A Distributed Parallel Reconstruction Method for Mobile Terminals


College of computer science & engineering, Jiangsu University of Science and Technology, Zhenjiang, Jiangsu, P. R. China

Abstract - With the rapid development of mobile Internet technology and real-time reconstruction technology, based on the multi-view of mobile terminal oriented 3D display provides remote interactive model reconstruction. The core of the algorithm is based on multiple images as input, and by calculating it generates sparse 3D point cloud through the expansion of that could generate 3D point cloud, and through the surface reconstruction it can accomplish the 3D model. But the high computing complexity and large data handling scale affect the quality and real-time performance of 3D demonstration. Against the above shortcomings, a distributed parallel reconstruction method for mobile terminals is proposed, which reconstructs model both in the server and in the client. The server uses levels of detail technology to control the scene’s complexity and generates initial reconstruction frames. The client uses image-based reconstruction technology to re-render the image, which can improve reconstruction quality. Experiments show that the method improves reconstruction speed, reduces the transmitted data size, and improves the image quality.

Keyword: Distributed reconstruction; Levels of Detail; Image based reconstruction; CUDA

1. Introduction

With the continuous development of virtual reality technology, the networked 3D demonstration shows highly realistic stereoscopic image by using real-time reconstruction and interaction of 3D models, which is totally different with traditional information communication methods based on text or image. However, it is a huge amount of work that reconstructing the 3D model with the associated 3D modeling software manually. At the same time, the cost of scanning equipment is expensive, so it is a hot research focus in the field of computer vision that how easy to obtain 3D model of the object from the real world[1]. Meanwhile, the further development of mobile intelligent terminal results in the transition of user’s terminal equipment from traditional personal computers to mobile phones, tablets, etc., which provides preconditions for the expansion of 3D display in the field of mobile applications.

Faced with increasingly reconstruction quality and real-time interactive requirement from mobile users, there are some shortcomings of traditional geometry-based reconstruction techniques for complex three-dimensional models[2,3]. On the one hand, for the scene composed by complex three-dimensional model, the larger data complexity causes more graphic hardware requirements of mobile terminal, which impacts the reconstruction real-time capacity.

Fig.1 Overall schematic diagram of the distributed parallel reconstruction method

On the other hand, the mobile network bandwidth
may not meet the speed requirement of the stereoscopic image display and interaction. With the development of massively parallel processing technology, the use of distributed parallel processing technology is an effective way to solve this problem.

In this paper a distributed parallel reconstruction method for mobile terminal is proposed to realize distributed reconstruction both in server side and terminal side. CUDA-based parallel reconstruction data processing is used in the method which achieves the goal of real-time complex three-dimensional model displaying on mobile terminals and reduction of network bandwidth consumption. The overall process of the method is shown in Figure 1, in which the server side generates LOD model by CUDA parallel computing and dynamic form the corresponding initial reconstruction image, also image-based reconstruction is used to realize the compression and transmission of the image, while the terminal side re-renders the image to improve the image fineness by CUDA-based IBR techniques.

2. Server side primary reconstruction

To improve the server’s ability of reconstruction complex three-dimensional model, a primary reconstruction strategy on the server side is proposed to ensure the real-time reconstruction capacity. The server pre-generates LOD simplified model of different resolutions by CUDA parallel computing. And in the reconstruction process, the appropriate LOD model is reconstructed based on the distance between the viewpoint and the model as well as the reconstruction frame rate, to improve the overall computing efficiency.

2.1 Parallel simplification of LOD model based on CUDA

CUDA is known as a parallel computing platform and programming model implemented by GPU. This paper uses CUDA to simplify model by means of edge collapse calculation. Edge collapse refers to choose two connected vertices and replace them with a single vertex, and all the vertexes connected to the two vertices will re-connect to the new vertex to maintain the triangle grid appearance. In this paper, we select one of the two connected vertices as the new vertex. Garland Quadratic Error Metrics (QEM) is used to compute the edge collapsing cost, which can efficiently preserve the features on the surface.

