Coarse View-Dependent Levels of Detail for Hierarchical and Deformable Models

Dieter Schmalstieg and Anton Fuhrmann
Vienna University of Technology, Austria
dieter|fuhrmann@cg.tuwien.ac.at

Abstract: While a large amount of work is geared towards multi-resolution modeling and rendering, the presented methods work almost exclusively with static models and scenes. The ASLOD (Animated Smooth Level Of Detail) approach presented in this paper is capable of rendering animated scenes consisting of mixtures of conventional and multi-resolution models. It features coarse view-dependent multi-resolution rendering by a consistent decomposition of large models into arbitrary regions together with an analytic solution to optimize polygon allocation to such assemblies of multi-resolution models. The multi-resolution model can be used together with animated hierarchical scene graphs. Finally, the multi-resolution model can be combined with a hierarchical skeleton and a real-time deformation algorithm for animating organic shapes rendered at any desired level of detail.

Keywords: level of detail, multi-resolution models, coarse view-dependent rendering, hierarchical modeling, scene graph, deformation, rigid bodies, articulated figures

1. Introduction

While levels of detail LOD modeling and rendering is indispensable for real-time performance in rendering, previous approaches have mostly ignored the issue of animation, i. e. dynamically changing scenes, and therefore are limited to static walk-throughs. In particular, the combination of LOD modeling and rendering with a hierarchical scene graph, articulated bodies and deformable meshes is an open challenge. We present a first approach towards addressing these issues.

1.1 Related Work

Since the conception of the LOD idea by Clark [Clark 1976], much research has been focused on LOD generation, leading to methods of increasing sophistication:

1. Discrete methods create multiple separate LOD representations for each object in the scene. For each object and frame of an animation sequence, one LOD is chosen and rendered. For details refer to some of the excellent surveys on the topic [Cignoni et al., 1998a; Heckbert & Garland, 1997; Rossignac, 1997].
2. Progressive LOD methods (e. g. [Hoppe, 1996; Schmalstieg & Schaufler, 1997]) create a multi-resolution representation from the original polygonal object. At runtime, a simplified polygonal representation is extracted from the multi-resolution representation at arbitrary precision. The difference to discrete methods is that the runtime system can select an arbitrary number of primitives rather than having to chose from a predetermined set of precomputed LODs. The creation process from [Garland & Heckbert, 1997] can also be used to construct progressive LOD models.
3. View-dependent progressive LOD methods (e. g. [Floriani et al., 1997; Gross et al., 1995; Guziec et al, 1998; Hoppe, 1997; Klein, 1998; Lindstrom et al., 1996; Luebke & Erikson, 1997; Xia et al., 1997]) are able to adapt the polygonal representation dynamically in a non-uniform manner. Depending on the viewpoint and possibly other parameters such as lighting, some areas of the object may be represented at a higher LOD than others. This behavior is achieved by an extended multi-resolution representation that allows local refinement of the geometry.

Discrete methods separate the generation of LOD models from the run-time selection. Multiple representations of a single model may be created using any algorithm independently of the rendering. When using discrete LOD models, many commercial implementations rely on static LOD selection such as VRML97 [Hartman & Wernecke, 1996] or simple feedback mechanisms such as Performer [Rohlf & Helman, 1994]. These approaches fail to achieve true output-sensitivity, as there is no upper bound on the number of polygons that are scheduled for rendering. [Funkhouser & Sequin, 1993] therefore propose a procedure that achieves output sensitivity by optimizing discrete LOD selection according to cost and benefit heuristics. They show that their approach is sufficiently precise in predicting rendering time even for sudden changes in scene complexity, so that frame times are always met.

1.2 Problem statement

Progressive and view-dependent progressive methods integrate generation and modeling of multi-resolution models. The LOD representation generated from the original models can only be rendered with the specific traversal/selection procedure designed for that data structure. An advantage of this approach is that frame times are easily met since the user may specify a target primitive count. While view-dependent progressive models solve the problem of output-sensitive rendering, some issues are not addressed:

Lack of support for animation: All examples to date demonstrate output-sensitive rendering of only one large, static model, such as in a terrain fly-over or a walkthrough. However, virtual environments are generally composed of larger assemblies of individually animated entities, such as moving avatars in a multi-user experience or machine parts in an educational animation. The scene of a virtual environment is therefore not a flat, static polygonal database, but a hierarchical assembly - a scene graph. While the method of [Funkhouser & Sequin, 1993] and the extension proposed in [Mason & Blake, 1997] are capable of allocating the polygon budget among multiple independent models, it is not clear how this can be achieved with view-dependent progressive models.

