Solid Texturing on a Per-Pixel Basis

Rüdiger Westermann, Thomas Ertl

University of Erlangen, Computer Graphics Group
Am Weichselgarten 9, D-91054 Erlangen, Germany
Email: wester@informatik.uni-erlangen.de

Abstract

Over the last years OpenGL has positioned itself as the favored API in high-end graphics programming. In particular, by providing access to hardware supported texture interpolation a fundamentally new class of algorithms was initiated. When 3D textures became available interactive rendering of volumetric data sets showed up as a potential application. However, space filling textures can be seen in a more general context. In this paper we will show an efficient way for carving arbitrary shapes out of solid textures by exploiting 3D pixel textures coming along as OpenGL extension on SGI Impact architectures. We make extensive use of the fragment operations in the rasterization stage thus providing interactive parameter control. Arbitrary changes in the orientation of the object to be textured relative to the solid block can be realized efficiently by our approach.

1 Introduction

Recent advances in the performance and functionality of graphics workstations [1, 6] spawned many new algorithms in computer graphics applications. Dedicated hardware now available on the desktop side provides massive resources on geometry processing, texture mapping and fragment operations. Simultaneously, OpenGL has been manifested as a de-facto standard in graphics programming allowing easy and intuitive access to advanced graphics operations. As a result, over the past years the benefits of high-end graphics workstations have been exploited in many different algorithms.

In particular, 3D texture interpolation and pixel blending operations have positioned themself as a potential machinery enabling real-time rendering of complex volume data sets [2, 10, 3]. Apparently, 3D texture mapping is performed on a per-vertex basis, texture coordinates interpolated properly across triangles.

However, when accomplished in this way accurate and direct modifications of texture coordinates across polygons can hardly be achieved. On the other hand, even when solid textures [7, 8, 4] are to be evaluated it is often desired to change the assignment of texture coordinates relative to the object, or to directly control the generation of texture coordinates in a more flexible and subtle way.

In this paper we propose an efficient technique for carving arbitrary shapes out of solid objects via 3D texture maps. In this way we

- *enable solid texturing in real-time* by taking advantage of hardware supported geometry processing, color interpolation and texture re-sampling
- *avoid topological constraints* by performing texture mapping on a per-pixel basis
- *provide direct control of texture coordinates* by exploiting simple pixel transfer operations.

The remainder of this paper is organized as follows. First, we briefly outline the benefits of 3D textures by exemplifying the traditional way they are used in volume rendering applications. Then the mechanism and the advantages of pixel textures are described. We demonstrate their use for solid texturing of arbitrary objects and sketch ways they can be exploited for real-time texture animations.

2 Texture mapped volume rendering

The basic idea in volume rendering via 3D textures [3, 2, 10] is to exploit texture mapping hardware for the re-sampling of material values defined on regular grids. Once the scalar data is loaded into a texture map, planes parallel to the image plane are clipped against the parametric texture domain (see Figure 1). The texture mapped cross-sections are blended appropriately into the framebuffer thereby approximating the continuous volume rendering integral.

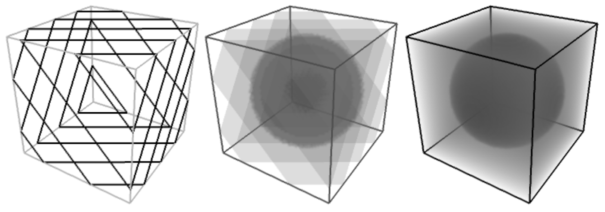
Figure 1: Volume rendering via 3D textures.

During rasterization texture coordinates issued at the vertices of each slice are smoothly interpolated across the polygon and mapped to the texture. Since texture coordinates are assigned on the per-vertex basis subtle control of texture values generated within polygons can hardly be achieved. By defining texture coordinates on the per-pixel basis the functionality of texture mapping can be improved considerably.

3 Pixel textures

On the SGI Impact architectures the **pixel texture** extension is now available which allows the user to directly control texture coordinates on the per-pixel basis. Specified in the same way traditional 3D textures are, rather than issuing texture coordinates at vertices they are issued at pixel locations. Before pixel data from main memory is drawn into the framebuffer the RGB components are interpreted as 3D texture coordinates and mapped appropriately. Finally, the re-sampled texture values are drawn.

