GPU-Accelerated Visual Hull Rendering

Bingfeng Zhou*, Donghai Xie**
Institute of Computer Science and Technology
Peking University
Beijing, P.R.China
*cczbf@pku,edu,cn
**xiedonghai@icst.pku.edu.cn

Abstract **-** In this paper, a method using programmable GPU to accelerate visual hull rendering is described. The method creates the geometry model of a static object. Then to explore the parallelism of the GPU, the triangle mesh model of the visual hull is fed into GPU for coordinate transformation and visual hull texture sampling. Using this method, real-time user interaction of the image-based virtual object is achieved.

Keywords: Image-based Rendering, Visual Hull, Hardware Acceleration, GPU, Cg

1 Introduction

Visual hull [4] is an efficient technique for image based rendering. Its idea can be traced back to a kind of object description method called shape-from-silhouette ([1],[2]). The visual hull method takes several photographs containing the image and silhouette of the target object from different view positions as input. The images are then used to construct the convex hull of the target object. Theoretically, the visual hull method calculates the intersection of the viewing cone defined by the viewing position and silhouette of each photograph and uses this intersection as the visual hull of the target object. By sampling the corresponding source-image pixels of each surface point on the hull, the color of the surface point is decided.

When implement the visual hull algorithm, the visual hull is obtained by using one dimensional CSG algorithm, that is, when rendering each pixel of a particular scenery image, the 3D surface point of the visual hull is decided by applying an 1D CSG operation of the parts of the viewing line mapped from each silhouettes. The complexity of this algorithm depends on the number of the source images and usually too slow to achieve the real time rendering,. In its original implementation, the real time rendering is achieved using parallel calculation among multiple interconnected computers [4].

Form the algorithm point of the view, the original form of visual hull algorithm [4] is parallelizable because the 1D CSG operations as well as the following texture sampling steps for each rendering pixel are independent of each other. This parallelism makes it possible for the algorithm to be implemented on a given hardware equipped with a kind of the parallel calculation mechanism, and the modern programmable GPU graphics hardware is just a suitable facility. GPU (Graphics Processing Unit) is a kind of specially designed hardware for fast rendering of 3D graphics. Its fundamental rendering mechanism is to perform the rendering calculation in parallel because most of the image generations in computer graphics can be parallelized on pixel basis. Besides its parallel mechanism, anther outstanding feature of modern GPUs is its programmability and the corresponding programming environments are also available now [3].

In this paper, we describe a method to render the image based visual hull using programmable GPUs. The target of this method is to achieve the real-time interaction of the virtual object generated using image based visual hull.

There are already some works on rendering visual hulls using GPUs [5]. But most of them only use the texture mapping functions of the GPUs directly and leaves the visual hull calculation to the separate networked computers to explore the parallelism on an image basis, which increases the complexity of the system.

In our work, instead of using multiple networked computers to reconstruct the visual hull, we employed a Marching Cube [6] algorithm to reconstruct the visual hull, and then using this pre-calculated visual hull model to render the visual hull image in a view-dependent way, which can fully utilize the parallelism of the GPU hardware. The method keeps the

advantage of "optimal texturing from multiple silhouette images" and can achieve real time interaction of static virtual objects.

The remaining of this paper will be organized as following: Section 2 will describe the overview of our rendering system. Section 3 will describe our method of visual hull reconstruction. Section 4 describes our method of using GPU to accelerate visual hull rendering. Section 5 is the experiment and results.

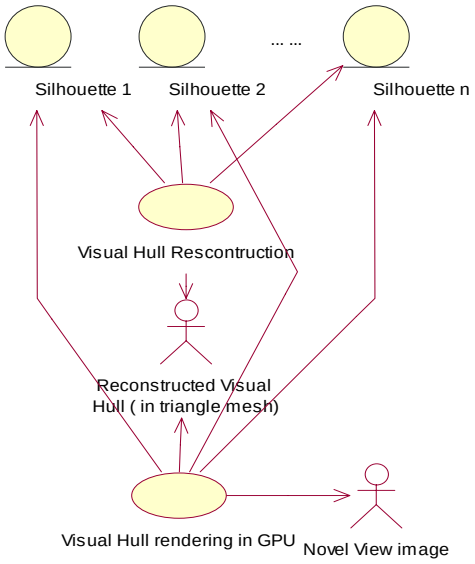

Figure 1 System overview

2 System overview

The organization of our rendering system is shown in Figure 1. In our system, the input is a set of images taken using an off the shelf digital camera. For each input image, the camera is automatically calibrated by the system for its position, view direction and internal parameters using a specially designed calibration chart (Figure 2). When the image is fed in to the system, the background is blue-screened manually, based on which the silhouette is extracted automatically and is represented in the form of polygon.

In our system, we chose to reconstruct triangle-meshed visual hull prior to the real-time rendering of the novel images. This is because that it is a most efficient way to for the GPUs to render the triangle-meshed model and the target of our system is to render the static object.

