LODESTAR: An Octree-Based Level Of Detail Generator For VRML

Dieter Schmalstieg
Vienna University of Technology, Austria
dieter@cg.tuwien.ac.at - http://www.cg.tuwien.ac.at/~dieter/

Abstract. Level of detail generation is important for managing geometric complexity of three-dimensional objects and virtual worlds. However, most algorithms that compute levels of detail do not deal with the special requirements of input data in VRML format. We report an algorithm called LODESTAR, based on octree quantization that robustly computes simplifications for objects in VRML.

Keywords: VRML, polygonal object simplification, levels of detail, octree quantization

1. INTRODUCTION AND MOTIVATION

Levels of detail (LODs) are approximations of an object with fewer geometric primitives, that are selected at run time to sustain interactive frame rates [1]. Reducing geometric complexity of geometric models is particularly important for VRML worlds for at least two good reasons:

- While 3-D rendering systems are notoriously too slow, this is particularly painful for low cost devices such as personal computers that are most frequently used for VRML applications.
- A single level of detail for a given object can be considered a form of lossy compression. Effectively, the size of the original model is reduced at the expense of quality, allowing for fast transmission and more effective distributed applications.

Many algorithms have been published on how to simplify geometric models, but they generally place assumptions on the data that hamper their application to VRML files. Many applications that create geometry can now export models in VRML format, and some of them support level of detail control. However, since VRML is most often used with Internet-based applications, in practice one is often confronted with the problem that the application that generated the model is no more available. Information on the internal structure of the model is lost, and so the task of generating levels of detail for input data that *comes* in VRML format is all but trivial. This situation is often deteriorated by export filters or translators that produce non-standard VRML files or introduce degeneracies (for example, non-planar or otherwise distorted polygons are commonly found).

2. RELATED WORK

The term „geometric primitive“ in the context of interactive rendering usually means polygons or triangles, but other geometric entities can be used as well (e.g., a bounding box of an object can suffice as a “cheap” coarse LOD). However, the algorithms that have been reported on only deal with polygonal objects. For a survey, see [5].

2.1 Surface algorithms

The methods that produce the highest quality work on the surface of polygonal objects, e.g. [9, 11, 2]. For the moment let us assume that we are only dealing with triangles. With information on which triangles are neighbors, local operations can be applied to remove triangles and fill the holes created by that process. Such algorithms can take into account local curvature and can generate simplifications with guaranteed error bounds. However, they are constrained to objects with well-connected surfaces. Unfortunately, this constraint is often not fulfilled by CAD models. Many of these algorithms are also constrained to preserve the genus of the object, and can therefore not simplify the objects beyond a model-dependent level.

2.2 Clustering algorithms

Real-world applications almost always involve ill-behaved data, and for very large scenes and slow connections, it should be possible to produce very coarse approximations as well as moderately coarse ones. More apt to this task are

LOD generation methods that ignore the topology of objects and force a reduction of the data set. The key idea here is to cluster multiple vertices of the polygonal object that are close in object space into one, and remove all triangles that degenerate or collapse in the process. The problem here is that exact control over local detail is not so easily possible, but such an algorithm can robustly deal with any type of input data, and produce arbitrarily high compression. Vertex clustering can either be done with a simple uniform quantization [6] or a binary tree [7].

3. OCTREE QUANTIZATION FOR LODS

We have investigated a level of detail generation algorithm called LODESTAR. It uses octree quantization [3] for vertex clustering. Octree quantization is superior in quality to uniform quantization and in speed to binary trees. The three-dimensional spatial structure represented by an octree allows simple clustering operations on three-dimensional samples in linear time. The method works well for colors (the three dimensions being the R, G, and B component), and has been adapted for (x, y, z) vertex coordinate tuples in this work. In the following, we outline the quantization method. We start by explaining how to create the octree data structure, and proceed with details on how to identify clusters, and how to select the representative for each identified cluster. Finally, we describe how to obtain the simplified model from the original model using the octree as an auxiliary datastructure.

