Building and Exploiting Levels of Detail: An Overview and Some VRML Experiments

J.L. Pajon, Y. Collenot, X. Lhomme
Institut Français du Pétrole, Rueil-Malmaison, FRANCE
(J_Louis.Pajon, Yves.Collenot, Xavier.Lhomme at ifp.fr)

N. Tsingos, F. Sillion
iMAGIS, IMAG, Grenoble, FRANCE
(Nicolas.Tsingos, Francois.Sillion at imag.fr)

P. Guilloteau, P. Vuyslteker, G. Grillon, D. David
Centre d'Infographie, Université de Marne-la-Vallée, FRANCE
(guillote, pvk, grillon, davidd at indy.univ-mlv.fr)

6 July 1995

Abstract

Interactive walkthroughs in voluminous graphical data are made possible on current computers only if various stratagems are used to increase frame rates. A very common sratagem is to model each object with several levels of detail and to choose appropriate levels at each frame. The choice of a specific level of detail can depend on various factors: performances must not be obtained at the cost of an important degredation of image quality, except perhaps when the viewer moves rapidly inside the scene. We review some techniques for simplifying geometries as well as for selecting the resulting simplified objects. These techniques can be used to generate VRML data and to manipulate them. VRML could be used for various industrial applications if it could include such tools as well as elaborate methods for specifying behaviours, interactions and navigational tasks.

Introduction

In industrial applications we are interested in at the French Institute of Petroleum (such as engineering and more specifically building of oil refineries), great amounts of data are generated. To achieve great interactivity on current graphics workstations, the number of displayed grapical entities such as polygons must be bounded. We have tested various software packages to cope with such problems and we have been surprised by the inefficient approaches used in most of them and by the fact that many methods described in the literature are not avalailable today. We think that future versions of VRML could give interesting solutions to such problems. These solutions could be based on recent advances in this field of work that we would like to summarize in this article.

A method for displaying more objects at interactive frame rates is to temporarily simplify the objects during 3D manipulations and to restore complete geometries when objects are in a fixed position. Several levels of detail, i.e. several polygonal models of various complexities, can thus be precomputed for each object and then the choice of the model to be displayed depends on the distance between the object and the viewer as in VRML 1.0. More elaborate criteria could be used.

Usually, different levels of detail are obtained by simplifying (i.e. removing polygons from) a given initial model. Many methods can be used for achieving such a task. Simplifications of parametric surfaces are easy to obtain. For more general unstructured surfaces, various algorithms can be designed. Some can be very brute-force and dramatically reduce the number of polygons of each model by subsampling points in a systematic way, without taking in account its topology and the curvature of specific regions. More elaborate algorithms have been designed to retain as much as possible the specific features of the initial object. Unfortunately, these algorithms cannot guarantee in all cases a significant reduction of the number of polygons.

The first part of this article gives a brief summary of some other methods for reducing the work of the graphics processors.

In the second and main part of this article, we review the methods described in the literature for performing geometric simplifications and we compare their various benefits and disadvantages. We use one of these methods to generate from any OpenInventor scene several levels of detail that could be saved in VRML. Some examples can be found at URL: *http://www.univ-mlv.fr/~pajon.*

In the last part of this article, we review some criteria that can be used to select appropriate levels of detail during visualization. Some of these criteria could be taken

into account in future versions of VRML. We intend to include them in the viewer *webview* of **SDSC**.

In conclusion, we would like to stress the fact that in our mind and certainly in many other minds VRML must not only be a tool for describing 3D scenes but also for describing how to navigate inside these scenes. In contrast to many graphical softwares like *webspace* which provide predefined navigational methods, we think that it would be very useful to be able to specify in some simple language the various methods that are available to explore 3D scenes. In particular, it could be interesting to imitate the way humans walk inside buildings by constraining them to walk on specific delimited surfaces. Such constraints can be specified but not always as simply as we would like in advanced virtual reality toolkits (such as Clovis of Medialab) we are testing at the French Institute of Petroleum with the data provided by the leading French engineering company TECHNIP.

