Multiresolution 3D Rendering on Mobile Devices

Javier Lluch, Rafa Gaitán, Miguel Escrivá, and Emilio Camahort

Computer Graphics Section
Departament of Computer Science
Polytechnic University of Valencia
Camino de Vera s/n, 46022. Valencia, Spain

Abstract. We present a client/server system that is able to display 3D scenes on handheld devices. At the server we extract the geometry that is visible for each client and send it. We also extract texture and material information. The clients, running on mobile devices, use that information to render realistic images. Our geometry extraction algorithm employs multiresolution and view-dependent simplification. We present results of our system running on PocketPC 2003 PDAs.

1 Introduction

Over the last few years mobile computing platforms have made important advances. We can see handheld devices with increasing processor speed and integrated wireless technologies. Computer graphics are also rapidly advancing in mobile devices. Still, these devices have limitations of memory and processing power. Also, most of them do not offer hardware accelerated graphics. This makes it difficult to render and interact with large 3D scenes on these devices.

A different issue is the size of the geometric data sets. Recent advances in 3D design, acquisition and simulation have led to larger and larger geometric data sets that exceed the size of the main memory and the rendering capabilities of current graphics hardware. Therefore, it is necessary to apply techniques like multiresolution and simplification in order to reduce the amount of geometry. Still, multiresolution models can only be applied to data sets that do not exceed the size of the main memory.

To solve this problem new models have been developed that store large scenes in secondary memory (*out-of-core*). We can still render these models at interactive speeds using view-dependent simplification. One of the immediate applications of this type of models is rendering of three-dimensional scenes on mobile devices.

We have developed a client-server system for remote rendering on mobile devices. Our system uses multiresolution, view-frustum culling and out-of-core techniques to deliver geometry from a PC server to one or more mobile devices. Our system has the advantage that it only sends the changes in geometry from the server to the client. Additionally, our system can handle and send textures to the clients.

In this paper, we present the system and the algorithm used to extract and send geometry from the client to the server. The paper is structured as follows. First, we introduce OpenSceneGraph(OSG) and multiresolution modeling, two components used in our system. Then, we present our client-server system. In the following Section we introduce an algorithm to efficiently extract geometry differences at the server. We also explain how textures are sent to the client. Finally, we present some results, conclusions and directions for future work.

2 Background

In this section we present the background of our system and our implementation. First we summarize OSG and the benfits of its use. Next we explain how we have merged OSG and the Multitesselation(MT) library.

2.1 OpenSceneGraph

OSG [8] is an open source, cross platform graphics toolkit for the development of high peformance graphics applications such as flight simulators, games, virtual reality and scientific visualization. Based around the concept of a scene graph, it provides an object oriented framework on top of OpenGL, freeing the developer from implementing and optimizing low level graphics calls. OSG provides many additional utilities for rapid development of graphics applications.The key strengths of OSG are its performance, scalability, portability and the productivity gains associated with using a fully featured scene graph.

2.2 Multiresolution

Multiresolution meshes are a common basis for building representations of geometric shapes at different levels of detail. The use of the term multiresolution means that the accuracy (or level of detail) of a mesh in approximating a shape is related to the mesh resolution, i.e., to the density (size and number) of its cells. A multiresolution mesh provides several alternative mesh-based approximations of a spatial object (e.g., a surface describing the boundary of a solid object, or the graph of a scalar field).

A multiresolution mesh is a collection of mesh fragments, describing usually small portions of a spatial object with different accuracies. It also includes suitable relations that allow selecting a subset of fragments (according to user-defined accuracy criteria), and combining them into a mesh covering part or the whole object. Existing multiresolution models differ in the type of mesh fragments they consider and in the way they define relations among such fragments. The reader is referred to the surveys [4, 7, 2] for a detailed description of multiresolution mesh representations.

2.3 Rendering on Mobile Devices

In [5] we address the problem of rendering complex three-dimensional scenes on mobile computer devices. The issue is that the scene does not fit in the devices's main memory. So we use out-of-core storage and view-dependent simplification to keep the smallest possible number of polygons stored in the mobile device for rendering. To achieve this goal we employ multiresolution meshes.

We propose applying current multiresolution, viewing frustum culling and out-of-core [3] techniques to 3D graphics rendering on mobile devices [1, 6]. We present a client-server system that delivers simplified geometry to a PDA over a wireless connection. The server stores the scene graph and extracts levels of detail to be rendered at the client. Careful selection of appropriate levels of detail allows rendering at interactive rates on a handheld device.

