* &= 500\left(\left(\frac{X}{X_n}\right)^{\frac{1}{3}} - \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}}\right) \\
b* &= 200\left(\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_n}\right)^{\frac{1}{3}}\right)
\end{aligned}
\tag{2}
$$

In equation 2, if any of the ratios $\left(\frac{X}{X_n}\right)$, $\left(\frac{Y}{Y_n}\right)$, or $\left(\frac{Z}{Z_n}\right)$ is less than or equal to 0.008856, it is replaced in the formulas with

$$7.787F + \frac{16}{116}$$

where F is $\left(\frac{X}{X_n}\right)$, $\left(\frac{Y}{Y_n}\right)$, or $\left(\frac{Z}{Z_n}\right)$, as appropriate.

The L value forms the basis for our cut line. While creating the cut line, we want to make sure that we do not create long horizontal and vertical lines. This is achieved by limiting the cut line to travel no more than a prespecified number of pixels (default 5) in a given row or column.

Let us see the creation of the line along the west edge. We will create a mask using a cut line from south to north starting at the southwest corner, and another cut line from north to south starting at the northwest corner.

Let us first create the line from south to north. This line extends all the way from southwest corner to the northern edge. For each pixel, we examine the five neighboring pixels (if they exist) in the directions west, northwest, north, northeast, and east. We select the pixel whose L-value differs the least from the current pixel's and select it for our cut line. We stop our line when we reach the northern edge.

The second line starts at the northwestern corner and moves in the southeastern direction but stops as the first line is encountered. At this point, we clean off the line from south to north that is north of the point of intersection. The resulting mask on the base texture is shown in Fig. 4.

The cut line along the southern edge is computed similarly with the first line starting at the southwestern corner and moving in the northeastern direction up to the eastern edge, while the second line starts at the southeastern corner and moves towards the first line in the northwestern direction. The mask of this cut line on our sample image is shown in Fig. 5.

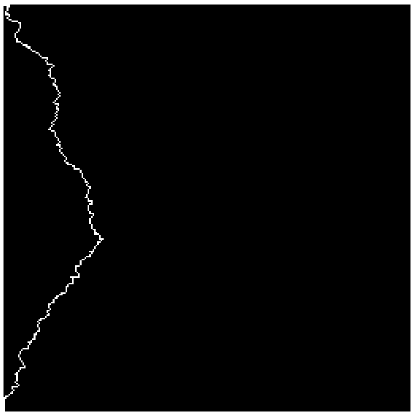

Figure 4: Cut line along the western edge

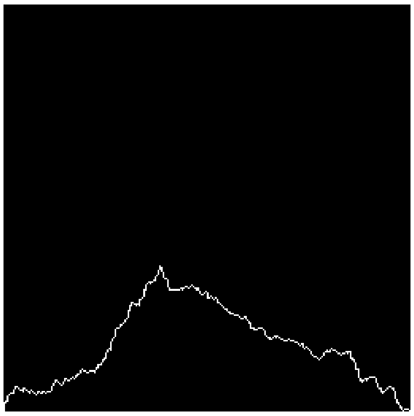

Figure 5: Cut line along the southern edge

The cut lines in the two masks are used to create the final texture pattern.

Creating the Texture Pattern

The final texture pattern is created on the extracted texture (Fig. 3). We use the mask of Fig. 4 and replace the pixels in base texture that correspond to the pixels to the west of the cut line in the mask. Then, we fill the pixels that correspond to the pixels towards the south of the cut line in the mask of Fig. 5. Let us look at filling of the pixels on western edge.

We fill the pixels along the western edge one row at a time. We start at the leftmost column of the texture and copy the corresponding pixel from the original image (same row) that is just right of the base texture identified (see Fig. 3). If the distance of cut line from the left edge of texture is more than the distance between the right edge of the extracted image and the right edge of the original image, the remaining pixels are filled by using the mirror image of the original image placed to the right of the original image.

After filling the pixels in the row, we blend the pixels around the cut line to allow for uniform transition from one type of pixels to another. This is done by selecting a blending region of pixels (default five pixels on either side of the cut line). The blending is performed using the principles outlined in Perlin noise. (Ebert *et al*, 1998; Hertzmann and Perlin, 2000)

The pixels in the texture that are to the south of the cut line in the mask are filled in a similar manner by going through each column from left to right. The final texture is shown in Fig. 6.

Figure 6: Final texture

The texture in Fig. 6 is rendered in a 4×4 pattern in Fig. 7.