3.1 Building the octree

Octree quantization was originally developed to select the entries for a color lookup table that optimally represent a given image. Instead of color pixels, we enter the vertices of the model into an octree. Intermediate nodes of the octree represent subdivisions of the object space along the x, y, and z direction. The goal is to place exactly one vertex in each subvolume. The octree is successively refined by further subdivision of leaf nodes when entering new vertices until this criterion is satisfied. Theoretically, this can generate arbitrarily deep octrees, but in practice a certain octree depth is never exceeded as the input data comes in finite precision floating point numbers.

When entering a new vertex, the octree is recursively traversed by comparing the coordinates of the new vertex against the coordinates stored in the octree node, and traversing the link to the appropriate child node until a leaf or a nil pointer is encountered. Three cases must be distinguished:

Case 1: The selected link is a nil pointer, so the corresponding subvolume is empty, and we can simply create a new leaf node and store the vertex in that node (Figure 1).

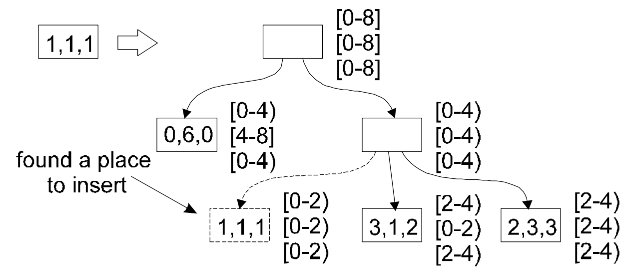

Figure 1: Inserting a vertex into an empty subvolume

Case 2: The link points to a leaf node, and the new vertex is equal to the vertex stored in the leaf: No new node is created, but the vertex counter of the existing node is simply incremented. Note that this automatically sorts out doublets in the vertices, which are a major defect of many VRML models found today, because only one copy of each vertex is finally output (Figure 2).

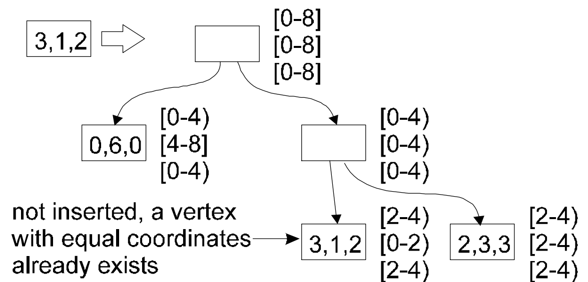

Figure 2: Inserting an already existing vertex

Case 3: The link points to a leaf node, but the new vertex is not equal to the vertex stored in the leaf: The leaf vertex and the new vertex fall into the same subvolume, so the octree must be subdivided in that location. A new intermediate node is created, and the old leaf node and a new node containing the new vertex are inserted as children of the new intermediate node (Figure 3).

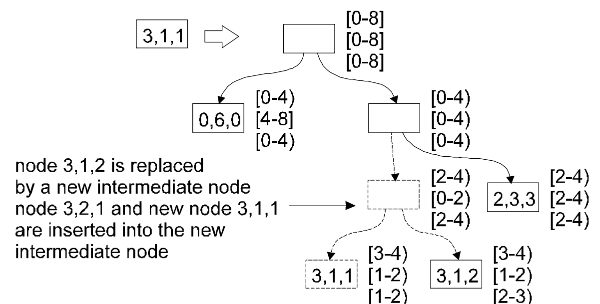

Figure 3: Inserting a vertex into an occupied subvolume

3.2 Vertex clustering

The number of vertices is reduced by combining multiple close vertices into one cluster. For such a cluster, a representative is chosen from the set of vertices represented by that cluster. This has the advantage that no new vertex must be synthesized, and the original set of vertices can be kept unchanged.

Vertex clustering is done by replacing leaf nodes that share an intermediate node as common parent with that parent, setting the vertex count of the parent to the sum of the vertex counts of its children. In selecting the cluster, the following criteria are relevant:

- From all clusters, select the one whose nodes have the largest depth within the octree, for they represent vertices that lie closest together.
- If there is more than one such cluster, additional criteria can be used for the selection according to the user's preferences: Selecting the cluster that represents the fewest vertices will keep the error sum small. Selecting the cluster that represents the most vertices will tend to generate coarser representations of finely tessellated areas with fewer vertices, but preserve small distinctive features instead. Experiments show that this latter strategy usually produces better results.