In the simplification process, LOD model generation introduces the concept of vertices importance as the trade-off of the sequence of edge collapse operation. Vertex importance reflects the geometry importance degree of triangle mesh. It is discussed in [4] that in the neighborhood of the vertex in the mesh, if the steeper the vertex is, the greater the impact on the mesh geometry is; while the sparser the vertex is, the bigger the vertex importance value is as well. So the vertex importance can be summarized as follows:

$$Q(v_j) = K_j \times \bar{I} = \sum_{v_i \in \text{neibors}(v_j)} \frac{c_{ji}}{m} \times \bar{I}$$

Call $c_{ji}$ the cosine of the angle between the normal vectors $v_j$ and $v_i$. $v_i$ is the element in the vertices collection in which the vertices are connected with $v_j$. $m$ is the total number of the set, and $\bar{I}$ refers to the average length of the associated edges. So $K_j$, the average cosine value of the vector angles, can be regarded as the curvature of $v_j$, and therefore $Q(v_j)$ can reflect the geometry importance of $v_j$.

We use Octree to divide the original mesh into several independent sub meshes. A generation method of LOD model is proposed by using CUDA to parallel simplify those sub meshes based on edge collapse operation. The process is shown in Figure 2.

In order to facilitate the subsequent edge collapse calculation, we storage all the vertex info in the sub mesh into an array $\text{Vtable}$, which is defined as follows:

```c
struct vernode
{
    float x,y,z;   //vertex three-dimensional coordinates
    long ver_importance;
    int foldingindex;   //the folding vertex index
} ;
vernode Vtable[vernum];
```

As is shown in Figure 2, the original model is partitioned by octree spatial decomposition on CPU at first, which regards the entire model bounding box as the octree root node. The bounding box is split into 8 sub cubes, in accordance with intermediate section in the
direction of the three-dimensional coordinates. Correspondingly, the model is divided into 8 sub meshes. Recursively decompose the sub cubes until the amount of triangle facets in the sub mesh is less than a given threshold. Finally the model is organized as an octree structure.

In GPU computing stage, parallel threads are created to calculate each vertex importance according to (1). After the vertex importance calculation is finished, the vertex is sorted by its importance in each sub mesh and the sequence number is stored in \( Vtable \). During parallel edge collapse operation, the sub meshes is simplified concurrently. For each sub mesh, edge collapse is implemented by vertex importance ascending sequence. For each edge collapse operation, the edge \((v_i, v_j)\) that contains vertex \(v_i\) which owns the lowest importance is replaced by the vertex \(v_j\). For the edges that all contains vertex \(v_i\), we calculate the edge collapse cost by QEM to select the appropriate one. Recursively simplify the sub mesh until the number of remaining vertices meets the requirement.

The parallel simplification algorithm of LOD model is as follows:

```
Algorithm 1 GPU parallel LOD generating
Input: mesh:the original model triangle mesh
Output: simplifiedmesh:the simplified mesh
1. submeshes ← CPU_OctreeMeshDivde(mesh) ;/*submeshes:array of divided submesh */
2. MemcopySync(submeshes,host->device); /*Load submeshes to GPU device memory*/
3. for i=0 to NUMsubmeshes-1 parallel do
4. Ver_importance ← verCalculating_kernel(submeshes);
5. __syncthreads(); /*threads synchronize */
6. Vtable ← importanceSort_kernel(Ver_importance);
7. __syncthreads(); /*threads synchronize */
8. simplifiedmesh ← edgeCollapse_kerem(submeshes,Vtable);
9. end for
10. MemcopySync(simplifiedmesh,device->host);
```

2.2 Dynamic LOD Scheduling stagey

While the server reconstruction, it should select the appropriate resolution level of LOD model based on the distance between the viewpoint in model and in advance\(^5,6\).

Shown in Figure 3, while the distance is \(d_1,d_2,\ldots,d_n\), the corresponding LOD level is LOD1,LOD2,\ldots,LODn. The server adjusts LOD model according to the distance in order to keep its reconstruction efficiency. Also the last frame reconstruction time cost is took into consideration. If the frame rate is lower than the real-time requirement, a more simplified LOD model is selected until the frame rate meets the requirement. After the frame is reconstructed, the server will transmit the result in form of image to the terminal for the further reconstruction and displaying.

