Real-time Stereo Visual Hull Rendering Using a Multi-GPU-accelerated Pipeline

Jie Feng* Yang Liu Bingfeng Zhou
Institute of Computer Science and Technology, Peking University

Reference images

Figure 1: Stereo visual hull rendering result using our improved pipeline. With 47 input reference images, a rendering speed of about 30 fps can be achieved on a double-GPU personal computer.

1 Introduction

Visual hull is an efficient technique for image-based rendering. It takes a group of images as input, produces an approximate geometry model of the object, and renders it from arbitrary new viewpoints. Explicit visual hull models (as polygon meshes or point sets) are usually insufficient in accuracy. Image-based visual hull (IBVH) method [Matusik 2000] reconstructs implicit models, and produce view-dependent rendering results with higher quality.

There have been a number of hardware-accelerated visual hull rendering methods, some of which are also able to reconstruct and render dynamic objects [Li 2004]. However, most of these methods adopt complex distributed systems, which are composed of several client PCs for data acquiring and preprocessing, and a server for reconstructing and rendering.

In this paper, we propose a novel real-time IBVH rendering pipeline, which enrolls a single PC with multiple GPUs instead of a distributed system. We first optimize the rendering algorithm to fit the parallel computing. Then, with the help of multiple GPUs, new views from different viewpoints of left/right eyes can be rendered synchronously, producing stereo images of the object with a rendering speed of about 30 fps.

2 Standard Image-Based Visual Hull Rendering

The input of the IBVH method is a group of calibrated images of the object from different directions. Given a novel viewpoint, a new view is generated with the following steps [Matusik 2000]:

1. For each pixel of the desired view, a viewing ray from the virtual camera center is calculated, and projected onto a reference image to get an epipolar line.
2. The epipolar line intersects the 2D silhouette (formed by a group of line segments) on the reference image, and generates a group of 2D intervals.
3. The 2D intervals are projected back to 3D space, producing corresponding 3D segments on the viewing ray. Then the intersection of all the 3D segments from all the reference images indicates the visual hull boundary at current pixel.

Here, the intersection operations are restricted to 2D planes, and thus have lower computational complexity. Finally, the depth of each pixel is calculated and its color is extracted from the nearest reference images.

* The work was supported by the NSFC grants (No. 60973054).
Email: fengjie@icst.pku.edu.cn

3 Algorithm Optimization for Parallel Computation

Though the standard IBVH algorithm has relatively high efficiency by calculating view-dependent implicit visual hulls, its rendering speed is not fast enough for real-time applications on common PCs. Fortunately, when generating a novel view, the calculation for each pixel is independent. Hence the algorithm will benefit a lot from GPU-acceleration.

We first adjust the original algorithm to make it more suitable for running on GPUs. The rendering task is decomposed into multiple parallel threads, each for one desired pixel, which are fed to GPUs simultaneously.

Moreover, there are several important optimizations that we make to obtain higher efficiency:

(1) To accelerate the intersection of epipolar lines and 2D silhouettes, the original algorithm sorts all silhouette line segments in *bin* structures in preprocessing. That consumes a lot of CPU resources and is not suitable for GPU implementation for its sequential feature. Instead, we perform a simple linear intersection test between each line segment and the epipolar line. Only when the two endpoints of the segment lie on different sides of the epipolar line, the intersection operations are performed.

(2) In step 2, 2D intersection points on reference image planes are projected back to the 3D viewing ray. Note that the epipolar line and the viewing ray lies in the same epipolar plane. Hence this 3D projection operation can be converted to a 2D intersection in this plane. The result for each pixel is a sorted 1D depth array.

(3) Since all the 3D segments from all the reference images lies on the same viewing ray, their final intersection can also be simplified as an 1D Boolean operation on the ray, based on the depth arrays from (2).

We implement the optimized algorithm using CUDA. To further reduce the cost of storage and data exchange, we pack all the reference images into a CUDA 3D texture, and load them to GPU memory at one time during initialization. Common parameters, including 2D silhouettes, camera parameters, etc., are also packed as textures and stored in the constant memory of GPUs, which can be rapidly accessed by all GPU threads. We also use CUDA page-locked host memory, which has higher bandwidth, to transfer rendering results between GPU and main rendering thread with less cost.

