Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

An Adaptive Octree Textures Painting Algorithm

Gang Dang, Hang Li
School of Computer Science
National University of Defense Technology
Changsha, P.R. China
gangdang@nudt.edu.cn, donald_leed@163.com

Zhi-Quan Cheng†, Kai Xu, Yan-Zhen Wang, Bao Li
PDL Laboratory
National University of Defense Technology
Changsha, P.R. China
{Cheng.zhiquan†, Kevin.kai.xu, Yanzhen.Wang, sibao.li}@gmail.com

Abstract—**Traditional Texturing using a set of two dimensional image maps is an established and widespread practice. However, it is difficult to parameterize a model in texture space, particularly with representations such as implicit surfaces, subdivision surfaces, and very dense or detailed polygonal meshes. Based on an adaptive octree textures definition, this paper proposes a direct reverse-projecting pixel-level painting approach which has less storage requirements to general octree textures maps. In addition, it depends on texture lookup in the GPU, which particularly lookup faster than the non-GPU program.**

Keywords—**Painting, reverse-projection, adaptive octree textures, GPU**

I. INTRODUCTION

Interactive painting of complex surfaces is an important problem in the digital film community and 3D computer graphics field. This has recently led to several commercial applications based on the well-accepted mesh painting metaphor. To modify a given high resolution mesh, the designer simply uses a brush of adjustable size to directly paint the transformations onto the model's surface. The transformations itself can be as diverse as texture or color painting.

Painting color or texture information onto meshes was first introduced by Hanrahan and Häberli [1], and later extended to painting geometric details. Traditionally, texturing a model with image data requires setting up a 2D parameterization of a model. However, this is a difficult work to the programmers. Parameterizing a model in texture space can be very difficult, particularly with representations such as implicit surfaces [2], subdivision surfaces, and dense or detailed polygonal meshes.

The traditional volume texture define the volume around the model and create the coordinates of voxels automatically. It is no necessary to parameterize the model. However, they occupied so many memories, which made them impractical for general use and confined their use to primarily medical and scientific imaging [3].

Octree textures only store the subset of the volume that actually intersects the surface of a model [4]. Thus a sparse texture is created. The octree structure provides an efficient way to store and look up these textures.

We address the interactive painting problem with an efficient reverse-projecting approach that stores paint in an octree-like GPU-based adaptive data structure. Our algorithm allows interactive 3D painting of complex, unparameterized models. It differs from previous work [4,5,6] in two important ways: first, it uses an adaptive Octree data structure implemented entirely on the GPU, and second, it enables interactive performance with high quality by reverse-projecting pixel-level painting and fast data accesses.

II. RELATED WORK

Texture mapping was first introduced by Catmull [7]. Essence of which is the mapping from a 2D texture plane to the surface of a 3D model. This concept was extended with the solid textures. The programmers were liberated from the tedious work of the parameterization of a model. But the volume texture occupied so many memories which could make them unrealizable.

Benson and Davis [4] presented the octree textures and used the sparse octree structure to store the the octree cells which intersect an object's surface. Their method saved a lot of memories and became a standard data structure in the texture mapping. DeBry et al [5] developed a similar approach to texture mapping using octrees. And octree textures only cost the same or even less memory than the 2D textures. Octree is a regular layer data structure. The first node which is a cube is the root. Each node has eight children or none. The children were created when the parent node was subdivided in each axial. If the node contains empty space, then it's no necessary to subdivide. If the node intersects the surface of the model, it is divided into eight parts. The steps repeat until the target of resolution is reached. Now the octree textures have been realized on the GPU [8,9]. The octree is stored in the texture memory, and could be called by the fragment program. It depends on texture lookup in the GPU and can lookup faster than the program.

III. ADAPTIVE OCTREE STRUCTURE AND USAGE

A. Storage

To build an adaptive octree, a regular octree which has a low depth such as 1 or 2 should be built first. Then the volume was subdivided according to the target of resolution. The high resolution area has more nodes while the rest has less ones. We store the adaptive octree into a 3D texture which is used to save the value of RGBA with 8 bits. This 3D texture is called indirect pool. The indirect pool is divided into indirect grid. The grid, corresponding to one node, is a cube composed of 2×2×2 cells.

The main construction of the node is a data array which stores the index pointing to the children or the value of the color of the leaf. The alpha channel is a sign to judge whether the node is a child or a leaf. So the data and the index are both RGB form whose values range from 0 to 255. And the root of the adaptive octree is stored at the (0, 0, 0) of the indirect pool.

