Incremental Encoding of Polygonal Models

Dieter Schmalstieg
Vienna University of Technology, Austria
dieter@cg.tuwien.ac.at

Gernot Schaufler
Kepler University Linz, Austria
gs@gup.uni-linz.ac.at

Abstract

This paper presents a method to generate a stream of data which continuously refines a polygonal model from the coarsest representation to the original high-fidelity model. Unlike conventional level of detail algorithms that use only a small number of discrete approximations for each object, our method allows a smooth, continuous refinement of the object over time while the stream is transmitted. The approach is based on a hierarchical clustering algorithm which produces the model representation of steadily increasing detail and has several advantages over conventional models with few, discrete levels of detail: The model can be transmitted and decoded incrementally, yielding a valid approximation at any stage of the process. The incremental encoding is extremely compact, so models are smaller than in their original form. Any desired level of detail can be selected during rendering at interactive speed. It is also possible to have variable detail resolution within a single model, which is useful for rendering models with large extent.

1. Introduction

In real-time computer graphics and virtual environments polygonal models are most commonly used to describe the objects in the environment because of available hardware support for rendering. Despite recent performance advances of hardware accelerators the model complexity frequently exceeds hardware capabilities, resulting in degraded system performance often at no gain in image quality because the small detail in the models is not captured by rasterized images.

Overload of the graphics hardware and object detail which will not be visible in the final image is commonly avoided through the use of so-called levels of detail (LOD): models are approximated with decreasing accuracy using a decreasing number of triangles. Numerous methods to generate a small set of such levels of detail have been developed, that all aim at a high data reduction without noticeable degradation of the resulting image quality. Coarser levels of detail should only be used for small or distant objects, so that the difference in image quality cannot be noticed by the observer, but frequently models are to complex for the available rendering capacity, so that a coarser than desired approximations has to be drawn to prevent the image generator from stalling. In such cases, switching from one level of detail to another in an is particularly distracting and annoying for the user.

Use of 3-D models in networked computer systems is growing fast as Internet-based graphics standards such as VRML and distributed virtual environments are being adopted by large user communities. In this context, transmission of object models is a major issue as soon as the simulated environment is complex enough to make storing full copies of the environment on every computer impractical. LODs with progressively higher detail will be transmitted as the participant is approaching an object. However, only the last completely transmitted level can be displayed, and as data sizes increase with the LOD quality, delays between model refinements increase rapidly. Such stalling negatively affects the participant's experience of the simulation.

This paper proposes a solution for the mentioned problems. We propose the generation of a representation for an object which can be displayed with continuous level of detail. The transmission of this object representation over computer networks makes the latest transmitted data immediately available for display during the simulation.

After reviewing related work in section 2, the algorithm is motivated in section 3. Section 4 introduces the hierarchical cluster tree used to represent the smooth LODs. Manipulation of the cluster tree is explained in section 5. Section 6 defines a suitable network protocol, section 7 deals with object reconstruction and presents methods for display. Section 8 presents results and section 9 draws conclusions.

2. Related Work

The gains in rendering speed achievable by approximating small, distant or otherwise unimportant objects instead of rendering them in full quality were identified by Clark as early as 1976 [1]. Since then, researchers have investigated several methods for the automated generation of so-called levels of detail (LODs); a comprehensive overview can be found in the paper by Heckbert and Garland [4]. The task of LOD generation so far has been understood as a search for a series of progressive simplifications of a polygonal object, that have fewer primitives (polygons), but closely resemble the original object.

Polygonal simplification can be categorized by the property that is examined to compute the reduced data set: Topological algorithms such as. [3, 5, 11, 12] allow relatively precise control over the result, including preservation of the topology of the object and error bounds on the quality of the approximation. Unfortunately, implementation of these algorithms is often highly complex, and generally requires objects to be topologically well-behaved and manifold, a requirement rarely met by real-world CAD data.

Geometric algorithms, however, are less sensible to the input data and easier to implement. Since they are not constrained to preserve any topological properties of the data, they can be forced to achieve arbitrarily high data reduction (at the expense of object fidelity). The LOD generation algorithm on which this paper is based belongs to the latter class.