Our system uses a geometry cache that manages two copies of the set of visible objects. One copy is located at the server and the other at the client. The server updates the cache with the visible geometry when the scene or the viewing parameters change. A synchronization process mantains the coherence between the server and the client copies of the cache. With this approach we only send geometry updates from the server to the client, thus reducing the latency and bandwidth requirements of our system. It also has the added advantage that we can use advanced geometry culling methods at the server side.

Our server system also supports multiresolution and view-dependent simplification, thus allowing the selection of a suitable level of detail for rendering each desired view. The system can render scenes with any number of polygons. It improves on previous systems because we do not store the entire geometry at the mobile device. Neither do we render at the server and send the rendered frames to the client. Instead we make sure that the client only receives those parts of the scene graph that are visible. These parts are currently rendered using a software library. Once hardware accelerated rendering is available we will be able to render larger amounts of geometry on the mobile device.

3 Geometry and Texture Extraction

The system just described caches geometry at the mobile client so that only new geometries need to be sent from the server. Still, the architecture needs to send all the multiresolution geometry every time there is a change in the resolution level. This behavior causes the latency to increase. To solve this problem we propose an algorithm that efficiently computes differences between two extracted levels of detail. This reduces the latency between client and server with multiresolution geometries. We also show how to extract textures at the server in order to send them to the client for realistic rendering.

3.1 Geometry Extraction

We have a library that binds OSG and the MT library. We have improved the library by adding new methods that extract only the differences between two different geometry resolutions.

MT works by extracting two arrays of triangle identifiers. One for the geometry inside the interest volume and the other for the geometry outside the interest volume. Until now our server was extracting all the geometry for every change in the resolution and sending all the new geometry information to the client. Between two extractions of geometry the information differs only in a few vertices and of course in their triangle configuration. We propose to reuse this information to avoid sending all the geometry information to the client.

We use an algorithm that is made of three steps:

1. First we extract the geometry and save the triangle identifiers and the index pointer (of each vertex) to the vertex array inside the triangle. The data structure that we use

to save the information is a hash map(*index hash map*). This step is necessary because the next step uses the saved information to calculate the differences between geometries.
2. The second step obtains the new arrays of triangle identifiers from the MT geometry, and compares each triangle with the triangles that are already stored at the server. For those triangles we generate the index array with the index pointer saved in the *index hash map* during the previous step. For those triangles not saved in the hash map we use a different hash map (*difference hash map*) to save the triangle identifier, and the vertex and normal information. We must have the difference information available because the next step will use it to generate the new geometry.
3. The third step takes the *difference hash map* with the triangles not yet cached and appends the triangles to the vertex and normal arrays. The index hash map is also updated with the new added triangle identifier and the indices of its vertices in the vertex and normal arrays. It is also necessary to update the geometry index array with the index pointers added to the *index hash map*. Finally we clean the difference hash map and go to step 2 to calculate differences again.

Once steps 2 and 3 have been run, we can draw the complete geometry with a different level of detail. Algorithm 1 shows the methods used for computing the geometry differences (left) and reconstructing the geometry from those differences (right).

The server system has been modified to extract differences between two continuous levels of detail. When the cull process traverses the scene, if the geometry is a multiresolution geometry then we run (i) step one if is the first time we find the geometry or (ii) step two if the geometry had already been created. Next we add the geometry to the local geometry cache to make the syncronization between client and server.

The cache has been modified in order to support this kind of new geometries. We have extended our cache to save the differences for multiresolution geometries and to calculate the geometry using step three of the algorithm.

Algorithm 1. Calculate Differences and Geometry Extraction by Differences

```
procedure CALCULATEDIFFERENCES
    $act\_tri \Leftarrow MT\_ExtractActiveTiles()$
    for all $t$ in $act\_tri$ do
        if $t$ is in indexHashMap then
            indices.add(indexHashMap[$t$].$i_0$)
            indices.add(indexHashMap[$t$].$i_1$)
            indices.add(indexHashMap[$t$].$i_2$)
        else
            diffHashMap[$t$].$v_0 \leftarrow t.v_0$
            diffHashMap[$t$].$v_1 \leftarrow t.v_1$
            diffHashMap[$t$].$v_2 \leftarrow t.v_2$
        end if
    end for
end procedure
```

```
procedure CALCGEOMBYDIFFERENCES
    $act\_tri \Leftarrow MT\_ExtractActiveTiles()$
    for all $t$ in diffHasMap do
        vertices.add($t.v_0$,$t.v_1$,$t.v_2$)
        $lenv \leftarrow vertices.size()$
        indices.add($lenv - 3$)
        indices.add($lenv - 2$)
        indices.add($lenv - 1$)
        $leni \leftarrow indices.size()$
        indexHashMap[$t$].$i_0 \leftarrow (leni - 3)$
        indexHashMap[$t$].$i_1 \leftarrow (leni - 2)$
        indexHashMap[$t$].$i_2 \leftarrow (leni - 1)$
    end for
    diffHashMap.clean()
end procedure
```

3.2 Texture Extraction

In order to improve the realism of our system we implement functionality to extract textures at the server and send them to the client for rendering. To accomplish this task, we had to modify the **geometry extractor**, the **client** and the **scene cache**. Figure 2 contains some screenshots of our system.

