

## Procedural Texture Mapping on FPGAs

Andy G. Ye and David M. Lewis

Address: Dept. Of Elect. And Comp. Engineering, 10 King's College Road,  
Toronto, Ontario, M5S 3G4

Tel: (416) 978-1652 E-mail: {yeandy, lewis}@eecg.utoronto.ca

In many computer graphic applications, polygon meshes are used to model geometrical surfaces. Procedural texture mapping increases the level of surface detail of polygon meshes by determining their surface colouring using computer algorithms. These procedural texture algorithms typically model the structures of materials like concrete, wood, and marble. They can be defined in 3-D space and be parameterized using input variables defining additional attributes other than the texture coordinates.

Procedural texture algorithms, when executed in software, often cannot achieve the real time performance demanded by many computer animation applications. While 2-D textures can be stored in RAM, 3-D and parameterized textures require excessive memory. In this presentation, we present a new approach to synthesizing procedural textures in hardware in which FPGA hardware is used to provide high performance implementations of procedural texture algorithms. The primary technique used is to compile the procedural algorithms into hardware structures that can be programmed into FPGAs. This approach is more memory efficient than storing pre-generated textures in memory, since only the algorithms are stored. It is also more flexible than fixed hardware, since the reconfigurability of FPGAs can be used to exploit the parallelism presented in each individual algorithm.

A procedural texture generator was designed using the FPGA approach. It is flexible enough to synthesize a variety of procedural textures in high speed, and is small enough to be implemented in one or two modern FPGA chips. The procedural texture generator was implemented using the Transmogripher-2 (TM-2) rapid prototype system [LGI<sup>+</sup>97], as a part of a 3-D computer graphic rendering system design. The rendering system achieved a performance of over 3 million pixels per second (MPPS). Procedural textures can be synthesized at a rate of 7 MPPS. For textures implemented, we estimated that the FPGA implementations require only 4% to 5% of silicon area of comparable texture memory implementations. Our FPGA implementations also outperform comparable software implementations by 17 times.

The block diagram of the rendering system is shown in Figure 1. The system is designed to be implemented in a mixture of software, ASICs and FPGAs. The world to screen space transformation unit operates on a per triangle basis. It transforms objects in 3-D world space into triangles in 2-D screen space. It is implemented in software as commonly done in many graphic accelerators. The screen to texture space transformation unit calculates the texture coordinates for each texture mapped pixel. An independently derived screen space to texture space transformation algorithm is used. This unit and the frame buffer are designed to be implemented in ASICs. Although in our prototype system, we emulated both in FPGAs. The procedural texture generator synthesizes procedural textures directly in FPGAs.

Six procedural textures, wood, marble, brick, fog, cloud, and fire, were implemented on the system. Two are shown in Figure 2 and 3. Although procedure texture mapping is typically used for complex geometrical shapes like spheres or vases, a simpler cube is texture mapped in both figures. The correctness of a texture implementation can be verified by checking the consistency of the texture across all six faces of the cube. To render a image, a user simply need to load a pre-designed texture into the FPGAs and then specify the object in both world and texture spaces.

Both marble and wood textures use the fractal function  $Fractal = \sum_{i=0}^3 2^{-i} P(u^i, v^i, w^i)$ , where  $P(x, y, z)$  is the Perlin noise function [EMP<sup>+</sup>94]. We modified the basic Perlin noise algorithm to achieve a fast and efficient FPGA implementation. Major improvements to the algorithm include generating pseudo random numbers on the fly in XOR tables [Rau91] and using a 3-D linear interpolation and the smooth function  $3x^2 - 2x^3$  to replace the original wavelet interpolation.

The marble texture,  $M((Fractal(u, v, w) + v) \bmod 128)$ , is created by indexing fractal values into,  $M(x)$ , a 128 entry colour table storing a range of marble colours. The wood texture,  $W((u^2 + v^2 + aw) + Fractal(u, v, w) \bmod 128)$ , is based on the geometrical model of annular rings. The annular rings are simulated using the function  $(u^2 + v^2 + aw) \bmod 128$ . Fractal values are used to simulate the irregularity of tree growth.  $W(x)$  is a colour table of 128 entries storing a range of wood colours. Minimum precision fixed point arithmetics are used for all six textures.

### References

- [LGI<sup>+</sup>97] David M. Lewis, David R. Galloway, Marcus van Ierssel, Jonathan Rose, and Paul Chow, "The Transmogripher-2: A 1 Million Gate Rapid Prototyping System", *IEEE Trans. on VLSI Systems*, pp 188-198, June 1998.
- [EMP<sup>+</sup>94] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Worley Steven, *Texturing and Modeling: A Procedural Approach*, AP Professional, Boston, 1994.
- [Rau91] B. Ramakrishna Rau, "Pseudo-Randomly Interleaved Memory", *ACM*, 1991.

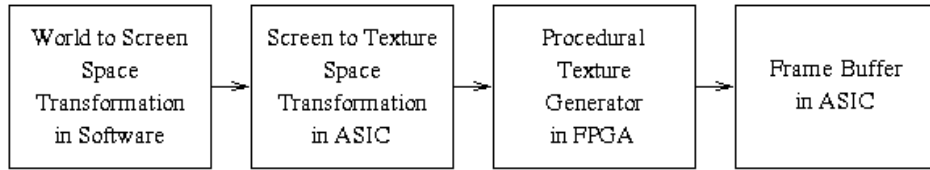


Figure 1 3-D Rendering System Design

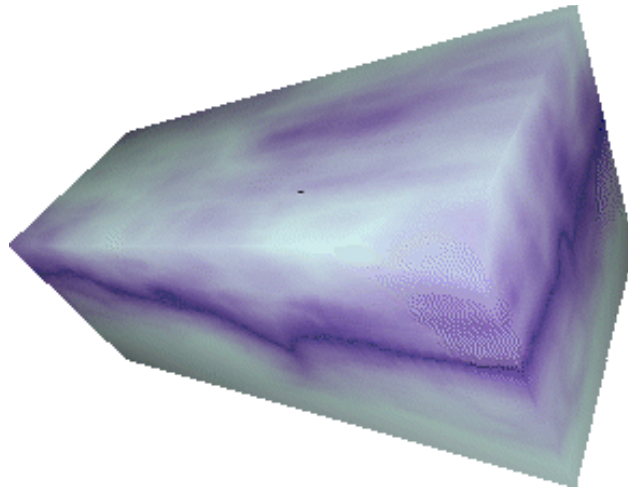


Figure 2 Marble Texture Mapped Cube

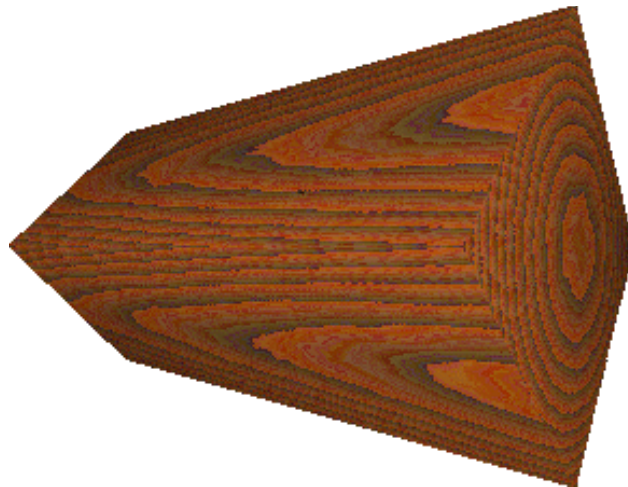


Figure 3 Wood Texture Mapped Cube