3.3 Selecting the cluster representative

The remaining problem is which strategy to use to select the representative vertex from the vertices in the cluster. To do so, we use three different heuristics with user defined weight. Let the *involved triangles* be those triangles that have at least one vertex in the cluster.

Error area is an attempt to measure the change in the extent of the object's surface: If a cluster of vertices is replaced by a representative, the areas of most involved triangles change.

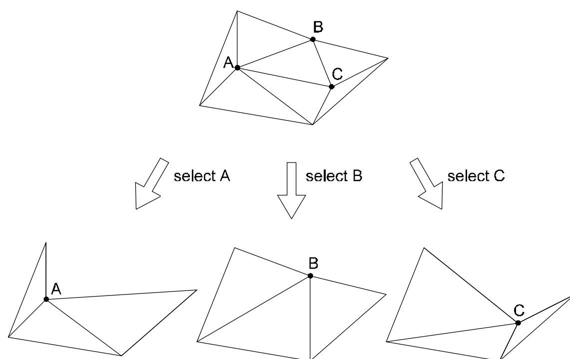

Figure 4: Different choices of the representative influence the area of the resulting triangle mesh

The error area is defined as the difference in the summed area of the involved polygons before and after the clustering. The vertex that produces the smallest error area is chosen.

Error volume is an attempt to measure the object's change in volume: For every involved triangle, we construct the tetrahedron from the three original vertices and the potential representative. The volume of such a tetrahedron is zero if one of the vertices is elected the representative. The summed volume of all such tetrahedrons is taken as the error volume, and the vertex with the smallest error volume is elected. One disadvantage of this approach is that all volumes are zero if the vertices lie in a plane, so it is only useful in combination with another heuristic.

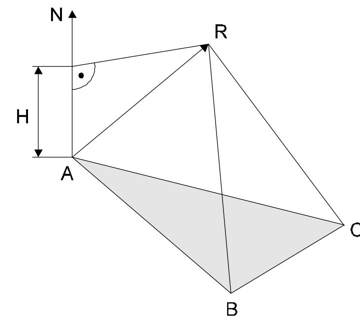

Figure 5: The error volume is computed from the tetrahedron with the original triangle ABC as a base and the chosen representative R as the top

Weighted mean is an attempt to find the vertex that "best" represents the other vertices: An average vertex is synthesized from the cluster as a weighted mean, where the weights are the vertex counts of the nodes in the cluster (remember that leaf nodes represent a single vertex, intermediate nodes represent all leaves in their subtree). The vertex that is closest to the mean is chosen. Unfortunately, this does not take into account any surface properties, and our experiments show that results using this heuristic are visually not as appealing as the other two heuristics. Weighted mean was kept for the sole purpose of handling indexed line sets (see below).

3.4 Computing the reduced triangle set

After the number of vertices has been reduced by the desired amount, the set of triangles associated with the reduced vertex set must be reconstructed. For every triangle, its vertices are replaced by the representative chosen for that vertex. This process may produce doublets (triangles with identical vertices) for which only one instance is kept. Triangles may also collapse into lines, most of which are identical to the edge of another triangle and can be discarded. The remaining lines are usually

important for the appearance of the model and are thus saved. Sometimes triangles collapse into points, which are removed from the model.

4. DEALING WITH VRML SPECIFICS

Up to now we have silently assumed that the geometric model consists of an unstructured set of triangles, and we have neglected in the discussion a variety of properties specific to VRML models.

4.1 Non-polygonal nodes

VRML models do not only consist of triangles or polygons, but also of other primitives like spheres or text, and of context-defining nodes such as transformations. However, the essential structure of VRML scenes is the IndexedFaceSet and its helper nodes Coordinate3, Normal, TextureCoordinate2, and Material. Large amounts of geometric primitives are almost exclusively specified using IndexedFaceSets, and therefore it is reasonable to concentrate on this node for level of detail generation. Level of detail generation dealing with VRML geometry other than IndexedFaceSets may become an interesting area for future research, but this is beyond the scope of this paper.

4.2 Scene graph structure and output format

VRML models and scenes are not "flat", but are rather arranged in a hierarchical scene graph, so an algorithm dealing with a single set of polygons is not sufficient. The simple yet effective solution that was used in LODESTAR is to traverse the VRML model and apply the LOD generation to every IndexedFaceSet individually, producing for each a new LOD node (details on how to deal with multiple IndexedFaceSets are given in section 5).