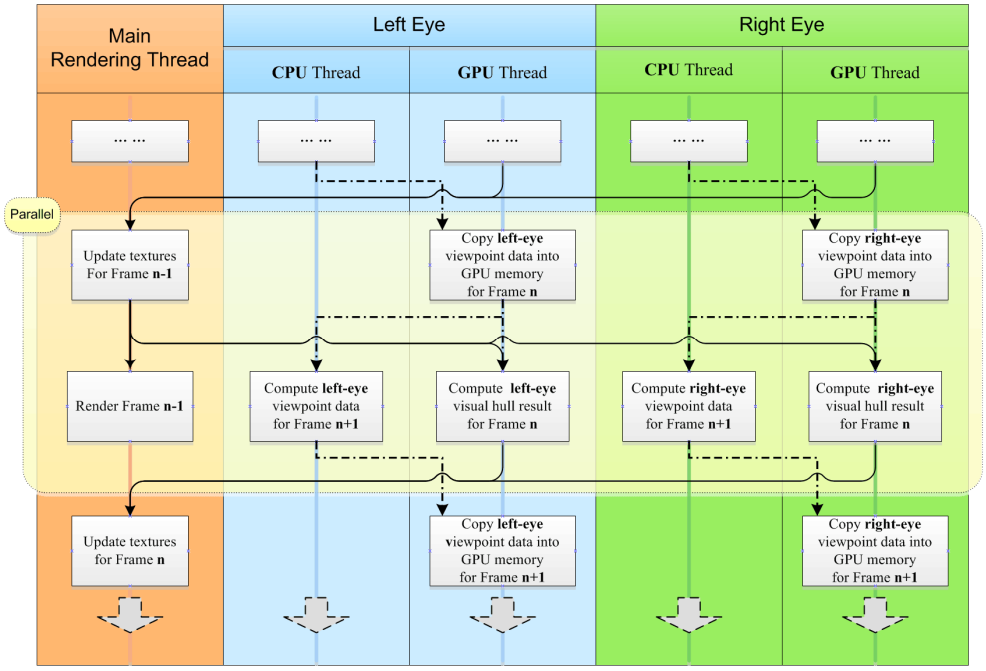

Figure 2: An illustration of our stereo visual hull rendering pipeline. Two channels control the computation of new views of left and right eyes, respectively. The arrows indicate the synchronizing signal flows. A module starts only after it receives all the signals it is waiting for.

Simple GPU-based parallel computing strategy can increase the rendering speed from less than 1 frame per minute to about 5 frames per second. After our optimizations, the rendering speed can even boost to about 40 fps on a PC with single GPU. That provides more possibility for real-time stereo rendering.

4 Multi-GPU Pipeline for Real-Time Stereo Rendering

On the basis of the optimized real-time IBVH rendering algorithm, we propose a novel two-channel stereo rendering pipeline which adopts a PC with two GPUs. The pipeline can simultaneously calculate new views from the left eye and the right eye, and produce stereo image of the object.

As illustrated in Fig. 2, the pipeline is made up with a *left-eye channel*, a *right-eye channel* and a *main rendering thread*. The two channels work with the same work-flow and different input, utilizing different GPUs. Each channel has a thread that controls GPU computations for visual hull reconstruction (ab. *GPU thread*) and a *CPU thread* preparing parameters and data. The main rendering thread collects the computing results from two GPU threads, and renders them on display devices. All these threads work concurrently, synchronized by signals.

When the computation of Frame *n*-1 has been completed, the main rendering thread collects the computing results from GPUs and updates textures to be rendered for Frame *n*-1. Meanwhile, the viewpoint data of Frame *n*, including left/right eye viewpoint positions, virtual camera parameters, fundamental matrices of reference images, etc., are copied to GPU memories.

After these data are ready, the main rendering thread release a signal to the GPU threads to start the computing of visual hulls for Frame *n*. At the same time, Frame *n*-1 is rendered to display devices and the viewpoint data for Frame *n*+1 are computed in the CPU threads in parallel. Again, when the computation of Frame *n* ends, the main thread receives signals from the GPU threads, collects new results and begins another cycle of rendering.

5 Conclusions

We implemented our optimized IBVH rendering method on a PC with Quad CPU 2.5GHz, 2.75GB RAM, and double GeForce GTX260+ graphic cards. Fig. 1 shows an example with 47 input reference images. The object is rendered in left/right eye channels synchronously, and a sphere panorama is used in environment mapping. The rendering results are output to a double-projector stereo displaying system, and a rendering speed of about 30 fps can be achieved. Users can also apply interactions such as rotating, scaling, translating and parallax adjusting in real-time.

There are several points that we should pay more efforts on in future works:

Firstly, we have not added a visibility testing algorithm in our method, and some artifacts may appear when the shape is complicate. Therefore we will implement an efficient algorithm to detect self-occlusions and improve the rendering quality.

Secondly, our current pipeline works well on static objects. In fact, with the help of multiple GPUs, it is also possible to deal with dynamic objects. Synchronous videos can be used as input data, and the processing of each video stream can be assigned to different GPUs. The high rendering speed can guarantee real-time stereo displaying and user interactions, while the quality of the rendered results will exceed existing methods.

References

MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., AND MCMILLAN, L. 2000. Image-based visual hulls. In *Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques (SIGGRAPH 2000)*, 369-374.

LI, M. 2004. Towards Real-Time Novel View Synthesis Using Visual Hulls, *Ph.D. thesis, Max-Planck-Institut für Informatik*.