Creating Large Isotropic Textures Using Image Quilting

Sanjiv K. Bhatia
Department of Mathematics & Computer Science
University of Missouri – St. Louis
St. Louis, MO 63121, USA
sanjiv@acm.org

ABSTRACT

Image quilting is a texture synthesis technique to create a large texture by wrapping around patches of a small texture in a way that the repetition of small texture is not noticeable. The basic algorithm is to randomly select small patches in a given texture. These patches are then positioned in a large texture to be synthesized and blended across boundaries to remove the appearance of boundaries across patches. The algorithm is useful to create large isotropic textures from small isotropic textures. We have extended the algorithm to create large isotropic textures from a given anisotropic texture by using only the desired areas in the synthesized texture.

Keywords

Texture Synthesis, Isotropic Textures, Visualization

1 Introduction

In a number of computer graphics applications, we need to create textures that grow, almost without limit, in all directions. For example, modeling of generic terrain in flight simulators and game engines requires texture models that can be grown from small textures. In such applications, a forest texture can be used to build a large terrain to represent a forest. However, a limited number of texture patterns creates a noticeable repetition of those patterns in the final rendering. This problem can be partially solved by creating *tileable* textures, also known as *isotropic toroidal texture* [Bha03]. The *isotropic* part of the phrase refers to the fact that the texture represents a single type of object – it could be a patch of grass or forest, or a picture of bricks laid on the wall. The *toroidal* part refers to the fact that if two of the same textures are laid against each other, the textures fit seamlessly from left to right and from top to bottom, or that the textures are rotationally invariant. Typically, such a texture is designed by graphics artists who have to take painstaking measures to ensure a seamless transition from one side to the next. These textures are nonprocedural as they are not based on a mathematical or algorithmic model that creates a procedural texture [Ebe98].

WSCG SHORT Communication papers proceedings
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

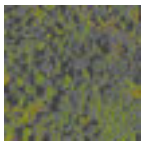

Figure 1: Forest texture

An example of a 128×128 pixel texture of a forest is shown in Figure 1. This figure is rendered in a small scale to show the creation of larger textures in later figures (Figures 2, 4) while preserving the scale. In

Figure 2, we present a rendering of this texture by repeating the same texture 4 times along each row and 4 times along each column. Since it is an isotropic texture, there are no visible seams but even on a 4×4 rendering, the repetition of texture is clearly visible.

In image quilting, our goal is twofold. One, we want to create the larger rendering of the texture without the visible repetition while still preserving the seamless transition from one instance of texture to next. Second, we want to work with textures that may not be toroidal. In this paper, we'll describe an algorithm to achieve this objective. After creation of such a rendering, we'll extend the technique to ignore undesired portions in the input texture.

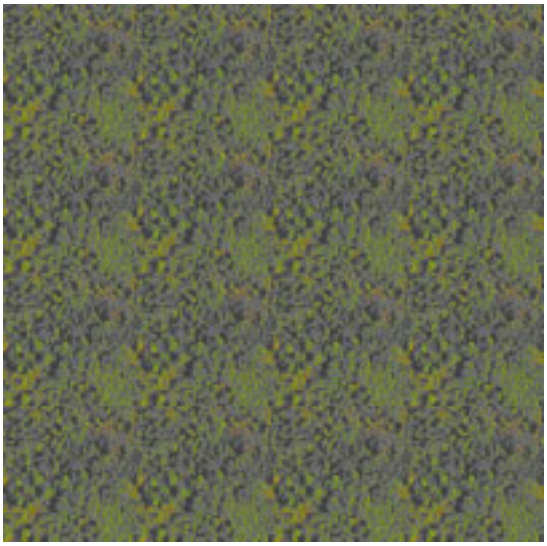

Figure 2: 4×4 Rendering of forest texture

Our algorithm is motivated by and is a slight improvement over the one proposed by Efros and Freeman [Efr01]. They proposed an algorithm that selected fixed-size blocks from an input texture at random and placed those in the final texture. The blocks were then blended by computing an error surface between the already chosen blocks and the new block.

The selection of blocks in the algorithm by Efros and Freeman [Efr01] is constrained by an error tolerance. This could be a bottleneck in the algorithm's performance. Our algorithm removes this constraint, allowing for a block to be selected completely at random, and the blending performed using a *minimum resistance cut line* on the newly selected block.