Fig.3 Viewpoint distance-dependent LOD strategy

3. Terminal side secondary reconstruction

To further reduce the server computing load and the amount of data transmitted between the server and the terminal, the terminal uses CUDA to refine initial image from the server and utilizes the difference between geometry-based reconstructed image and reconstructed image via IBR\(^7\).

3.1 IBR-based compression and secondary reconstruction algorithm

As shown in Figure 4, call \( L \) the generated view of the model corresponding to position \( V \) of the viewpoint.

So \( L(x) \ (x \in Q = [0,W] \times [0,H]) \) can be seen as a two-dimensional array of pixels, where every pixel \( X = [x,y,z] \) refers to the 3D position in the reference system of \( V \). When the viewpoint changes into \( V' \), the generated view is \( L'(x') \ (x' \in Q' = [0,W'] \times [0,H']) \).

In this situation, the pixels \( X' \) in the reference system of \( V' \) can be divided into two categories. One can be seen as the transformed pixels in view \( V' \), the other are those new pixels first into the screen space. For the former one, pixels can be calculated by the three-dimensional transformation theory as follow:

\[
X' = [x',y',z(x',y')]^T = T(x,y,z(x,y)) \quad (2)
\]

\( T \) is a suitable 3D projective transformation obtainable by matrix \( T_g \) in homogeneous coordinate.

Denote \( \hat{L}(x') \) the view with respect to \( V' \) via IBR procedure, in order to distinguish with \( L'(x') \) the view reconstructed from the 3D model with respect to \( V' \).
For the pixels in $\hat{L}'(x')$, one may obtain:

$$x' = t(x), \quad x \in Q$$  \hspace{1cm} (3)

The two pixels set $I_1 = Q' \cap t(Q)$ and $I_2 = Q' - I_1$ in view $\hat{L}'(x')$ can be written as:

$$\hat{L}'(x') = \begin{cases} L(t^{-1}(x')) & x' \in I_1 \\ 0 & x' \in I_2 \end{cases}$$  \hspace{1cm} (4)

So the only difference between $L'(x')$ and $\hat{L}'(x')$ are the pixels in $I_2$. To reduce the amount of data transferred between the server and the terminal, the server only need to transmit the correction data (pixels in $I_2$) to the terminal instead of the whole view $L'(x')$, if the terminal can calculate $\hat{L}'(x')$ itself. $E(x')$ can be written as:

$$E(x') = L'(x')\chi(x'), \quad \chi(x') = \begin{cases} 1 & x' \in I_1 \\ 0 & x' \in I_2 \end{cases}$$  \hspace{1cm} (5)

During the secondary reconstruction process in the terminal, the server pre-renders fine reference view $L_{\text{high}}$ with respect to $V$ based on high-resolution LOD model selected by dynamic LOD stagey. For views $L'$ with respect to subsequent viewpoints $V'$, the server replaces with a relative low-resolution LOD model. The view $L'$ generate by the terminal with respect to $V'$ via IBR procedure can be also written as:

$$\hat{L}_{\text{high}}'(x') = \begin{cases} L_{\text{high}}(t^{-1}(x')) & x' \in I_1' \\ 0 & x' \in I_2' \end{cases}$$  \hspace{1cm} (6)

As can be seen, both $L'$ and $\hat{L}_{\text{high}}'$ are the descriptions of the same original model. The difference is that $\hat{L}_{\text{high}}'$ does not contain the new pixels which are put into the screen space by the viewpoint transformation, but for pixels in $I_2'$, the view $\hat{L}_{\text{high}}'$ contains more model details of $L'$. So the terminal can re-render the view as follow:

$$L'(x) = \begin{cases} \hat{L}_{\text{high}}'(x) & x \in I_1' \\ L'(x) & x \in I_2' \end{cases}$$  \hspace{1cm} (7)

### 3.2 Parallel secondary reconstruction in the terminal based on CUDA

Since the parallel secondary reconstruction in the terminal is based on image-based reconstruction, the standard practice is to reset the reference frame of every $p$ frames to reduce the prediction error[8]. A scheme of principle for reconstruction $p$ views $(L_1, L_2,..., L_p)$ of a 3D model is as follows:

1. In reconstruction preprocessing stage, the server uses GPU to implement parallel LOD model generation.

2. $L_1$ is set as the reference frame view, and the server reconstructs a relative high-resolution simplified model selected by dynamic LOD stagey and sends the fine view $L_1$ to the terminal.

