Smooth Levels of Detail

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

Levels of detail (LODs) are used in interactive computer graphics to avoid overload of the rendering hardware with to high numbers of polygons. While conventional methods use a small set of discrete LODs, we introduce a new class of polygonal simplification: Smooth LODs. A very large number of small details encoded in a data stream allows a progressive refinement of the object from a very coarse approximation to the original high quality representation. Advantages of the new approach include progressive transmission and encoding suitable for networked applications, interactive selection of any desired quality, and compression of the data by incremental and redundancy free encoding.

1. Motivation

When rendering complex three-dimensional scenes, it is commonly the case that many objects are very small or distant. The size of many geometric features of these objects falls below the perception threshold or is smaller than a pixel on the screen. To better use the effort put into rendering such features, an object should be represented at multiple levels of detail (LODs). Simpler representation of an object can be used to improve the frame rates and memory utilization during interactive rendering. This technique was first described by Clark already in 1976 [1], and has been an active area of research ever since.

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. Frequently models are too complex for the available rendering capacity, so that a coarser approximation than the one desired must be drawn to prevent a reduction of the frame rate. In such cases, switching from one level of detail to another is particularly distracting and annoying for the user.

With the increasingly widespread use of 3-D graphics in distributed applications and over the Internet, 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. As data sizes increase with LOD quality, delays between model refinements increase rapidly. Such stalling negatively affects the participant's experience of the simulation.

Adding levels of detail partly addresses the rendering problem for large and complex objects, but makes overall model size even larger. The reason for this problem is that the standard approach of representing polygonal data as lists of vertices and triangles is not powerful enough. Instead, we need a more capable data structure that can address the mentioned shortcomings.

The model 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. 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.

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 be able to make immediate use all the data received up to a certain moment, and render a model not yet fully transmitted. This is important for progressive refinement of large models that take an extended period to transmit, and allows continuous operation in case of network failures.

The smooth LODs data structure should support selection and rendering of any specific LOD in real-time allowing to vary the level of detail (both coarser and finer) at interactive speeds (during rendering).

It is preferred if the smooth LODs data structure introduces no overhead in model size compared to the original, uncompressed polygonal model. Ideally, the introduction of smooth LODs should yield compression instead of increasing the model size.

All these properties can be addressed by a novel object representation called smooth levels of detail that is

presented in this paper. After reviewing related work, we present how to create, manipulate and render smooth levels of detail. We also show how they can be used for geometry compression, and present some results from our implementation.

2. Related Work

Generating levels of details addresses the the problem of finding a series of progressive simplifications of a polygonal object, that have fewer primitives (polygons), but closely resemble the original object.

2.1 Topological algorithms

The methods that produce the highest quality work on the surface of polygonal objects, e.g. [4, 10, 11]. 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 Geometric 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 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 [7], octree quantization [6, 17] or a nearest neighbor search [8, 9].

2.3 Progressive representations

Two approaches have been developed concurrently, that draw from the a similar basic idea as the approach presented in this paper, namely to abandon the use of a small set of discrete levels of detail in favor of a progressive representation that efficiently encodes a large number of LODs. Eck et al. [3] develop a wavelet-based representation of polygonal geometry, which is extended in [12] to allow interactive multi-resolution surface viewing. Hoppe's representation - progressive meshes [13] - is based on incremental topological operations on the object's surface. The major difference of these algorithms to ours is that we use a geometrical rather than a topological method to reduce object complexity, which is simpler, more robust and efficient, but does not yield as tight bounds on the visual error.

2.4 Selective refinement

The approaches presented in [12] and [13] allow selective refinement of the model as opposed to choosing one LOD per object. This property is also supported by Lindstrom et al. in their terrain rendering model [14], by the wavelet-based meshes from [16] and by the simplification enveloped presented in [15].

2.5 Compression

As far as compression of geometry for storage and transmission is concerned, some work is relevant for our approach: Deering [2] introduces a compression method for polygonal data sets. Levoy [5] proposes a combination of geometry and compressed image data to preserve bandwidth with a compressed video stream.

3. The hierarchical cluster tree representation

Hierarchical clustering for LOD generation, as first presented in [8], 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. 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*.

3.1 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 which 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. Finding the closest pair of clusters can efficiently be done with a BSP tree.