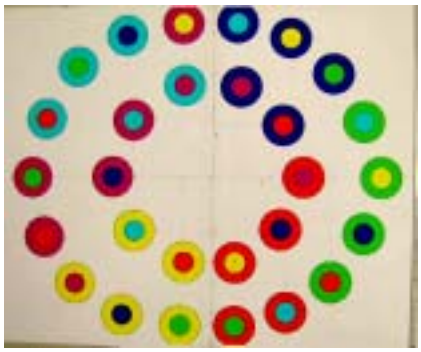

Figure 2 Camera calibration pattern

After the visual hull reconstruction, the reconstructed model is represented in the form of triangle mesh; the triangle mesh is then fed into the GPU for the visual rendering, which is exactly based on the method given by Matusik et al [4]. Since this algorithm is parallel-able and "optimized with respect to the view", we obtained a close-to-realism virtual object rendering with a real-time user interaction.

3 Visuall hull reconstruction

3.1 Image-based visual hull and its reconstruction

In the original form of image-based visual hull rendering [1, 4], no visual hull exists explicitly in the whole process. When rendering, the surface point of the visual hull corresponding to the pixel of the rendering plane is calculated using 1-D CSG operation. Given the 3D coordinate of the surface point and the camera parameters of each silhouette image, the point can be easily mapped on to the pixels of each silhouette images and the optimal rendering color value can be chosen.

This original form of the method is quite suitable for the real-time rendering of dynamic scene captured with several video cameras. It can also be implemented in a multi-computer context to achieve real-time rendering. But with respect to the GPU implementation, this algorithm can not be efficiently implemented due to the limitations of current GPU programmability.

The GPU is designed to rendering 3D geometry models efficiently. So the most efficient form of the data they can process is triangles. Given this nature of GPU, we choose to generate the explicit model of visual hull instead of calculating it in rendering time.

If the visual hull is treated as the intersection of viewing cones defined by the silhouettes, there are many works available for the visual hull reconstruction. In [7], viewing cones are constructed and the visual hull is obtained using 3D boolean operation. The work described in [2] subdivides the space in to voxels and obtain the result by mapping these voxels into each silhouettes and collecting those voxels falling inside all the silhouettes. In our system, we choose the "Marching-Cube" method described in [6] to obtain the explicit geometry model, that is, the triangle mesh, of the visual hull.

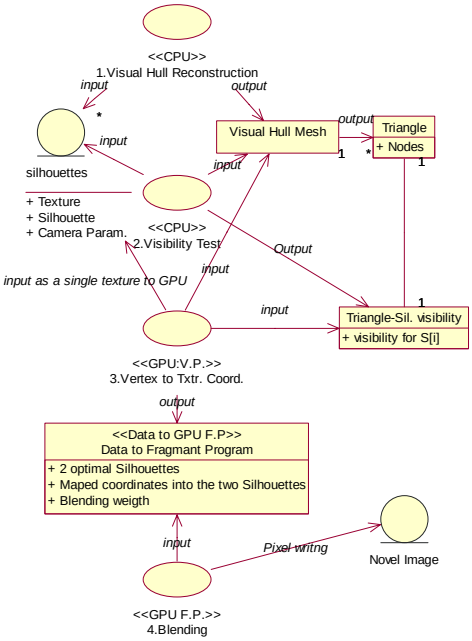

Figure 3 Procedures of our visual hull rendering

When reconstructing the geometry model using Marching-Cube method, the 3D space occupied by the target object is subdivided into small regular cubes. Each cube is then intersected with the surface definition of any target geometry (in our situation, the visual hull) and the obtained intersection surface constitutes the surface of the geometry. To obtain a triangle mesh of the visual hull, the cube is further divided into 6 tetrahedrons [8], and thus the obtained the surface model is a triangle mesh.

In the visual hull context, the visual is defined implicitly. To obtain the intersection of each edge of the cube, we used a binary subdivision scheme [8]. In this process, the subdivided edge is mapped into the each silhouette images to decide, in a similar way to the one in [2], whether it falls inside the visual hull, and we abandon those that fall complexly inside or outside the visual hull. When the remaining edge is shrunk small enough, it is considered as a point, and it is exactly the intersection point wanted.

4 Hardware accelerated visual hull rendering

Visual hull reconstruction is the first stage in our rendering process. This stage is carried out in a off-line way, which means, for each virtual object, the stage is performed only once. For the remaining part of the visual hull rendering, except the silhouette-visibility test [4], we'll put them into GPU to achieve the real-time user interaction (Figure 3).

4.1 Silhouette-visibility test

In the original form of the image-based visual hull rendering [4], after the 3D coordinate of a surface point is decided, the 3D coordinate is mapped into each silhouette images to sample the corresponding color values. Since silhouette images are taken around the target object, a 3D point may not be seen from every viewing position of the silhouette images. So a visibility test of the 3D points with respect to a silhouette image is necessary. This test is novel-view-position-independent, so it can be done in an off-line way.