Figure 2 demonstrates the basic mechanism. First, a squared polygon is rendered. At each of the four vertices the color values are assigned as shown. Then the framebuffer is read into main memory and it is written back with enabled pixel texture. Color values are mapped to the specified texture and the corresponding texture values are drawn.

Obviously, the same effect can be achieved by specifying texture coordinates at the vertices and rendering the textured square. However, let us

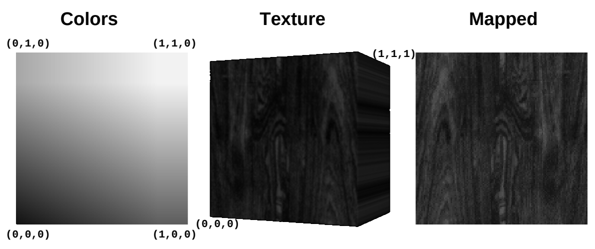

Figure 2: Texture mapping via 3D pixel textures.

assume that we want to arbitrarily distort texture values within the square. When texturing is performed on the per-vertex basis this can only be achieved if the square is represented by a large number of smaller elements for which the texture coordinates have to be modified. The benefit of pixel textures is that we just have to modulate the color values appropriately before they are written into the framebuffer.

Modulation can be accomplished in software, but more efficiently pixel transfer operations can be used. If the OpenGL color matrix extension is supported, when color components are written they are first modified by multiplying them with a 4x4 matrix. Then each component is scaled and biased by specified values. Arbitrary distortions can thus be achieved by initializing the matrix and both vectors properly.

In the following we will outline an efficient way to evaluate solid textures often used in photo-realistic rendering on the per-pixel basis.

4 Solid textures

Solid texturing can be viewed as creating a 3D texture space in which the object to be textured is included. The object is being sculptured out of the texture volume. For each surface point the texture value is evaluated and mapped to the object (see Figure 3). Maybe the most common examples are marble and wood textures used to simulate natural objects.

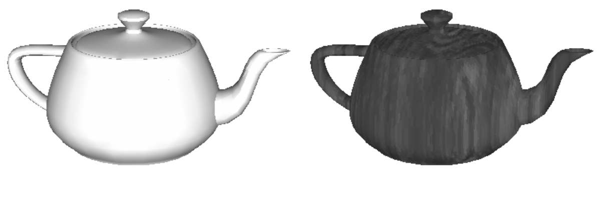
Figure 3: Solid texturing.

Traditionally, solid textures are evaluated by procedural shaders [5, 9]. Once a procedural description to generate the texture has been found, for each point to be textured the shader is evaluated based on the objects geometry. In this way explicitly storing the texture can be avoided and modifications of the texture can be accomplished by simply varying parameters in the procedural definition.

However, the shader has to be evaluated for each surface point, which slows down the rendering process considerably. Obviously, solid texturing in real-time can be achieved by using 3D textures in the usual way. Texture coordinates are assigned at vertex locations and used to map the texture to the object. But to change the mapping the entire geometry has to be traversed thereby updating the texture coordinates. As a consequence, display lists can not be generated in advance, and the size of details which can be directly controlled depends on the size of the textured primitives.

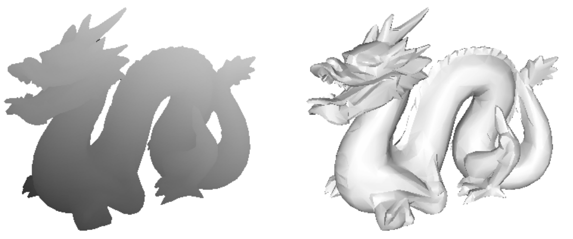
Figure 4: Coding texture coordinates into color values.

Pixel textures provide an even more efficient alternative. Therefore, we store an additional copy of the object to be textured, and we assign color values at each of its vertices. The RGB color components are taken to be equivalent to the vertices local coordinates in the [0,1] texture space. Then the object is rendered with disabled lighting thereby smoothly interpolating the color values across triangles (left image in Figure 4).