The algorithm starts with a cluster for each vertex, with the vertex serving as the representative. In each step, it finds the two clusters with the closest representatives, and replaces the two clusters identified in step 1 by a joint cluster. For the joint cluster, a new representative is selected. This procedure is repeated until only one cluster containing all vertices remains.

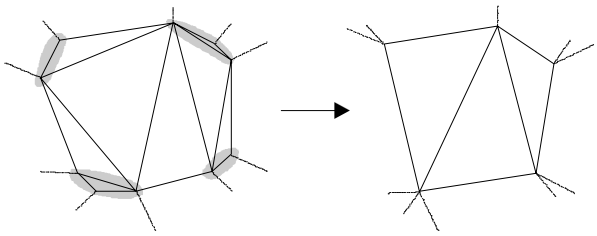


Figure 1: The clustering process: A mesh is mapped onto a vertex cluster tree, which is used to group vertices. From the reduced vertex set, a simplified model is computed.

Various heuristics are possible to select the new representative of a cluster among the candidate vertices. We used the vertex with the largest distance from the object's center to avoid shrinking the object as vertices are moved together. An alternative was proposed in [7] and tries to identify vertices that are visually important.

The cluster tree contains instructions for a continuous simplification of the model, and therefore can 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 extended cluster tree alone.

When two clusters are joined and consequently on representative vertex is eliminated, the events (changes) are recorded in the triangle database. The reversed application of these events can be used to reconstruct the triangle database by evaluating the events node by node.

3.2 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, three cases can be distinguished:

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. A list (the *update list*) of all such triangles is kept in the cluster node (Figure 2a).
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 2b).

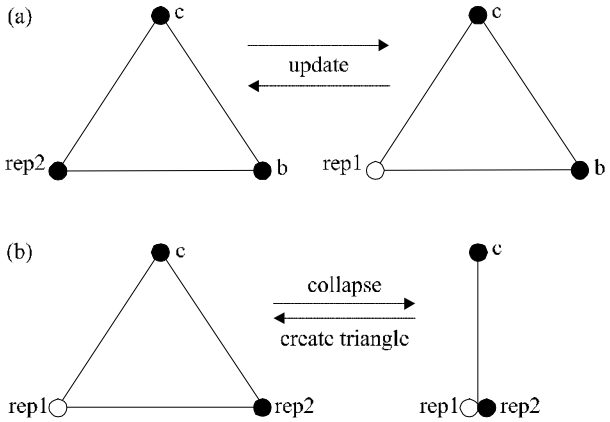


Figure 2: Two events in the triangle database during clustering are of interest for the reconstruction of the original triangles: Collapsing triangles (a), and triangles whose vertices are updated (b).

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. Stepping from one LOD to the next is done by adding only one vertex (adding one cluster, see Figure 3).

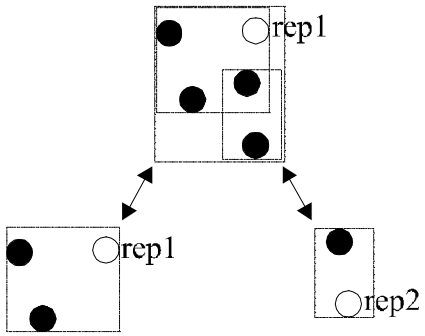


Figure 3: During the clustering, two vertex clusters are joined into one, and the effect on the triangles is recorded. The inverse operation, cluster expansion, uses the recorded data to reconstruct the triangles.

The involved changes are small, so coherence between LODs is exploited by storing only the changes in the update list and collapsed list at each node. A cluster tree containing the cluster representatives and the information on triangle changes (update list and collapsed list) completely encodes the original model, plus instructions how to create all intermediate levels of detail.

4. Manipulation of the cluster tree

While the cluster tree has the desired property of compactly representing 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). Moreover, a tree is also not suitable for network transmission, it must be linearized first. 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 (linearization, model reconstruction, LOD selection, and rendering) can easily be constructed.

4.1 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 reverse order. 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.

4.2 Reconstruction of the polygonal model

The original polygonal model, consisting of a vertex list and a triangle list, can be reconstructed using the cluster tree traversal. The root introduces 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). At the same time 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 (reversing the process by which 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 has been completely restored.

4.3 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 to terminate the reconstruction process 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. A modified traversal algorithm no longer continues until the active node set is empty, but terminates if the biggest cluster size of any such node is smaller than the given threshold. The reconstructed triangle and vertex lists up to that point represent the desired level of detail and can directly be used for rendering.