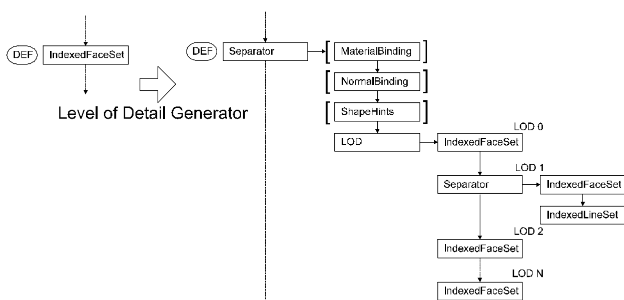

Figure 6: A single indexed face set is converted into a subtree with a single LOD node

LODESTAR replaces every IndexedFaceSet in the original file with a subtree containing the computed LODs. This subtree contains a single LOD node. If the structure of the file requires that additional nodes (such as bindings) are output, the whole structure is wrapped in an additional Separator. The children of the LOD node are the IndexedFaceSets containing the computed levels of detail.

If any triangles collapsed to lines are produced as a result of the clustering process, an additional IndexedLineSet is generated to complement the IndexedFaceSet, and the resulting structure is wrapped in a Separator.

4.3 Triangulation

As already mentioned, most level of detail algorithms including LODESTAR can only deal with triangles as an input. The triangulation is necessary because after a vertex clustering operation, any n-sided triangle (with n>3) almost certainly becomes non-planar. Therefore all n-sided polygons are triangulated first by using the algorithm from [4]. As a side effect, all concave polygons are removed from the model, which allows the use of algorithms that are simpler, more robust and faster.

The exception are quadrilaterals, for which the error is often small and hence tolerable (i.e. non-visible). It is necessary, though, to check any quadrilaterals for validity after a modification of its vertices. Concave quadrilaterals or quadrilaterals which are distorted in space more than a user-specified threshold (measured as the maximum angle between the normals at the vertices) are split into two triangles (Figure 7).

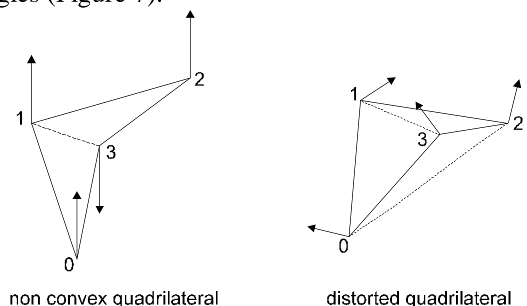

Figure 7: Degenerated quadrilaterals must be split

Triangulation increases the number of polygons and can involve a performance penalty. However, most 3-D rendering engines triangulate all geometry internally [10], so with the use of triangle strips, a performance penalty can be avoided. This consideration of course assumes that the renderer detects and uses triangle strips, which unfortunately cannot be influenced from within a VRML file.

4.4 Lines

IndexedLineSets can be treated almost like IndexedFaceSets: Vertices are clustered with octree quantization, and a new IndexedLineSet is constructed from the reduced vertex set for every level of detail. The output is equivalent to the structure depicted in Figure 6, except that IndexedFaceSets are replaced by IndexedLineSets. However, for IndexedLineSets the only applicable heuristic for representative selection is weighted mean.

4.5 Bindings

Non-indexed bindings impose a one-to-one relationship between entries in the IndexedFaceSet fields and the corresponding helper nodes. They cannot be maintained if multiple levels of detail are to share the same materials, normals etc. Therefore non-indexed bindings are transformed into the corresponding indexed bindings, and an index will be synthesized. In this case, an additional MaterialBinding or NormalBinding is generated.

4.6 Range values

The selection of a LOD in VRML is performed by comparing ranges. A viewer switches to the next LOD if the distance of the object to the viewpoint is greater than or equal to a specified value. For satisfactory performance, the LOD generator has to compute reasonable range values.