While some approaches allow to a certain extent interactive resolution modification [Cignoni et al., 1998a] or editing [Lau et al., 1998; Zorin et al., 1997] of multiresolution models, these features are costly and can only be applied to moderate models in an interactive environment (e. g., Zorin et al. report 3 second preprocessing for every modification in their multiresolution editor).

Runtime efficiency: Maintenance and extraction of an ideal polygonal model from a view-dependent multi-resolution model introduces significant computational complexity and may affect scalability of the system, in particular if the CPU (on low-cost systems) also has to perform geometric transformations.

1.3 Contribution

This paper addresses the mentioned restrictions by introducing a new multi-resolution representation called ASLOD (Animated Smooth Level of Detail). Major contributions of our approach are:

- Integration of progressive multi-resolution models with hierarchical scene graphs: The ASLOD method allows to use progressive LOD models (section 2) to create a scene graph and select detail for each model individually, while the scene graph maintains all its usual properties and can be used for animation in dynamic virtual environments. Thus existing applications of virtual environments can be fitted with progressive LOD rendering without the need to modify the scene structure.
- This integrated approach is enabled by a simple analytical run-time LOD selection strategy for such scene graphs (section 3). This strategy allows to allocate a given polygon budget over a large number of objects.
- Coarse view dependence (section 4): We believe that extracting an ideal representation from a view-dependent progressive LOD model is computationally costly and not strictly necessary. Instead, we propose to approximate the ideal view-dependent representation with *coarse view-dependence* that is based on blending multiple non view-dependent progressive LODs and can be customized by the user to the desired precision.
- Finally, ASLOD supports multi-resolution rendering of deformable models animated with skeleton animation (see section 5). By using this feature, organically changing shapes frequently found in high-quality animations can be rendered with a fixed polygon budget.

Our approach is illustrated with details on the prototype implementation (section 6), results (section 0) and conclusions (section 8).

2. Basic progressive LOD model

The progressive LOD model upon which our system is built is derived from earlier work described in [Schmalstieg & Schaufler, 1997]. It is summarized here for clarity. A given triangular mesh is

successively simplified by local modifications of the geometry. This simplification introduces local error (the simplified model deviates from the original error) measured by a suitable metric. By repeatedly applying local simplification, the polygon count is iteratively reduced, yielding a sequence of approximations to the original model with increasing error.

The underlying data structure is a binary tree called cluster tree. The algorithm starts with a candidate set of one cluster for each vertex. In each step, it finds two source clusters with the closest vertices, and creates a new cluster node with the two source clusters as children. The clustered vertices may or may not be required to share an edge, depending on the type of input model that must be supported.

For the new cluster node, a representative is chosen from the set of vertices it represents. The new cluster then replaces the two source clusters in the candidate set, and the process is repeated until only one cluster remains.

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.

Therefore, when two clusters are joined and consequently one representative vertex is eliminated, this information is stored in the new cluster. The order in which vertices are collapsed is recorded in a *multiresolution sequence* similar to the one proposed by [Hoppe, 1996]. At runtime, the desired LOD is chosen by moving back and forth in this sequence and using the information in the cluster tree to update (i. e., add, remove or change triangles) a separate polygonal *cache* (a straightforward BREP) which is used for rendering.

3. Level of detail selection

The dominant modeling paradigm for virtual worlds is not a single polygonal model, but a scene graph composed of individual objects that are related to each other by assembling them in a hierarchy. A scene graph can be used to assemble and animate collections of individual objects, but also to represent complex animated objects such as articulated rigid bodies.

