A 3D Multiresolution rendering engine for PDA devices

Claudio Zunino Fabrizio Lamberti Andrea Sanna

**Dipartimento di Automatica e Informatica,
Politecnico di Torino, corso Duca degli Abruzzi 24,
I-10129 Torino (Italy)**

{claudio.zunino,fabrizio.lamberti,andrea.sanna}@polito.it

ABSTRACT

One of the main goal in the context of computer graphics is to create and display scenes as much realistic as possible, but realism requires complex and highly detailed models that can strongly affect the interactivity of the user with the scene. On the other hand, a new category of devices, known as Personal Digital Assistant (PDA), have been quickly spreading since end of the nineties. PDAs cannot still provide a graphics capacity such as last generation PCs, however the request for quality graphics for portable devices is one of the most new, important and attractive field of research in this area. The paper proposes a continuous multiresolution rendering engine for PDAs, based on the PocketGL library, able to dynamically adjust the level of detail of objects according to a frame rate value set from the user. The frame rate is considered as an "objective function" that determines both the quality of the scene and the interaction capability; objects nearer the user are represented by means of a larger number of polygons with respect farther elements in order to guarantee the best trade-off between quality and rendering frame rate.

Keywords: PDA, dynamic multiresolution rendering, PocketGL, 3D graphics on mobile devices.

1. INTRODUCTION

The realistic visualization of large and complex models like cars, trains, airplanes, and so on is one of the most attractive challenge in the context of computer graphics and virtual reality. Although realism is a major goal, a large spectrum of applications require real-time interaction or, at least, a good level of interactivity between user and the graphics interface. In these cases, a trade-off solution between quality and rendering frame rate has to be addressed.
Multiresolution techniques provide an effective tool to tackle this issue. Each object is maintained at different levels of detail (LOD) or approximation, here the level of detail may be different in distinct areas of the object. At any given moment, every object is visualized with a level of detail; if the scene is too complex and the set frame rate value cannot be guaranteed, object resolutions can be scaled down.
A good multiresolution rendering algorithm has to be able to obtain the best quality for the objects nearer to the observer affecting in a larger way the resolution of far and less visible elements.
Two main categories of multiresolution modelling can be denoted: discrete and continuous. Discrete multiresolution modeling algorithms compute off-line a set of representations for every object; the rendering engine can switch between two consecutive representations in order to satisfy frame rate and quality specifications. This approach is simple and effective but the gap between two different LODs might strongly affect the visualization of the scene. Continuous algorithms dynamically compute an infinite number of resolutions according both to the point of view and to a set of cost functions that determine the criteria for geometry modifications. These techniques are computational more expensive than discrete ones but, generally, provide better results.

Multiresolution rendering engines can be of great value for devices not provided of accelerated graphics hardware like PDAs. PDAs are handheld computers that originally were designed as personal organizers; the basic features of any PDA are a date book, address book, task list, and memo pad.
Nowadays PDAs are composed by a input unit that is a stylus for navigation and data entry via a handwriting recognition system. The typical display size is 160x200 pixels, even if some products reach 240x320 pixels of resolution.
More expensive PDAs have a color display while the medium price devices have gray scaled display. The large diffusion of these devices is the stimulus for developing new software and 3D graphics applications is one of the major challenge in this field.
This paper if one of the first attempts to investigate issues related to 3D graphics engine for PDAs, since a few works are known in the literature in the context of this research area. For instance, Elite [Eli02a] is a fast 3D rendering engine for small devices running JAVA. The engine provides a framework to create and display 3D wireframe models. PocketGL [Pok02a] is a 3D graphics library (similar to OpenGL) for pocket PC written in C and C++ and it allows to draw 3D objects and manage 3D transformations.
In the proposed work, a continuous multiresolution modeling technique has been used to obtain a satisfactory trade-off between quality and interactivity. The proposed methodology allows the user to set a desired frame rate value that is used as an objective function. The scene complexity is scaled down when the current frame rate is lower than the threshold value, while the model complexity is increased if the frame rate trade is positive. Geometry decimations/refinements are performed according to cost functions able to take into account both the distance of the objects from the observer and the impact of topological changes on the visualization.
The paper is organized as follows. Section 2 reviews main multiresolution modeling techniques and briefly summarizes PocketGL technical characteristics. Section 3 describes in the details the proposed rendering engine, while Section 4 presents examples and results.