The fundamental idea of data reduction based on geometry is to reduce the number of vertices of a polygonal model (usually a triangle mesh). Due to perspective distortion individual vertices of an object move closer together on the screen as the distance to the observer increases until they finally fall into one pixel. By creating a cluster of such close vertices and replacing all cluster members by a representative vertex, the number of vertices is reduced. The set of triangles is modified to include only the vertices in the new set. In the course of that process, triangles will degenerate to lines or points and can be removed. Therefore the set of triangles is reduced, and any such intermediate data set can be used as an individual LOD.

Several selection criteria have been presented to choose the vertices that are to be clustered. Rossignac and Borrel [8] propose a simple, yet efficient uniform quantization in 3-D. Schaufler and Stürzlinger [9] use a hierarchical clustering method, on which the work presented in this paper is based. Luebke [7] presents a data structure derived from an octree, that is capable of computing any desired level of detail during display, which incrementally adapts to changes in the viewpoint.

As networked virtual environments and other distributed graphical applications involving more than one computer and display system become increasingly commonplace, it is no longer sufficient to provide LODs for better utilization of the rendering pipeline only. Transmission of the geometric database over the network becomes the bottleneck as large data sets introduce intolerable lag. Schmalstieg and Gervautz [10] discuss a strategy for network distribution of individual LODs on demand that reduces bandwidth requirements. Deering [2] introduces a compression method for polygonal data sets. Levoy [6] proposes a combination of geometry and compressed image data to preserve bandwidth with a compressed video stream. The wavelet based multi-resolution modeling presented by Eck et al. [3] lends itself to progressive refinement of objects, similar in spirit to the approach presented here.

3. Motivation for Smooth Levels of Detail

Frequently polygonal models are very large, which is at conflict with rendering capacity and network throughput. Adding levels of detail partly addresses the rendering problem, but makes overall model size even larger. The reason for this dilemma is that the standard approach of representing polygonal data as lists of vertices and triangles is not powerful enough. In this paper, we present a data structure that does not suffer from the mentioned shortcomings and fulfills the following requirements:

- **Smooth LODs.** The data structure should represent many levels of details (not only 3-6, but hundreds or thousands of LODs), so that a continuous (or almost continuous) refinement of the model is possible by repeatedly adding small amounts of local detail to the model.
- **Incremental decoding.** Decoding of the smooth LODs should be incremental, i. e. the next finer LOD should be represented as the difference to the current LOD. By reusing all the data from the coarser LODs, model size can be kept small despite the large number of LODs.
- **Interactive LOD selection.** The smooth LODs data structure should support selection and rendering of any specific LOD, allowing a chance to the next LOD (both coarser and finer) at interactive speeds (during rendering).
- **Incremental transmission.** It should be possible to incrementally transmit the model over the network, starting from the coarsest approximation and progressing to the original model. In particular, rendering should immediately be able to use all the data received up to a certain moment, and render an incomplete. This is important for progressive refinement of large models that take an extended period to transmit, and allows continued operation in case of network failures.
- **Compact representation.** It is preferred if the smooth LODs data structure introduces no overhead in the model size compared to the original, uncompressed polygonal model. Ideally,

the introduction of smooth LODs should yield compression instead of bloating the model.

- **Variable resolution within the model.** If the many LODs within the data structure have only local influence on the appearance of the model, the corresponding details can be selected individually, resulting in variable resolution within a single model. This is particularly useful for models with a large extent (e.g. a large building seen from the inside), where parts close to the observer should have high fidelity, whereas distant parts can be represented by a coarse approximation.

Our data structure is based on a binary tree that is created by hierarchically clustering vertices of the original model, thereby constructing a *cluster tree*. Every clustering operation simplifies the model, and therefore every node of the cluster tree represents a single level of detail. A linearization of the tree in the inverse order of the clustering process yields a sequential representation of the data structure that is suitable for network transmission. It also incrementally encodes the model, and therefore fulfills our requirements. The next section discusses the creation of this data structure in detail.

4. Representing the model as a cluster tree

Hierarchical clustering for LOD generation, as presented in [9], is based on the idea that groups of vertices which project onto a sufficiently small area in the image can be replaced by a single representative: a many-to-one mapping of vertices. As a consequence, the number of triangles is reduced when the triangles' vertices are replaced by their representatives from the reduced vertex set, and collapsed triangles are filtered out. Repeated application of the clustering operation yields a sequence of progressive simplifications (LODs). If exactly two clusters are combined in every step, the result is a binary tree, the *cluster tree*.