For scenes modeled with discrete LODs, [Funkhouser & Sequin, 1993] have formulated the task of finding a good selection of LODs as a discrete optimization problem. They propose the following heuristics:

- Cost is constrained by the system's polygon and pixel fill rate. Current systems use powerful rasterization hardware, while affordable hardware acceleration for geometric computation is only beginning to become affordable [nvidia, 1999]. Therefore simplification literature generally assumes polygon rate the bounding factor and consequently tries to reduce polygon count only.
- Benefit is determined by the object's size on the screen and the number of polygons. This heuristic can be modulated by a variety of factors including shading, focus, user priorities etc.

Their discrete optimization problem is formulated as follows: Maximize cumulative benefit while keeping the associated cost within a given limit (polygon count). In contrast, most literature on progressive LOD generation does not discuss polygon allocation. View-dependent progressive LOD models implicitly solve the allocation of polygons by truncating refinement of the model extracted from the multi-resolution representation when the polygon count is exhausted (although this fact is hardly ever discussed in the literature). However, this does not necessarily explain how the polygon budget is subdivided among two or more separate LOD models.

In the following, we describe an analytic method to compute the polygon count that should be allocated to a collection of progressive LOD models assembled in a scene graph. For simplicity, assume that the scene for which the polygon budget must be subdivided is composed of a set of objects, each of which is small enough so that perspective distortion can be neglected and projected size gives a reasonable estimate as to what LOD should be selected for the object (as used for basic progressive LOD models). If objects in the scene do not fulfill this property, they must be broken down into smaller pieces. Section 4 explains how this can be done while keeping the borders between the pieces seamless.

Clearly, benefit (i. e. contribution to the image) of an object increases as more polygons are allocated to that object. However, it is not desirable to make the relative benefit increase (in mathematical terms, the derivative of the benefit function) constant, as adding more and more polygons yields diminishing returns in terms of quality improvement.

As the difference in benefit increase is certainly subjective, it was decided to incorporate a parameter that lets the application designer or user modify the system's policy of polygon allocation. Therefore, the following simple formula uses an attenuation exponent α to compute the polygon count p_i allocated to object i out of n objects given an overall polygon count P and weights w_i (which are dependent on the projected area of a single small object) for each object:

$$p_i = P \frac{w_i^{\frac{1}{\alpha}}}{\sum_{i=0}^{n} w_j^{\frac{1}{\alpha}}}$$

The rationale of this formula is that the polygon budget should be allocated proportional to the fraction an object's projected area has in the total projected area of all objects. Different strategies are possible in the choice of α and w_i. For example, setting $\alpha = 3$ and $w_i = 2A_i c$ where A_i is the projected area and c is an arbitrary user-defined constant yield the actual equivalent of the discrete optimization from [Funkhouser & Sequin, 1993]. For details on the parameter selection, refer to [Wimmer & Schmalstieg, 1998]. The projected area of objects is estimated by computing the area of an tight oriented bounding box (OBB) [Schmalstieg & Tobler, 1999] of the object after projection onto the screen.

4. Model decomposition for coarse view-dependence

A basic progressive LOD model allows to choose one LOD per object. This may be insufficient for large objects. After perspective projection, some parts of a large model that are in front appear larger and should consequently be rendered with more primitives. While basic progressive LOD models are completely view-independent and fail to adapt sufficiently to such situations, view-dependent approaches allow to select the degree of detail for every surface location separately, requiring a large number of decisions for every frame even if modifications are applied incrementally.

We therefore present an approach on how to decompose a large polygonal model consistently into independent and disjoint regions of arbitrary size, without holes or cracks at the borders between two regions. This approach is related to the decomposition scheme proposed in [Hoppe, 1998], but differs in two important aspects:

- Rather than partitioning the faces, we are partitioning the vertices of the mesh into disjoint sets.
- Due to this partitioning strategy, each resulting region can be simplified completey independent without the requirement of [Hoppe, 1998] that region borders must remain at full resolution.

Assume such a partitioning of a mesh into two regions. If there exists an edge from a vertex belonging to one region to a vertex belonging to another region, these two regions are called *neighbor regions*. In that case, there are a number of triangles between neighbor regions that have two vertices in one region, and one vertex in the neighbor region. Such triangles belong to the region where they have two vertices, and are called the *border triangles* of that region. The vertices from these triangles are called *border vertices*, and the edges between regions are called *border edges*. Note that the border between regions effectively cuts *through* border triangles: each border triangle has two vertices on one side of the border and the third vertex on the other side (Figure 1).

