Parallelized Generation of Photon Texture and Real-time Rendering on GPU

Liu Xiaoping1 Chen Hao1,2 Xie Wenjun1 Wang Qijun1
1) VCC Laboratory School of Computer and Information University of Technology
   Hefei 230009
2) The 38th Institute of China Electronics Technology Group Corporation
   Hefei 230031

Abstract

In the basis of photon texture data generated by parallelized algorithm, we propose the real-time rendering algorithm. This method stores pre-computed results of photon mapping as textures before rendering, and looks for adjacent photons by programming on GPU(Graphics Processing Unit). The algorithm takes full advantage of the Computing capacity of GPU to accelerate the process of searching photon map. The experiment has proved that this algorithm can enhance efficiency of rendering for real-time interaction, also ensure rendering effect.

1. Introduction

What is always strived in computer graphics is realistic rendering, among which algorithms used to render realistic graphics rapidly is the hot spot and also difficult problem yet. Actually, the complex light-computing is hard to resolve in realistic rendering. In 1996 Henrik Wann Jensen proposed formally the photon mapping algorithm which was developed in the next few years. The most advantage of this algorithm is that it can calculate the diffuse reflection during objects and the caustic phenomenon caused by refracted rays. These phenomenons can not be solved easily before photon mapping was putting forward. Photon mapping algorithm ingeniously solves questions like caustic rendering and medium scattering by twice rendering consisted of generating photon and evaluating of radiosity.

Photon mapping algorithm has been one of the most prevalent algorithms of global light computing for the advantages it has. A large mount of studies on photon mapping algorithm emerges both here and abroad. Bibliography [2] proposed a so-called STF algorithm which uses photon mapping for pre-computing. The author applied it to rendering translucent object of its second surface and finally gets a better effect. Accord to this function in photon mapping algorithm of rendering medium, Bibliography [3] gives out one way to compute speed of photon when it spread in medium. Bibliography [4] realize photon mapping on a programmable hardware with which it accelerates algorithm. The main problem in photon mapping algorithm is how to find out the photons that around the specified photon among the enormous photons. There are a lot of ways were proposed to solve such a question. For instance, KD tree can be used to accelerate the process of searching by its space division algorithm. In bibliography [5], only a few photons were needed to be calculated to get the final color of the sampler point for each photon’s attribution was pre-computed before. Bibliography [6] also reduces the searching time the process need by hashing all the photons around the specified one. Clustering method was used in Bibliography [7] to deal the caustic photon. PRT[8] (pre-compute radiometry passing) is a representative pre-computing algorithm beyond which a lot of investigator have discussed in many aspects. For instance, Wang Jing finished its algorithm called full frequency shadow algorithm on GPU with his teammates. And Anders has proposed that it is welcome to realize global lighting-rendering for a large scene based on pre-computed local radiated passing. But we need to compress the enormous data pre-computed and decompress it when rendering. All these ways, in varying degrees, enhance the efficiency of time, but it still cost a lot resource when looking for adjacent photon around the sampler point so photon mapping algorithm can not basically be able to achieve real-time rendering.

As the development of graphic processor, GPU also start to display its characteristics such as supporting 32bit floating computing, rendering to texture and its flexible ability of be programmable. And now, more and more general tasks could be run on GPU[11].
General computing based on GPU is widespread applied to realistic rendering to enhance rendering realistic feeling, improve rendering efficient. For example, we could achieve radiation algorithm, light-tracing algorithm and light-projecting algorithm or hair’s realistic rendering on GPU. According to these characteristics, this dissertation proposed an algorithm based on photon texture. First, we pre-compute simultaneously the photon texture data in the specified scene. Due to the characteristic that GPU can index texture conveniently, we store the pre-computed results of photon mapping into textures. When comes to render, we get the photon by index so as to accelerate the rendering process. Parallelized algorithm succeeds to generate photon texture simultaneously, and it also improves the computing speed.

2. Rendering algorithm based on photon texture

In the basis of photon texture, this algorithm condenses the second computing in photon mapping algorithm. And it takes full advantage of GPU’s programmable ability so as to find the way that is to look for the photon in texture instead of searching for the entire photon map. As texture index is a linear computing, it speed rendering up a lot while using photon mapping. The algorithm can be divided into three steps: photon tracing, photon texture computing and light computing. Here is the detail:

2.1. Photon tracing

The first step in photon mapping is to spread photons into the entire scene with regard to static light source. The different between the traditional photon mapping and the new one is that the results were stored into photon texture structure for rendering conveniently. The elaborate steps of light tracing as follows:

Store the pre-computed results into photon texture structures. Send photons from the source under certain rules. Generally, point source will send photons to random directions but face source will send photons as a cos- function.

