Available online at www.sciencedirect.com

SciVerse ScienceDirect

Physics Procedia 33 (2012) 548 – 554

Physics Procedia

2012 International Conference on Medical Physics and Biomedical Engineering

Fast parallel algorithm of triangle intersection based on GPU

Zheng wang[a], Gaojun Ren[a], Liangeng Zhao[a], Meijun Sun[b]

[a]*School of Computer Software,Tianjin University,Tianjin, China*
wzheng@tju.edu.cn
[b]*School of Computer Science and Technology,Tianjin University,Tianjin, China*
sunmeijun@tju.edu.cn

Abstract

As triangular intersection of ray tracing algorithm is of the significant proportion in the calculation, improving the light and triangle intersection calculation speed has a significant role in accelerating the ray tracing algorithm. This paper presents a fast light and triangle intersection parallel algorithm based on GPU. The algorithm reduces the amount of triangle intersection test computation in the way of octree subdivision of space, and simplifies the triangle intersection test by using the triangle barycentric coordinates. According to the experimental results, the algorithm has a great speedup than calculated in the CPU.

Keywords:component; GPU; triangle intersection; ray tracing; barycentric coordinates; octree

1.Introduction

Ray tracing algorithm plays an important role in generating 3D realistic graphics. It can reproduce natural light reflection and refraction in the scene, and calculate the total intensity of its role, resulting in realistic visual effects. Currently, the algorithm has been widely used in various games, commercial animation software.

Since the optical paths are reversible, most of the ray tracing algorithms simulate light via tracking light emitted from the point of view. Tracking each light emitted from the point of view invokes a large number of intersection tests. Compared to traditional scan line algorithm, ray tracing algorithm needs a very large amount of computation with about 75% to 95% of the amount used to calculate the intersection operation. Therefore, speeding up the intersection testing in ray tracing algorithm plays an important role in improving the overall efficiency of the ray tracing algorithm. Because each light is independent of each other and each triangular patch is also independent, it is possible for parallel processing computing..

2.Research Status

Today the technologies of realistic rendering mainly include the following[3][8][9] : rasterization, ray casting, radiosity algorithm, ray tracing algorithm. Of these rendering technologies, ray tracing algorithm can simulate the reflection, refraction, scattering, dispersion and other advanced optical effects, which is hard for other rendering technologies.

Ray tracing algorithm is a computationally intensive algorithm, which includes following steps, light generation, space traversal, intersection testing, rendering[7]. Among these steps, intersection testing needs the most large amount of calculation. In ray tracing, improving the speed of rendering is often at the expense of realism results[10]. To overcome this problem, many studies have focused on hardware-accelerated algorithm[7]. As a result, in recent years the use of GPU-accelerated ray tracing algorithm has become an important trend.

3.Ray and Triangle Intersection Acceleration

3.1.Theory of GPU Computation

GPU and CPU architectures vary widely. GPU has an astonishing ability to handle floating-point operations with relatively less functional modules which makes most of transistors composed of various types of dedicated circuit, numbers of pipelines.

Compared to CPU, GPU has more memory bandwidth and has a larger number of execution units. We use CUDA as a programming tool, which is NVIDIA GPGPU model. In the CUDA architecture, the program is divided into two parts, namely host side and device side. The smallest unit of execution at device side is thread and several threads form a block. Threads under the same block can access the same shared memory, and can be synchronized quickly.

3.2.Algorithm Overview

We use octree spatial subdivision algorithm to subdivide the space of the scene. The algorithm divides the space cube in three directions which contains the entire scene, into eight sub grid, or bounding box, organized into an octree. A further subdivision will be done if the number of vertices the bounding box contains is more than a given threshold. This process will continue until the number of vertices each leaf node contains is less than a given threshold and the leaf node holds the information of triangular patches surrounded by vertices. When the light goes through the scene, a simple bounding box intersection operation can be done to skip those bounding boxes which has nothing to do with the tracking light, reducing the number of intersection operations. As a result, the computation time can be reduced. Once we find the leaf node intersected with the tracking light, the intersection computation operation can be done with the tracking light and triangular patches contained in the leaf node. The vertices in the triangular patches can be represented in the form of triangle barycentric coordinates. By computing the intersection point with tracking light and the triangular patch, we can known whether the tracking light intersects with the triangular patch according to that whether the intersection point is within the triangle.

3.3.Hardware Accelerated Algorithm

3.3.1 Build Octree

As a method of scene organization, octree subdivision algorithm is widely applied in the computer graphics systems which can significantly reduce the time of sorting the polygons in the scene.

- Regular octree structure used to represent spatial information, stores the index of each node's child nodes. This avoids integer arithmetic to find child nodes, thus improving efficiency. Generally, algorithms like scanning or sorting all nodes in the octree can make each node have similar number of elements, keeping the balance of octree and improving the efficiency of intersection testing operation. We use the quick sort algorithm to subdivide the element and build octrees. The algorithm steps are as follows:
- Sort the vertices in the scene with X axis, Y axis, Z axis coordinates and we get the data needed to represent a bounding box, namely Max_x, Min_x, Max_y, Min_y, Max_z, Min_z.

