A Hybrid LOD Based Rendering Approach for Dynamic Scenes

Karsten Hilbert
Chemnitz University of Technology
Department of Computer Science
Computer Graphics and Visualization
09107 Chemnitz, Germany
karsten.hilbert@informatik.tu-chemnitz.de

Guido Brunnett
Chemnitz University of Technology
Department of Computer Science
Computer Graphics and Visualization
09107 Chemnitz, Germany
brunnett@informatik.tu-chemnitz.de

Abstract

In this paper we present a novel LOD based rendering approach that integrates geometry based and image based LOD mechanisms in one system. We assume that every LOD object in the scene is represented by a textured polygonal mesh. In a preprocess we generate a continuous multiresolution model for every LOD object in the scene. During runtime we first check for every LOD object if an appropriate cached approximation generated for former frames is available. If so, this approximation object can be used for rendering the next frame. Otherwise, we extract a new suitable approximation of this object out of its continuous multiresolution model depending on view-dependent criteria. After that, we replace this adaptive geometry-based approximation by an image-based impostor if this is reasonable and efficient. To test efficiency we use resolution-dependent criterion and perform a cost analysis that compares costs for using a geometry-based approximation with costs for using an image-based approximation. Object approximations that have been created in this step are saved in a approximation cache using a LRU-mechanism. The selected approximations of LOD objects are used for rendering the next frame.

1 Introduction

1.1 Definitions

Level-of-Detail (LOD) based rendering approaches process a multiresolution model either for the whole scene or for each object in the scene. In a preprocess a multiresolution representation (that is a data structure out of that approximations can be extracted) is produced for each object or the whole scene. At runtime approximations are extracted out of these multiresolution models that are used for rendering. One of the key issues in this context is the proper choice of the approximation which should be less detailed (i.e. contain less polygons) but should look very similar to the original object at least from the current viewpoint. If this can be achieved high quality images can be rendered efficiently with the LOD approach.

For the generation of approximations two different approaches have been suggested.

Geometry-based approximations are represented by polygonal meshes and can have a constant (non-adaptive approximations) or a varying Level-of-Detail (adaptive approximations). Impostors, billboards, sprites with depth and layered depth imaged (LDI) are examples for image-based approximations. An impostor is a simple primitive that can be used for the creation of different object views within a small range. It possesses the most important visual features of the original object. It is represented by a few textured quadrilaterals in most of the cases or in special cases by a few simple textured polygonal meshes [4]. At least an RGBA texture and optional a depth map are associated with each quadrilateral or mesh. If available the content of the depth map is used to generate a warped texture that is applied to the quadrilateral or the mesh. Impostors are very similar to billboards which also consist of textured single- or multi-layered, world- or screen-aligned polygons. The main difference is that the billboard texture is used in the creation of arbitrary views of the object while the impostor texture has been created to look reasonable for a small range of object views.

1.2 Previous Work

The first proposed LOD based rendering approaches[1][2] are based on discrete multiresolution models. Each of such models contains a series of discrete object approximations that were generated by repeated polygonal simplification. At runtime one

object approximation out of the discrete multiresolution model is selected for rendering depending on view-independent criteria.

Sets of pre-generated impostors have been used in [3][4][5][6][7][8] as approximations of objects or object groups. At runtime for each object or object group an appropriate impostor is selected for rendering depending on view-independent criteria.

Approaches based on discrete multiresolution models suffer from visual artifacts when a currently used object approximation is replaced by another one (popping artifacts). Because only a small number of approximations is available for each object if discrete multiresolution models are used the approximation used for a special view configuration is not always optimal for this view configuration.

Therefore current LOD based rendering approaches process a continuous multiresolution model for each single object or the whole scene. This avoids popping artifacts. A continuous multiresolution models is an abstract data structure out of that an approximation that is suitable for the current view can be extracted. There are two main categories of continuous multiresolution models: incremental and hierarchical continuous multiresolution models. Incremental multiresolution models[9][10] contain a very reduced base model and a sequence of refinement operations. At runtime a subsequence of this refinement stream is determined depending on a view-independent criteria and applied to the base model to construct a suitable object approximation that can be used for rendering. The major disadvantage of this method is that the refinement operations in the refinement stream of dynamic incremental multiresolution models can only be executed in that order they were entered in the refinement stream. So it is impossible to extract an adaptive approximation out of an incremental continuous multiresolution model. This disadvantage can be avoided by using a hierarchical continuous multiresolution model. A hierarchical continuous multiresolution model[11][12][13] represents a dependency graph of refinement or simplification operations. Here the extracted adaptive approximation that is used for rendering corresponds to cut that is moved through this hierarchy depending on a mostly view-dependent criterion. So the extracted approximations are always optimal for the current viewing configuration. However, extracting an adaptive approximation out of an dynamic hierarchical multiresolution model is very time-consuming.

Some image-based rendering approaches suggested in the last decade use dynamically generated impostors [14][15] or other dynamically generated image-based approximations[18][19] for objects or object groups. Before rendering the next frame for each object a decision is made whether a cached impostor of the object can be reused, a new impostor has to be generated for the object or if the original geometry is used for rendering. If dynamically generated impostors can be used the number of polygons that have to be rendered is reduced dramatically. Again the high computational cost for impostor generation based on the original geometry is the main drawback of this approach.