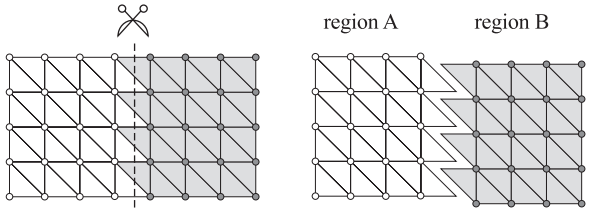

Figure 1: The mesh is decomposed into connected regions. Every vertex belongs to exactly one region, and each triangle belongs to the region where it has at least two vertices.

Edge collapses are only performed when both vertices belong to the same region. No edge crossing a border may be collapsed. Nevertheless border regions can be simplified: edges of border triangles lying completely in either region can be collapsed, thereby reducing the number of border triangles. This behavior is implemented via an additional lookup table for edge vertices, which maps border triangle vertices from one region onto vertices of the other, occasionally mapping them to the same vertex, when an edge has been collapsed in the other region. Thereby tears and other inconsistencies (T-vertices etc.) between independently simplified regions are avoided.

A slight disadvantage from this approach is that border edges cannot be collapsed. However, border triangles and border edges can still be decimated to any desired degree by collapsing edges between two adjacent border vertices within one region. While this is not quite as flexible as arbitrary edge collapses of involving border vertices, we found the restriction to be negligible in practice, as the proposed data structure makes only sense if the interior of regions is significantly bigger in the number of involved

primitives than the border. Figure 2 depicts different LODs on connected regions. Note how the border region does not exhibit artifacts.

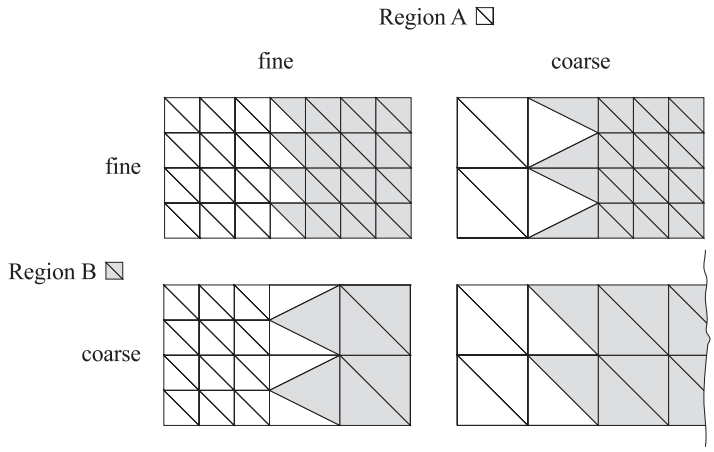

Figure 2: The level of detail of two adjacent regions A (light polygons) and B (dark polygons) can be varied independently without creating tears or cracks

To summarize, coarse view-dependence can be achieved by decomposing large polygonal objects into sufficiently small regions for which a uniform level of detail can be chosen independently at runtime with little computational overhead, while retaining most of the benefits of view-dependent rendering.

5. Animating deformable structures

Animating large structures by manipulating the positions and orientations of the objects – and their LODs – is easily implemented. For more sophisticated animations, especially animations of human bodies, the simple approach of assembling an animated structure from rigid bodies is not satisfying. It results in a typical "wooden puppet" appearance, where joints are depicted as intersections of rigid bodies.

We present a solution for animation of deformable bodies which is applicable for animation of humans in real-time. To satisfy the real-time criterion, our approach uses keyframe skeleton animation to reduce the information necessary for animating the human model and ASLODs to allow view and pose dependent rendering of the model.

5.1 Skeleton animation

Organic structures such as humans are covered with a smooth skin that deforms if the creature is animated. However, such a skin is difficult to animate directly, as the exact specification of the deformation of every small part of the skin is too labor-intensive. Consequently, a higher level control structure is used to specify deformation indirectly. Today's animation models are composed of a *skin* (either a polygonal model or a free-form surface) and a *skeleton* defining the underlying structure, which is conceptually similar to a scene graph for articulated rigid bodies. One node of the skeleton (associated with one transformation) is called a *bone*.