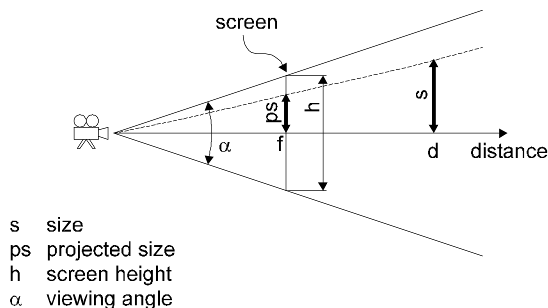

Figure 8: A heuristic is used to compute the range values required for the LOD node

The next level of detail is computed by moving vertices from a cluster to a selected representative. The maximum visible error introduced by this operation is equal to the maximum distance a vertex can move in screen space due to a clustering operation (deviation). The goal is to compute LOD switching ranges in such a way that this maximum visible error does not exceed a user defined threshold, that is specified as a percentage of the screen height.

The viewer must switch LODs if the maximum deviation *s* projected onto the screen is greater than error range specified as a fraction of the height of the screen. Let *rootsize* be the extent of the cube associated with the octree root (note that the actual size is computed from the local coordinates in the octree modified by the current scale factor from preceding Scale or Transform nodes!) and *depth* be the level of the octree corresponding to level of detail being computed:

$$s = rootsize \times \sqrt{3} / 2^{depth-1}$$

The height of the screen *h* is computed from the camera height angle *alpha* and the focal length *f*:

$$h = 2 \times \tan(\alpha/2) \times f$$

Given the desired *errorrange* (in percent), we can compute the maximum projected deferral *ps* as:

$$ps = errorrange/100 \times h$$

Finally, from the relation $d/s = 1/ps$, we can compute the range *d* as

$$d = s / (errorrange/100 \times 2 \times \tan(\alpha/2))$$

To take into account the extent of the cluster, for the actual range one has to add the radius of the bounding sphere of the cluster to *d*.

5. JOINING NODES

Often VRML files are produced with primitive converters that generate many IndexedFaceSets in sequence, each containing very few polygons.

Computing levels of details for every IndexedFaceSet of such a model has a tendency of ripping apart the model and produces useless LODs (see Figure 9), a problem also reported by [5].

Fortunately, most of these degeneracies can be cured with a very simple algorithm that joins sequential IndexedFaceSets into one if possible. This algorithm does not even require knowledge of the involved geometry but can operate in purely syntactical way on the VRML file. It does not work in every case (this would require a deep analysis of both model structure and geometry), but it cures most of the degeneracies that we have encountered so far, and even more importantly, it works very fast.

5.1 Basic joining algorithm

For the components of a boundary representation (IndexedFaceSet, IndexedLineSet Material, Normal, TextureCoordinate2, Coordinate3) and Separators, Groups and Bindings, subsequent nodes of the same type are joined unless the second is tagged with DEF.

In case of multiple Separator or Group nodes, the sub-groups can be joined. In this process, the components of a boundary representation that span multiple sub-groups are joined into one node of that type, so a single IndexedFaceSet can be synthesized.

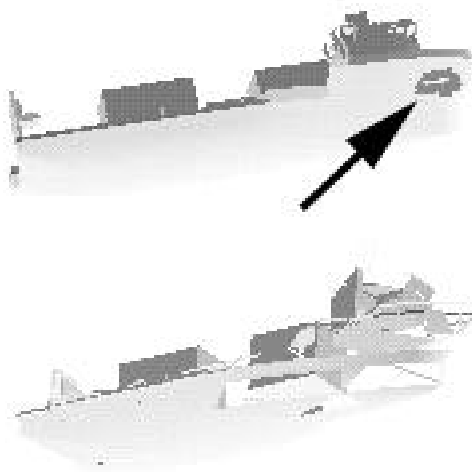

Figure 9: The hull of the ship model was represented by many small IndexedFaceSets (one shown in upper image). Holes can be suppressed by joining nodes.

Example 1

```
Separator {
  Material { diffuse 0.5 0.6 0.4 }
  IndexedFaceSet { coordIndex
                   [1,2,3,-1] }
}
Separator {
  Material { diffuse 0.3 0.3 0.3 }
  IndexedFaceSet { coordIndex
                   [4,5,6,-1] }
}
```

becomes

```
Separator {
  Material {
    diffuseColor [0.5 0.6 0.4,
                  0.3 0.3 0.3]
  }
  MaterialBinding { value
                  PER_FACE_INDEXED }
  IndexedFaceSet {
      coordIndex [1,2,3,-1,4,5,6,-1]
      materialIndex [0,1]
  }
}
```

Joining two Separator nodes with different Material sub nodes and possibly different IndexedFaceSet sub nodes. Note that it is necessary to insert a new MaterialBinding so that the synthesized Material node is put in correct relation to the synthesized IndexedFaceSet.