Hierarchical image caches[16][17] are a very sophisticated application of image-based and geometry-based rendering techniques. These systems use a hierarchy of impostors to reduce the number of times an impostor needs to be regenerated. The basic idea is to partition the scene into a hierarchy of boxes and create an impostor for each box. Before a frame is rendered the impostors that have become invalid are updated and propagated towards the root of the hierarchy. For objects that lie in the view frustum and in the same box as the viewer polygonal meshes are used for rendering. For boxes that lie in the view frustum the impostors of the boxes are used for rendering. Note that this approach can not be used to process dynamic scenes. Furthermore it seems very difficult to parallelize it.

1.3 Our Approach

We are interested in using LOD techniques as a necessary ingredient for real time rendering of dynamic scenes in virtual reality (VR) applications. Therefore we assume the scene to be given as a scene graph that allows access to each individual object in the scene. Furthermore, we require that all objects are represented as textured polygonal meshes of arbitrary topology.

Our approach combines geometry-based and image-based LOD-techniques and then avoids the disadvantage of existing approaches mentioned in 1.2. In a preprocess we create a continuous hierarchical multiresolution model for each LOD object in scene. Among such models we found the vertex hierarchy [13] to be most appropriate for our application. This scheme does not require any knowledge about the polygonal meshes. Manifold topology is neither assumed nor preserved. Out of the vertex hierarchy an adaptive approximation can be extracted based on view-dependent criteria. Our data structure differs in the following respects from the original one suggested by Luebke and Erikson. In addition to the vertex tree and the list of active triangles we assume normal and color information to be given as textures (represented by a

series of mip maps). At runtime we compute adapted texture coordinates for the vertices of active triangles to obtain color and normal information. In this way we avoid the occurrence of discontinuities in the texture of an approximation. Note also that we create a vertex hierarchy for each object in the scene instead for the whole scene as suggested by Luebke and Erikson[13]. This allows us to apply the approach to dynamic scenes. In order to reduce the computational expense for the generation of approximations we realized a caching mechanism both for geometry- and image-based approximations.

Before rendering a frame we perform the following operations: First we check if there exists a valid approximation in the approximation cache for each LOD object. The validity of an approximation is checked using a bounding-box-based view-dependent. For an object with a valid approximation in the approximation cache this cached approximation can be used for rendering. For all other LOD objects a new approximation has to be created. For these objects we first extract a suitable adaptive approximation out of the object's vertex hierarchy. Then we decide whether it is reasonable and efficient to replace the geometry based approximation by an impostor. This decision is based on three criteria. First of all we check if the object can be separated from all other objects in the scene. Only in this situation the replacement by an impostor is reasonable. This is checked using a bounding-box-based collision detection mechanism.

Second an object should not be approximated by an impostor if its distance to the viewer falls below a given threshold. This depends on the ratio of resolution of the impostor texture and the resolution of the output device.. Third it is only efficient to create an impostor for the object if the impostor can be reused for a sufficient number of frames. To check this we estimate the number of frames the impostor will be reused based on a cost analysis similar to Shade et al[17]. Former approaches only used the original geometry to create an impostor. That is not efficient. If all three criteria are fulfilled we use the created adaptive approximation to generate an impostor for the object.

If enough memory for saving the approximation and its parameters in the approximation cache is not available a LRU (Last Recently Used) mechanism is used to identify approximations of the object that can be deleted to free memory. Therefore a timestamp is saved with every approximation of every object. Then the created approximation and several important information is saved in the approximation cache for further usage.

Finally the selected approximations of LOD objects are used for rendering the next frame.

Although it is not the focus of this paper we would like to mention that our approach that is currently implemented on a single processor system will be implemented distributed on a PC cluster in the near future.

1.4 Structure of this Paper

First the used continuous multiresolution model of each LOD object and its creation will be described in section two. The technique for extracting object approximations out of such a multiresolution model will also be explained in this section. In section three we describe how to create impostors based on these adaptive approximations. Furthermore we define the criteria to decide wether it is reasonable to create the image-based approximation. In section four the used caching mechanism is described. The computation of adapted texture coordinates for obtaining color and normal information is explained in section five. In section six we will explain how the algorithm can be modified such that dynamic objects can be processed. The results are described in section seven. Finally, we summarize and discuss the results in section eight.

2 Creating and Processing a Vertex Hierarchy for an LOD Object

If all LOD objects are available as textured meshes a continuous multiresolution model can be created for these objects. Our multiresolution model of an object consists of a vertex tree, a list of active triangles and color and normal textures (represented by series of mip maps).

A vertex tree is a continuous hierarchical multiresolution representation of a scene's geometry [13]. In contrast to [13] we create a vertex tree for each LOD object in the scene. Coordinates of vertices are saved as coordinates of object coordinate system. All computations are done in object coordinate system. This enables use to support dynamic scenes. The multiresolution models of LOD objects are produced in a preprocess. For each vertex of the object a weight value w can be determined by a local criterion computed based on local curvature and lengths of adjacent edges. The higher the local curvature and the higher the average length of adjacent edges are the higher the weight value of an vertex is.