When the server traverses the scene to collect all the geometries to send to the client, we need to accumulate the state of OpenGL (*textures, lights, ...*). So, when a geometry is found, the current accumulated state is added to the server cache with its associate geometry. Then the cache is synchronized with the client cache, and the new geometry and the associated OpenGL state are sent to the client.

The client synchronizes the scene cache with the server's cache. When this happens, the new geometry and the material state are received by the client. Then, the client adds the received geometry to its cache.

Both parts, client and server, have a scene cache that must be synchronized. The scene cache has been modified to support the material state and textures. We need to send those materials through the network, but only new materals are sent in order to improve cache synchronization speed. The new geometries and textured materials are received at the *back buffer cache*, so when the synchronization process finishes, the system swaps the cache and the new geometries, so that they can be used for rendering.

4 Results

To test the implementation of our system we have run the server on a desktop PC and the client on different devices. We run the client on a laptop with wireless access and on an HP ipaq 4150 running PocketPC 2003.

We used different scenes to test our system. Figure 2(b) shows a screenshot of the system running our win32 client. Figure 2(c) shows a scene with several multiresolution objects rendered using OSG. Figures 2(e) and 2(f) show the client running on a PDA.

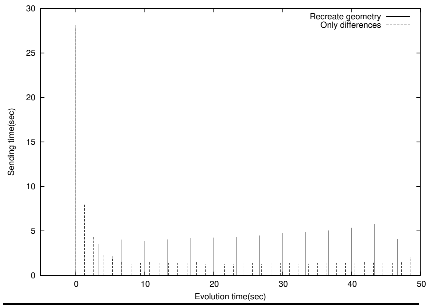
To analyze our system we compare the evolution of the time of extraction between the difference algorithm and the complete extraction. We also show the latency of the system with multiresolution geometries and textured materials. The comparisons have been made moving the camera along a couple of pre-generated paths.

Table 1(a) shows a comparison between the average, maximum and minimum of extraction time of one multiresolution geometry. We observe that the average extraction time is substantially better using the difference algorithm. This allows us to greatly reduce the time needed to prepare the geometry to send it to the client. Table 1(b) shows a comparison of the transmission times required by the difference algorithm and the algorithm that sends all the geometry. We observe that the average transmission time of the difference algorithm is 2.5 times faster than sending all the geometry. Table 1(c) shows a comparison of sending static geometries with textured materials through two different types of network.

Plot 1(b) shows latency time with multiresolution geometries. Note that the difference algorithm runs faster. Reducing latency time allows more extractions thus reducing popping artifacts. Plot 1(c) shows the evolution of latency time when rendering static geometry and textured materials. The textured scene has three geometries with the same

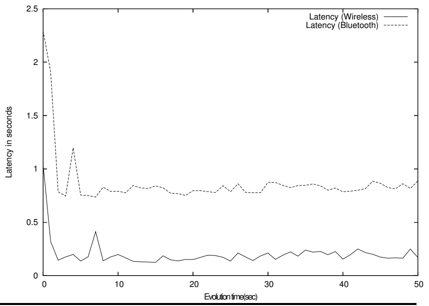
	Recreate	Differences
Average	$0.0057sec$	$0.0026sec$
Max	$0.0187sec$	$0.0078sec$
Min	$0.0038sec$	$0.0021sec$

(a) Comparison of extraction time of a single multiresolution geometry when using the difference algorithm and the complete reconstruction algorithm. The test scene has roughly 69000 triangles, viewing as many as 4000 triangles in the PDA.



	Complete	Differences
Average	$6.053sec$	$2.412sec$
Max	$28.173sec$	$27.791sec$
Min	$3.525sec$	$1.236sec$

(b) Latency times for multiresolution geometries. The test scene has more than 960000 triangles, viewing as many as 10000 in the PDA. Maximum values are obtained at the initial time.



	Bluetooth	Wireless
Average	$0.870sec$	$0.211sec$
Max	$2.292sec$	$1.035sec$
Min	$0.736sec$	$0.124sec$

(c) Evolution of latency time when rendering static geometry and textured materials across two different types of network. The test scene has 1500 triangles.

