Adaptive Solid Texturing for Web Graphics

Bing-Yu Chen and Tomoyuki Nishita[*]
The University of Tokyo

1 Introduction

Solid texturing [Peachey 1985; Perlin 1985] has become a well-known computer graphics technology since it was first presented more than fifteen years ago. However, solid texturing still remains problems today, because it consumes too much time and has a very high memory requirement. Although some methods have recently been proposed to solve these problems, almost all of them need the support of hardware accelerators. Hence, these methods could not be applied to all kinds of machines, especially the low-cost ones available over the Internet. Therefore, we present a new method for procedural solid texturing in this paper. Our approach could almost render an object with procedural solid texturing in real-time using only a software solution. The basic idea of this approach is similar to the cache mechanism used for main memory control. Furthermore, to demonstrate that our approach is widely applicable we choose pure Java for it's implementation, since this could not receive any benefit from the hardware and could be executed on the Internet directly.

2 Cache Access Storage Methodology

Basically, in order to cache the calculated texture data of a texturing function in memory, a "cache cube" is used, as other 3D-texture mapping methods require. The cache cube is empty initially, since there is no texture data has been calculated. When rendering an object with procedural solid texturing, the system checks the cache cube first to see if there is already any corresponding texture data for the drawing pixel, which is contained in a voxel. If the voxel does exist, the pixel is rendered using it, otherwise the desired texture data is calculated and stored into the cache cube as a voxel at its corresponding position. The pixel is then rendered with the calculated data.

Since the required texture data is calculated on demand, it is not necessary to generate a cache cube before implementing the raster process. Moreover, if the cache cube is pre-generated, even if we assume it is a low-resolution one containing just 128 x 128 x 128 voxels, it still costs an un-compressed file size of more than 8MB. This kind of huge data size is difficult to transmit through the narrow Internet bandwidth and requires a lot of memory to store, and it cannot even offer good quality visual effects as Fig. 1 left.

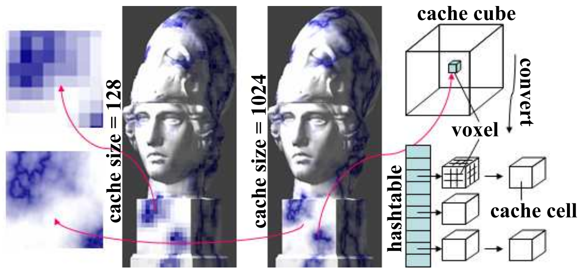

Figure 1: System diagram and results of different cache cube size.

Fortunately, when rendering an object, the visible area is only a small part of the whole shape, because the back and inside faces is obviously not rendered. This rule is independent of the object's complexity, the number of polygons, and the calculation of the texturing function. Therefore, there is only a relatively small amount of texture data needed to be calculated and stored in the cache cube, i.e. the rest of the cache cube remains empty. To store the calculated texture data in the sparse cache cube, we separate each texture coordinate into two parts, one of which is global index, and the other is local index. Therefore, the sparse cache cube is subdivided into several cells due to the global index. Each cell is classified as either a cache cell or just as an empty cell if there is no voxel located within it. To make the retrieval of the voxel efficient, we take the size of the sparse cache cube to the power of two as the requirement for some graphics libraries such as OpenGL. Therefore, we can use some efficient algorithms, like Hashtable, to store the cache cells by using the converted global index, to get good performance for insertion and traversal, as shown in Fig. 1 right. Morefore, according to the LOD (Level-of-Detail) parameter, which is based on the well-known definition of MIP-Mapping [Williams 1983], the system could provide the most suitable levels of several sparse cache cubes for the pixel on the screen. Additionally, since there are several low-cost machines over the Internet, we also provide a mechanism to control the resolution of the cache cube automatically in accordance with the capability of the client machine.

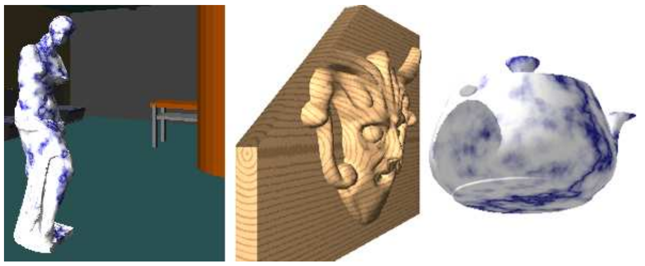

Figure 2: The results rendered with our approach.

3 Conclusion

In this paper, we have proposed a method to render an object with procedural solid texturing for almost all kinds of machines over the Internet[1]. The results are shown in Fig. 2. Although the implementation only uses pure Java, the user could also achieve an almost real-time interactive response. Our approach could be applied to arbitrary models and even their inner surface due to clipping, since it is a general solution for solid texturing, and the capability of the client machine is detected before rendering for Internet heterogeneous environment.

References

PEACHEY, D. R. 1985. Solid texturing of complex surfaces. *Computer Graphics (Proceedings of SIGGRAPH 85) 19*, 3, 279–286.

PERLIN, K. 1985. An image synthesizer. *Computer Graphics (Proceedings of SIGGRAPH 85) 19*, 3, 287–296.

WILLIAMS, L. 1983. Pyramidal parametrics. *Computer Graphics (Proceedings of SIGGRAPH 83) 17*, 3, 1–11.

[*]e-mail: {robin, nis}@is.s.u-tokyo.ac.jp

[1]http://nis-lab.is.s.u-tokyo.ac.jp/~robin/jST/