Trace the movement of photon and get the point photon intersect with scene.

According to the plane belong to the attribution of two objects intersected; the algorithm demands the photon’s next position: reflected or digested. And the diffuse reflection will blend the color of the plane with the color of itself.

If the photon was absorbed, stored the photon to the photon texture. (The structure will be introduced latter).

After all this steps, we get the distribution of the photons around the object’s surface.

2.2. Photon texture

After all the photons were spread into the scene, there may be more than one photon in each texture unit. So a texture structure should be design to store all the information of several photons into just one texture. Photon texture is composed of index layer and data layer and figure 2 shows how it looks.

The layer structure is good for storing photon graphics. We could adjust the size of photon texture according to the amount of photons to achieve the purpose that each texture unit store more than one photon. On the other hand, to a certain extent, this structure saves the usage of memory. The layer of index includes two textures, the first one store the index of photon corresponded to the texture unit. The second one stores the number of photons. And the layer of data includes three textures: photon position, photon power and photon direction. They all have the same type of GL_FLOAT. These three divisions was separately stand for the position, power (RGB instead) and the direction of photon(X value, Y value, Z value).

And all these were used to compute light when rendering. Generally, due to the confine of ordinary texture, the stored texture data layer have the type of GL_TEXTURE_RECTANGLE_ARB which can use texture with arbitrary size, that is, each channel of RGBA in texture is stored by a 32bit float number.

After getting index and number of photon in the index layer, we can locate the coordinate of texture in data layer and further get the correspond photons.

Figure 1 structure of photon texture
2.3. Rendering based on photon texture

The results of photon tracing are stored in the textures for the faces of objects, and the illustration compute based on programmable GPU while rendering. The evaluating formula of radiosity goes as below:

\[ L_i(x, \Psi_r) \approx \sum_{p=1}^{N} f_i(x, \Psi_r, \Psi_{i,p}) \frac{\Delta \Phi_p(x, \Psi_{i,p})}{r^2} \]

This formula is consistent with that for evaluating radiosity with photon mapping, and the difference is the method of searching for N photons. In photon mapping, the whole space or all its divided nodes should be searched for N photons, however in our method, only textures index of neighbor space need to be searched for photons. Therefore our method is linear and greatly improves the efficiency of rendering. The procedure goes as follows:

1. While shading rasterised fragments, input models and their photon textures (including index texture and data texture), and the fragment pixels’ texture coordinates are computed by the vertex programs.
2. According to the texture coordinates, texture values are accessed of index texture in a range from the near to the distant, then use the accessed values to query the data texture in order to get the photon’s energy and directions.
3. Summarize all photons’ contribution with the render formula to obtain the final illustration.

It will be a long process if the generation of photon textures is computed serially. The generation is the process of emitting multi-particles into the scene from the light source, and the tracing of each photon’s reflection, refraction and absorption is independent. Obviously, the tracing of each photon can be treated like a cell procedure. The whole process’ parallelization can be achieved by compute parts of photons in nodes. The generation of photon textures is done in the control node after collecting each node’s result.

In this paper, a master-slave form is used for parallelization of photon textures with clusters. The procedure goes as figure 2.

The distribution of tasks and models and the collection of results are done on the control node. Besides, the compute nodes finish their tasks and reply the results.

If there are N compute nodes, a free node will be find first and be delivered 1/m of all tasks, therefore this node traces 1/M photons of all (for the balance of loads, \( M \geq N \)).

Figure 2 structure of Algorithm of photon texture

If there is no free node, it’s considered that all compute nodes are working, then the control node will be suspended and waiting for results.

If a compute node’s result is received, the control node will consolidate it with final result and deliver 1/M of remaining tasks to that node. If the 1/M of remaining tasks a less than a threshold value, the task of the threshold value will be assign to the node. If there are small numbers of photons, the computing will be very fast, therefore it has less influence in the load balance.