B. *Lookup*

After the tree is stored in the pool and realized on the GPU, it would be called in the fragment program. If we want to get the value of a point *M*, we find the root of the tree and the child node which contains the point. These steps are repeated until a leaf was found at last and then the color was obtained.

Let the coordinates of point *M* be (X_D,Y_D,Z_D) at the depth *D*, I_D be the index of the visited grid, and S be the number of the grids in the indirect pool, the lookup coordinates *P* is

$$P = \frac{I_D + (X_D, Y_D, Z_D)}{S}$$

Then, we get the RGBA value which is stored at point P. According to the value of alpha channel, we get the color if the node is leaf or the next index if the node is child (Fig. 1). The Fig. 1 also illustrates how to obtain I_{D+1} from I_D.

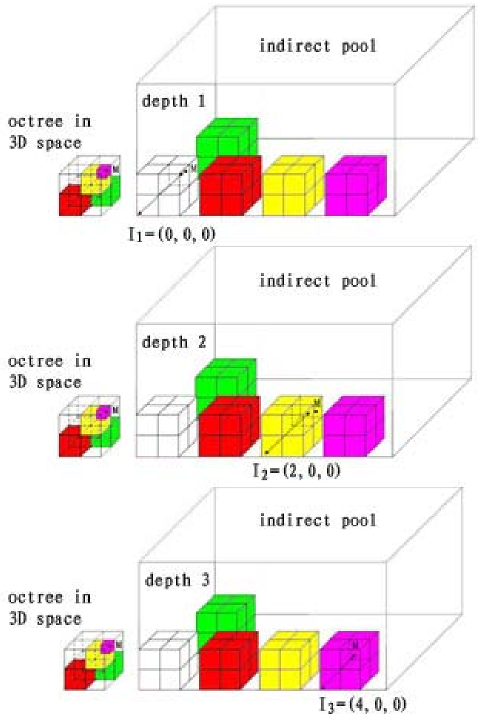


Figure 1. Lookup in the adaptive octree textures with depth 3.

The lookup ends at the leaf. In fact, most hardware can support limited recursion. The program must limit the depth of the adaptive octree to prevent from the break that always appears over the acceptable visited times.

C. *Interpolation*

To maintain a continuous field of color, tri-linear or tri-cubic interpolation is used. Since the adaptive octree texture is a kind of volume textures, the interpolation need eight samples. If some of the samples are not in the tree, distortion will be caused. We add these nodes into the tree to solve this problem. To do this, the program create two bounding box to each node. One is regular, another is extended. If the extended box of a sample contains a node of the tree, the program will add this sample into the tree (Fig. 2).

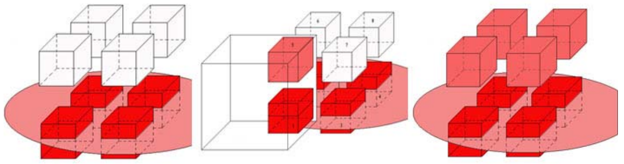


Figure 2. The tri-linear interpolation add the necessary node into the adaptive octree.

D. *Comparison with Volume and Octree Textures*

Octree texture is unrealizable for the models with huge datas or large surface. Adaptive octree textures can handle with some of these models which have a few high resolution areas, and keep the detail as well as the octree textures.

For comparison, we paint a point on a model which is in the resolution of 256×256×256. If we use the volume textures and each voxel store a set of RGBA value, the memory occupied should be 256×256×256×4B = 64M. To keep the same detail, if the depth of the octree textures was 8 and the children's number of each node was 4, the whole number of the nodes would be:

$$\sum_{k=0}^{7} 4^k = \frac{4^8 - 1}{4 - 1} = 21845$$

Each node stores 8 sets of RGBA value. So the cost of the memory is about 683K. Thus the octree textures save more memories than volume textures. If we use the adaptive octree textures, only one node of each layer need to be subdivided while other conditions are the same with the octree textures. The depth of the initial octree is 2. Then the number of the nodes is 5+8×6=53. And the memory is about 1.7K.

The formulas (Fig. 3) about the number of nodes and the depth are:

Volume: $y = \left(2^{x-1}\right)^3 = 8^{x-1}$, x=1,2,3,…,8 (1)

Octree: $y \approx \frac{4^x - 1}{4 - 1} = \frac{4^x - 1}{3}$, x=1,2,3,…,8 (2)

Adaptive octree: $\frac{4^x - 1}{3} \ge y \ge 5 + 8(x-2) = 8x - 11$, x=2,…,8(3)

From the analysis, it is observed that: The adaptive octree textures cost less memory than the octree textures at the same resolution, and can present high quality details at the higher resolution when the regular octree textures are un-realizable.

