Adaptive Solid Texturing for Web3D Applications

Bing-Yu Chen and Tomoyuki Nishita
The University of Tokyo
{robin, nis}@is.s.u-tokyo.ac.jp

Abstract

Solid texturing is a well-known computer graphics technology, but still has problems today, because it consumes too much time if every pixel is calculated on the fly or has a very high memory requirement if all of the pixels are stored in the beginning. Although some methods have been proposed, almost all of them need the support of specific 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 solid texturing in real-time using only a software solution. Furthermore, to demonstrate that our approach is widely applicable we choose pure Java for its implementation, since it could not receive any benefit from the hardware and could be executed on the Internet directly.

1. Introduction

Rendering an object with solid texturing [2] [4] [5] is a useful method of showing an object realistically, since it uses 3D coordinates as the parameters to represent the appearance of the shape, but texture mapping only maps a 2D image onto the surface. Unfortunately, to calculate the corresponding texture data on the fly is time consuming, since the texturing function could involve a lot of computational time. Many people wish to pre-generate a 3D image in the beginning so that they can use the 3D-texture mapping functions in graphics libraries. However, if the resolution of the 3D image is low, there is an aliasing problem. If we increase the resolution of the image, the storage requirement becomes a big problem to handle. Moreover, for a solid texturing program running on the Internet, it is also difficult to download a huge 3D image.

2. Adaptive Solid Texturing

Utilizing the cache technology to store some information for re-using is a common hardware technique. For procedural solid texturing, people may use a lookup table to store a sequence of random numbers to enhance the performance [3]. Basically, in order to cache the calculated texture data in memory, we use a 'cache cube' just like other 3D-texture mapping methods require. Initially, the cache cube is empty, since there is no texture data has been calculated. When rendering an object, the system checks the cache cube first to see if there is already any corresponding texture data, which is contained in a 'voxel'. If the voxel does exist, the pixel is rendered using the existing voxel, otherwise the desired texture data is calculated and stored into the cache cube at its corresponding position as a new voxel, and then the pixel is then rendered with the newly calculated data.

Since the required texture data is calculated on demand, it is not necessary to generate a cache cube before the raster process. Moreover, if the cache cube is pre-generated, even if we assume that it is a low-resolution one containing just 128x128x128 voxels, we still have 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 Figure 1 (a).

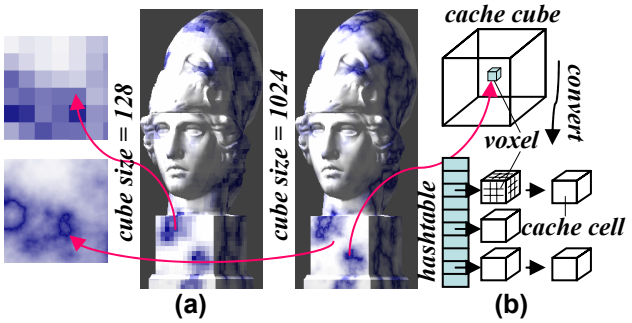

Figure 1: (a) Results of different cache cube size and (b) cache cube conception

Fortunately, when rendering an object, the visible area of the object is only a small portion of the whole shape. Therefore, there is only a relatively small amount of texture data that needs to be calculated and stored in the cache cube, so that the cache cube is sparse. To store the calculated texture data in a 'sparse cache cube', we separate each texture coordinate into two parts, one of which is the 'global index', and the other is the 'local index'. Therefore, the cache cube is subdivided into several cells due to the global index. Each cell is classified as either a 'cache cell' which contains one or more voxels according to the size of the local index or just as an 'empty cell' if there is no voxel located within it. Therefore, we can use a hashtable to store the cache cells by using the converted global index, since it could give us good performance for insertion and traversal as Figure 1 (b).

Following the well-known definition of MIP-Mapping [6], we define the LOD (Level-of-Detail) parameter of our system, so the most suitable level of the cache cube for

each pixel on the screen is decided by it. Moreover, since we also provide the LOD mechanism in our system, multiple cache cubes with different resolutions are used.

The implementation is achieved by modifying the kernel of jGL [1], which is a graphics library for Java and also developed by us. When our system is started up, it first checks which levels of the cache cubes are currently used by interrogating the LOD parameter. Initially, the maximum size of the cache cube is not set, and the number of local index bits is fixed. Once the system receives an out-of-memory exception, the current maximum level of the cache cube will be released, and the maximum size of the cache cube will be set to be one level lower than the current maximum level. Therefore, the client could achieve the optimum quality of the procedural solid texturing commensurate with its performance.