1 Review of some strategies for interactive walkthroughs

Apart from the display of various levels of detail, there are various other strategies for improving frame rates. We would like to stress three approaches which are based on the same idea. To alleviate the work of the graphics processors, they send only a fraction of the scene to them. This fraction is an estimated superset of the polygons that can be viewed. Unfortunately these methods are inefficient each time all objects are visualized.

At each frame, the viewing space is defined as a truncated pyramid in world coordinates. If the bounding box of an object is outside this space, this object can be omitted by the graphics hardware. Such simple culling strategies are often very efficient and their systematic use can only be very helpful. They are available in some graphics libraries like *Performer* and *OpenInventor* of **SGI**.

In the design process, it is often easy to identify separate worlds or universes that are invisible from each other. When the user is in one of these worlds, he cannot see anyone of the objects that lie inside another world and that therefore do not need to be sent to the graphics processors. To access from one world to another one, specific elements or portals can be defined. Such strategies have become very popular in virtual reality applications, in particular for building walkthroughs.

Based on the previous idea, a more systematic approach consists in subdividing the space into numerous cells and to precompute for each cell the set of all cells that are visible from it. The algorithms described in the litterature [1,4,13] are very specific to architectural models because they rely on the fact that most walls are aligned along two orthogonal axes. In this case, they are very efficient because they use precomputed structures to detect many invisible objects, and they avoid the storage in memory of the entire data, but their usefulness in more general cases can be discussed. Various other visibility precomputations can be imagined but they can be very costly because of their high combinatorial complexity.

2 Review of some algorithms for polygonal simplification

2.1 Application Areas and Main Objectives

In various application areas, there is a need for simplifying polygonal models. We can make a distinction between two different ways of generating data, by sensor acquisition or by modelling.

Many data are obtained from physical objects by acquisition techniques such as 3D scanners. Medical scanners produce volume data giving the value of some physical property in each cell of a regularly subdivided parallelepipedic domain while laser range scanners produce clouds of points distributed on the scanned surfaces. Let us summarize the problems encountered in these two cases.

Volume data contain very often millions of voxels (it is not uncommon to have more than 100 images of resolution 512×512) that are often difficult to exploit with direct rendering methods, although many solutions exist now for some particular types of data, owing to a lot of algorithmical research, some progresses in graphics hardware like $3D-$texturing on particular Silicon Graphics systems and some efficient commercial softwares. It is often appropriate to extract surfaces of interest, and in particular polygonal models, from these data. The models generated by most usual methods such as the Marching Cubes algorithm contain too many polygons and the cannot be efficiently rendered on many current machines. This is the case in particular for segmenting medical data or reverse-engineering. Here the main aim is really to replace an initial model by a more concise one that will be the only one to be analysed.

Range data are made of rectangular grids of distances from the sensor to an object. The grid spacings are independent from the shape of the object, such that regions of same extent with low or high curvature are digitized with the same number of points. In order to adapt the grid to the shape characteristics and to use as few points as possible to have a concise but nevertheless accurate representation, simplification methods are required [2].

Many other data are manually obtained by designers using modelling software. In this case, generally the initial data is always present in the visualization process and can be brought at any moment, allowing the user to concentrate on it and not on its simplified models. Such applications include interactive navigation through CAD data [9], visual simulators [8] and video-games.

Although our main interest is in visualization of CAD data, the understanding of simplification techniques and strategies is improved by considering the particular features of the various preceding application areas. Depending on these areas, various sets of rules and constraints can be used for modifying polygonal data.

For example in which way are the initial topology and geometry respected ?