The skin is made functionally dependent on the underlying skeleton. Animation of the skeleton results in a deformation of the skin, which is in turn rendered, while the skeleton remains invisible. Deforming a fully detailed triangular representation is computationally expensive. This is a reason why objects in virtual environments are often composed only from articulated rigid bodies.

Furthermore animating a skeleton requires much less information than animating every vertex of a model. By interpolating keyframe data provided every few frames or only for significant poses, bandwidth for distributed virtual environments can be preserved. The data for the pose of the underlying skeleton can also be generated by motion-tracking a user, which provides highly detailed animated avatars.

5.2 Model decomposition for deformation

We observe that deformation does not occur uniformly over the skin of a deformable object, but that skin areas in the vicinity of joints are strongly deformed, while other areas remain unaffected or only marginally affected by the animation and need not be deformed. This observation allows us to introduce a

simplification which speeds the animating calculations significantly while only slightly reducing the visual quality of the result: we *combine* our model from rigid and deformable regions.

A rigid region would be for example a lower arm, which would deform only minimal under normal circumstances, while the attached elbow needs to be deformable to act as a flexible connection to the - rigid - upper arm. A model composed in this way allows a wide range of poses without breaking up.

Compared to procedural animation techniques used for feature films these simplifications of course lack in realism: muscle bulges, sinews and physical simulations of flesh cannot easily be simulated in this way. However, regarding the real-time requirements and application areas we formulated above the simplification do not seem to be overly constraining for the animations

The main advantage of the decomposition is that no computational power must be invested in those portions of the model that do not require deformation but only transformation, and that isolating regions of rigid geometry makes room for potential optimization (such as exploiting hardware display lists) that cannot be applied to regions which must be deformed in every frame.

The fundamental idea of making a multi-resolution model compatible with deformation of a polygonal skin is therefore to decompose the skin into regions according to its deformation properties, i. e. the functional dependency on the skeleton, then apply LOD selection to each region individually. Influence of the bones on the skin is assigned on a per vertex base, so connected vertices with the same dependency on bones form one region. For simplicity, we allow a vertex to be dependent only on exactly one or two bones. Dependency on two bones is sufficient to construct ball joints, hinges and even sliding joints. A vertex that is dependent on only one bone is part of a rigid region which need not be deformed.

This decomposition is achieved in the same manner as described in section 4 and allows the designer/animator to maintain control over the local influence of deformation. A region will be either rigid or deformable. For articulated figures, rigid and deformable regions will usually interleave in the hierarchy, so that two rigid regions can always rely on a deformable region to connect them (Figure 3).

While it is principally possible to fully automate the decomposition into regions e.g. using distance to a joint as criterion whether a region is rigid or deformable, we found it more useful to let the animator perform this step manually using suitable modeling software. This approach may be more labor-intensive, but leaves control in the hands of the animator and yields more satisfactory results.

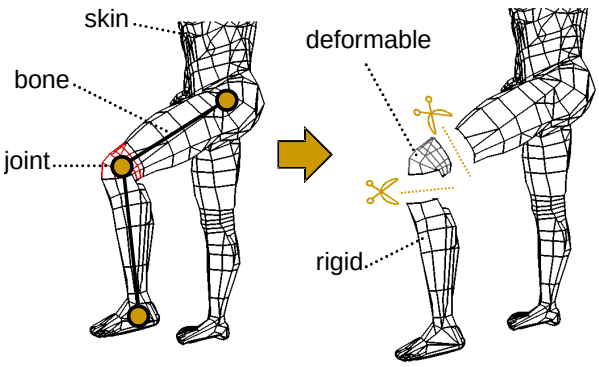

Figure 3: Decomposition of a deformable model

5.3 Real-time deformation algorithm

At runtime, a new position is computed for very deformable vertex of the skin from the position of the joints and bones, and *weights* (not to be confused with the weight from section 3) assigned to the vertices. The weights are a percentage used to describe how much each vertex is dependent on either bone. They are determined according to a scheme described in section 5.5.