Construction of the cluster tree. The cluster tree is built by successively finding the two closest cluster in the model and combining them into one. The combined cluster is stored in a new node that has the two joined clusters as its children. The process is repeated until only one cluster containing all the vertices remains, which is the root of the cluster tree.

For each new cluster, a *representative* is chosen from the set of vertices in the cluster. More precisely, we chose the representative to be one of the two representatives of the child clusters. The distance of two clusters (used to find the closest clusters) is computed as the Euclidean distance of the two childrens' representatives. This value is also stored as the *cluster size* in the new cluster's node for further use.

Step 0 (Initialization): Form a cluster for each vertex, with the vertex serving as the representative

Step 1: Find the two clusters with the closest representatives

Step 2: Replace the two clusters identified in step 1 by a joint cluster, select new representative

Step 3: If more than one cluster remains, go to step 1

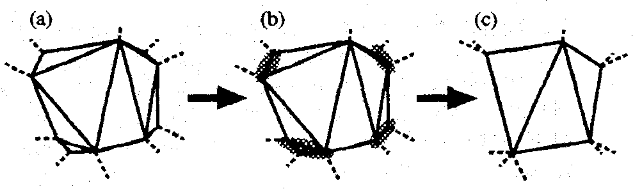


Figure 1: The clustering process: A mesh (a) is mapped onto a vertex cluster tree, which is used to group vertices (b). From the reduced vertex set, a simplified model (c) is computed.

The cluster tree has the desired properties that it contains instructions for a continuous simplification of the model, and can therefore be used to construct a sequence of smooth levels of detail. However, in its form described above, it only stores the vertices of the model, but not the triangles. To use the cluster tree as an alternate representation of the original polygonal model, the triangles must also be encoded and stored in the cluster tree in a way so that the original model (or any desired level of detail) can be reconstructed from the cluster tree alone.

This is done by recording the events (changes) in the triangle database when two clusters are joined (and consequently one representative vertex is eliminated). The inverse operation of these events can be used to reconstruct the triangle database by reconstructing the cluster tree node by node. If the events are appropriately recorded, the smooth LODs can be generated by a simple traversal of the cluster tree in the inverse order of the clustering process with appropriate output.

Triangle event recording during clustering. When the clustering stage combines two clusters into one, those triangles which have at least one vertex in the new cluster must be changed accordingly. For each such triangle, one of three events can happen:

1. The triangle has one vertex in the new cluster, and this vertex is elected the new cluster representative. Therefore, no change is made to the triangle at all, and the event need not be recorded.
2. The triangle has one vertex in the new cluster, but this vertex is *not* elected the new cluster representative. This vertex must be changed to the new cluster representative. An *update list* of all such triangles is kept in the cluster node (Figure 3a).
3. The triangle has two vertices in the new cluster. Therefore it collapses to a line which is discarded

from the triangle set. A list (the *collapsed list*) of all collapsed triangles is kept in the cluster node (Figure 3b).

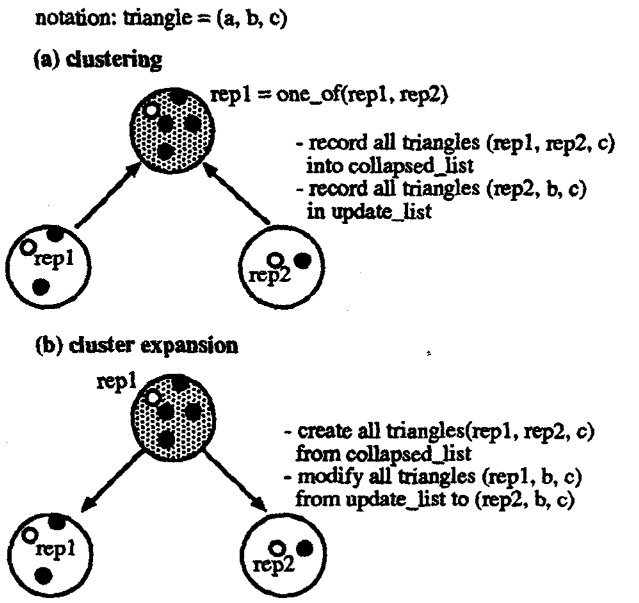

notation: triangle = (a, b, c)
(a) clustering
rep1 = one_of(rep1, rep2)
- record all triangles (rep1, rep2, c) into collapsed_list
- record all triangles (rep2, b, c) in update_list
rep1
rep2
(b) cluster expansion
rep1
- create all triangles(rep1, rep2, c) from collapsed_list
- modify all triangles (rep1, b, c) from update_list to (rep2, b, c)
rep1
rep2