Regarding topological constraints, we can distinguish two great classes of methods, according to the respect or not of the initial topology. Although it can seem very important to preserve topology in applications where, as said previously, focus is concentrated on the simplified data, in other applications where these data are visualized only temporarily during fast motions until the initial model is restored for more profound examination, the topology has not to be preserved absolutely. For example, in the algorithm described in [9], small distinct objects that are close can be merged together, and any object can be reduced to a segment if its shape is elongated or to a point if its shape is spherical. A further distinction [6] can be made between the methods that explicitly require input data to consist of only one mesh embedding a 2-manifold, or any type of polygonal data of various topologies.

Regarding geometrical constraints, there are also various strategies. For example, it can be crucial in some cases to preserve sharp features and therefore to keep edges and points that contribute to these features. Such edges can be identified as determining dihedral angles exceeding some threshold value.

To discuss the algorithms, we distinguish two great classes of methods:

1) Clustering Algorithms: an initial polygonal model is subsampled by regrouping primitives to form only one primitive in the final model. In this case, the two methods that will be described do not guarantee topology consistency.

2) Iterative Algorithms: an initial polygonal model is simplified or adjusted step by step until some criteria are fulfilled. A subset of the initial points is looked for, or a new surface can be fitted to the initial points.

Other distinctions can be made. In particular, some algorithms constraint the final model to be made of a subset of the initial points and some others do not deal with such constraints and in this case it is possible to have old or new points in the final model. We can also notice that some algorithms have been adapted to allow a sort of "continuous interpolation" between different levels of detail. The data structures required by these algorithms can also be considered.

We will also distinguish the way in which these methods deal with the two basic tasks they perform: modification of geometry (points positions) or of connectivity. Either these two tasks are made simultaneously at each iteration, or they are made successively in two separate steps.

2.2 Clustering Algorithms

In the two methods described in this section, the initial topology of the model is not constrained to be preserved in the coarser models. Old points can be replaced by a subset of them or by new points, that can be computed by averaging methods (which have often the disadvantage of reducing the size of the objects).

2.2.1 Regular space partitioning [9]

The basic idea of this algorithm consists in regularly subdividing the bounding box of the scene and to merge into one point the vertices which lie inside each of the cells of this subdivision. The resulting polygonal models have less vertices. This algorithm is very effective in reducing the number of polygons very rapidly and it is easy to program as it requires only simple data strustures and simple traversals of them. Continuous transition from one model to another one is allowed. The shape of simple surfaces is preserved, but the topology of more complex objects can change during simplification and in particular many holes can be introduced (aliasing effects). This sampling process eliminates all details that are smaller than the size of the grid cell, independently from their semantic importance, and it cannot suppress all redondancy: if a planar component is made of polygons that extend over several cells, it cannot replace all these polygons by only one. Futhermore, results are not invariant to translations or rotations of the model.

We wrote two programs to test this algorithm. GL library is used in the first one, which is faster than the second that was written in OpenInventor. This second program can be used to save VRML files (only the header has to be modified).

Results of the first program are summarized in the table below and in a figure which can be found at URL: *http://www.univ-mlv.fr/~pajon/levels.rgb*. Three levels of detail are defined for two examples, a sphere and a mechanical part.

	State 0		*State 1*		*State 2*	
	Points	Rate	Points	Rate	Points	Rate
Sphere	99452	6	419	20	101	30
CAD	14455	10	598	20	117	30

The second program as well as some VRML resulting scenes are available at the same URL: *http://www.univ-mlv.fr/~pajon*. In these examples, we can see that this algorithm is not very efficient if it separately deals the very small components of some part of an oil refinery.

2.2.2 Hierarchical object partitioning [10]

The idea of this algorithm is also to eliminate small details. Instead of removing all details contained inside cells of the same size, it replaces by one point sets of points that are close to each other. To perform this operation, it stores points in a specific tree where each node admits as children a cluster of points characterized by a dissimilarity measure: the strongest distance between any two of these points. All the points in a cluster can be replaced by a representative if the corresponding dissimilarity measure is smaller than the smallest detail size chosen by the user.