In fact a vertex hierarchy can be produced by using several simplification operations. We use an octree-clustering-scheme suggested in [13]. This scheme does

not require any knowledge about the polygonal mesh. Manifold topology is neither assumed nor preserved. An disadvantage of this vertex tree generation strategy is that meshes with degeneracies (as cracks, T-junctions, and missing polygons) may appear. We start with the axially parallel bounding box of the object. A vertex of the object with the highest weight value *w* can be found. That is the representative vertex of this octree cell. Now the axially parallel bounding box of the object is subdivided into eight axially parallel octants of equal size. Every octant contains at least one of the object's vertices. In every octant there is a vertex whose weight value is maximal. That is the representative vertex of this octant. During runtime all vertices contained in this octant are collapsed to this vertex or the vertex is split into the vertices contained in the octant depending on a view-dependent criterion. Not the octants themselves, but the bounding boxes of the vertices contained in the octants are now subdivided in axially parallel sub-octants and these sub-octants are processed in the same way as mentioned above. This results in a faster termination of subdivision recursion and speeds up generation of multiresolution models. The recursive subdivision process is continued for each sub-octant. The recursion stops at octants that contain less vertices of the model than a prescribed threshold. Each node of a vertex tree corresponds to an octree cell generated by recursive subdivision of the object's bounding box. So in each node of the vertex hierarchy information about the position of the corresponding octree cell in the octree structure, its status, the representative vertex of the corresponding octree cell, the bounding sphere of the vertices contained in the corresponding octree cell, pre-computed parameters for fast evaluation of a view-dependent criterion, about fully and partially included triangles and some additional information are saved. In the following the data structure [13][22] containing all information we save for each subcell is described:

```
struct Node {
BitVec id;
Byte depth;
NodeStatus label;
Coord repvert;
Coord texcoords;
Coord center;
float radius;
Coord coneNormal;
Float coneAngle;
Tri * tris;
Tri * subtris;
Node * parent;
Byte numchildren;
Node ** children;
};
```

- **id**: a bit vector which labels the path from the root of the vertex tree to the node. For the vertex octree described here, each 3-bit triple in the vector denotes the correct branch at that level.
- **depth**: the depth of the node in the vertex tree. The depth and id together uniquely identify the node.
- **label**: the node's status: *active*, *boundary*, or *inactive*.
- **repvert**: the coordinates of the node's representative vertex. All vertices in boundary and inactive nodes are collapsed to this vertex.
- **texcoord:** texture coordinates of representative vertex
- **center, radius**: the center and radius of a bounding sphere containing all vertices in this octree cell.
- **coneNormal, coneAngle:** define a cone that contains the Gaussian image of all normals of the triangles at least partially included in the octree cell
- **tris**: a list of triangles with exactly one vertex in the node. These are the triangles whose vertices must be adjusted when the vertex tree node is folded or unfolded.
- **subtris**: a list of triangles with two or three vertices within the octree cell, but no more than one vertex within any child of the octree cell. These triangles will be filtered out if the node is folded, and re-introduced if the node is unfolded.
- **parent, numchildren, children**: the parent and children of this node in the vertex tree.

Figure 1. data structure used for saving nodes of the vertex hierarchy

This data structure is slightly different to the one suggested by Luebke and Erikson in [13]. They save RGB values for every node in the vertex tree. In contrast to this we compute texture coordinates for every vertex of an active triangle depending on its position in object coordinate system at runtime to obtain color and normal information because our approximations are represented by textured meshes. So discontinuities in the approximation's texture can be avoided. Further approaches often introduce discontinuities in the approximation's texture. Further on we save object coordinates of each representative vertex instead of world coordinates. That is done because we do generate a vertex hierarchy for each object and perform all computations in object coordinate system. That enables real-time display of complex dynamic scenes. Further approaches that used vertex hierarchies as multiresolution model for a whole scene were not able to process dynamic scenes. In addition to the vertex tree and the list of active triangles we produce a series of mip maps of a color texture and a series of mip maps of a normal map. Less complex versions of object's geometry can be used for rendering high quality images if these simple versions of object's geometry are used in conjunction with these

textures. The dynamically computed texture coordinates avoid discontinuities in these textures.

The extraction of a suitable adaptive object approximation requires the following steps:

The list of active triangles contains the triangles that form the current adaptive geometry-based approximation of the LOD object. Each triangle of this list is described by a data structure suggested in [13]. The list of active triangles corresponds to a cut that is moved through the vertex tree depending on a view-dependent criteria at runtime. The cut through the vertex tree is formed by a node front. Nodes that are located between the root and this node front are called active nodes. Nodes of the vertex tree that belong to the node front are called boundary nodes. Nodes that are located in subtrees beyond boundary nodes are called inactive nodes.

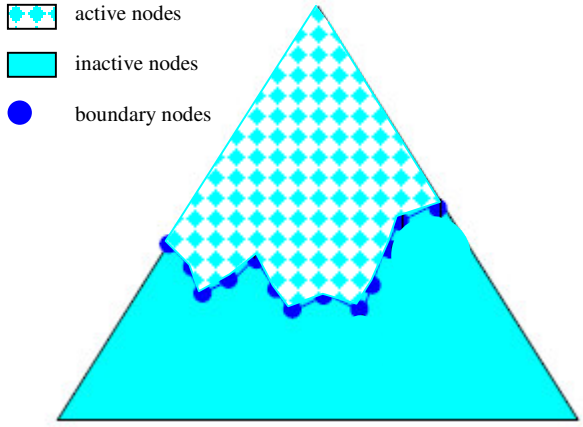

Figure 2. sketch of an vertex tree

Before rendering the first frame the object's list of active polygons is empty. It will be filled during creation of the first adaptive approximation of the object.