Figure 2: During the clustering, two vertex clusters are joined into one, and the effect on the triangles are recorded (a). The inverse operation, cluster expansion, uses the recorded data to reconstruct the triangles (b).

The lists kept for events of type 2 and 3 make it efficient to perform the construction of the new triangle list for each generated level of detail.

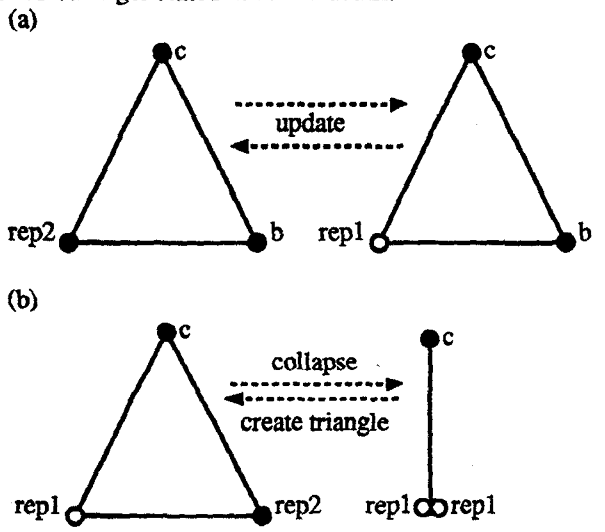


Figure 3: Two events during clustering are of interest for the reconstruction of the original triangles: Collapsing triangles (a), and triangles whose vertices are updated (b).

Stepping from one LOD to the next is done by adding only one vertex (adding one cluster). The involved changes are small, so coherence between LODs is exploited by caching changes in the update list and collapsed list at each node.

A cluster tree containing the cluster representatives and the information on triangles (update list and collapsed list) completely encodes the information contained in the original model, plus instructions how to create all intermediate levels of detail. In the next section, we describe basic operations on the cluster tree.

5. Manipulation of the cluster tree

While the cluster tree has the desired property that is efficiently represents the original model plus all its levels of detail, it is not directly usable. For rendering, it is still necessary to reconstruct a vertex list and triangle list (either for the original model or for a level of detail). A tree is also not suitable for network transmission, it must be linearized first. Furthermore, a simple method for selecting an arbitrary level of detail is required. Therefore, we define a number of basic operations on the cluster tree, from which the required functions (model reconstruction, linearization, LOD selection) can easily be constructed.

Traversal of the cluster tree. During the hierarchical clustering process, the nodes of the cluster tree were generated in the order of increasing cluster size. Traversal of the cluster tree is done in the exact inverse order of the creation. A set of active nodes is maintained to reflect the current status of the traversal. Starting with the root of the cluster tree, the algorithm processes the cluster tree node by node, in the order of increasing cluster size. Every visited interior node is replaced by its two children. The following pseudocode sketches the algorithm:

```
activeNodeSet = root
while not empty(activeNodeSet)
  current = get node from activeNodeSet
            with biggest cluster size
  if(not isLeaf(current->left))
    add current->left to activeNodeSet
  if(not isLeaf(current->right))
    add current->right to activeNodeSet
endwhile
```

Reconstruction of the polygonal model. The original polygonal model, consisting of a vertex list and triangle list, can be reconstructed using the cluster tree traversal function. The root introduced the first vertex. With every visited node, one new vertex is introduced and added to the vertex list (the other child inherits the parent's representative). Concurrently, the triangle list is reconstructed by processing each visited node's collapsed list and update list. Every entry in the collapsed list introduces a new triangle into the triangle list (in the inverse process, this triangle was collapsed and removed). Every triangle in the update list contains the parent cluster's representative, which must be replaced by the new vertex mentioned above. When all nodes have been visited by the traversal, the original model is completely reconstructed.