Another texture synthesis algorithm that is similar to the work presented in this paper is by Ashikhmin [Ash01]. However, that algorithm is based on pixel level synthesis in scanline order while we work with a user-selectable block size. An algorithm with block-level synthesis was proposed by Praun, *et al* [Pra00]. They select random blocks from an input texture and grow the blocks to cover the surface. Dischler, *et al*, perform block-level synthesis by using correlation between different *texture particles* [Dis02]. Neyret and Cani have performed texture mapping using triangular texture samples [Ney99]. Liu, *et al* [Liu01] have proposed a bidirectional texture function for real-world texture synthesis.

In the next section, we present the algorithm to determine the minimum resistance cut line. In Section 3, we present our algorithm for quilting. Section 4 concludes the paper by comparing our results with those in reference [Efr01].

2 Minimum Resistance Cut Line

The minimum resistance cut line is a line along an edge formed by randomly traversing the pixels along that edge under the constraint that the line is continuous in 8-neighborhood of each pixel. It forms the basis for our blending across selected sub-blocks and provides us with the flexibility to modify a number of parameters. It also eliminates the straight line edges that may cause a problem in creating large textures.

Consider an $m \times n$ block of texture extracted from the overall input texture. We'll use this extracted block to create a minimum resistance cut line along the right edge and the bottom edge. We'll describe the creation of cut line along the left edge of the texture. The cut line along the bottom edge can be created in a similar manner.

The cut line along the left edge is created by starting at the bottom left corner of the extracted texture block, and traversing the pixels toward the top right corner of the block. The traversal is performed by selecting a pixel that differs the least from the current pixel in its neighborhood in a selected color space, such as RGB, HSV, or L*a*b* [Ald92, Mur96, Poy95]. The selected pixel is from the set of neighboring pixels that are in one of the following directions of the current pixel: left, top-left, top, top-right, and right. The pixels below the current pixel are ignored so as to prod the line traversal towards the top. The traversal could result into long horizontal or vertical lines which is not desirable. This problem is solved by limiting the cut line

to traverse no more than a predefined number of pixels in a given row or column of the texture. We stop the traversal when the last pixel on the texture toward top or right edge is encountered.

After computing the line from bottom left to top right, we do the same exercise starting at top left and moving toward bottom right. This line stops when we encounter the earlier line that was traversed from bottom left to top right. The portion of the prior line from the current intersection point towards top right is then discarded yielding a cut line towards the left edge. The cut line towards the bottom edge is created in a similar manner.

An example of minimum resistance cut line toward the left and bottom edges in the texture of Figure 1 is presented in Figure 3. It should be noted that the maximum distance of the cut line from the edges can be controlled by a configurable parameter that specifies the maximum length of the straight lines that is allowed in vertical or horizontal direction. In Figure 3, this maximum length has been limited to five pixels. A smaller length will bring the cut line closer to the edge.

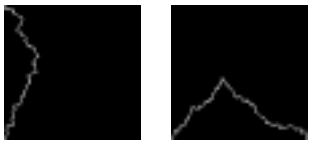

Figure 3: Minimum resistance cut line toward left and bottom edges

The two cut lines form the basis for our image quilting algorithm as described in the next section.

3 Image Quilting Algorithm

The basic algorithm for image quilting involves picking up small sub-blocks of texture from the input texture at random and arranging them in the output texture. All the selected sub-blocks are of the same width and height, and must be smaller than the input texture. This obviously creates a problem in that it will lead to straight lines along the edges where blocks are placed. Thus, our quilting algorithm is based on providing a blending method across the boundaries of the sub-blocks using the minimum resistance cut line described earlier.

The quilting algorithm starts by selecting a sub-block at random from the input texture and placing it in a corner, say bottom left. Now, we *grow* the texture along the row and column specified by this placement. If we place the sub-block in bottom left corner, we grow the entire bottom row and the entire left column by placing randomly selected sub-blocks. It is also important to remember the position of the sub-blocks in the original texture to facilitate seamless blending. This position can be given by the coordinates of a corner of the sub-block in the original texture, and its use is described in the blending part below.