3. The server replaces the model with a lower-resolution one, computes and sends $L_2$ to the terminal.

4. For frame view $L_i, 2 < i \leq p$
   
   a. At both, server and terminal compute:
   
   $$\hat{L}_i(x) = \begin{cases} L_2(t^{-1}(x)) & x \in I_1 \\ 0 & x \in I_2 \end{cases}$$

   b. At server’s side, compute $E_i(x) = L_i(x)\chi(x)$ and send $E_i(x)$ to the terminal.

   c. At terminal’s side, compute:
   
   $$L_i'(x) = \hat{L}_i(x) + E_i(x)$$

5. At terminal’s side, update:

   $$L_i'(x) = \begin{cases} L_2(t^{-1}(x)) & x \in I_1' \\ L_i(x) & x \in I_2' \end{cases}$$

CUDA-based parallel reconstruction in the terminal is supposed to send $L_1$ and $L_2$ to GPU memory at first, and then generate parallel GPU threads for reconstruction the corresponding pixels calculation[9], which is described as follows:

**Algorithm 2 GPU parallel secondary reconstruction**

**Input:**

- `frameImage`: the ordinary frame view $L_i$ from the server side;
- `refImage`: the reference frame view $L_1$;
- `IBRimage`: the ordinary frame view $L_2$;
size: the size of frameImage;

Output:
Sec_frameImage: the secondary reconstructed view

1. MemcpySync(refImage, host->device); /*Load refImage to GPU device memory*/
2. MemcpySync(IBRimage, host->device); /*Load IBRimage to GPU device memory*/
3. for i=0 to size parallel do
4. local_Image ← IBRcalculating_kernel(IBRimage); /*calculating the IBR view on GPU*/
5. end for
6. frameImage ← receive(); /*get frameImage difference E(x) from the network*/
7. MemcpySync(frameImage, host->device);
8. for i=0 to size parallel do
9. local_Image ← imageCorrection_kernel (IBRimage, frameImage);
10. Sec_frameImage ← secondReconstruction_kernel(local_Image, refImage);
11. end for
12. MemcpySync(Sec_frameImage, device->host);

4. Performance analysis

For the tests reported in this paper, the server node was equipped with CPU: Intel(R) Xeon CPU E5504 2.00GHz, GPU: Nvidia tesla S2050 and the terminal application was run on an Nvidia Tegra TK1 equipped with Nvidia Kepler GPU by OpenSceneGraph platform. Figure 5 is the example of the server side initial frame view and the terminal side secondary frame view.

![Fig.5.1 (a) Primary reconstruction in server](image1)

![Fig.5.2 (b) Secondary reconstruction in terminal](image2)

To compare the calculating efficiency, GPU and CPU are used to simplify the model and image-based secondary reconstruction. The time cost is shown in Figure 6. We can see that with the increasing size of data processing, GPU-based parallel computing can effectively reduce the time consuming and improve the real-time capacity.

![Fig.6.1 (a) time of LOD generating](image3)

![Fig.6.2 (b) time of Secondary reconstruction](image4)

When the view resolution is 320*240, the frame view data size within a certain period of time is shown in Figure 7, where each cylindrical size represents the total volume of the frame view, and the marked region means the size of image component. From the result, we can see the IBR-based image compression can effectively reduce the data volume and the bandwidth requirements.

Experiments show that the distributed parallel reconstruction method can guarantee the view reconstruction quality, improve real-time capacity as well as reduce the data transmission.
5. Conclusion

This paper describes a distributed parallel reconstruction method for mobile terminals, which implements the distributed reconstruction on both server and terminal, as well as parallel reconstruction data processing by GPU. The server dynamically schedules the LOD model to realize real-time reconstruction, combined with the image-based reconstruction to compress the frame view. While the terminal further re-renders the view to improve the view fineness. Experiments show that this method can improve the overall reconstruction efficiency, reduce the bandwidth requirements and improve the reconstruction speed.

6. References