Selection of a LOD. The original model is only the most detailed version of a large number of LOD approximations. A convenient way to select any desired LOD from the available range is required. Therefore the traversal process is modified to terminate when all nodes belonging to a particular LOD have been visited. The desired LOD is specified as a threshold that is compared to the cluster size contained in every node. The modified traversal algorithm no longer continues until the active node set is empty, but terminates if biggest cluster size from any node in the active node set is smaller than the given threshold. The reconstructed triangle list and vertex list up to that point represent the desired level of detail and can directly be used for rendering.

Refinement. For refinement of the model, the fundamental operation is to switch from a given level of detail to the next finder one. A particular LOD is defined by an active node list of the cluster tree, and the corresponding vertex list and triangle list. This is achieved by unfolding the node which is selected for refinement into its two successors, and using the information contained in that node to extend the triangle list and vertex list. This is an incremental operation that typically requires only a small amount of processing and can be carried out at interactive speeds. Selection of a LOD as previously mentioned is nothing else than the repeated application of refinement, starting with an initially empty vertex list and triangle list.

Simplification. The inverse operation to refinement is simplification, which is used to switch from a given level of detail to the next coarser one. Two nodes are folded into their common parent node. One vertex is removed from the vertex list, and references to that node in the triangle list are updated. Collapsed triangles are filtered out, thereby simplifying the model.

Linearization. The traversal can not only be used to reconstruct the model for rendering, but also to generate a sequential version of the cluster tree suitable for network transmission. Nodes are visited in the same order as for LOD selection, but instead of reconstructing the original model, the information contained in the node is directly output into a sequential data stream. During that process, triangles and vertices are automatically renumbered in the order in which they are visited, so that references always point back to available valid indices and incremental decoding becomes possible.

6. Transmission Protocol

Using the linearization operation introduced in the last section, it is very simple to create the stream of packets required for network transmission. No redundant information is stored in the network packages, so the requirement of compactness is satisfied by the network protocol, which actually represents the smooth LODs model in less bytes than the original model (see section 8 for results).

Our protocol currently consists two types of packets, one introducing a new vertex and encoding the updates on the existing triangles necessary for that node. The other packet type introduces a new triangle and is repeatedly used for every new triangle introduced with a new vertex. Such a unit of one new vertex packet and multiple new triangle packets incrementally encodes one smooth level of detail.

1. **new_vertex**(parent_node, x, y, z, update_list): A new vertex is introduced. One node of the cluster tree is replaced by its two children. The coordinates of the representative of one of the new clusters are encoded in this package. The parent_node field is an integer that identifies the cluster that is being split in two. The (x,y,z) tuple gives the coordinates of the new vertex in absolute single precision numbers (32 bit per value).

   new_vertex also encodes the update list associated with the parent node. Already encoded triangles which contain the parent cluster's representative can either continue to use that representative or from now on use the new vertex. This information must be encoded to be able to update the triangles correctly. The update is simply the replacement of the parent clusters representative with the new vertex within the triangle. One bit is sufficient to indicate for each candidate triangle containing the parent cluster's representative whether the update should take place or not. These bits are compactly stored as a bit list.

   A variable length bit list is used to encode these updates. Since the number of candidate triangles as well as the order of the triangles given by their position in the global triangle list is known to both sender and receiver. the update process is well defined.

2. **new_triangle**(vertex, orientation): As the reconstruction of the object from the network data steam is the inverse operation of the clustering stage, for every new vertex encoded by new_vertex, the triangles stored in the parent node's collapsed list must be re-introduced as *new* triangles. This is done by a sequence of new_triangle packets. The triangle in question collapsed because new vertex and the parent's representative were clustered, so

two of the original vertices are already known. The missing third vertex is encoded in the packet as a 16-bit index into the array of vertices. The new triangle has either the orientation (new_vertex, parent_rep, vertex) or (parent_rep, new_vertex, vertex), distinguished by the orientation bit.

The packets are headed by a one-bit tag indicating whether a new_vertex or another new_triangle packet is to follow. As typically only a few new triangles are introduced with each new vertex, the overhead associated with the tag is small. The use of tags will also make it easier to extend the protocol. Currently 16-bit indices for vertices and cluster nodes are used, which is sufficient for the test cases given below. The size of the indices might need to be increased for larger models.