A further octree subdivision will be done with the bounding box represented in the last step.

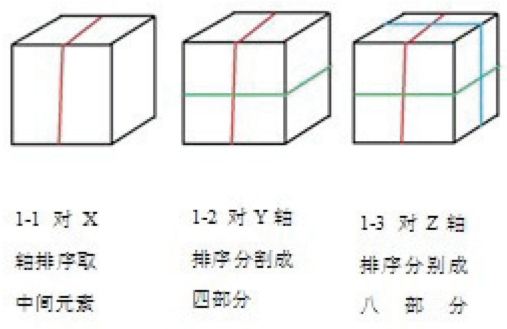

1-1 对 X 轴排序取中间元素
1-2 对 Y 轴排序分割成四部分
1-3 对 Z 轴排序分别成八 部 分

Figure 1. Process of building a Octree

First, we subdivide the X axis after sorting the X axis coordinates to obtain the index of middle element , dividing the scene into two parts. Then we subdivide the Y axis of every part after sorting the Y axis coordinates to obtain the indexes of each middle element, dividing the scene into four parts. At last, the same operation will be done to the Z axis to divide the scene into eight parts. During the subdivision , the data needed to represent the sub bounding box will be recorded.

- Do recursion with the step two until the number of vertices in node is less than a given threshold (set threshold as 8 for example).
- Octree is built.

3.3.2 Intersection with Bounding Box

After the scene subdivision with the octree algorithm, the intersection between tracking light and the scene turns into the intersection between tracking light and the bounding box. Let the equation for the tracking light as $L = E + t * D$, where E is the coordinate of the point of view or the start point of the light, D is the unit direction vector of the light, t is the parameter of the equation. When the light and the outer bounding box intersect, there are two intersection points on the parallel side on the outer bounding box. We can get the two parameters t as t_(xyz)max and t_(xyz)min, via changing the equation of the light. If the three intervals of value t have a intersection, the tracking light and the bounding box intersect. Otherwise they do not intersect. According to the above arguement, the light and bounding box intersection testing algorithm is as follows:

- According to the unit direction vector D, compute the each intersection intervals between the tracking light and the parallel triangular plane.

- Determine the intervals in each direction:

Whether the intervals [t_(xyz)min, t_(xyz)max] intersect.

- Return the status of whether the scene and the tracking light intersect.

3.3.3.Intersection with Patch

After determining the smallest bounding box intersected with the tracking light, we need to compute the intersection between the tracking light and the triangular patches in the bounding box.

With the barycentric coordinates, the vertice P on the triangular patch can be represented as $p(a,b,c) = a*X + b*Y + c*Z$ where X, Y, Z are the triangle's three vertices, and a, b, c are the coefficients greater than zero with $a+b+c=1$. Let $a = 1-c-b$ and the above equation can be represented as: $p(b,c) = X + b*(Y-X) + c*(Z-X)$, turning the three-dimensional problem into two-dimensional, which reduces the problem complexity.

Let the light equation as $L = E + t*D$ and the three triangular vertices as X, Y, Z. The tracking light and the triangular patch will intersect when the following condition is met.

$$E + t*D = X + b*(Y-X) + c*(Z-X)$$

$$(b+c<1), b>0, c>0$$

Expanding its vector form results in three linear equations:

$$E.x + t*D.x = X.x + b*(Y.x - X.x) + c*(Z.x - X.x)$$

$$E.y + t*D.y = X.y + b*(Y.y - X.y) + c*(Z.y - X.y)$$

$$E.z + t*D.z = X.z + b*(Y.z - X.z) + c*(Z.z - X.z)$$

The value of t, b, c can be get via the theory of numerical analysis.

Above the pseudo code of the intersection testing between tracking light and the triangular patch is as follows:

```
Bool triTest()
{
 Bool Intersect = false;
While( has triangular patches in the bounding box)
{
Compute the value of t, b ,c;
If( t < t0 || t > t1) continue;
If( c < 0 || c > 1) continue;
If( b < 0 || b > 1 - c) continue;
                    $t\_x\max = (Max\_x - E.x)/D.x$
                    $t\_x\min = (Min\_x - E.x)/D.x$
                    $t\_y\max = (Max\_y - E.y)/D.y$
                    $t\_y\min = (Min\_y - E.y)/D.y$
                    $t\_z\max = (Max\_z - E.z)/D.z$
                    $t\_z\min = (Min\_z - E.z)/D.z$
Intersect = true;
}
return Intersect ;
}
```

Figure 2. Process of patch intersection

4.CUDA-based Algorithm

4.1.Experimental Environment

Computers for experiment are configured as follows. CPU: Inter(R) Core(TM) Duo CPU E8400 @ 3.00GHz, 2GB memory, GPU: NVIDIA GeForce 9800 GT with 512MB memory , OS: Microsoft Windows 7 Ultimate , CUDA 3.0，coding Tools: Microsoft Visual Studio 2005, C/C++。

4.2.Host and Device Functions

4.2.1.Host part

The functions in host are mainly used for initialing the devices, such as initialing data for CPU and GPU, controlling the processing of implementation, calling the kernel functions and cleaning the memory.