The placement and blending of a sub-block to the right of a sub-block in the bottom row proceeds as follows. Select a sub-block at random from the original texture, and create a corresponding mask by drawing a minimum resistance cut line along its left edge. The area in the new sub-block to the left of the cut line is filled by the extension of the sub-block that has been placed already. This extension is from the area in the original texture just to the right of the sub-block that was previously placed. The extension is easily determined from the position of the previous sub-block (as remembered when the block was selected) and the width of the sub-block which is constant for all sub-blocks during a given run. After filling in the area, the pixels around the cut line are blended to remove any abrupt transitions along the cut line. After completing the row of sub-blocks in the final texture, the column is grown in the same way.

The edges are simple as we do not have to worry about blending in two directions. As we move away from edges, each sub-block has to be blended with existing sub-blocks on two sides. In the example being considered, we have to blend the sub-blocks toward left and bottom edges for the new sub-block being placed. It seems to be a simple problem if we can first blend the sub-block toward one edge (left) and then toward the other (bottom). However, this creates abrupt transitions because the edge toward the bottom has some pixels brought in from the extension toward right of the sub-block toward bottom left. Similarly, the edge toward left has pixels brought in from the extension toward top of the sub-block toward bottom left. Therefore, the proper blending toward the bottom left corner of the new sub-block has to account for four sub-blocks – the current sub-block, the right extension of sub-block toward left, the top extension of sub-block toward bottom, and the top right extension of sub-block toward bottom left.

A filling of all sub-blocks, with proper lending as described above, completes the output texture. If the ex-

tension of randomly selected sub-block moves past the input texture edges, it can be accounted for by inverting the direction movement, or extending the input texture by its reflection. An example of a larger 512×512 pixel texture created from the input texture of Figure 1 is presented in Figure 4. This rendering can be compared with the simple repetitive rendering achieved by placing the isotropic toroidal textures (Figure 2) to see the degree of randomness achieved. As the texture size grows, the image quilted texture removes the uniformity that becomes even more obvious in the rendering using multiple instances of toroidal textures.

Figure 4: 4×4 rendering of forest texture by image quilting

3.1 Extracting A Specified Texture Type

In certain applications, we may need to extract a particular area of the texture to create a large texture. For example, consider the aerial photograph of an airport taxiway with some grassy areas shown in Figure 5. This is a 1000×517 pixel photograph. We want to extract a texture to represent grass from this photograph using our image quilting algorithm.

Our image quilting algorithm works remarkably well for isolating the selected texture if we can convey to it the range of colors to be ignored. Thus, the roads, represented by a dark color, can be ignored by specifying a configurable parameter. We can create a set of blocks that are to be picked and note them in a data structure before starting to select the blocks for quilting. This simply adds a preprocessing phase to the algorithm. The selection is based on a simple scan of pixel array of the input texture and marking each pixel as valid (selectable for quilting) or invalid (not selectable). For this selection, we start at the top right corner and whenever a pixel in the undesired range is encountered, make that pixel as the top right corner of an undesired block, based on the size of sub-blocks to be selected for quilting. Effectively, the previously valid pixels to the left of this current pixel become invalid, as do pixels that have not even been examined yet, and that are towards the bottom of the texture from current pixel. Then, all the pixels in the undesired block are marked as invalid and the scan continues with the next unassigned pixel. The image quilting algorithm now selects a sub-block only if its top right corner has been marked as valid. A 512×512 pixel

Figure 5: Aerial photograph of airport taxiway and grass

texture of grass created from the photograph of Figure 5 is presented in Figure 6.

Figure 6: Grass texture

4 Conclusion

We have presented an algorithm to create a large texture from a given small texture by randomly selecting sub-blocks of an original texture and blending them. Our algorithm removes the matching of sub-blocks required by the algorithm proposed by Efros and Freeman [Efr01]. The results are visually acceptable and compare well with the existing algorithm. Our algorithm is closer to the one proposed by Ashikhmin [Ash01] with the innovation that the block selection is random and blending between blocks is achieved on the basis of a cut line to avoid straight line artifacts. The algorihm has been used to generate larger textures that form the basis of "universal texture" in a commercial flight simulator.