Fig. 1. Results obtained with our system running on a PocketPC 2003 PDA

material. Note that the high latency at the beginning occurs because we send two geometries and the associated material. The following high latency in the plot appears at roughly second 8 is the sending of the third geometry.

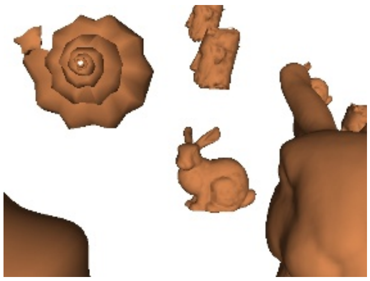
5 Conclusions and Future Work

In this paper we present a client-server system for remote rendering on mobile devices. Our system uses multiresolution, view-frustum culling and out-of-core techniques to deliver geometry from a PC server to one or more mobile devices. Our system has the advantage that it only sends the changes in geometry from the server to the client. Additionally, our system can handle and send textures to the clients.

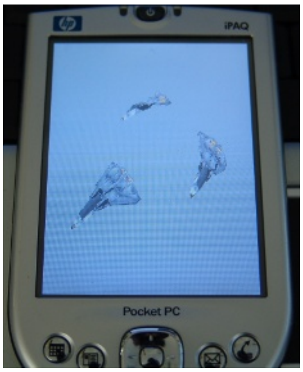
The system's server uses *OSG* and multiresolution meshes for scene management. Given the camera parameters delivered by the client, the server culls the scene graph and sends to the client the geometry to be rendered. We propose a new algorithm that ex-

(a) A container terminal

(b) Detailed view of the same terminal


(c) A large scene with several multiresolution models. The test scene used in 1(b).

(d) A scene with several texture-mapped objects.


(e) Texture-mapped objects.

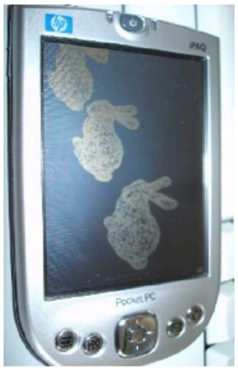

(f) A scene with multiresolution objects.

Fig. 2. Example renderings of our client system running on *top* and *middle* a PC, and *bottom* a PDA

tracts the differences between two geometry levels of detail. With this algorithm only the difference in geometry needs to be sent to the client, thus substantially reducing latency.

Our algorithm also allows extracting textures in addition to geometry. Texture and material data can be extracted, cached and delivered at the client exactly like geometry. This produces an increased realism in the renderings generated at the mobile client.

We need to improve the cache in order to avoid overloading if we continuously append differences.

We want to implement an out-of-core multiresolution model for PDA systems and we want to compare it with the system just described. We are also preparing some benchmarks to measure the transmisssion efficiency of different network connections.

Acknowledgement

This work was partially supported by grants: TIN2005-08863-C03-01 of the Spanish Ministry of Science and Technology, GV04B-497 04-05 of Generalitat Valenciana and IST-004363 of the 6th Framework Program of the European Union.

References

1. B. D'amora and F. Bernardini. Pervasive 3d viewing for product data management: IEEE Computer Graphics and Applications (March/April 2003), Volume: 23, Issue: 2, 14-19.
2. Claudio Zunino and Fabrizio Lamberti and Andrea Sanna: A 3d multiresolution rendering engine for pda devices. In SCI 2003, volume 5 (2003) 538-542.
3. J. El–Sana and Y.-J. Chiang: External memory view-dependent simplification. In EuroGraphics volume 19 (2000), 139-150.
4. J. El–Sana and A. Varshney: Generalized view-dependent simplification. In Proceedings EuroGraphics (1999), 83-94.
5. J. Lluch, Rafael Gaitán, Emilio Camahort and Roberto Vivó: Interactive Three-Dimensional Rendering on Mobile Computer Devices. Proceedings of ACM ACE (2005) 254–256.
6. Khronos Group: http://www.khronos.org/. OpenGLES - The Standard for Embedded Accelerated 3D Graphics (2004).
7. Leila De Floriani and Paola Magillo and Enrico Puppo: Multiresolution representation of shapes based on cell complexes. In Discrete Geometry for Computer Imagery, number 1568 (1999) 3-18.Lecture Notes in Computer Science. http:// www.disi.unige.it/person/MagilloP/MT.
8. OSG Community: http://www.openscenegraph.org. OpenSceneGraph - Open Source high performance 3D graphics toolkit (2005).