Conclusion

We have described an algorithm to create toroidal textures using an arbitrary image. We have been able to generate both rectangular as well as square texture patterns using this algorithm while obeying the toroidal property. The generated textures, if isotropic, have been used to create multiple textures that fit seamlessly against each other by using a modification of this algorithm. The multiple textures are useful in creating vast expanses of terrain such that the repetition of textures is not visible while still avoiding sharp boundaries at the edges. The process, with slight modifications, has been used to generate multiple instances of isotropic toroidal texture patterns for terrain models used in flight simulators.

Acknowledgement

The work reported in this paper was performed at Visual Simulation Systems Division of FlightSafety International. Andrew Lindberg helped with a number of ideas.

References

Aldus Corporation. (1992). TIFF Revision 6.0, Seattle, WA.

Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., and Worley, S. (1998). Texturing and Modeling – A Procedural Approach. Academic Press.

Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F. (1990). Computer Graphics: Principles and Practice. Addison-Wesley.

Hertzmann, A. and Perlin, K. (2000). Painterly Rendering for Video and Interaction. Proceedings of NPAR 2000 (pp. 7–12).

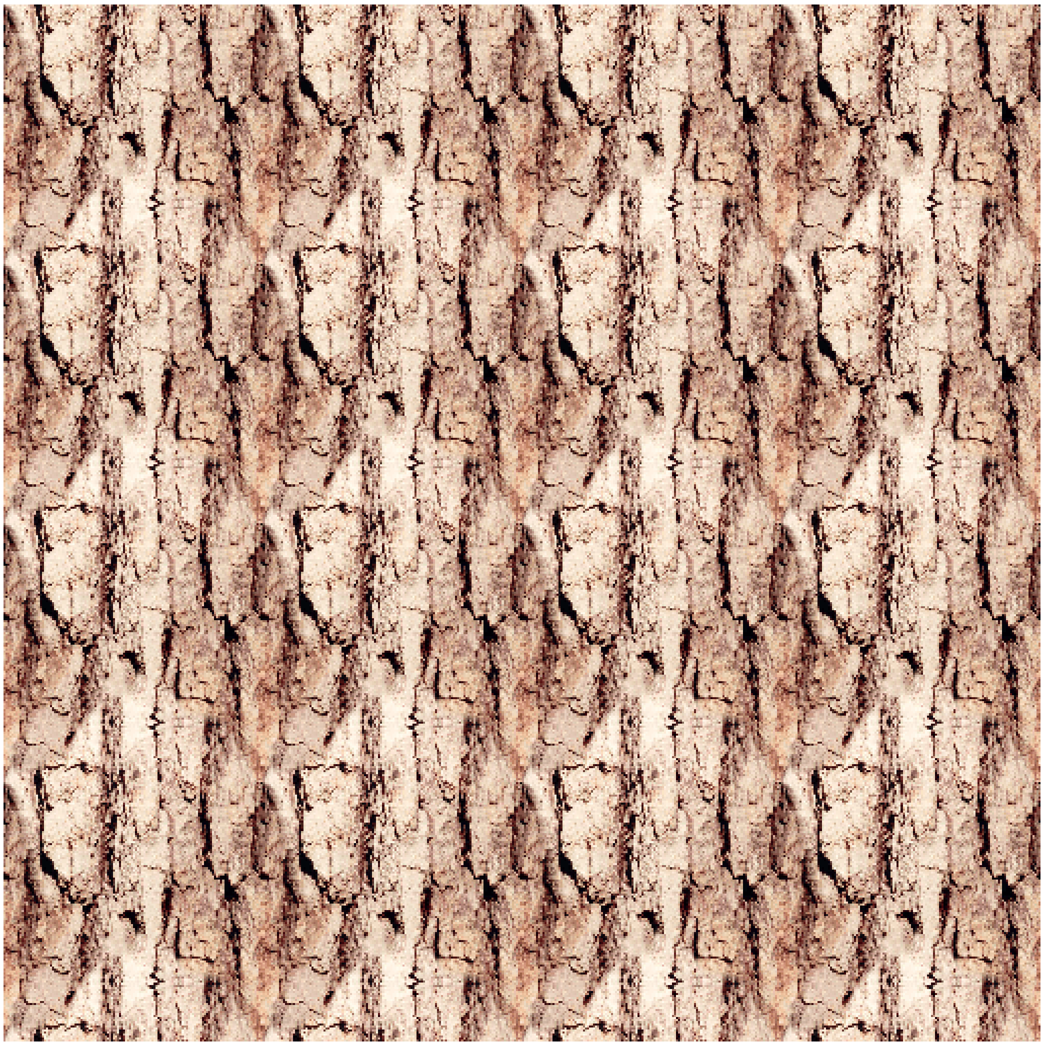

Figure 7: Rendering of texture