We show a number of textures rendered by our algorithm in Figures 7 and 8. The textures are the same as used by Efros and Freeman to visualize the performance of their algorithm. These figures show that the larger textures generated by our algorithm are well blended on the edges and show a nice rendering as large textures while eliminating the repetition from the small textures. These figures may show some artifacts near the blended region in textures that have sharp edges but no such artifacts are visible in textures that lack sharp edges. Even then, the artifacts are barely noticeable and the generated textures are perfectly useable in low resolution visualization applications such as universal texture in flight simulation.

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. The textures are used with permission from FlightSafety and William Freeman.

References

[Ald92] Aldus Corporation, Seattle, WA. *TIFF Revision 6.0*, June 1992.

[Ash01] Ashikhmin, M. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[Bha03] Bhatia, S. K. Creating isotropic toroidal texture patterns. In *Proceedings of the IMAGE 2003 Conference*, Scottsdale, AZ, July 2003.

[Dis02] J.-M. Dischler, K. Maritaud, B. Levy, and D. Ghazanfarpour. Texture particles. *Eurographics*, 21(3), 2002.

[Efr01] Efros, A. A. and Freeman, W. T. Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001*, Los Angeles, CA, 2001.

[Ebe98] Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. *Texturing and Modeling – A Procedural Approach*. Academic Press, 1998.

[Liu01] Liu, X., Yu, Y., and Shum, H.-Y. Synthesizing bidirectional texture functions for real-world surfaces. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 97–106. ACM Press / ACM SIGGRAPH, 2001.

[Mur96] Murray, J. D. and vanRyper, W. *Encyclopedia of Graphics File Formats (2nd ed.)*. O'Reilly, Sebastopol, CA, 1996.

[Ney99] Neyret, F. and Cani, M.-P. Pattern-based texturing revisited. In Alyn Rockwood, editor, *SIGGRAPH '99, Computer Graphics Proceedings*, pages 235–242, Los Angeles, 1999. Addison Wesley Longman.

[Pra00] Praun, E., Finkelstein, A., and Hoppe, H. Lapped textures. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 465–470. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[Poy95] Poynton, C. A guided tour of color space. In *New Foundations for Video Technology: Proceedings of the SMPTE Advanced Television and Electronic Imaging Conference*, pages 167–180, San Francisco, CA, February 1995.