The actual position of a vertex after deformation is computed using a simple linear interpolation scheme depicted in Figure 4: Each vertex has a position in the original undeformed mesh in Figure 4a. Let P_A and P_B be the positions of the vertex from the original mesh expressed in the local coordinate systems of the two bones *A* and *B*, respectively, and let *T* be transformation matrix defining the current position of two bones. Let further *s* be the vertex weight (in the range 0..1). Then the actual position *P* of the vertex is computed by a simple linear interpolation (Figure 4b):

$$P_d = s\,P_A + (1\text{-}s)\,P_B T$$

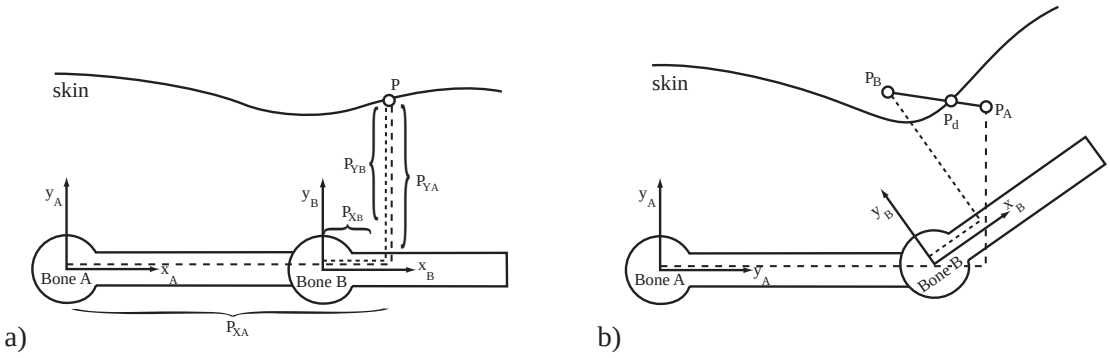

Figure 4: a) undeformed skin with underlying bones, b) deformed skin with linear interpolation operation

While numerous more sophisticated schemes exist in the computer animation literature that are superior in quality to the proposed deformation scheme, the associated computational cost makes them generally unsuitable for real-time rendering of large scenes. For a study, see [Stalpers & Overveld, 1999]. We have found the proposed method sufficient for our purposes, and leave a more advanced real-time deformation for future work. An example of the deformation is shown in Figure 5.

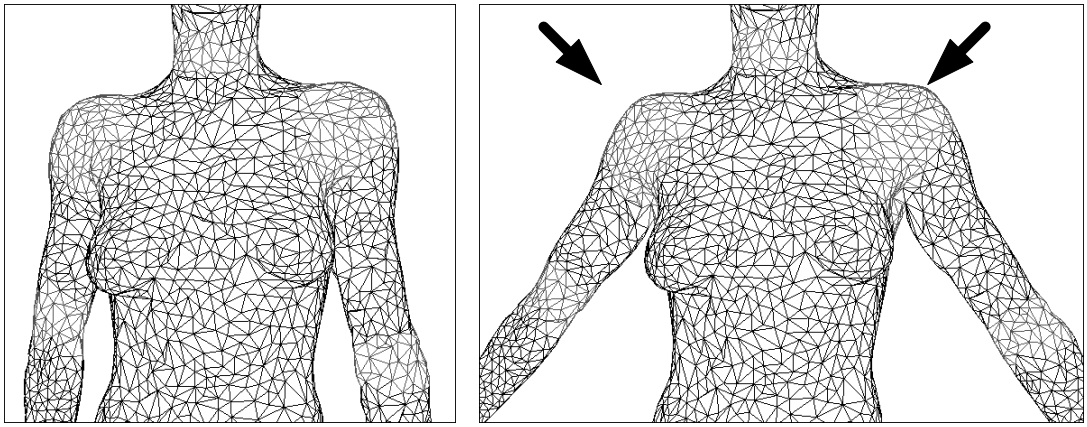

Figure 5: In this example, the shoulder of the articulated figure are deformed with a simple linear interpolation scheme

5.4 Runtime algorithm

At runtime, three steps must be carried out:

1. **LOD selection**: For each region, it is checked whether the currently selected LOD is a reasonable representation with respect to the current viewpoint. If a certain error threshold is exceeded, a new desired polygon count is computed and the polygonal cache is updated according to the corresponding data from the multiresolution sequence.
2. **Deformation**: Deformation is computed for the polygonal cache of a region if the associated bones have changed position since the last deformation. As the cache only contains the polygons of the current LOD, only those vertices which are actually used in the rendering are deformed, which can mean a significant saving in terms of computational effort. Sometimes, the LOD selection algorithms may later switch to a higher resolution involving more (undeformed) vertices, in which case the remaining vertices must be deformed as well. More often however, a new deformation will be computed together with a such a change in LOD, in which case the deformation must be re-computed in any case.
3. **Rendering**: After the necessary LOD selection and deformation has taken place, standard rendering is performed for the polygonal cache of each region.

The following pseudo code summarizes the procedure:

```
for every region R
  p = polgonBudget(projectedArea(R,eyepoint))
  Δp = p-currentPolygonBudget
  deformFlag = FALSE
  if Δp > ε
    extract Δp triangles from sequence(R) to cache(R)
    deformFlag = TRUE
  if Δp < ε
    remove Δp triangles from cache(R) according to sequence(R)
  if animated(bones(R)) or deformFlag
    for every vertex v referred by cache(R)
      deform v
  render cache(R)
```

Note that since every LOD refers only to a strict subset of the original vertices (no averages vertices are produced during LOD generation), the original table of vertices is stored only once for each object. All vertices in the multiresolution model are stored as references to this table. However, the deformed vertices resulting from step 2 above are temporarily stored in the region's cache and used for rendering, avoiding costly re-evaluation of deformation when possible.

5.5 Weight assignment

The weights of each vertex in a deformable region determine how much its position is influenced by each of the two underlying bones. Intuitively, vertices that are closer to one bone should be influenced more by that bone than the other. Border vertices – vertices contained in both the deformable and one of the bordering regions – therefore have a weight of 0 or 1. Between these maxima, the weight should gradually shift from 0 to 1.

The algorithm uses a shortest path search along the edges of the polygonal region as a heuristic to find the desired weight. With a prioritized breadth search using the summed length of visited edges as path length, the shortest path from one border to the other is computed. Weights along such a path are then assigned to vertices proportional to the ratio of the traveled path length to one bone to sum of path lengths (Figure 6).

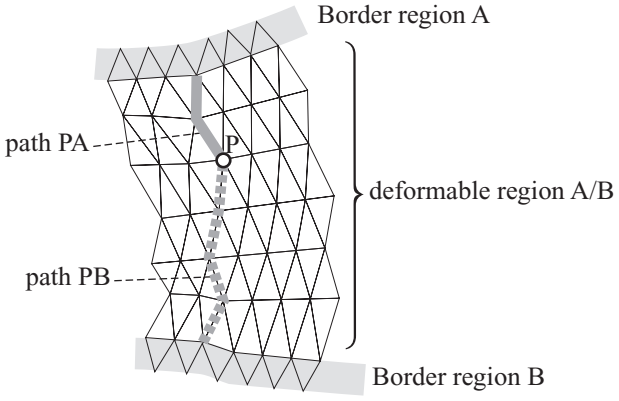

Figure 6: Vertex weight for the deformation are computed by shortest distance search from one border region to the other

The weight for vertex *P* in the figure above calculates as follows:

$$s = \frac{|PA|}{|PA| + |PB|}$$

Note that the vertex weights are precomputed during the modeling process and remain constant during runtime. Vertices from coarser LOD representations are always a subset of the original vertices, consequently the precomputed weights can be used.

6. Implementation

A prototype ASLODs modeler and rendering tool was implemented in C++ using the Open Inventor (OIV) graphics library [Strauss & Carey, 1992]. OIV provides facilities for scene graph management, which was extended to build the data structure to represent a skeleton, while the multi-resolution model itself was implemented with proprietary code [Hey, 1998]. Modeling and animation was performed in 3D Studio MAX, complemented with a variety of custom modeling tools. In particular, mesh segmentation was simply performed using mesh coloring in 3D Studio MAX.