The thresholds for controlling the cut movement through the vertex tree have to be given. The correct choice of these thresholds is discussed in the context of the used view-dependent criterion.

Because we perform all computations in the object coordinate system we need the current model transformation of the object to be able to transform all necessary parameters in this coordinate system. That causes additional computational effort, but it enables us to process dynamic scenes. Therefore the viewpoint and the viewing direction have to be transformed into the object coordinate system by using the inverse of the object's current model transformation matrix. Knowledge about the current viewing configuration is necessary to be able to extract the correct adaptive geometry-based approximation of an object.

Our multiresolution model of an object consist of a vertex tree, a list of active triangles and textures that contain attributes. As mentioned before a cut through the vertex tree separates active nodes from inactive nodes of the vertex tree. This cut is moved through the vertex tree depending on a view-dependent criterion and at the same time the list of active polygons is updated incrementally. Moving the cut through the vertex tree means finding a new labeling (active, boundary, inactive) of vertex tree nodes depending on current viewing conditions. Updating the list of active triangles incrementally means adding or removing only some triangles to or from this list. Early implementations of the approach that extracts approximations out of an vertex hierarchy do not update the list of active triangles incrementally. That cause high computational cost for extracting a approximation. Updating the list of active approximations incrementally is less expensive. Because we perform the cut movement incrementally we start the traversal for finding the new labeling of vertex tree nodes at the boundary nodes forming the last determined cut. During this traversal some nodes have to be collapsed and others have to be expanded.

To adjust the cut through the vertex tree we use a view-dependent criterion[13] that is based on screen-space error and takes care of silhouette regions. That means that in interior regions of an object's surface we do a more drastic simplification than in silhouette regions of the object's surface. So the silhouette of all approximations looks very similar and several approximations can not be distinguished very well.

3 Creating and Processing An Impostor for an LOD object

The algorithm has to do the following steps to create an impostor for an object. First it determines the plane P located in the middle point C of the object and whose normal n points from C to the viewpoint v_0. After that the smallest rectangle R in P has to be determined that bounds the projection of the object onto the plane P. Using the rectangle R and the viewpoint v_0 a viewing volume can be defined. In contrast to [14] where the impostor texture is rendered using the original geometry we use an adaptive approximation of the object that was created before to render the impostor texture using the determined viewing volume. During this process we assign an alpha value *1.0* to every vertex of the approximation. So only for pixels where a part of this adaptive approximation was projected onto an alpha value *1.0* is

written in the frame buffer while an alpha value *0.0* stays assigned to every pixel where nothing was projected onto. Now we read this RGBA image back from frame buffer to get the impostor texture. Finally, the position of the bounding rectangle's vertices in object coordinate system are determined. The determined quadrilateral textured with the RGBA texture forms the impostor. The created impostor can be re-used to render following frames while a validity criteria is fulfilled.

Creating an impostor for an object is expensive. The textured mesh representing the current valid approximation of the object has to be rendered and a time-consuming read back from the frame buffer has to be done. It isn't reasonable and efficient to do this in every situation. It is reasonable and efficient to create an impostor if the object can be separated from all other objects, if its distance to the viewer is big enough so that its flat character isn't noticeable and if the costs for impostor creation amortize.

To check if an can be separated from all other objects a bounding-box-based collision detection mechanism can be used.

If the ratio of determined resolution of the impostor texture to resolution of output device does not exceed a given threshold an object's distance to viewer is big enough that it is worth to create an impostor for the object. So the ratio of determined resolution of the impostor texture to the resolution of output device can be used to decide if it is worth to create an impostor or not. This criterion is based on criteria used in [14][16][17]. These two criteria are used to decide if it is reasonable to create an impostor.

After that a cost analysis is performed to decide if it is efficient to create an impostor. Here we use a criterion that was suggested by Shade et al. in [17] in the context of hierarchical image caches. It is worth to create an impostor for an object if the costs for using an impostor in one frame are smaller than the costs for using an geometry-based approximation of the object in one frame. The costs for using the impostor in one frame consist of the cost for rendering the impostor once and a part of the costs for creating the impostor. The costs for rendering the impostor once comply with the time used for rendering the impostor once. The part of costs for the impostor generation are determined by dividing the time used for creating the impostor by the number of frames the impostor will be valid. So we have to estimate the number of frames the impostor will be valid for. Depending on an angle-based criterion a distance can be estimated the impostor can move relative to its current position without losing validity. Using this distance and the current velocity of the object a time span can be determined the impostor will be valid. Using this time span and the current frame rate the number of frames the impostor will be valid for can be estimated. The time span used for impostor creation divided by the number of frames the impostor will be valid for defines one frame's part of costs for generating an impostor.

$$\frac{\text{Cost to Create Impostor}}{\text{est. number of valid frames}} + \text{Cost to Render Impostor Once} < \text{Cost to DrawGeometry based approximation}$$

Equation 1. criterion for efficiency of using an impostor, based on [17]

So we further reduce an adaptive approximation of an object to an impostor only

- if the object can be separate from all other objects in the scene,
- if the ratio of needed texture resolution to resolution of output device is smaller than a given threshold and
- if the costs for using an impostor are smaller than the costs for using the adaptive approximation.