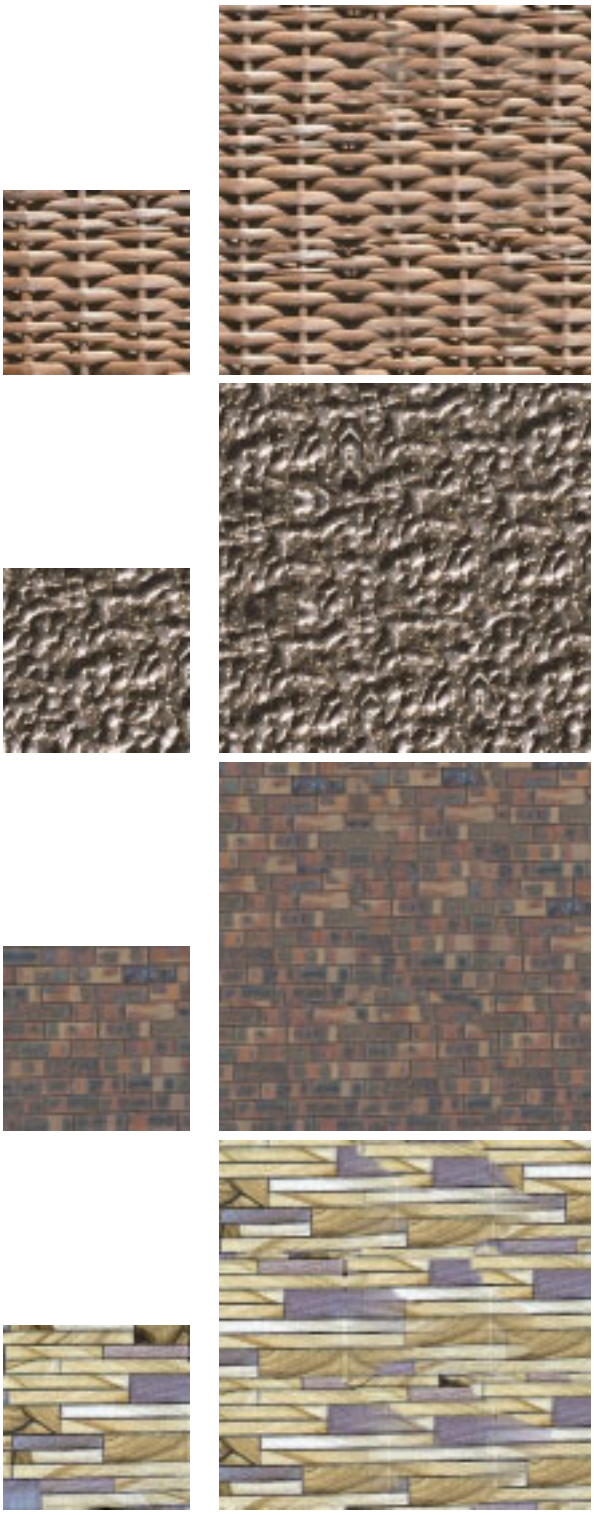

Figure 7: Some results with textures used by Efros and Freeman

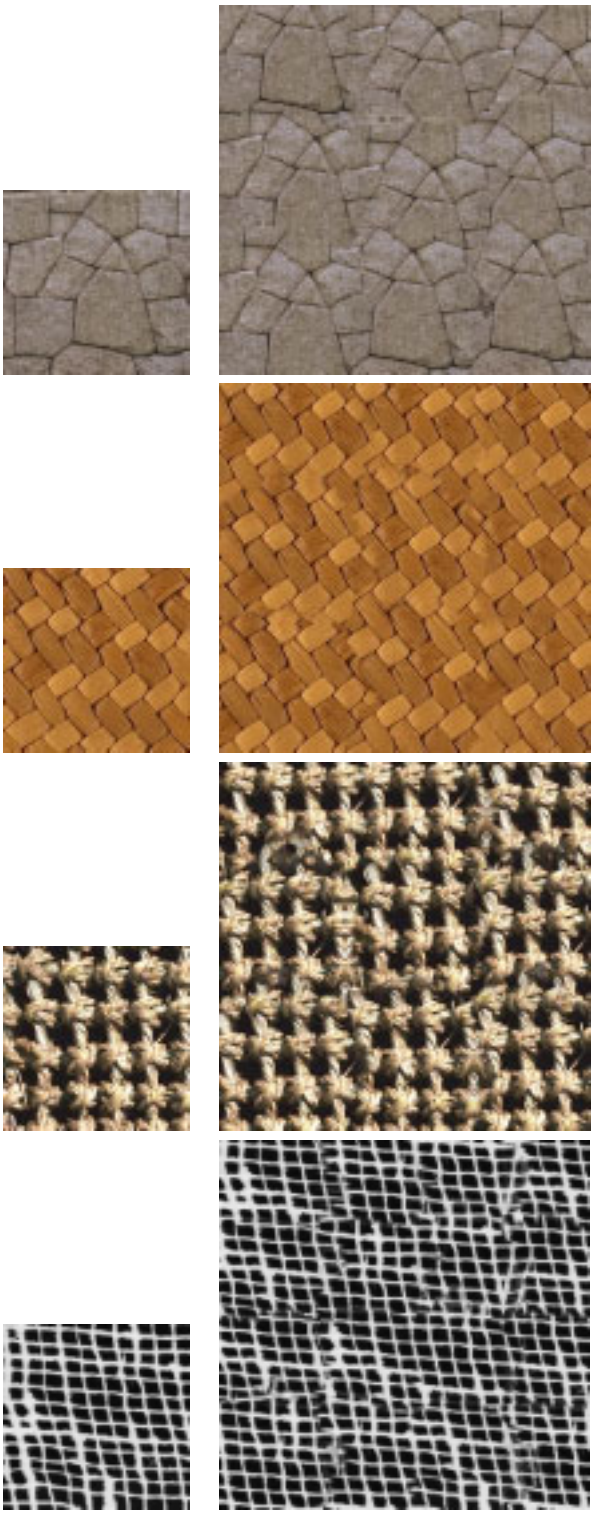

Figure 8: More results with textures used by Efros and Freeman