The compute nodes have simpler jobs: waiting for tasks while being free, start computing while get task and return the result after computing.

3. Results and Analysis

The method of generating photon textures is implemented on the Dell Power Edge 6800 (4 CPUs) with 10,000,000 photons. The efficiencies of different threshold level (P) and number of tasks (M) was compared. The time of computing goes like table 1.

As above, deliver 1/M of tasks to free node is in the course of load balance. Both M and P influences the rate of acceleration. If M approaches the number of compute nodes, the slowest node will affect the efficiency, meanwhile, the load balance will be very bad with too large P. If M is big and P is small, the tasks will be divided too tiny, and greatly increase the communications between control node and compute nodes. The efficiency will be affected even if the time of calculation is balanced.
Table 1 Compare of Time Using Photon Texture

<table>
<thead>
<tr>
<th>Para Algorithm (s)</th>
<th>Two Nodes Time (s)</th>
<th>Two Nodes Ratio</th>
<th>Four Nodes Time (s)</th>
<th>Four Nodes Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>P=100,000 M=5</td>
<td>3,285</td>
<td>1.866</td>
<td>893</td>
<td>3.52</td>
</tr>
<tr>
<td>P=100,000 M=10</td>
<td>1,776</td>
<td>1.85</td>
<td>893</td>
<td>3.68</td>
</tr>
<tr>
<td>P=1,000 M=40</td>
<td>1,815</td>
<td>1.81</td>
<td>885</td>
<td>3.71</td>
</tr>
</tbody>
</table>

By a serial of experiments, the tendency of M and P’s value, the rate of acceleration is first increased and then decreased relative to M. For a good rate of acceleration, M should be twice of the number of compute nodes, and P should be $\frac{1}{M^2}$ of the number of tasks.

Figure 3 Scene using local lighting
Figure 4 Pre-compute results of Photon
Figure 5 Scene render by photon texture
Figure 6 Another Effect

On a computer with Intel 3Ghz, Geforce 8600 and 1G ram, the expected results of algorithm are shown by textures. Take 1,000,000 photons for example, the scene is as figure 3, contains 2993 triangles. The distribution of photons of pre-computing is as figure 4. Results with different photon numbers are shown by figure 5 and figure 6. Real time rendering can be achieved while the radius of searching is 1. But because of the limitation of the radius, wrong result appears in the edge of texture units, it can be resolved by enlarging search radius.

Render speeds of different search radius is shown by table 2. The more neighbor photons it used the more accurate result it achieves. Therefore with the fixed density of photon textures, accuracy of illustration can be improved by enlarging search radius. Compare to the photon texture with search radius 2 and searching of 50 neighbor photons, to the photon mapping based on kd-tree with searching of 50 neighbor photons, it can be seen that the algorithm in this paper makes a great improvement of efficiency.
Table 2 Efficiency of Rendering

<table>
<thead>
<tr>
<th>Scene</th>
<th>Pre-Compute Time(s)</th>
<th>Speed (Photon Texture) (ms)</th>
<th>Speed (KD Tree) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>r=1 15 Photons</td>
<td>r=2 50 Photons</td>
</tr>
<tr>
<td>Scene 1</td>
<td>2993</td>
<td>326</td>
<td>119</td>
</tr>
<tr>
<td>Scene 2</td>
<td>2960</td>
<td>263</td>
<td>104</td>
</tr>
</tbody>
</table>

In applications, because of the limitation of graphics card, using 2D textures can store more data. As in this paper, the card Geforce8600 has the max texture width of 8192, so nearly 7,000,000 photons can be stored. It is enough for normal scenes, but in complex scenes, more photon textures are needed and affect the efficiency. The implementation is the same as single texture.

4. Summarize and Prospect

In this paper, the storage structure of photon texture is designed and the algorithm based on photon texture is implemented. The index structure for single texture unit can implement multi-photons’ storage, and resolved the storage for results of pre-computing, and accelerated the search of neighbor photons while rendering.

For the diffuse illustration between objects, inevitably, photon texture need longer time for pre-computation, it can be accelerated by parallelization or the faster structure of space division. Otherwise, for the popular render of projected z-buffer mode of VR, the algorithm now support only static light source, and calculate the distribution of photons of dynamic light by GPU general computing.

5. Acknowledgements

Our work was supported by National Science Foundation (No. 60673028).

6. References