These conditions assure that a created impostor can be reused for at least some frames.

4 Caching Approximations of LOD Objects

Caching the created image-based or geometry-based approximations of LOD objects in an approximation cache is useful because approximations often can be reused for several frames and don't have to be re-created for these frames. This causes an additional speed up of frame rate. Note that it is not necessary to distinguish between geometry-based and image-based approximations in this context because both are represented as textured polygons.

To determine if a cached approximation is available for rendering the next frame the validity of cached approximations of the object has to be checked. Therefore we save several parameters with each approximation of each LOD object after creation. We save the created approximation together with an object id, a time stamp, the viewpoint v_0 it was created for and the vertices of bounding box the object it approximates. These additional information are used to check the validity of this approximation. We check the validity of approximation with the same object id as the processed

LOD object using an angle-based criterion similar to the one suggested by Shade et al. in the context of hierarchical image caches[17]. All computations are done in object coordinate system. So the current viewpoint and the viewpoint the approximation was created for have to be transformed into this coordinate system. For each approximation of an object we know the viewpoint v_0 it was created for and the bounding box of the object it was created for. For every vertex of this bounding box we compute two vectors. The first vector points from the bounding box vertex B_i to the current viewpoint v. The second vector points from the bounding box vertex B_i to the viewpoint v_0 the approximation was created for.

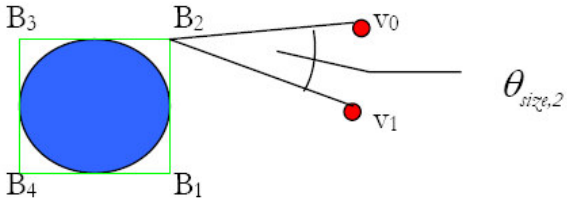

Figure 3. angle-based criterion for checking the validity of an approximation

For each bounding box vertex B_i we compute the angle $\theta_{size,i}$ between these two vectors;

$$\theta_{size,i} = \angle(v_0, B_{i,wc}, v_1)$$

Equation 2.

If the maximum of these eight angles exceeds a given threshold the object approximation is considered to be invalid and will not be used for rendering the next frame.

We don't use only one angle to decide if an approximation is valid because several movements of the viewer are possible where one of these angles stays constant while the impostor or the adaptive approximation isn't suitable for this constellation. If no valid approximation of the object is available a new approximation of the object has to be created that will be saved in an approximation cache. Before an approximation is saved in this cache it is checked if enough cache memory is available. If this is not the case a LRU mechanism is used to identify approximations of the object that can be deleted to free the needed memory. To determine these candidates all approximations of the object (identified by the same object id) are ordered depending on their time stamp. Now candidates with the oldest time stamp are deleted until enough memory is available. Last recently used (LRU) approximations stay in the cache. Finally the created approximation and its parameters are saved in the approximation cache.

5 Computing Adaptive Texture Coordinates at Runtime

After generation of approximation and before rendering a frame we compute the texture coordinates of an approximation's vertices. If the approximation is an impostor the determination of texture coordinates is trivial. If the approximation is an adaptive geometry-based approximation the following operations have to be performed. Color and normal information are sampled in color and normal textures in such a way that the correct color and normal information for an object's vertex can be obtained by using texture coordinates that are computed based on the position of vertex in object coordinate system.

After extracting an adaptive approximation out of the object's multiresolution model we can use the inverse of the current modelview matrix and the position of vertex in object coordinate system to determine the corresponding position in texture space where the correct color and normal information for the vertex is saved.

6 Supporting Dynamic Objects

The proposed LOD based rendering approach supports display of dynamic scenes. More precisely, scenes that contain objects that can be transformed during simulation and whose geometry is constant can be rendered using our approach. The majority of possible scenes can be rendered using our approach. Sometimes scenes contain objects whose geometry is changed by a script or by interaction with a user. The modifications of our approach that are necessary to support this case will be described in the next paragraphs.

We have to distinguish three cases. First there are scenes that contain dynamic objects that consist of several sub-objects whose geometry is constant and that are transformed by script operations. Before writing this object into the scene database it has to be split into its sub-objects manually. So our approach can process each of these subobjects like a single stand-alone object.

Furthermore there are scenes that contain objects whose geometry is changed time-dependent by script operations. Several versions of the original object can be generated by executing the script. In the scene graph the object is replaced by a selection node that selects versions of the object time-dependently and whose

children are time-dependent versions of the object. For each of these object versions a multiresolution model has to be produced. Before rendering the next frame the appropriate version of the object is selected. Based on the multiresolution model of these object version an approximation of the object is generated and used for rendering.