In contrast to the previous method, this method gives results invariant to rigid motion, as it relies on the intrinsic spatial properties of the model and not on an arbitrary subdivision of space. The authors state that this method

can only deal with small models comprising less than some hundreds of points because the building of the hierarchical clustering tree can take some time. As the preceding one, it doesn't suppress all redondancy. An initial step performed with one of the following algorithms could be used for this task.

2.3 Iterative Algorithms for Selecting a Subpart of a Set of Initial Points

2.3.1 Decimation of triangle meshes [12]

The basic approach starts from a given polygonal model and then tries to improve this model by progressively removing points from it and by filling with less triangles the holes thus created. The criterion for removing a point is its distance to a plane approximating the surface where its neighboring points are lying. If this distance is smaller than a specified value, the point is a candidate for removal. But it will be removed only if it is possible to find two vertices of the border of the hole such that the "split plane" which contains these points and which is orthogonal to the average plane is not intersected by the border edges of the hole. Arbitrary topologies are accepted and preserved by this algorithm. For exemple, an edge that is shared by more than two triangles is acceptable but it will not be removed. Each iteration modifies both geometry and connectivity.

2.3.2 Adaptive Subdivision, Polygon Growth [2]

These methods have been described in the case of quadrilateral meshes commonly obtained in 3D digitizing techniques. Particular algorithms have been devised in this case. Rectangular areas of points are approximated by polygons. An adaptive algorithm allows to recursively subdivide such polygons if they do not satisfy some fitting criterion. Conversely, polygon growth algorithms allow to combine several polygons together until this same criterion is not satisfied.

2.4 Iterative Algorithms for Fitting Surfaces to a Set of Initial Points

The following algorithms are restricted to surface meshes.

2.4.1 Mesh Optimization [7]

As in preceding iterative approaches, the algorithm starts from an initial polygonal mesh, that is progressively modified to contain less vertices while preserving the initial shape. These two criteria are expressed in a mathematical way in order to allow numerical algorithms to be explicitly performed. The topology and geometry of the mesh are considered separately. For a fixed mesh, an inner, continuous minimization is made by varying locally the geometry, i.e. by modifying the location of some points. For a fixed geometry, an outer, discrete minimization is made by using three basic transformatioms (edge collapse, edge split and edge swap) for varying the mesh structure. Local heuristics are used to simplify the inner minimization algorithm. Nevertheless, this algorithm seems to involve complex computations.

2.4.2 Re-Tiling [14]

The retiling algorithm is a multi-step process that generates from an initial mesh a more concise representation. In an initial step, an arbitrary number of points (that will be the points of the surface that is looked for) is randomly distributed on the initial surface. These points are then submitted to repulsive forces that move them farther from each other in order to distribute them more evenly. This process can be adaptive to concentrate more points in regions of high curvature. These points are then added to the initial ones to form a *mutual tessellation*. The old points are then removed as in the decimation algorithm. Continuous interpolation from the initial model to the simplified one is possible. As in the previous algorithm, but in a very different way (in two succesive steps and not in two imbricated loops) the two essential steps of this algorithm (location of new points on the surface by point repulsion, generation of a new mesh by mutual tessellation) allow to deal separately, in two successive steps (in contrast to the previous algorithms), with the geometry and then the connectivity of the mesh.

2.4.3 Deformable Models [3]

The basic idea is to approximate a set of points by a deformable mesh, which has initially a simple shape (a sphere or a cylinder for example) and which is progressively deformed to fit the data. This mesh is attracted towards the points of interest by forces that decrease at each iteration in order to minimize an energy functional. If these points form a surface, the final mesh gives a concise representation of this surface, as its number of points can be fixed arbitrarly low. Interactive tools allow also to add more points, to concentrate points on some regions, to cut the mesh and change its topology in order to obtain a more accurate fitting. To allow real time computations, a special class of mesh is used, a 2-simplex mesh: each vertex has a three-connectivity (it is linked to three vertices), which leads to hexagonal faces. This algorithm proceedes in exactly the reverse order of the preceding one: it deals first with the connectivity and second with the geometry of the mesh.