2. BACKGROUND

In the first part of the Section we present a brief survey of the most important multiresolution modeling techniques. In Subsection PocketGL a description of the most important characteristics of PocketGL library is presented.

A polygonal model M is composed of a fixed set of vertices $V = (v_1; v_2; \ldots; v_t)$ and a fixed set of faces $F = (f_1; f_2; \ldots; f_n)$. It provides a single fixed resolution representation of an object. Without loss of generality, we can assume that the model consists entirely of triangular faces, since any non-triangular polygons may be triangulated in a pre-processing phase.

Let us suppose to have a polygonal model *M* and an approximation *M'* has to be obtained. While this approximation will have fewer polygons than the original, it should also be as similar as possible to *M*. The goal of polygonal surface simplification is to automatically produce such approximations. User supervision is generally not feasible. Simplification is naturally targeted towards large and complex datasets which would be very difficult to manually manipulate.

The primary aim of simplification is to produce a surface approximation which is as similar as possible to the original. In order to the quality of an approximation, we need some means of quantifying the notion of similarity. Given a polygonal model *M* and an approximation *M'*, an error metric *E(M;M')* measures the approximation error of *M'*. First, the notion of deviation between the original and the approximation has to be generalized. When comparing general surfaces, it will be measured distances between closest pairs of points. The distance from a point **v** to the model *M* is defined to be the distance to the closest point **w** on the model:

Equation 1

$$d_v(M) = \min_{w \in M} \|v - w\|$$

where $\|\cdot\|$ is the Euclidean vector length operator. One commonly used geometric error measure is the Hausdorff distance [Pre85a], that is defined as:

Equation 2

$$E_{\max}(M_1, M_2) = \max(\max_{v \in M_1} d_v(M_2), \max_{v \in M_2} d_v(M_1))$$ This error measures the maximum difference between the two models. In a similar way we can define E_{avg} :

Equation 3

$$E_{avg}(M_1, M_2) = \frac{1}{w_1} \int_{v \in M_1} d_v^2(M_2) + \frac{1}{w_2} \int_{v \in M_2} d_v^2(M_1)$$ This measures the average squared distance between models, where w_1, w_2 are the surface areas of M_1, M_2. These error metrics can be very expensive to compute, thus it is common to use a discrete set of points X_1, X_2 on the surfaces of M_1, M_2 that contains, at a minimum, all the vertices. Some simplification algorithms have used these metrics, for example the E_{dist} energy term used by Hoppe *et al.* [Hop93a] [Hop96a] is similar to E_{avg}.

The two main methodologies in surface simplification are refinement and decimation. The first one is an algorithm which begins with an initial very simple approximation and adds elements at each step; the second begins with the original surface and iteratively removes elements at each step.

One of the more used algorithms is the vertex decimation, a simplification algorithm proposed by Schroeder *et al.* [Sch92a]. At each step, a vertex is selected for the removal, the faces adjacent are removed and the resulting hole is triangulated. The original version of the algorithm used conservative estimate of approximation error while more recent algorithms [Cia97a, Kob98a] use more accurate error metrics.

The major class of algorithms is based on iterative contraction of vertex pairs [Gar97a, Gar98a, Hop96a, Pop97a, Lau98a]. A edge contraction $(v_1, v_2) \rightarrow v$ modifies the surface with the movement of vertex v_1 and v_2 to the position of vertex v, the replacement of all v_2 with v_1 and the removal of v_2 and of all faces that have become degenerated. An example of an edge contraction is shown in Figure 1.

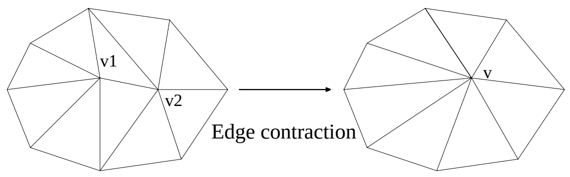

Figure 1: (v1, v2) → v edge contraction.