Table 1 summarizes the packets including their parameters (field sizes in bit given in parenthesis).

Table 1: Protocol packets with parameters & sizes

Packet Id	Parameters	total
new_vertex (1)	parent_node (16)	
	x,y,z (96)	
	update_list (N)	113+N
new_triangle (1)	vertex (16)	
	orientation (1)	18

7. Model reconstruction and rendering

Model reconstruction. At the receiver's side, the geometric model must be reconstructed from the data stream. The cluster tree is de-linearized by successive refinement (i. e. node unfolding) operations, where the child nodes are created from the data fields of the network packages. At the same time, a vertex list and triangle list can be reconstructed. The reconstruction process is incremental and fast, which is necessary, because it must be possible to perform decoding and rendering in parallel, always displaying the best approximation possible with the data received so far.

Rendering. The representation of the model as a cluster tree allows more than one way of rendering. The more conventional approach is to take "snapshots" of the reconstructed triangle list after a certain amount of data has arrived, thereby obtaining conventional, discrete levels of detail. In that case, the reconstruction of the cluster tree can be omitted. The advantage of this possibility is that the resulting LODs can be used by an existing LOD renderer without any modification.

Interactive selection of smooth LODs. The cluster tree makes it possible to select smooth levels of detail on the fly during rendering, which is a more powerful method than simply creating a small set of LODs. In an initial step, the LOD selection operation is used to create a triangle list for display.

For every successive frame, a new threshold is chosen according to the new viewpoint, and the corresponding smooth LOD is selected. Depending on whether the new LOD is finer or coarser than the previous one, the refinement or simplification operation is used to modify the active node set and the vertex list and triangle list. Usually only few manipulations have to be carried out, so the incremental LOD selection can be carried out at interactive speed.

The selection of smooth LODs from the cluster tree also works if the transmission is still incomplete, because every partially created tree is consistent in the set of vertices, triangles and clusters. As soon as new data arrives and is "mounted" in the tree, the model can be refined to incorporate the new data, if desired.

Variable resolution within one object. The comparison of the cluster size against the threshold can also be made by estimating the cluster's projected screen size. This allows to make a different selection for every node, depending on the distance of the cluster to the observer. The displayed model allows non-uniform simplification and automatically adapts to the user's position. Those parts of the object that are further away from the observer will be displayed coarser than those that are near. Consequently, the polygon budget is better exploited. However, neither cluster size nor update list can be precomputed anymore, but the incurred performance penalty can be kept within tolerable limits (compare [7]).

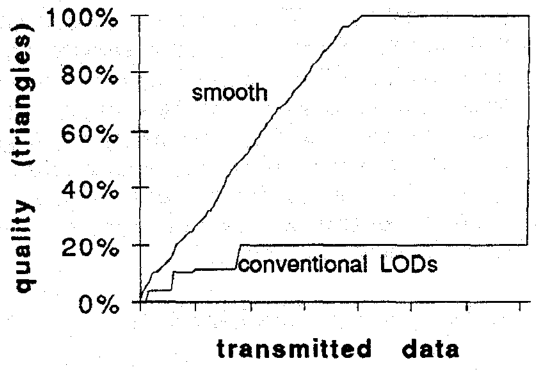
8. Results and Comparison

Comparison of model sizes. Table 2 compares the sizes of models encoded as a smooth LOD packet stream as detailed in section 6 to the original models (vertex list and triangle list) with and without levels of detail (see Figure 7 for images). Every model is listed with its vertex (*#vericest*) and triangle (*#triangles*) count, the original size (*size orig.*), computed from 12 byte per vertex and 6 byte per triangle, assuming 16 bit indices for vertex references in triangles). The next column (*size w/ LOD*) lists the size of the model with 5 conventional LODs including the original object (additional LODs only increase triangle count, vertices are reused from the original model with the approach described in section 4!). These values should be compared to the size of the corresponding smooth LOD model (*SLOD size*), stored in the format given in section 6. The size of the smooth LOD model is also given as a percentage of the original model (*% orig.*) and level of detail model (*% w/ LOD*).