Finally there are scenes that contain objects whose geometry is changed by interaction with an user. New multiresolution models have to be produced for these objects during simulation. This is possible only if the proposed LOD based redering approach is implemented in a distributed manner. In this case a separate task produces updated multiresolution models for objects based on a copy of the original scene graph. Until no updated multiresolution model of the object is available the old multiresolution model will be used for extracting object approximations. If the updated multiresoltuion model is available all cached approximations of the object are deleted and this new multiresolution model is used. This will be content of further studies.

7 Results

We have implemented our approach on a single processor machine with an Intel Pentium 4, 1 GB RAM and an Nvidia GeForce FX 5900 Ultra installed. It was implemented on a Windows XP system. Functionality of the library VDSlib[21] implemented by David Luebke is used by our implementation for generating hierarchical multiresolution models of objects and extracting adaptive approximations out of them. OpenGL is used for rendering. The current implementation of our approach processes the objects of a scene successively. We have tested it with several scenes. One scene used for testing contained 36 copies of a plane consisting of 5.706 polygons. A second scene used for testing contained several copies of an archaeological consisting of 600.000 polygons. A third scenes used for testing contained four copies of turbine blade each consisting of 1.765.388 polygons. In all cases a smooth interaction with the scene at high frame rates was possible when the scenes were rendered using our approach. Furthermore we reached a high image quality using our approach. It is very hard to distinguish between the original object and its approximation. It is not noticeable if an object is approximated by a geometry-based approximation or an impostor.

8 Summary and Discussion

We have suggested a LOD based rendering approach that enables rendering of high quality images of a scene. We use a combination of geometry-based and image-based LOD techniques controlled by three selection criteria. Geometry-based approximations of objects processed by our approach are adaptive approximations that are extracted out of a hierarchical continuous multiresolution model. We use a vertex hierarchy[13] as multiresolution model for each object. Extraction of approximations out of such a vertex hierarchy is controlled by a view-dependent, silhouette-based criteria[12]. The image-based approximations used by our approach are dynamically generated impostors[14]. We use a bounding-box-based collision test, a resolution-dependent criteria and a cost analysis[17] to decide which form of approximation is reasonable and efficient for the given view configuration. Our approach supports caching of object approximations produced for former frames. This should cause an additional speed up of rendering.

Our experience with our current implementation on a single processor machine show that it is possible to render complex scenes at high frame rates. Furthermore the proposed approach can be applied to dynamic scenes because we process a continuous multiresolution model for each object. At this time an implementation of the approach on a single processor machine exists that process the objects of a scene sequential. In the near future we will realize a distributed implementation of this method that also includes an improved caching mechanism. Further on an occlusion culling mechanism should be integrated in the rendering approach. This should cause further speed up of rendering. Our approach needs a lot of memory. The memory requirements of our approach should be reduced. This can be done by optimizing the used data structures.

9 References

[1] James H. Clark. „Hierarchical geometric models for visible surface algorithms“, *CACM 19(10)*, October 1976, pp. 547–554.

[2] Thomas A. Funkhouser and Carlo H. Séquin. „Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments“, *Computer Graphics (* SIGGRAPH ’93 Proc.*)*, 1993.

[3] P. W. C. Maciel and P. Shirley, “Visual Navigation of Large Environments Using Textured Clusters.”, In P. Hanrahan and J. Winget, editors, *1995 Symposium on*

Interactive 3D Graphics, ACM SIGGRAPH, April 1995, pp. 95–102.

[4] F. Sillion, G. Drettakis, and B. Bodelet, "Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery.", *Computer Graphics Forum 16(3)*, Proceedings of Eurographics '97, August 1997, pp. 207–218 .

[5] Daniel G. Aliaga, "Visualization of complex models using dynamic dynamic texture-based simplification", *IEEE Visualization '96*, IEEE, October 1996.

[6] Daniel G. Aliaga and Anselmo Lastra, "Architectural Walkthroughs using Portal Textures", *Proceedings of IEEE Visualization '97*, IEEE, 1997, pp. 355-362.

[7] Rebecca Xiong, "CityScape- A virtual Navigation System for Large Environments", *Master Thesis*, MIT, 1996.

[8] L. Darsa, B. Costa and Amitabh Varshney, "Walkthroughs of Complex environments using image-based Simplification", *Computer & Graphics 22(3)*, February 1998, pp. 55-69.

[9] Hugues Hoppe, „Progressive meshes", *SIGGRAPH '96 Proc.,* ACM, August 1996, pp. 99–108.

[10] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery and Werner Stuetzle, „Multiresolution analysis of arbitrary meshes", *SIGGRAPH '95 Proc.*, ACM, August 1995, pp. 173–182.

[11] Hugues Hoppe, "View-dependent Refinement of Progressive Meshes" *Computer Graphics, SIGGRAPH `97 Proceedings*, ACM, August 1997, pp. 189- 198.

[12] Julie Xia, J. El-Sana and Amitabh Varshney, "Adaptive Real-Time Level-Of-Detail-based Rendering for Polygonal Models", *IEEE Visualization and Computer Graphics*, IEEE, June 1997, pp.

[13] David Luebke and Carl Erikson, "View-Dependent Simplification of Arbitrary Polygonal Environments", *Computer Graphics (31), SIGGRAPH `97 Proceedings*, ACM, August 1997,pp.