3 Management of levels of detail in real-time visualization

Many approaches can be used to select levels of detail and to make transitions between them. The software designer must take care of the various strategies that will be at the disposal of the user as well as of the degree of control that will be left to him in order to adapt these strategies to his specific needs.

We can list here some alternatives that can be thought of and some proposed solutions.

- Is there an optimal number of levels of detail for each application ? Can it be computed ? Can the user choose this number ? Usually there are approximately four or six levels of detail, which can include the initial model, its convex hull, its bounding-box, its center point and no point at all. A great number of levels of detail can require huge storage capabilities.

- Are all objects displayed with the same level of detail as is done in some softwares (such as *Review* of **CadCentre** or *Megavision* of bf MatraDatavision) or is each object displayed with a particular level of detail depending on its characteristics and its location ? This second solution seems preferable. Let us consider now this option.

- Which rule can be used to change the level of detail of a particular object ? The most basic approach is to make it depend only on the distance of this object to the viewer as is done in visual simulators or on the size in pixels of each object (the two cases are the same if all objects have identical size). More complex functions than this distance can be evaluated at each visualization step in order to determine which level of detail to use. A general framework has been established in [5] to describe the situation and design efficient strategies. For each object O, let us suppose that several levels of detail L and several rendering algorithms R are available and that with each "tuple" (O, L, R) we can associate a benefit $B(O, L, R)$ and a cost $C(O, L, R)$. To have the maximum of detail while maintaining some selected frame rate, the set S of "tuples" to be chosen must maximize the benefit $\sum_S B(O, L, R)$ with the constraint that $\sum_S C(O, L, R) < TargetFrameRate$. Another constraint is that no object can be rendered several times. $B(O, L, R)$ mainly depends on the perceived size of object O but it can also depend on the accuracy on its representation, its particular meaning in the application, its location relative to the center of the window, etc. $C(O, L, R)$ depends on the number of polygons and points that are displayed as well as on the number of pixels covered by O. This constrained optimization problem guarantees a constant frame rate without degrading too much the image quality, but it is NP-complete as this is a continuous multiple choice knapsack problem. A greedy algorithm to approximate the solution it is to calculate the function $B(O, L, R)/C(O, L, R)$ for each tuple and to select the tuples that have the highest values until the maximum cost is reached. Frame to frame coherence can be used to update this function incrementally at each step.

  Is it possible to use fuzzy logic and genetic algorithms to ease the implementation of the preceding methods, as is suggested by some authors [11] ?

- Let us suppose that the only used criterion for switching models is distance. Is it possible to compute automatically at which distances switchings can be made ? Can simple rules be applied to obtain these distances ? Can they change during animation ? Futhermore, which rule can we apply to select the part of polygons that is retained from one distance to the following ? In [8], the author suggests to divide space into several parts of equal sizes (except perhaps the two regions nearest to the user, that can be two times smaller) and to reduce polygon density in proportion to the square of the distance.

  We can notice also that such computations of distances can be made either in real time during visualization or in an initial step in the following manner for example: the space is subdivided into cubes of equal size and to each one of these cubes we associate for each object the level of detail that is appropriate. Objects contained in a cube would appear in full detail when the viewer is in this cube while objects that are farther on can be rendered with less details or no detail at all.

  As said previously, a more constraining and effecient rule than distance comparision for selecting levels of detail would be for example to bound the maximal number n of polygons that can be displayed, in order to achieve a constant frame rate. A proposed solution would be the following. If there are five levels of detail, the last one consisting of void models, we can define for each of them an arbitrary maximum number of polygons such that these numbers comes to n. Let us suppose also that the objects are ordered according to their distances to the viewer. It is then easy to associate the first level of detail with the first objects until the number of cumulated polygons is greater than the specified bounding number, and so on for the following objects and the other levels of detail. This algorithm is simpler than the one described in the previous session and it can give perhaps unnecessarily less details, as the maximum number of polygons for each level of detail is fixed. We intend to include this algorithm in the VRML browser *webview* of **SDSC**.