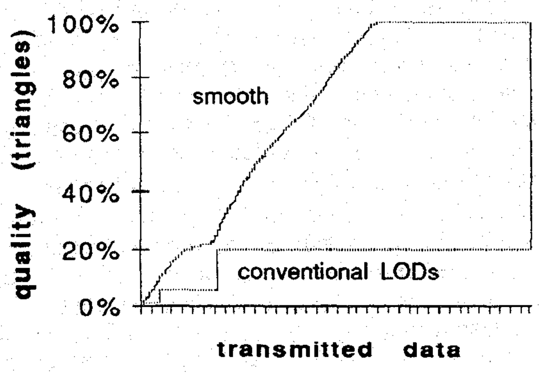
A smooth LOD model is not only smaller than the model with LODs, but also smaller than the original model. As far as model size is concerned, smooth LODs come for free! This is particularly impressive as the applied compression is lossless. There is still much potential in lossy compression [2].

Table 2: Comparison of model sizes - smooth LODs against conventional models (sizes given in bytes)

name	#vertices	#triangles	size orig.	size w/ LOD	SLOD size	% orig.	% w/ LOD
lamp	584	1352	13968	17712	11485	82.2	64.8
tree	718	1092	15168	20460	12830	84.6	62.7
shelf	1239	2600	30228	37188	24014	79.4	64.6
plant	8228	13576	179352	200154	149214	83.2	74.5
stool	1024	1600	21864	30528	18916	86.5	62.0
tub	3422	5404	73488	84906	64584	87.9	76.1
sink	2952	4464	62208	81558	55513	89.2	68.1
ball	1232	2288	28512	39420	24127	84.6	61.2
curtain	4648	8606	107412	109770	89684	83.5	81.7



(a) shelf



(b) plant

Figure 4: Comparison of visual effect of smooth vs. conventional LODs (1 noth on x-axis ≈ 5 KB).

Comparison of the visual effect. Our experience shows that the refinement of a model with smooth LODs is superior to the coarse-grained switching between a few (typically 4-6) conventional LODs. However, such a subjective statement is hard to prove formally. If we assume that image quality is roughly proportional to the number of triangles from the original model, we can compare smooth to conventional LODs by plotting available triangles as a function of transmitted bytes for both methods. Figure 4 shows two such examples.

The maximum triangle count is reached much earlier using the smooth LODs than using conventional LODs because of the smooth LODs' more compact representation (compare the *r_l%* column in Table 2). This difference is also obvious when comparing the obtained images. (compare Figure 5).

Note that the roughly linear correspondence between transmitted data (x-axis) and available triangles (y-axis) is very suitable for networked virtual environments, where an object is approached at constant velocity, while its geometric representation is still being transmitted over a network of constant bandwidth.

9. Conclusions and Future Work

We have presented a new approach for representing polygonal models designed for interactive rendering and transmission in networked systems. A hierarchical clustering method which has been used to compute conventional simplifications of triangle meshes is extended to yield a continuous stream of approximations of the original model. A very large, practically continuous number of levels of detail is computed. The result can be represented in an extremely compact way by relative encoding. The resulting data set is smaller than the original models without levels of detail. If the data set is transmitted over a network, a useful representation is available at any stage of the data transmission. The data set can be used to compute conventional levels of detail, or the underlying hierarchical structure can be exploited to generate and incrementally update any desired approximation for rendering at runtime.

Experiments with the method show that conventional and smooth levels of detail perform roughly alike, if the LODs are used as intended, that is, a coarser approximation is only used if the model is so small on the screen that the visible difference to the high fidelity model is barely noticeable. However, when running real world applications on low cost systems, this assumption is almost always violated because of insufficient rendering performance (see Figure 5 and Figure 6). Furthermore, slow network connections such as Internet downloads make the user wait for completion of transmission while the model is already displayed at full screen resolution. In these situations, our approach is clearly superior, because it makes new data immediately visible (compare Figure 4) and due to its compression finishes earlier.

Future work will involve improving the compression ratio of the smooth LODs.

Acknowledgments. The authors would like to thank Michael Gervautz, Eduard Gröller and Wolfgang Stürzlinger for proofreading and giving useful suggestions. This research was sponsored by the Austrian Fonds zur Forderung der Wissenschaftlichen Forschung under project no. P11392-MAT.

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

[2] M. Deering, "Geometry Compression", *Proceedings of SIGGRAPH'95*, 1995, pp. 13-20.