Thus for all visible surface points the coordinates are now present in the RGB framebuffer values. We read the pixel data into main memory and render the original object as usual (right image in Figure 4). Finally, the pixel data is written back into the framebuffer with enabled pixel texture. Pixel values are first mapped to the active 3D texture before they are blended with the values already in the framebufffer. By choosing an appropriate blending function the modulation of the objects surface properties by the solid texture can be flexibly changed. The result is shown in Figure 5.

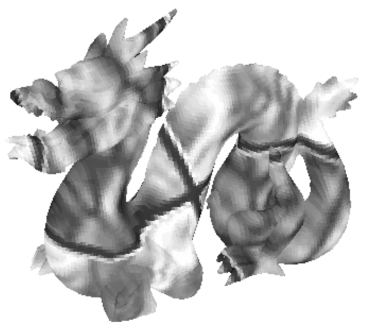
Figure 5: Modulating the dragon's surface by a solid texture.

To restrict the drawing of pixel values to those locations to which a surface point was projected the OpenGL stencil buffer is used. When the shaded object is rendered the stencil buffer is set wherever a pixel is drawn. At every other location it is locked for further modifications.

5 Real-time animations

By applying pixel transfer functions effects can be simulated which can hardly be achieved when texture coordinates are assigned on the pervertex basis. For example, to arbitrarily rotate the object within the texture space only the color matrix has to be initialized properly.

When the pixel data has been read all visible surface points are available in the color components, each of which is within the unit interval. For the rotation to proceed properly they have to be scaled to the range [-1,1]. Then the currently selected rotation is applied. Finally, before the transformed surface points get written to the framebuffer they have to be scaled back to the range [0,1]. The color matrix to be applied looks as follows:

$$CM = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} M_{rot} \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 & -1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note, that even without drawing the object multiple times we can simulate movements of the texture around the object by simply drawing the pixel data successively with the color matrix updated in each pass.

6 Conclusion

In this paper we have shown a novel approach for solid texturing arbitrary objects in real-time. General ideas have been proposed to exploit advanced features offered by high-end graphics workstations. A non usual way to perform the mapping of 3D textures has been demonstrated on the per-pixel basis. The use of pixel textures allows flexible control of texture coordinate generation thus providing an efficient mechanism for the simulation of interactive changes in the orientation of the object or the texture. Furthermore, texture lookup tables allow direct modifications of the solid textures to be used.

References

[1] K. Akeley. Reality Engine Graphics. *ACM Computer Graphics, Proc. SIGGRAPH '93*, pages 109–116, July 1993.

[2] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *ACM Symposium on Volume Visualization '94*, pages 91–98, 1994.

[3] T.J. Cullip and U. Neumann. Accelerating Volume Reconstruction with 3D Texture Hardware. Technical Report TR93-027, University of North Carolina, Chapel Hill N.C., 1993.

[4] Ebert, D. and Musgrave, F. and Peachey, D. and Perlin, K. and Worley, S. *Texturing and Modeling, A Procedural Approach.* Academic Press Inc., 1994.

[5] P. Hanrahan and J. Lawson. A Language for Shading and Lighting Calculations. *ACM Computer Graphics, Proc. SIGGRAPH '90*, pages 289–298, August 1990.

[6] J. Montrym, D. Baum, D. Dignam, and C. Migdal. Infinite Reality: A Real-Time Graphics System. *Computer Graphics, Proc. SIGGRAPH '97*, pages 293–303, July 1997.

[7] D. Peachey. Solid Texturing of Complex Surfaces. *ACM Computer Graphics, Proc. SIGGRAPH '85*, pages 279–286, July 1985.

[8] K. Perlin. An Image Synthesizer. *ACM Computer Graphics, Proc. SIGGRAPH '85*, pages 287–296, July 1985.

[9] Upstill, S. *The RenderMan Companion.* Addision Wesley, 1990.

[10] O. Wilson, A. Van Geldern, and J. Wilhelms. Direct Volume Rendering via 3D Textures. Technical Report UCSC-CRL-94-19, University of California, Santa Cruz, 1994.