[14] G. Schaufler. "Dynamically Generated Impostors.", D. W. Fellner, editor, *Modeling - Virtual Worlds - Distributed Graphics*, MVD'95 Workshop, November 1995, pp. 129–136.

[15] G. Schaufler. "Per-Object Image Warping with Layered Impostors", N. M. G. Drettakis, editor, *Rendering Techniques '98, Proceedings of the Eurographics Workshop,* Eurographics, Springer, Vienna, Austria, June 29-July 1, 1998, pages 145–156.

[16] G. Schaufler and W. Stürzlinger. "A Three Dimensional Image Cache for Virtual Reality", *Computer Graphics Forum 15(3) Proceedings of Eurographics '96*, Eurographics, August 1996, pp. 227–236.

[17] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder, "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments." H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, *Annual Conference Series*, ACM SIGGRAPH, Addison Wesley, New Orleans, Louisiana, 04-09, August 1996, pp 75–82.

[18] J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. "Layered Depth Images." M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, *Annual Conference Series*, ACM SIGGRAPH, Addison Wesley , July 1998, pp. 231–242.

[19] G. Schaufler. "Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes.", J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, Eurographics, Springer , Wien , New York City, NY, June 1997, pages 151–162.

[20] David Luebke, "Simplification", *Course Notes of CS551/651 Real-Time-Rendering*, University of Virginia,2002

[21] VDSlib Homepage, University of Virginia, http://vdslib.virginia.edu

[22] Karsten Hilbert, "Adaptives Level-of-Detail für die Bilderzeugung in VR-Anwendungen", *Diplomarbeit*, TU-Chemnitz, Juli 2003

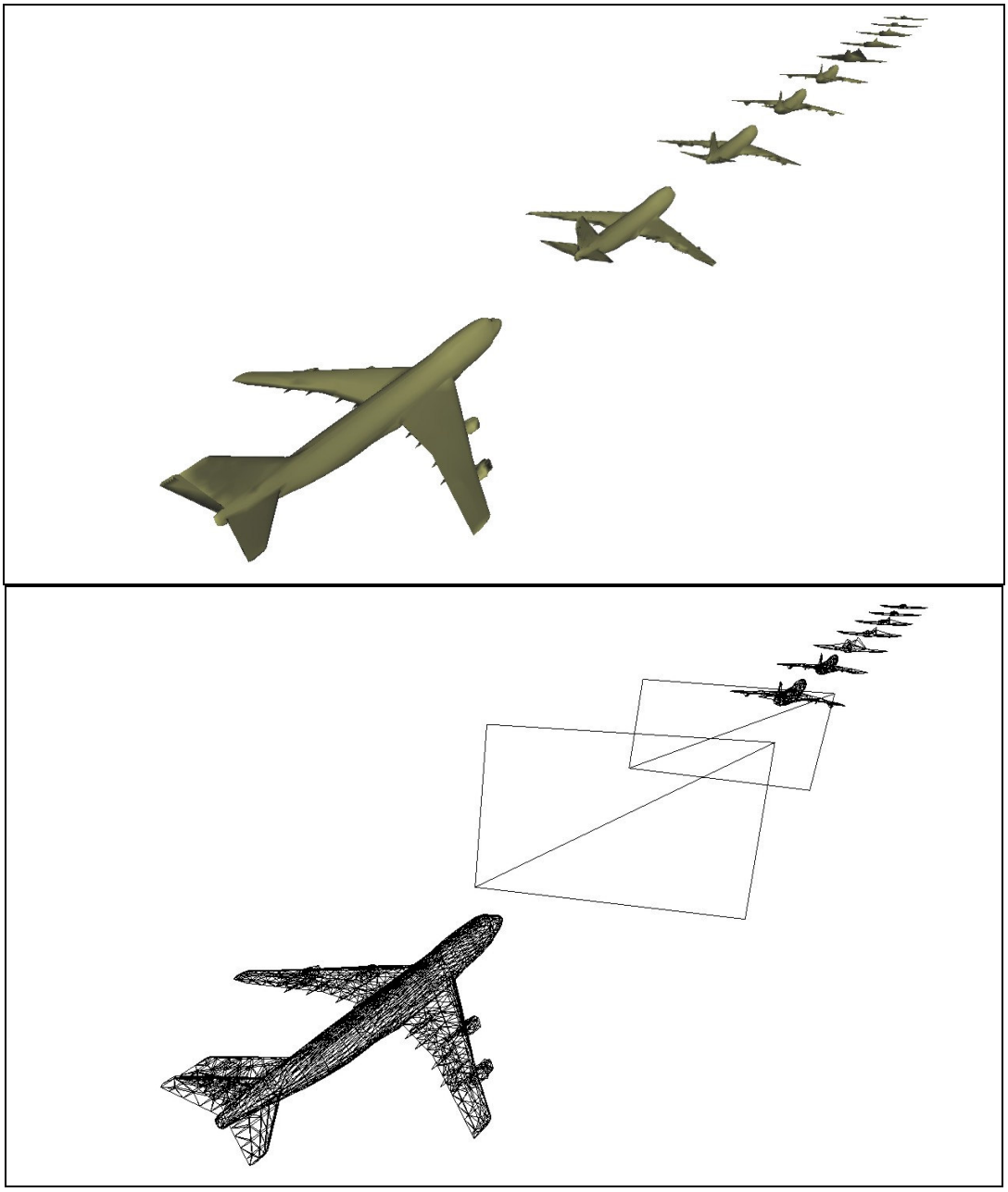

Figure 4. This figure visualizes the operating principle of our approach. For the plane in front of the viewer an adaptive geometry-based approximation is used because it is not efficient to replace its geometry-based approximation by an impostor. The geometry-based approximations of the following two planes are distant and complex enough that it is reasonable and efficient to approximate them by impostors. The complexity of the geometry-based approximations of the last seven planes is too low . So the expense for replacing them by impostors is greater than that for using the geometry-based approximations with low complexity. Notice that all approximations of the plane look very similar.