For the selection of the edge that has to be contracted, a cost function is used: generally this cost reflects the approximation error introduced by the contraction. At each algorithm iteration, the lowest cost edge is contracted.
A particular algorithm developed by Guéziec [Gue95a, Gue96a] maintains a tolerance volume to guarantee that the approximation lie within that volume.
An exhaustive survey of the more recent multiresolution modelling techniques can be found in [Gar99a] and in [Lue01a].

PocketGL library

The PocketGL library has been used to develop the visualization application. PocketGL is a 3D graphics library for PocketPC written in C and C++ which allows to draw 3D objects and manage 3D transformations.
The more important characteristics are:

- Perspective correction.
- Dynamic lightening.
- Alpha transparency.
- Mini OpenGL functions set to works with Triangles, Quads, and Vertex Array.
- BSP and functions for 1st person camera games.
- Landscape mode support.

A game for Pocket PC, Ghost World, has been developed with these library; a screen shot is shown in Figure 2.

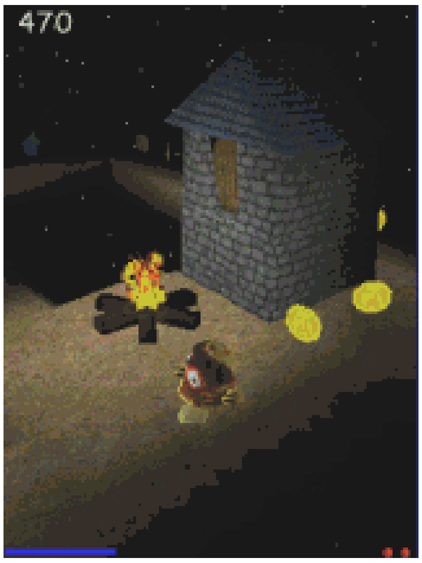

Figure 2: Ghost World screen shot.

3. THE MULTIRESOLUTION ENGINE

In the proposed algorithm the edge contraction is based on the cost function of Melax [Mel98a]:

Equation 4

$$\mathrm{cost}(u,v) = \|u-v\| \times$$
$$\times \max_{f \in T_u} \left\{ \min_{n \in T_{uv}} \frac{1 - f.normal \cdot n.normal}{2} \right\}$$

where T_u is the set of faces that contains the vertex u, T_{uv} is the set of faces that contains both the vertex u and the vertex v. The function cost in equation 4 defines the cost of collapsing an edge as the length of the edge multiplied by a curvature term. The curvature term for collapsing an edge uv is determined by comparing dot products of face normals in order to find the triangle adjacent to u that faces furthest away from the other triangles that are along uv. In this algorithm there is a collapse of one vertex to another.
The data structure is composed by a list of vertices and by a list of faces (triangles). This one contains the indexes into the vertex list (see Figure 3 a)). For the reduction there is a iteration until the desired number of vertices is reached. The pseudo code [Mel98a] for this operation is:

```
while(vertices.num > desired) {
    Vertex *mn = MinimumCostEdge();
    Collapse(mn,mn->collapse);
}
```

this cycle reduces the number of vertices analysing what is the edge with the minimum cost (function MinimumCostEdge) and then collapsing the vertex **mn**. During this process, the information about the vertices that are collapsed is stored. In particular it is sufficient to store in which vertex a vertex is collapsed and to order all the vertices in same order as they are collapsed. For example in Figure 3 are visible three vertices **a**, **b**, **c** where **c** collapses in **b** and **b** collapses in **a**.

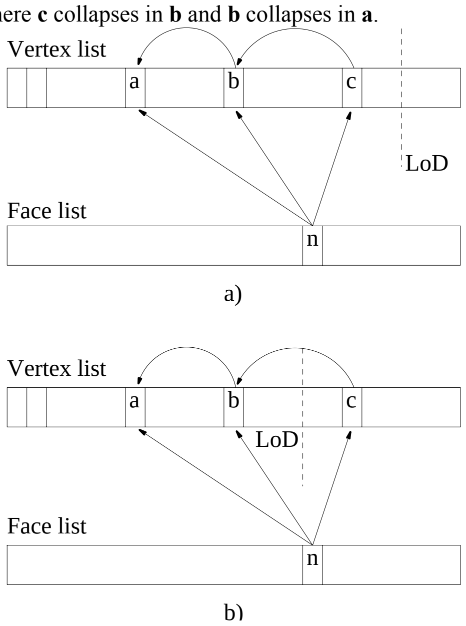

Figure 3: Data structures and vertices collapsing.