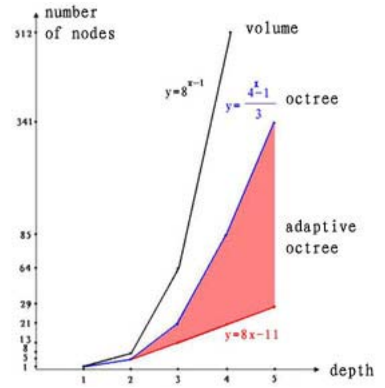


Figure 3. The comparison of the volume, octree and adaptive octree.

IV. Pixel-level painting

Based on the adaptive octree textures, we propose a reverse-projecting pixel-level painting algorithm. The basic procedure of the algorithm is "input a picture file and a model first, get the color of each pixel and save the value within an array, then paint the colors on the model one by one". This paper chooses the BMP file to be the input picture file. It is easy to extract the value of the RGB color of each pixel.

A. *Reverse-projection*

Generally, the visualization on the screen is the 2D projection of 3D real objects. In this paper, we reverse this process: put the BMP file at the screen position then re-project it to the model (Fig. 4).

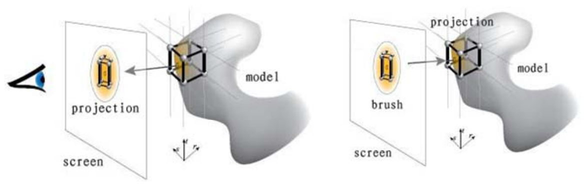


Figure 4. Traditional projection (left) and reverse-projection (right).

Just illustrated in the section III, we can deal with the adaptive octree textures at pixel-level. Consequently, we can repaint the model with the reverse-projection in theory and practice. Because of the complex surface or mask, it is not easy to repaint a model perfectly with just one picture. Some times we may need a lot of photos of an object to repaint its model.

We use a simple center justifying reverse-projection scheme. Based on the pixel-level adaptive octree textures, the program could automatically paint the model line by line just like the CRT.

B. *Fixing the breaking point*

There are some breaking points in two situations. One is the big angle between the screen and the surface of a model. The other is that the size of the pixel of the screen is not adapted with the size of the leaf well (Fig. 5).

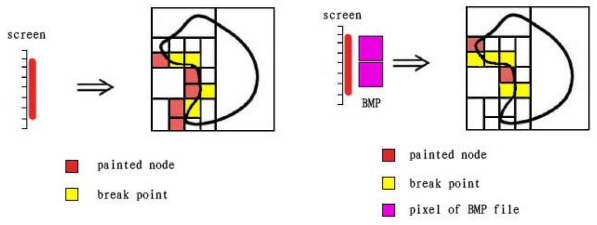


Figure 5. The reasons of breaking points. Left, big angle between screen and surface can cause breaking point. Right, the pixel is not equal to leaf will cause breaking point too.

If the surface of a model is plane and parallel to the screen, and their size are equal, breaking points won't be caused. But most of the real objects have complex surface, especially at the edge of the model. To solve the first problem, this paper uses a repairing method which gets the color around the brush, computes the interpolation, and paints the color on the breaking point. During the implementation, the coordinate of the brush's position is get first. Then, it is input into the GPU program to lookup table and find the target point. Next the GPU program computes the average value of the colors of the nodes around the point. At last, it paints the average color at the point.

For the second problem, we adopt a regional painting method. When the process of painting on the model, we may move, scale or rotate the model. Thus the size of the pixel of screen may bigger than the size of the leaf of the adaptive octree textures or smaller. If the pixel is smaller than the leaf, one more pixel's color will paint at one leaf point. Some of the color will be lost. If the pixel was bigger than the leaf, the node between the pixel will not be painted. We first scale the model until its leaf is a little smaller than the pixel of the screen. Then we enlarge the brush area. Not to paint only one node, but to paint a region around the node (Fig. 6).

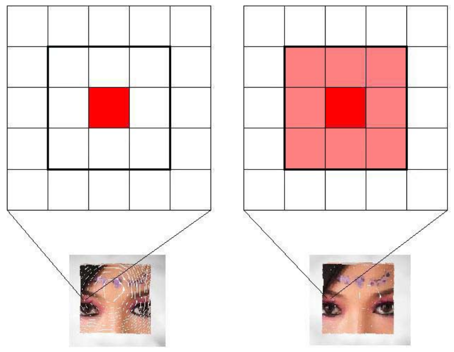


Figure 6. Paint a region instead of paint a node.