For the similar reason, not every silhouette image is valid for sampling for a given 3D surface point, and so some silhouette images should be neglected for the sampling, this is also done in the same stage as the visibility test.

In our implementation, we employed the rendering mechanism of the available graphics system such as OpenGL to do the visibility test. By rendering the z-buffer of the mesh from the viewing position of a silhouette image and by comparing the z coordinate of the 3D point in the viewing coordinate system with the corresponding z value of the z-buffer, the visibility of that 3D point can be easily decided. We perform the visibility test on a triangle basis. The visibility of a triangle with respect to each silhouette image is attached to the triangle data. As for the valid silhouette image selected, only two optimal images are chosen (Figure 3).

4.2 Coordinate transformation in GPU vertex program

The remaining calculation of the visual hull rendering for a novel viewing point is performed per pixel, and so they can be performed by GPU in parallel. According to the programming architecture of the GPUs [3], these calculations are divided into two steps. The first is to perform the coordinate transform from vertex coordinate to the texture coordinates for silhouette image sampling. This step is performed in the vertex program of the GPU. The second part of the calculation is the image blending between the two optimal silhouette images, which will be described in the next section.

When implementing this step in vertex program, considering the programming limitation, the multiple silhouette images are programmed in a single texture to be fed in to the vertex program, the camera parameters are send in to the vertex program as the *uniform* parameter of Gg [3], visibilities are send into the vertex program in the place of a spare geometry data "*normal*".

4.3 Texture blending in GPU fragment program

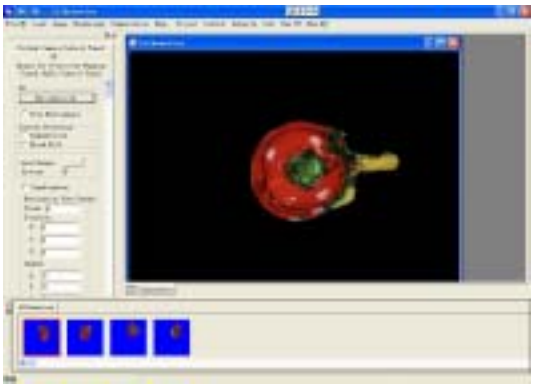

Figure 4 Results (Pentium4 2.1G，512MB RAM，GPU: NVIDIA GeForce FX 5900XT)

The result of the GPU vertex program includes two texture coordinates indexing into the texture buffer for the sampling of the two silhouette images. The sampled color values are to be blended by a weight calculated in the vertex program, the weight is calculated using the following formula [5]:

$$W_k = 1/\arccos(\vec{d}_k \bullet \vec{d})$$

and the blending is done by :

$$C = \left[\sum_{k=1}^{N} W_k * T_k\right] / \sum_{k=1}^{N} W_k$$

where C is the blended color value to be send to the output pixel; $\vec{d}_k$ is the viewing direction of the silhouette k; $\vec{d}$ is the view direction of the novel view; T_k is the sampled color value from silhouette k; N is total number of silhouettes take part in the blending, in our situation, N=2.

5 Results

An rendering system is implemented accroding to the descriptions in the previeous sections. Figure 4 is one of its output. The visual hull is constructed using 23 silhouette images. When render for the real time user interaction, only four images are are enough. In this experiment, the frame rate of the real-time user interaction is 50 fps.

References

[1] A.Laurentini, "The visual hull concept for silhouette-based image understanding", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol.16, No. 2, pp. 150-162, Feb. 1994.

[2] R.Szeliski, "Rapid octree construction from image sequences", *CVGIP: Image Understanding*, Vol.58, No.1, pp. 23-32, July 1993.

[3] R. William, R. Mark, S. Glanville, K. Akeley, M. J. Kilgard, "Cg: A System for Programming Graphics Hardware in a C-like Language", *ACM Transactions on Graphics*, Vol. 22, No. 3, pp. 896-907 , July 2003

[4] W.Matusik, C.Buehler, R Raskar, S.Gortler, and L.McMillan, "Image-Based Visual Hulls", Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2000), pp. 369-374, July 2000

[5] M.Li, M.Magnor, and H-P.Seidel, "Hardware-accelerated visual hull reconstruction and rendering", Proc. Graphics Interface 2003, pp. 65-71, 2003

[6] W.E.Lorensen , H.E.Cline, "Marching cubes: A high resolution 3D surface construction algorithm", *ACM SIGGRAPH Computer Graphics (SIGGRAPH1987)*, Vol. 21 No. 4, p.163-169, July 1987

[7] S. K. Srivastava, "Octree generation from object silhouettes in perspective views", *Computer Vision, Graphics, and Image Processing*, Vol. 49, No.1, pp.68-84, Jan. 1990

[8] J.Bloomenthal, “An Implicit Surface Polygonizer”, Graphics gems IV, pp. 324 - 349, 1994.