The simplification has been executed in the initialization phase: there is a reduction of the model to the suitable minimum number of vertices, obtaining the ordered vertex list.
Afterwards, in the rendering phase, it is sufficient to access to the desired level of detail. In particular there is a cycle for rendering all faces, but each face is rendered only if all its vertices are not collapsed. In Figure 3 a) there is the face **n** that is composed from **a**, **b**, **c** and the actual level of detail is higher than the position of vertex **c**: in this case the face **n** is rendered. In case b), instead, the level of detail is lower than the position of vertex **c** and, because **c** collapses in **b**, the resulting face is degenerate and is not rendered. The number of vertices to render each frame is computed according to the computational power: there is a frame-rate threshold to be reached and this goal is achieved using different model resolutions, also according to their distance from the observer.
In the following there is a description of the rendering phase of the algorithm, executed for each frame:

1. Rendering of non-multiresolution models of the scene.
2. Computation of the frame-rate (this one is computed with an average on more frames for a more stable value).
3. If the observer is moving there is a computation of the percentage of vertices to visualize:
    a. if the frame rate is reducing, there is a reduction of the number of vertices to visualize (proportionally to the distance of the value of the frame-rate from the threshold);
    b. if the frame rate is increasing or is over the threshold, there is an increase of the number of vertices to render (proportionally to the distance of the value of the frame-rate from the threshold).
4. If the observer is still the resolution is the maximum.

5. Update of the position of the observer and of the frame-rate.
6. Distribution of the percentage of vertices that have to be rendered among the visible models in the scene in a proportional way based on their distance from the observer. In particular, a parabola function of the distance is used: for each model, from the farthest to the nearest, a different number of vertices is removed until the number of vertices to render is reached.
7. Rendering of the models.

4. IMPLEMENTATION AND RESULTS

The effectiveness of the proposed 3D multiresolution rendering engine has been evaluated on a iPaq H3630 PDA equipped with the Microsoft PocketPC operating system. Some basic features of the selected handheld device are: 206 MHz Intel StrongARM CPU, 12-bit color depth display, 240x320 pixels (2.26x3.02 inches) TFT touch screen, 32 MB RAM and 16 MB Flash ROM. The traditional keyboard is replaced by a directional pad and by four application buttons. The directional pad acts as the arrow keys on a standard keyboard while the application buttons behaves as programmable launch buttons used to start preferred applications. The PocketPC application wrapping the rendering engine has been endowed with a navigation interface designed towards this ``poor'' input device. By means of the directional pad, camera position can be adjusted in order to perform zoom and rotate operations. The application buttons allow the user to increase and decrease the value of the threshold on the frame rate. Figure 4 shows user interaction with the portable device.

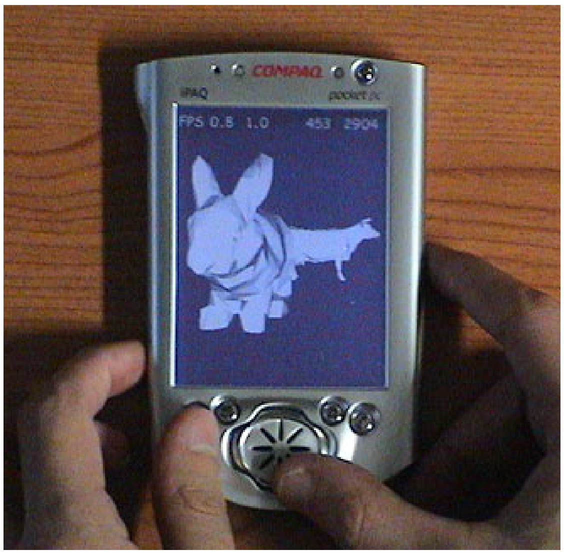

Figure 4: User-PDA interaction.