Example 2

```
Separator {
  Coordinate3 { point
   [ 10 11 12, 13 14 15, 16 17 18] }
  IndexedFaceSet { coordIndex
                   [0,1,2,-1] }
}
Separator {
Coordinate3 { point [ 20 21 22,
             23 24 25, 26 27 28] }
IndexedFaceSet { coordIndex
                 [0,1,2,-1]}
}
```

becomes

```
Separator {
  Coordinate3 { point
    [10 11 12, 13 14 15, 16 17 18,
    20 21 22, 23 24 25, 26 27 28] }
  IndexedFaceSet { coordIndex
              [ 0,1,2,-1,3,4,5,-1] }
}
```

Joining two Separator nodes with possibly different Coordinate3 sub nodes and possibly different IndexedFaceSet sub nodes: The algorithm also works with Coordinate3, Normal and TextureCoordinate2 nodes:

5.2 Trailing Separators

The joining algorithms works by traversing the scene graph bottom-up from the leaves, so that joinability can be propagated upwards. To improve chances of joinability, trailing Separators are removed (a Separator node on the end of a list is not necessary).

```
Separator {
  Separator { IndexedFaceSet {
            coordIndex [0,1,2,-1] }
  }
}
```

becomes

```
Separator {
  IndexedFaceSet { coordIndex
                   [0,1,2,-1] }
}
```

Limitations. The joining algorithm is a heuristic that was developed after studying the kind of degeneracies that are commonly found. It only works for relatively simple cases involving direct relations between the components of an IndexedFaceSet. Care must be taken that no other node such as a Transform is present that forbids the joining.

6. IMPLEMENTATION

The algorithm described in this paper has been implemented under C++ and ported to a variety of platforms, including multiple flavors of Unix, OS/2, and DOS. Because of the design decisions outlined earlier, it runs very fast. It works reasonably robust on input files that do not exactly comply to the VRML specification (such as some Inventor files). Furthermore, the software can be used as a „cleanup“ filter for VRML files: As explained in section 3.1, the process removed doublets in the vertices, colors etc., so invoking the program with the „no levels of detail“ option cleans up redundant models. See the appendix for some sample results.

7. CONCLUSIONS AND FUTURE WORK

We have presented an algorithm that produces levels of detail for polygonal objects. This algorithms reads and writes VRML files and takes care of the particular needs of the hierarchical structure and other advanced features of this format.

Our current implementation was designed to work with the VRML 1.0 specification. Naturally, we intend to modify the implementation to to work with VRML 2.0 files, which will mainly involve adapting the parser.

The LODESTAR software is a partial result of a larger research project aimed at developing better methods to control the bandwidth requirements of large distributed virtual environments. It is used to create the levels of detail required by the *demand-driven geometry transmission protocol* [8] for client-server based virtual environments. To further increase network performance and avoid waste of bandwidth, we have also developed a custom compression method for VRML.

Acknowledgments. Many thanks to Reinhard Sainitzer and Herbert Buchegger who worked on the implementation of this algorithm.

Resources. More detailed results of LODESTAR and binaries for most popular platforms can be obtained free of charge for non-profit use at

http://www.cg.tuwien.ac.at/research/vr/lodestar/

References

[1] J. Clark: Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, Vol. 19, No. 10, pp. 547-554 (1976)

[2] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, W. Stuetzle: Multiresolution Analysis of Arbitrary Meshes. *Proceedings of SIGGRAPH'95*, pp. 173-182 (1995)

[3] M. Gervautz, W. Purgathofer: A simple method for color quantization: octree quantization. New Trends in Computer Graphics, *Proceedings of Computer Graphics International'88*, p. 219-231, Springer, Geneva. Reprinted in: *Graphics Gems I*, pp. 287-293 (1990)

[4] A. Narkhede, D. Manocha: Fast Polygon Triangulation Based On Seidel's Algorithm. *Graphics Gems V*, Academic Press, pp. 394-397 (1995)