4.4 Refinement

For refinement of the model, the fundamental operation is to switch from a given level of detail to the next finer one. A particular LOD is defined by a list of active node in the cluster tree, and the corresponding vertex and triangle lists. Refinement is achieved by expanding the node with the largest cluster size in the active node list 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 speed. Selection of a LOD as previously mentioned is nothing else than the repeated application of refinement, starting with an initially empty vertex and triangle lists.

4.5 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 clustered into their common parent node. One vertex is removed from the vertex list, and references to that vertex in the triangle list are removed. Collapsed triangles are filtered out, which simplifies the model.

4.6 Rendering

"Snapshots" of the vertex and triangle lists can be taken after reaching the desired model fidelity to obtain conventional discrete levels of detail, or the cluster tree and the vertex and triangle lists can be maintained in parallel for interactive selection of smooth levels of detail. In that case, the currently displayed object is constructed by adapting the cluster size threshold to changes in the viewpoint, applying simplification or refinement operations as appropriate, and modifying the vertex and triangle lists according to these incremental operations.

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 exploited more efficiently.

However, neither cluster size nor update list can be precomputed any more. Interactive frame rates can be achieved by adapting the incremental algorithm for simplification/refinement. The active node list is small compared to the total number of clusters. It's items need to be examined whenever the viewpoint changes. However, spatial coherence can be exploited by re-evaluating the projected cluster size only if the ratio of the distance to the cluster and the distance travelled since the last evalation exceeds a certain treshold. As projected cluster sizes change slowly for smooth image sequences the items in the active node list can be visited in a round-robin fashion only every n frames.

5. Binary format

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 containing the node is piped 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.

From the linearized model it is easy to construct a binary format that is very compact and suitable for network transmission. No redundant information is stored in the network packages, so the requirement of compactness is satisfied by the network protocol. Actually the packets represent the smooth LODs model in less bytes than the original model (see section 8 for results). Effectively, the protocol can be used as a compression method.

Recall that the following information must be encoded for every node in the cluster tree:

- the new vertex introduced by the refinement operation
- the update list encoding which triangles must be modified to contain the new vertex
- the collapsed list encoding which new triangles must be created when the new vertex is introduced.

The goal of the protocol was to encode the required information with as little data as possible. Our protocol currently deals with vertices, triangles and surface materials and consists of four packets types: VERTEX, TRIANGLE, MULTI-TRIANGLE, MATERIAL.

PACKET	TAG	FIELDS (length)
VERTEX	0	parent (variable) coordinates (variable) update list (variable)
TRIANGLE	10	vertex_id (variable) orientation (1 bit)
MULTI	110	duplicate_flag (1 bit) vertex_id (variable)
MATERIAL	111	material_id (8 bit)

Table 1: Protocol packets with parameters and sizes in bit

Packet headers are encoded using a variable length tag according to their frequency. Table 1 summarizes the packets including their parameters (field sizes in bits are given in parenthesis).

5.1 VERTEX

Format: VERTEX(parent, 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 other inherits the coordinates from the parent.

Parent cluster: The parent field indicates the cluster that is being split in two. Indices can only point to already existing clusters, so they can have variable length: As the number of clusters increases, more bits are needed to encode the index. This variable length encoding of indices saves more than 50% of the bits needed for indices.

Vertex coordinates: The (x,y,z) tuple gives the coordinates of the new vertex. Details on the encoding of the vertices are given in the next section.

Update list: 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 allow updating of the triangles correctly. The update is simply the replacement of the parent cluster's 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 or not the update should take place. These bits are compactly stored as a variable length 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.

5.2 TRIANGLE

Format: TRIANGLE(vertex_id, orientation)

As the reconstruction of the object from the network data stream is the inverse operation of the clustering stage, for every new vertex encoded by 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 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 an index into the array of vertices. Like cluster indices, vertex indices can have variable length.

The new triangle has either the orientation (new_vertex, parent_rep, vertex_id) or (parent_rep, new_vertex, vertex_id), which is distinguished by the orientation bit.

5.3 MULTI-TRIANGLE

Format: MULTI(duplicate_flag, vertex_id)

The clustering process may produce identical triangles that are not collapsed and consequently not removed. These doublets were intentionally left in the data, because removing them would greatly complicate the coding and decoding process. Instead, the MULTI package can introduce either 2 or 4 related triangles at once, which efficiently covers the most frequent cases produced by the clustering algorithm. If the duplicate flag is zero, 2 triangles with opposite orientation (new_vertex, parent_rep, vertex_id) and (parent_rep, new_vertex, vertex_id) are created. If the duplicate flag is one, 2 triangles of either orientation are created.