To assess the performances of the rendering algorithm, a 3D test scene composed by three low-complexity models with about 450 vertices (corresponding to about 900 polygons) has been loaded in the application. The frame rate threshold has been set to a default value of seven frames per second. In spite of the actual value of the threshold, the still scene is displayed with the maximum level of detail resulting in a frame rate of about two frame per second. When the scene is rotated or the distance of the objects from the observer is modified, the level of detail of the whole scene is decreased according to the required frame rate. As expected, farther objects are rendered using a lower number of polygons with respect to nearer ones which maintain an higher level of detail. Figure 5 shows that models simplification starts from the farthest objects in the scene and subsequently affects the remaining objects until the requested frame rate is reached. In the top left corner of each frame, the effective and the required frame rate as well as the current level of detail for each object (expressed as number of vertices) are highlighted. According to the expected model behaviour, it has been experienced that by reducing the scene resolution, the level of interactivity is significantly improved and, as a result, the application promptly react to user commands.

To evaluate the behaviour of the proposed approach with respect to more complex and CPU intensive rendering cycles, a scene composed by two models with about 900 and 5800 polygons (450 and 2900 vertices) respectively has been created. Some screen shots of the application handling multiresolution rendering operations devoted to maintain a frame rate of ten frame per second can be found in Figure 6. Due to the greater complexity of the scene and to the higher frame rate requested, the multiresolution algorithm deeply simplify the 3D models.

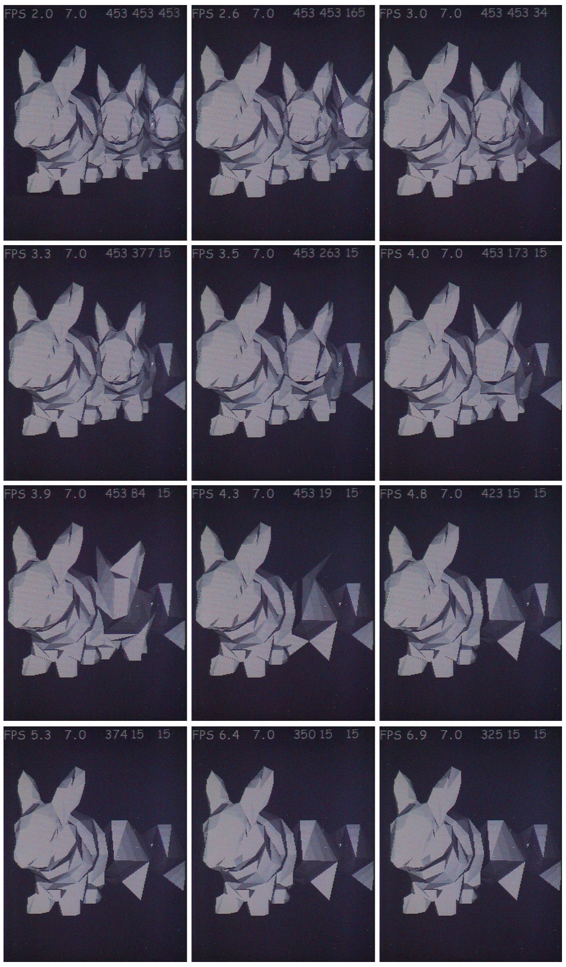

Figure 5: Twelve consecutive frames of an animation showing the convergence to to the selected frame rate threshold.

5. CONCLUSION AND FUTURE WORK

This paper is a first attempt to investigate and tackle issues related to 3D graphics on PDA devices. A continuous multiresolution modeling technique is used to achieve a trade-off between scene complexity/quality and interactivity/frame rate.