To do this, we test the program and get useful empirical value of some parameters. Such as the radius of the brush, threshold value *f* within which a node should be painted, and an attenuation coefficient β which is used to create the shallow color around the node.

$$f = Radius + \frac{1}{2^{depth}}$$

$$\beta = 1.0 - \max\left(0.0, \min\left(1.0, \frac{(d - Radius \times 0.5)}{Radius \times 0.5}\right)\right)$$

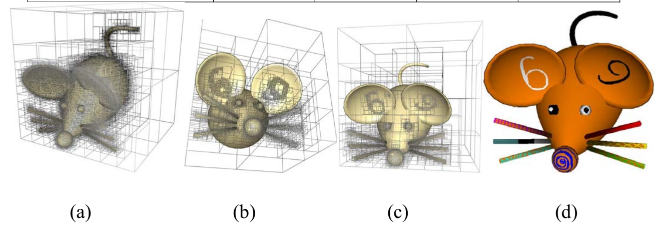
where $Radius = \frac{1.6}{2^{depth-1}}$.

V. RESULTS

First we compare the memory cost with the octree and the adaptive octree (Table 1 and Fig. 7). They show that using the adaptive octree textures would largely save much more memories than the octree textures.

TABLE I. THE DATAS OF OCTREE AND ADAPTIVE OCTREE

Model	***Memory***	***Building time***	***Max depth***	***Number of nodes***
Mouse(Fig. 7.a)	2.01M	170s	8	65964
Mouse(Fig. 7.b)	285K	1s	8	9119
Mouse(Fig. 7.c)	1.02M	1s	9	33410
Laddy(Fig. 7e)	562.5K	30s	8	18000
Laddy(Fig. 7f)	1.12M	1s	9	36781



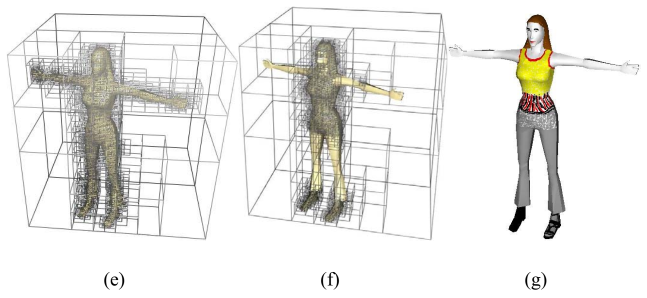


Figure 7. (a)(e)Traditional octree with the depth 8. (b)Adaptive octree with the max depth 8. (c)(f)Adaptive octree with the max depth 9. (d)(g)The painting result using the adaptive octree with the max depth 9.

Then we choose some models and test our pixel-level painting algorithm (Fig. 8). To get the results quickly, the depth of the adaptive octree textures is 9, which means the resolution is $512 \times 512 \times 512$. If we change the depth to 10 or further, we'll get a fine detail. The frame rates for viewing textured models in our 3D paint application (showed in the accompany video) are unaffected by the use of a single octree texture map. The frame rates are determined entirely by the complexity of geometry and varied between between 15 and 30 fps with models ranging in complexity from 50k to 1Mpolygons. Frame rates while painting depend on the size of the current brush, and we maintain highly interactive rates during painting.