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

[4] P. Heckbert, M. Garland, "Multiresolution Modelling for Fast Rendering", *Proceedings of Graphics Interface'94*, 1994, pp. 43-50.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, "Mesh Optimization", *Proceedings of SIGGRAPH'93*, 1993, pp. 19-26.

[6] M. Levoy, "Polygon-Assisted JPEG and MPEG Compression of Synthetic Images", *Proceedings of SIGGRAPH'95*, 1995, pp. 21-25.

[7] D. Luebke, "Hierarchical Structures for Dynamic Polygonal Simplification", *Technical Report TR-96-006*, Univ. North Carolina Chapel Hill, 1996.

[8] 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, Italy, 1993.

[9] G. Schaufler, W. Stürzlinger, "Generating Multiple Levels of Detail from Polygonal Geometry Models", *Virtual Environments'95*, Springer Wien-New York, 1995.

[10] D. Schmalstieg, M. Gervautz, "Demand-Driven Geometry Transmission for Distributed Virtual Environments", *Proceedings of EUROGRAPHICS '96*, Poitiers, France, August 1996.

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

[12] A. Varshney, "Hierarchical Geometric Approximations", *PhD Thesis*, Department of Computer Science, University of North Carolina, 1994.

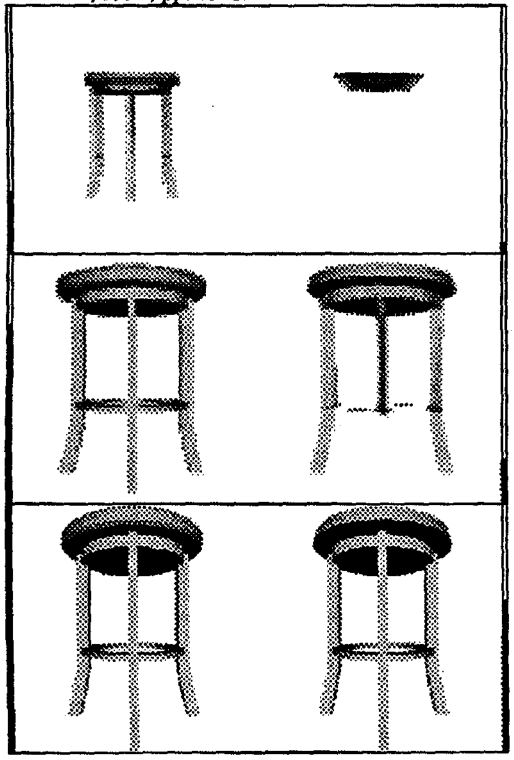


Figure 5 (left): Comparison of 3 development stages of a chair. The left column shows smooth LODs, the right column conventional LODs for corresponding amounts of data. Black bars on each side indicate the amount of triangles received and displayed.

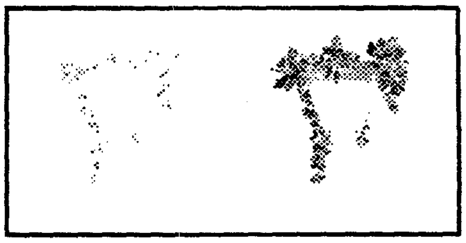


Figure 6: Difference image between two successive frames from an animation of the plant model. The difference for smooth LODs is very small (left). The conventional LOD model (right) switches LODs exactly between these consecutive frames, which produces an obvious difference. Note that this is an extreme example: Usually, LOD switching is not so much exposed because smaller images are rendered for coarse LODs.

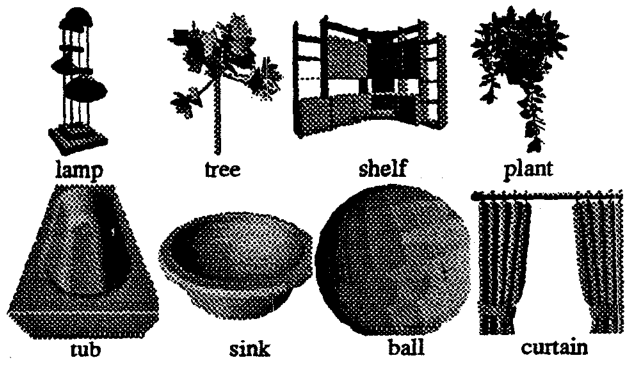


Figure 7: Models used for evaluation