- How to prevent transitions from one level of detail to another from being too distracting to the user ? In [8], the author suggests to progressively replace a more and more transparent object by a more and more opaque object. In order to avoid continual switching between two models when for example the viewer turns around the same object while maintaining approximately the same distance, the author suggests also to have two switching distances for each transition, the first one for allowing more detail and the second one for removing it.

Conclusion

In this paper we reviewed some geometry simplification techniques which are described in some recent papers and which seem to be promising for interactive navigation in voluminous data. Methods for exploiting the resulting levels of detail are also summarized.

There are two main classes of simplifying algorithms. Either they remove redundant polygons while respecting the initial topology of given surfaces but in this way there is some limit to polygon reduction, or they can be more brute-force by removing features of any size and even by reducing entire objects to almost nothing. These two types of methods have their own advantages. There is clearly a need for software tools which allow a full range of simplification results, and a lot of work remains to be done to choose the best algorithms from many existing ones in each class, to improve them, and to combine them in integrated software packages, which must answer the particular needs of various application areas (virtual reality, visual simulation, games, but also medicine, computational fluid dynamics where finite elements computations can be made in grids obtained from real data, etc). Difficult compromises have to be made to conciliate high quality rendering with high frame rates, especially in virtual reality applications where small user's motions are constantly recorded. The situation is simpler in desktop systems, where detailed models can be displayed when there is no user's action.

Nowadays, many modelers offer some tools for simplifying data (*MultiGen* of **Software Systems**, *3D-Studio* of **Autodesk**, *Explore* of **Wavefront**), which can then be used by visual simulation or virtual reality toolkits (*Performer* of **SGI**, *Vega* of **Paradigm Simulation**, *Clovis* of **Medialab**). Some virtual reality toolkits (*WorldToolKit* of **Sense8**, *dVS* of **DIVISION**) include their own utilities for achieving such tasks.

Some visualisation tools for engineering data (*Review* of **CadCentre**, *3DIX* of **IBM**) integrate also automatic simplification and levelling tools, which eases the final user's task. But these tools do not have the flexibility of virtual reality toolkits which can be used to build a large set of interactive experiments and to perform various human factors studies.

Methods for generating levels of detail and for using them to accelerate frame rates are very various from one software package to another one, and many research results do not seem to have been fully exploited yet. In particular, continuous interpolation from one model to another one (a particular case of morphing) is not usually found in these packages. We hope that satisfying solutions will be available very soon. In particular, we think that future VRML specifications give a good opportunity to unify the various approaches described in the literature and to exploit the most interesting features of each of them.

We think also that apart from the way large amounts of data are managed in a VRML visualization tool it could be interesting to specify in VRML the way navigation is done. For example, navigation can be constrained to some surfaces in such a way that the viewer stays always at the same distance of this surface and cannot leave this surface as if he was in a room. In the same way, hyperlinks between 3D scenes are a way of specifying how this observer can navigate from a place to another one. In typical virtual reality applications, transitions from a world to another one are only possible if the viewer can really reach some specific object in 3D world and not if he can select a 2D location in a 2D view. Such specific behaviors of the program have to be made possible in VRML to adapt it to realistic walkthroughs inside buildings.

More generally we think that one of the main interests of a language such as VRML is the ability to specify the behaviour of a program by providing in a simple language all the informations it needs to perform this behaviour. Nowadays, the behaviour of a program is ordinarily specified in one of these two ways: either this program has been built to achieve a set of predefined behaviors or, less frequently, the user of this program must give it the ability to achieve a suitable behaviour by writing some complex code that can be interpreted or recompiled. Obviously a simpler solution would be to use a simple language to specify complex behaviours. Will VRML become such a language ?