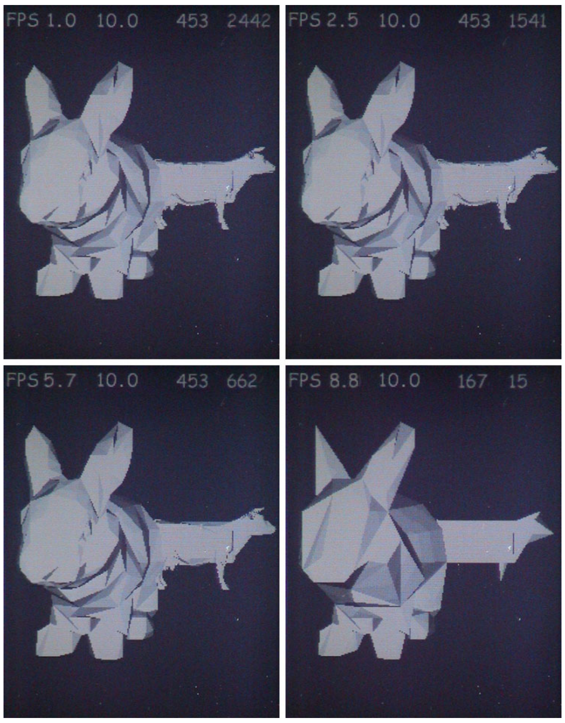

Figure 6: The resolution of a complex scene id dynamically adjusted.

A threshold frame rate value acts as an objective function; in this way, the object level of detail is dynamically adjusted to provide the best quality (near elements are displayed by a larger number of polygons than far ones) at the selected frame rate value.

Future work will be aimed to improve rendering engine functionalities by adding textures and other graphics effects, but, above all, to design a new and more intuitive navigation interface. A navigation compass, usable by the input stylus, could overcome problems due to the lack of keyboard and mouse.

6. ACKNOWLEDGMENTS

This work is supported by The National Research Council (CNR) inthe frame of the project *Strumenti e Tecniche per la Realizzazione di Negozi Virtuali Distribuiti*.

Moreover, we thank Ing. Ivan Perrini for his support in developing this work.

7. REFERENCES

[Cia97a] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. The Visual Computer, 13(5):228-246, 1997.

[Eli02a] Elite, http://home.rochester.rr.com/ohommes/Elite/

[Gar97a] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In SIGGRAPH 97 Proc., pp. 209-216 (August 1997).

[Gar98a] M. Garland and P.S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In IEEE Visualization 98 Conference Proceedings, pp. 263-269 (October 1998).

[Gar99a] M. Garland. Multiresolution modeling: Survey & future opportunities. In Eurographics '99 State of the Art Reports, pp. 111-131, 1999.

[Gue95a] A. Guéziec. Surface simplification with variable tolerance. In Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95), pp. 132-139 (November 1995).

[Gue96a] A. Guéziec. Surface simplification inside a tolerance volume. Technical report, Yorktown Heights, NY 10598. IBM Research Report RC 20440 (March 1996).

[Hop93a] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In SIGGRAPH '93 Proc., pp. 19-26 (August 1993).

[Hop96a] H. Hoppe. Progressive meshes. In SIGGRAPH '96 Proc.}, pp. 99-108 (August 1996).

[Kob98a] L. Kobbelt, S. Campagna, and H.P. Seidel. A general framework for mesh decimation. In Proc. Graphics Interface '98, pp. 43-50 (June 1998).

[Lau98a] R. Lau, M. Green, D. To, and J. Wong. Real-time continuous multi-resolution method for models of arbitrary topology. Presence: Teleoperators and Virtual Environments, 7(1):22-35 (February 1998).

[Lue01a] D.P. Luebke. A developer's survey of polygonal simplification algorithms. In IEEE CG&As, 21(3):24-35, 2001.

[Mel98a] S. Melax. A Simple, Fast, and Effective Polygon Reduction Algorithm. In Game Developer Magazine, pp. 44-49 (November 1998).

[Pok02a] PocketGL, http://pierrel5.free.fr/ThemePGL.htm

[Pop97a] J. Popović and H. Hoppe. Progressive simplicial complexes. In SIGGRAPH 97 Proc., pp. 217-224 (August 1997).

[Pre85a] F.P. Preparata and M.I. Shamos. Computational Geometry: an Introduction. Springer-Verlag, New York, NY, 1985.

[Sch92a] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. Computer Graphics (SIGGRAPH '92 Proc.), 26(2):65-70 (July 1992).