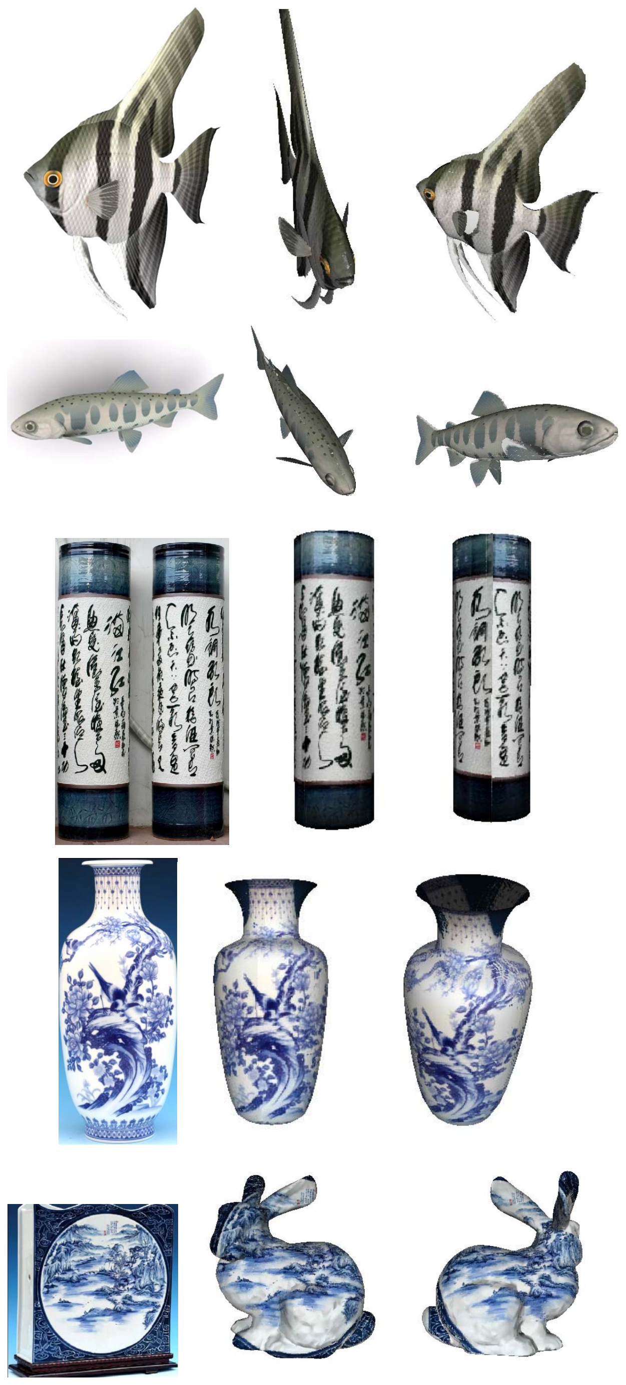


Figure 8. Final result of using the painting algorithm on some models. The left column are the BMP pictures. The rest are the painting results.

VI. CONCLUSTION AND FUTURE WORK

This paper presents the construction of adaptive octree textures, and compares the storage with the volume texture and the octree texture. Based on the construction, this paper proposes a novel reverse-projecting pixel-level painting algorithm which could paint a beautiful picture on the surface of a model. The experiments have shown the algorithm is robust, and use smaller in memory to the traditional octree textures at the same resolution.

There are some improvements that should be made to the current algorithm. Firstly, though the program can automatically paint each point, it will take a long time to paint millions of points. Most of the time was waste on subdividing the nodes. How to speed the construction of adaptive octree textures is a problem to be solved. Secondly, the program allows the user to move the model, so it's hard to match the pixel with the leaf exactly. The break point will be caused inevitably during the process. This produces an extra work to fix the break points. Thirdly, similar to the octree textures, the adaptive octree textures also lose the ability to paint directly on the maps. With the ability to add detail at any scale, even the adaptive octree textures can grow to be much large when it is needed to represent a particular set of texture on each point of the model.

REFERENCES

[1] P. Hanrahan, P. Häberli. Direct WYSIWYG painting and texturing on 3D shapes. In Proc. of ACM SIGGRAPH, 1990, pp. 215-223.

[2] J. Bloomenthal. "Polygonization of implicit surfaces," In Proc. of ACM SIGGRAPH, 1988, pp. 123-129.

[3] J. T. Kajiya and B.P.Von Herzen. "Ray tracing volume densities," In Proc. of ACM SIGGRAPH, 1984, pp. 321-329.

[4] D. Benson, J. Davis. "Octree Textures," In Proc. of ACM SIGGRAPH, 2002, pp. 785-790.

[5] D. DeBry, J. Gibbs, D. Petty, and N. Robins. "Painting and rendering textures on unparameterized models," In Proc. of ACM SIGGRAPH, 2002, pp. 763-768.

[6] N. A. Carr, J. C. Hart. "Painting detail". ACM Transactions on Graphics 23, 3, 2004, pp. 845–852.

[7] E. E. Catmull. "A subdivision algorithm for computer display of curved surfaces," Master's thesis, University of Utah, December 1974.

[8] M. Pharr. "GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation". Amazon Press, 2007, pp. 425-438.

[9] J. Kniss, A. Lefohn, R. Strzodka, S. Sengupta, J. D. Owens. Octree Textures on Graphics Hardware. In Proc. of ACM SIGGRAPH, 2005. (poster)