5.4 MATERIAL

Format: MATERIAL(index)

While polygonal models always contain geometry, they may or may not contain materials or colors. Our models consist of a small set of fixed materials, that can be encoded in an 8 bit index. A MATERIAL packet sets the current material of the following geometry to the new value until another material package is encountered. As our models use only a few different materials, such packets are relatively infrequent, and no further optimization efforts were taken. Material definitions are distributed once to all participating sites. If required, material definitions can be given in the header of the model. A more sophisticated shading support may include vertex colors for pre-shaded (e.g., radiosity) models or texture mapping. The latter would require to take into account the cumulated error in texture coordinates when computing distances between vertices, as shown by Hoppe [13].

6. Hierarchical precision encoding of vertices

About half the size of the model is due to the vertex coordinates. These are not affected by the algorithms and therefore are not yet compressed. Deering argues that while coordinate data is usually represented using floating point numbers, the finite extent of geometric models allows representation using compact fixed point numbers [2]. To minimize errors resulting from lossy compression via quantization, we have developed a hierarchical precision encoding scheme for the coordinate data. Our method still yields compression ratios of 1:2 to 1:3.

For every ordinate, a neighborhood is chosen by defining a fraction of the object diameter. If the new ordinate lies within the neighborhood of the corresponding ordinate of the parent cluster's representative, the ordinate is encoded with a relative offset to it. This offset is stored as a fixed point value ("relative encoding"). As the new vertex is expected to be in the vicinity of the parent's representative, most of the ordinates can be encoded relatively, thus saving storage. If the ordinate is not in the neighborhood, it is stored as an absolute (32 bit) single precision float ("absolute" encoding).

Typically we define the neighborhood to be a quarter of the extent of the model (computed separately for every axis), and consequently can bound the error to (1/4) * 1/(2^16) = 0.000004% of the model extent. At this precision, we use either 8 or 16 bit values (many relative values are small, and consequently 8 bit or less are sufficient).

Another method further reduces storage consumption: A special bit code indicates if the difference to the parent's ordinate is zero. In this case the specification of the 16 bit delta value can be omitted ("null" encoding). Often CAD models have edges aligned to the axises of the coordinate system, so this is frequently the case.

Note that while the use of fixed precision for relative encoding makes the compression scheme lossy, the inaccuracies introduced can be controlled by the user by selecting the fraction of the model extent which is to be considered as the neighborhood of parent vertices.

COORDINATES	TAG	FIELDS
relative16	0	16 bit fixed
relative8	10	8 bit fixed
null	110	(none)
absolute	111	32 bit float

Table 2: Protocol for encoding of coordinates

The distinction between the encoding variants is made by variable length tags. Table 1 gives an overview of coordinate encoding.

7. Results

Comparison of model sizes: Table 3 allows to compare 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. Every model is listed with its vertex and triangle count, the original object size, computed from 12 byte per vertex and 6 byte per triangle, assuming 16 bit indices for vertex references in triangles. The next column (*LOD size*) lists the size of the model with 5 conventional LODs including the original object (additional LODs only increase the triangle count, vertices are reused from the original model [8]). These values should be compared to the size of the corresponding smooth LOD model (*smooth LOD 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 (*% of obj. size*) and level of detail model (*% of LOD size*).

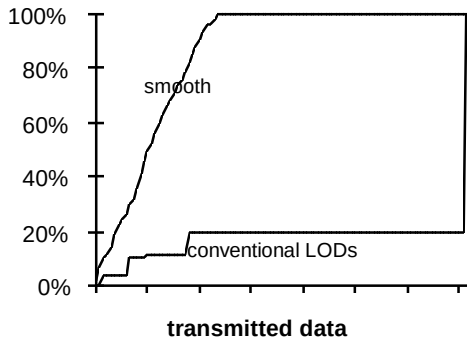
Note that the smooth LOD model is always not only significantly smaller than the level of detail model, but also smaller than the original model. As far as model size is concerned, smooth LODs come for free!