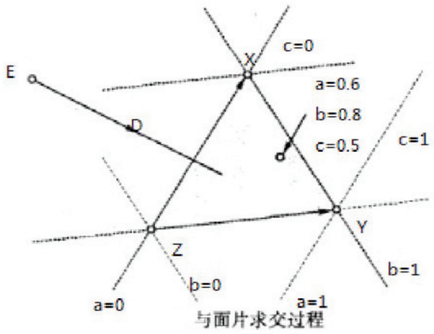
Calling the kernel functions Host is like the following:

```
for (int i= 0; i < scr->y; i+=block.y)
{
    Intersection<<<block,threads,0>>>(d_triangles,
d_vertex,d_aabbs,d_eye,
d_src,i,d_color,num_aabb,stack);
}
```

4.2.2.Device part

Functions in device are used to compute the intersection with the tracking lights, which start from the point of view and go through serval lines of pixels on the screen.

The declare of kernel functions is as follows:



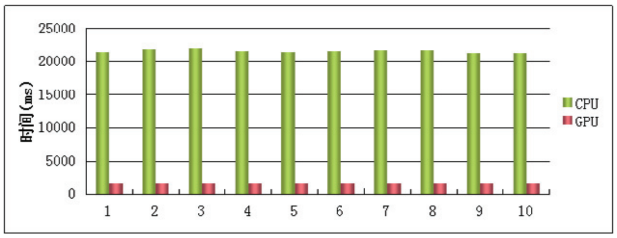
```
__global__ static void Intersection(Triangle* d_triangles, Vertex * d_vertex,
AABB *d_aabbs, float3* d_eye, Screen * d_src, int hy,
float3* d_color,int num_aabb,int * stack);
```

The declare of the function to compute the intersection with bounding box is as follows:

```
__DEVICE__ bool boxTest(AABB* d_aabbs, int curr, float3 vray, float3* eye);
```

The declare of the function to compute the intersection with triangular patches is as follows:
__device__ bool triTest(Triangle* d_triangles, Vertex * d_vertex,int index,
float3 vray, float3* eye, float3 &interPoint, float& t1,float t0)

5.Result



- When the size of screen is 800*600, the coordinate of the view is（0,0, Z）, loading 855 vertices, 1692 triangular patches, the results as running in GPU and CPU are as follows:

Table 1. Resulting data of the experiment

Z axis of view coordinate	700	600	500	400
CPU (ms)	22857.521484	22543.431641	22267.318359	20968.052734
GPU (ms)	1717.573730	1710.362305	1690.115356	1589.392456
Accelerate rate	13.308	13.180	13.175	13.192
Average of accelerate rate	13.21375			

- When the size of screen is 800*600, the coordinate of view is（0,0, 400）, loading 855 vertices, 1692 triangular patches, running in GPU and CPU each 10 times, the average accelerate rate is 13.599, the result is as follows:

Figure 3. Resulting data of the expriment

6.Conclusion

Ray tracing algorithm plays an significant role in realistic rendering so improving the computing efficiency is very important. We use the high-speed parallel computing power of GPU and the octree structure to speed up the intersection between light and triangle. Experimental result shows that this algorithm has a huge accelerate rate than in CPU.

References

[1] Yichen Han, Kejian Yang, Based on space partition algorithm for fast ray tracing. Computer Science and Technology, 2010,01-29

[2] Wenxi Wang, Shide Xiao, Wen Meng, Hong Dong, A space partition based on octree ray tracing algorithm technology. Computer Applications III 2008, 3

[3] Qing Lan, GPU-based ray tracing algorithm and implementation. Computer Science and Technology. 2009,05-11

[4] Hua Zou, Xinbo Gao, Xinrong Lu. Based on three-dimensional rendering hierarchical bounding box acceleration algorithm. Wuhan University Press ,Information Science, 2009, 3

[5] Timothy J.Purcell .，Ian Buck .，William R.Mark ，Pat Hanrahan.。Ray Tracing on Programmable Graphics Hardware。Acm Transactions on Graphics.21(3), pp. 703-712, 2002

[6] John Amanatides.，Andrew Woo。 A Fast Voxel Traversal Algorithm for Ray Tracing. EuroGraphics, 1987, pp. 1-10

[7] Qing Lan, GPU-based ray traversal algorithms KD tree, Computer and Communication, Hunan University. 2009, 04-07

[8] Apple A. Some techniques for machine rendering of solids. AFIPS Conference Proceedings. San Francisco, California,USA,1968,32,37-45

[9] Goral C, Torrance K E, Greenberg D. Modeling the interaction of light between diffuse surfaces. Proceedings of the 11th annual conference on Computer graphics and interactive techniques. Minneapolis, Minnesota, USA, 1984, 18(3), 213-22

[10] Glassner A S. An Introduction to Ray Tracing. San Diego, CA, USA: Academic Press, 1989, 341-352