OpenInventor format of **Silicon Graphics** or MAZ format of **DIVISION** are some examples for specifying interactions with objets or behaviours of objects (with the notion of engine in OpenInventor). But in such formats as well as in VRML 1.0 navigational tasks are not specified but depend on the visualization programs that are used as if these were ordinary tasks that could be performed in some standard ways, as is done for example in *webspace* or *webview*. In contrast, we think that such tasks devote a lot of attention and could be specified in a simple extension of VRML 1.0.

Acknowledgments

We would like to thank for their numerous contributions the teams of computer graphics experts of IFP Image group, iMAGIS goup at IMAG-INRIA in Grenoble and Computer Graphics Center of Marne-la-Vallée University, and in particular V. Bui-Tran, J. David and O. Dupuy. We thank also the Computer Division of TECHNIP, and in particular J.J. Avezou, H. Poissonier and F. Haynes for providing us with huge datasets containing millions of polygons that constitute a true challenge for our Onyx VTX graphics workstation. Unfortunately, these data are confidential and cannot be reproduced, but they will be included in a videotape.

References

[1] Airey J.M., Rohlf J. H., Brooks F.P., Jr. *Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments*, ACM Siggraph Special Issue on 1990 Symposium on Interactive 3D Graphics, Vol. 24, N.2, pp. 41-50

[2] DeHaemer M.J., Zyda M.J. *Simplification of Objects Rendered by Polygonal Approximations*, Computer and Graphics, Vol. 15, No. 2, 1991, pp. 175-184

[3] Delingette H. *Simplex Meshes: a General Representation for 3D Shape Reconstruction*, INRIA Research Report 2214, 1994

[4] Funkhouser T.A., Séquin C. H., Teller S.J. *Management of Large Amounts of Data in Interactive Building Walkthroughs*, ACM Siggraph Special Issue on 1992 Symposium on Interactive 3D Graphics, pp. 11-20

[5] Funkhouser T.A., Séquin C. H. *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments*, ACM Computer Graphics Proceedings, 1993, pp. 247-254

[6] Heckbert P.S., Garland M. *Multiresolution Modeling for Fast Rendering*, Proc. Graphics Interface'94, pp. 43-50

[7] Hoppe H., DeRose T., Duchamp T., McDonald J., Suetzle W. *Mesh Optimization*, ACM Computer Graphics Proceedings, 1993, pp. 19-26

[8] Latham R. *An Introduction to Model Switching* Real Time Graphics, Vol. 1, No. 1, 1992, pp. 6-13

[9] Rossignac J. and Borrel P. *Multi-resolution 3D approximations for rendering complex scenes*, Modeling in Computer Graphics, Facidieno B., Kunii T.L. (Eds), Springer-Verlag, 1993, pp. 455-466

[10] Schaufler G., Sturzlinger W. *Generating Multiple Levels of Detail from Polygonal Geometry Models*, Proceedings of Second Eurographics Workshop on Virtual Environments, 1995

[11] Schraft R.D., Neugebauer J., Flaig T. and Dainghaus R. *A Fuzzy Controlled Rendering System for Virtual Reality Systems Optimised by Genetic Algorithms*, Proceedings of Second Eurographics Workshop on Virtual Environments, 1995

[12] Schroeder W.J., Zarge J.A., Lorensen W.E *Decimation of Triangle Meshes*, ACM Computer Graphics, Vol. 26, No. 2, 1992, pp. 65-70

[13] Teller S.J., Séquin C.H. *Visibility Preprocessing for Interactive Walkthroughs*, ACM Computer Graphics, Vol. 25, No. 4, 1991, pp. 61-69

[14] Turk G. *Re-Tiling Polygonal Surfaces*, ACM Computer Graphics, Vol. 26, No. 2, 1992, pp. 55-64