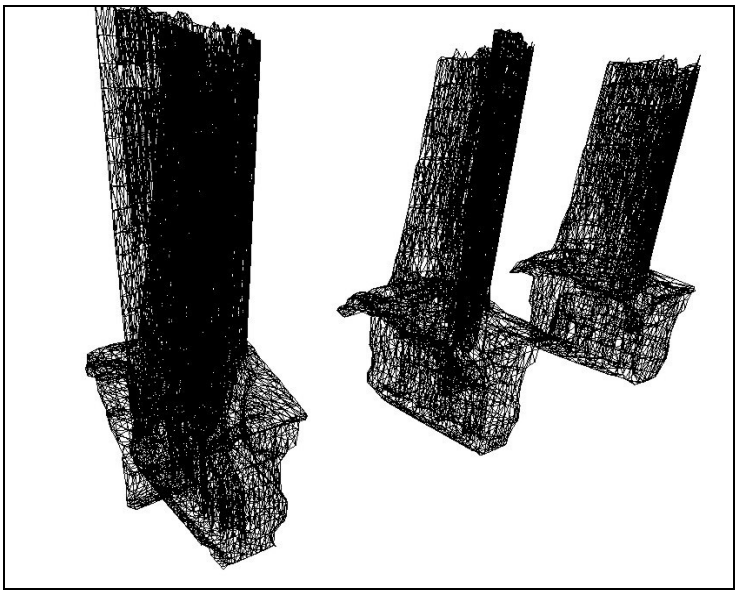

Figure 5. Scene containing three approximations of a turbine blade's model (1.765.388 polygons). A smooth interaction with the scene at high frame rates was possible using our approach.

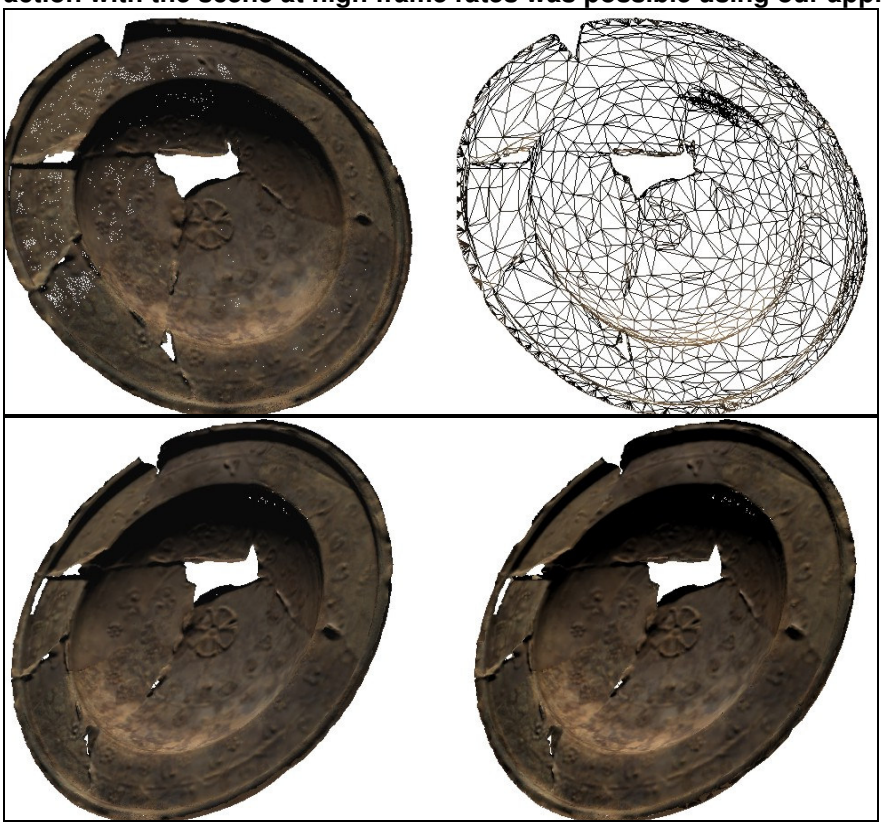

Figure 6. Top: original object (left) with 600.000 polygons and geometry-based approximation (right) with 6.000 polygons rendered in wire-frame mode, Bottom: original object (left) with 600.000 and geometry-based approximation (right) with 6.000 polygons rendered shaded. Notice that although the complexity of the approximation is very low it still looks very similar to the original object because detailed color and normal information can be obtained out of the textures. Because texture coordinates are computed at runtime no discontinuities in the textures appear.