[5] J.-L. Pajon, Y. Collenot, X. Lhomme, N. Tsingos, F. Sillion, P. Guiloteau, P. Vuyslteker, G. Grillon, D. David: Building and Exploiting Levels of Detail: An Overview and Some VRML Experiments. *Proceedings of VRML'95*, San Diego CA, pp. 117-122 (1995)

[6] Jarek Rossignac, Paul Borrel: Multi-Resolution 3D Approximation for Rendering Complex Scenes. *IFIP TC 5.WG 5.10 II Conference on Geometric Modeling in Computer Graphics* (1993)

[7] G. Schaufler, W. Stürzlinger: Generating Multiple Levels of Detail from Polygonal Geometry Models. *Virtual Environments'95*, Springer (1995)

[8] D. Schmalstieg, M. Gervautz: Demand-Driven Geometry Transmission for Distributed Virtual Environments. *Computer Graphics Forum (Proc. EUROGRAPHICS '96)*, Vol. 15, No. 3, pp. 421-433 (1996)

[9] W. Schroeder, J. Zarge, W. Lorensen: Decimation of Triangle Meshes. *Proceedings of SIGGRAPH'92*, pp. 65-70 (1992)

[10] SGI: OpenGL Programming Guide, Addison-Wesley (1993)

[11] G. Turk: Re-Tiling Polygon Surfaces. *Proceedings of SIGGRAPH'92*, pp. 55-64 (1992)

Appendix: Results

The LODESTAR code was tested with a large number of models downloaded via the Internet. Here we present a few quantitative results and images to give an impression of the performance of the implementation. The „Enterprise“ model (10 LODs) was computed in 5.4 seconds and the „Galleon“ model (8 LODs) was computed in 7.8 seconds on an SGI Indy R4400/150 workstation.

Table 1: „Enterprise“ model statistics

LOD	Tri's	Range	LOD	Tri's	Range
0	6343	36	5	1083	193
1	5020	42	6	553	355
2	3999	52	7	167	679
3	3182	72	8	51	1325
4	1960	112	9	16	2615

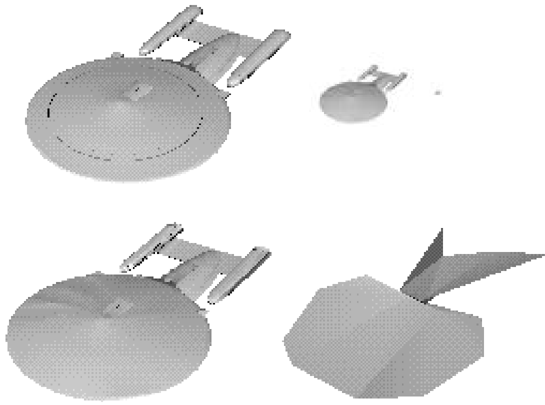

Figure 10: Three LODs of the enterprise mode (LOD # 0, 4, 8). If displayed in a size corresponding to the computed ranges, the quality degradation is no longer visible

Table 2: „Galleon“ model statistics

LOD	Tri's	Line	Range	LOD	Tri's	Line	Range
0	4698	0	1962	4	1478	8	12505
1	4142	0	2664	5	1478	8	12505
2	3686	0	4070	6	108	4	46122
3	2981	9	6882	7	24	0	90932

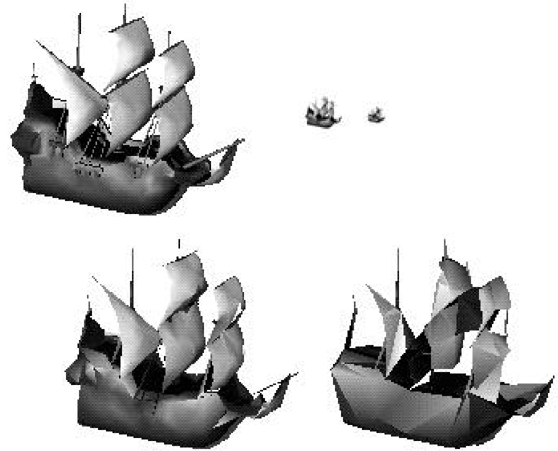

Figure 11: Three LODs from the galleon model (LOD # 0, 4, 5)