model name	# of verti-ces	# of trian-gles	object size	LOD size	smooth LOD size	% of obj. size	% of LOD size
lamp	584	1352	13968	17712	6106	43.7	34.5
tree	718	1092	15168	20460	7288	48.0	35.6
shelf	1239	2600	30228	37188	12635	41.8	34.0
plant	8228	13576	179352	200154	89921	50.1	44.9
stool	1024	1600	21864	30528	8406	38.4	27.5
tub	3422	5404	73488	84906	26993	36.7	31.8
sink	2952	4464	62208	81558	23743	38.2	29.1
ball	1232	2288	28512	39420	14099	49.4	35.8
curtain	4648	8606	107412	109770	44334	41.3	40.4

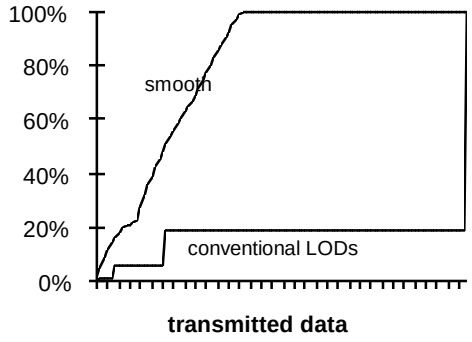
Table 3: Comparison of model sizes - smooth LODs against conventional models (sizes given in bytes)

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 3-6) 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 used for display, we can compare smooth to conventional LODs by plotting triangles available for rendering 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 (see the *% of obj. size* column in Table 3). This difference is also obvious when comparing the obtained images.



(a)



(b)

Figure 4: Comparison of visual effect of smooth vs. conventional LODs (a - shelf, b - plant). We measured the quality as the number of transmitted triangles for a certain amount of data (1 notch on the x-axis » 5 KB)

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 often approached at constant velocity, while its geometric representation is still being transmitted over a network of constant bandwidth.

8. Conclusions

We have presented a new polygonal model representation called smooth LODs 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 possible. 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.

When running real world applications on low cost systems, the constraint of using a coarse LOD only if the difference to the high fidelity model is not noticeable is regularly violated because of insufficient rendering performance. 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 Figure 5) and finishes earlier due to its compact representation.

Acknowledgments. This work was sponsored by the Austrian *Fonds zur Förderung der wissenschaftlichen Forschung* under project no. P11392-MAT.

Animations and visual results are available at http://www.cg.tuwien.ac.at/research/vr/smoothlods/

Figure 5 (right): Comparison of development stages of the chair and lamp model. 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.

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. Deering: Geometry Compression. Proc. of SIGGRAPH'95, pp. 13-20 (1995)

[3] 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)

[4] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle: Mesh Optimization. Proceedings of SIGGRAPH'93, pp. 19-26 (1993)

[5] M. Levoy: Polygon-Assisted JPEG and MPEG Compression of Synthetic Images. Proceedings of SIGGRAPH'95, pp. 21-25 (1995)

[6] D. Luebke: Hierarchical Structures for Dynamic Polygonal Simplification. Technical Report TR-96-006, Univ. North Carolina Chapel Hill (1996)

[7] 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)

[8] G. Schaufler, W. Stürzlinger: Generating Multiple Levels of Detail from Polygonal Geometry Models. Virtual Environments'95, Springer Wien-New York (1995)

[9] D. Schmalstieg, G. Schaufler: Incremental Encoding of Polygonal Models. Proceedings of HICSS-30 (1997).

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

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

[12] A. Certain, J. Popovic, T. DeRose, T. Duchamp, D. Salesin, W. Stuerzle: Interactive Multiresolution Surface Viewing. Proceedings of SIGGRAPH'96, pp. 91-98 (1996)

[13] H. Hoppe: Progressive meshes. Proceedings of SIGGRAPH '96, pp. 99-108 (1996)

[14] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, G. Turner: Real-Time Continuous Level of Detail Rendering of Height Fields. Proceedings of SIGGRAPH'96, pp. 109-118 (1996)

[15] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright: Simplification Envelopes. Proceedings of SIGGRAPH'96, pp. 119-128 (1996)

[16] M. Gross, R. Gatti, O. Staadt: Fast Multiresolution Surface Meshing. Proceedings of Visualization'95, pp. 135-142 (1995)

[17] D. Schmalstieg: Lodestar - An Octree-Based Level of Detail Generator for VRML. To appear in: Proceedings of the SIGGRAPH Symposium on VRML'97 (1997)