3. Results

To measure the performance of our approach with respect to the traditional methods, we used a marble texturing function and a simple cube as our 3D model in order to reduce the effects of the object complexity. In Table 1, we make comparisons of our new method and two previous methods, one of which calculates the texture data on the fly, and the other which generates a 3D image first and then renders with 3D-texture mapping which is also done by software. For accelerating the performance, we also use a lookup table to cache the random numbers. The test-platform is a notebook PC with an Intel Mobile Pentium III 850MHz CPU, 128MB memory. The applet window size is 500x500. Obviously, both of previous solid texturing and our approaches do not require the preparation of a 3D image, but this is essential for 3D-texture mapping.

	previous solid texturing	3D-texture mapping	single cache cube	multiple cache cubes
fill 3D image	0 ms	42.82 ms	0 ms	0 ms
first loop	0.82 fps	13.64 fps	12.96 fps	5.52 fps
rest loops	0.76 fps	13.95 fps	14.19 fps	9.72 fps
resolution	∞	128x128x128		1024x1024x1024

Table 1: Comparisons of adaptive solid texturing and two previous methods.

During the rendering state, the 3D-texture mapping and using single cache cube in our approach could achieve a real-time response, but the traditional solid texturing could not. To get a similar quality of the previous solid texturing, we use multiple cache cubes with LOD controlling. Although the performance is a little slower then using a single cache cube, but the appearance result is much better than using a low resolution one.

The result of viewing the inner texture is shown in Figure 2 left. Because our method does not only simulate the surface of the object, but also reserves a space for storing the interior texture data, showing the inner texture of an object is akin to showing an invisible portion. Moreover, an integrated example is shown in Figure 2 right, which is a museum and a marble Venus model is exhibited in it.

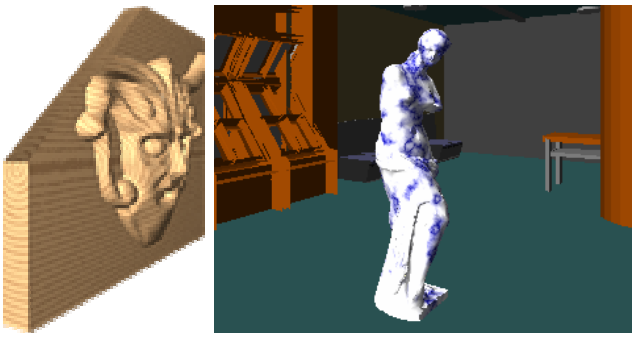

Figure 2: Rendering results.

For the Internet users, since we used only pure Java to develop all of the algorithms and the executable byte-code size is less than 40KB (jar-compressed), the use of our system on the web[1] is possible. Moreover, unlike 3D-texture mapping, our approach has no requirement to generate any cache cube at the out-set, and all of the calculation and resolution adjusting is done on the fly, so users will not need to take up a lot of time downloading huge image data files or waiting for something to be prepared.

4. Conclusion

In this paper, we have proposed a new and simple method to render an object with procedural solid texturing for almost all kinds of machines over the Internet. Although the implementation only uses pure Java, the user could also achieve an almost real-time interactive response. Since there are several low-cost machines over the Internet, we also provide a mechanism to control the resolution of the cache cubes automatically in accordance with the capability of the client machine.

References

[1] B.-Y. Chen and T. Nishita. jGL and its applications as a web3d platform. *Proc. of Web3D 2001*, pages 85-92, 2001.

[2] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and modeling: a procedural approach*, 2nd ed., AP Professional, 1998.

[3] J. P. Lewis. Algorithms for solid noise synthesis. *Computer Graphics (SIGGRAPH 89 Proc.)*, 23(3):263-270, 1989.

[4] D. R. Peachey. Solid texturing of complex surfaces. *Computer Graphics (Proc. of SIGGRAPH 85')*, 19(3):279-286, 1985.

[5] K. Perlin. An image synthesizer. *Computer Graphics (Proc. of SIGGRAPH 85')*, 19(3):253-262, 1985.

[6] L. Williams. Pyramidal parametrics. *Computer Graphics (Proc. of SIGGRAPH 83')*, 17(3):1-11, 1983.

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