No fundamental difference exists between a scene graph referring to rigid objects and a skeleton with a polygonal skin - both are essentially composed of a hierarchy of transformations. It is easily possible to combine the two types of models in the same hierarchy - certain sub-graphs of a scene can simply be marked as a skeleton associated with a particular skin, while other branches of the graph point to rigid objects. This has the additional advantage that deformable models can be placed anywhere in the scene and animated as a whole - even in groups - with conventional transformations, and that multi-resolution models can be intermixed with polygonal and other primitive objects.

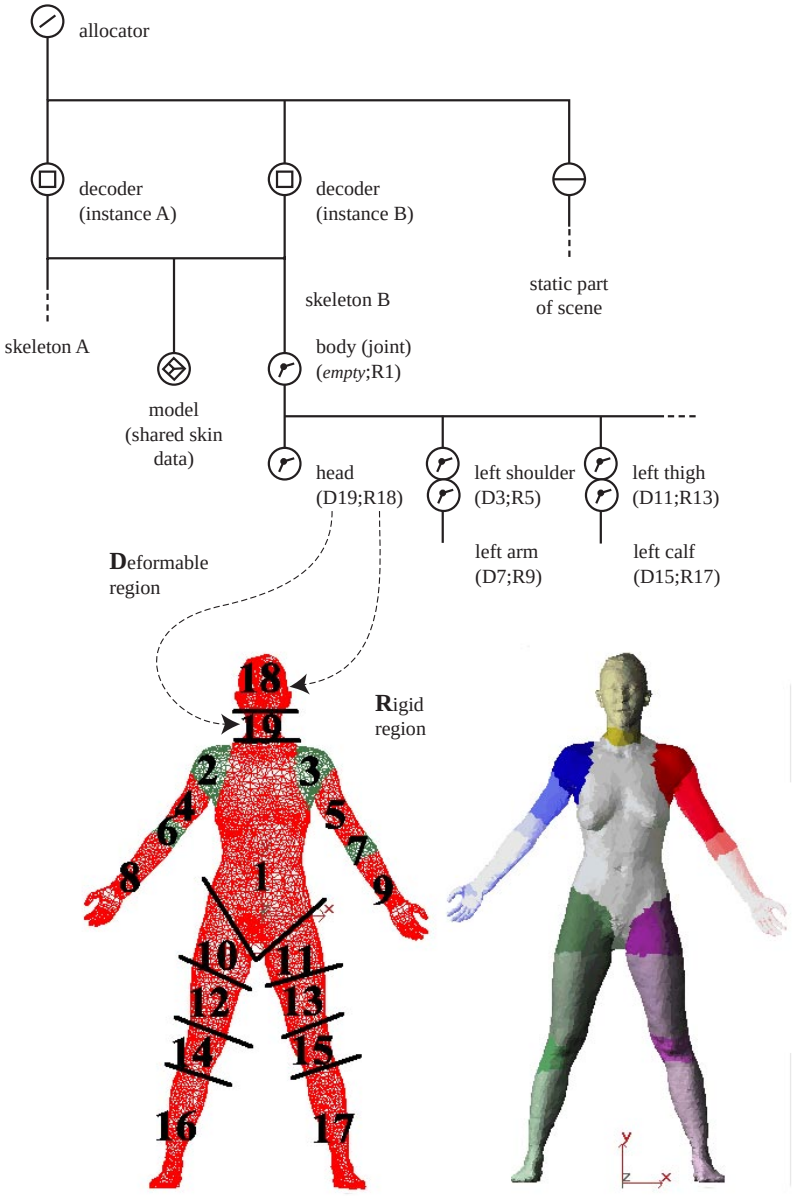

Figure 7: Modeling of an articulated mannequin - left: decomposition into rigid and deformable regions, right: shaded view of the resulting skin with color coded region number

By extending OIV's class hierarchy of scene graph nodes, it is thus possible to combine in the same scene graph conventional objects (non-polygonal and polygonal primitives) and deformable hierarchical

multi-resolution models. For the integration of multi-resolution models, the following new classes are necessary:

- ASLOD allocator node
- ASLOD controller node
- ASLOD model node
- ASLOD joint node

A scene graph hierarchy is built from group nodes (OIV's separators) with associated transformations. For a detailed example refer to Figure 7. Any sub-graph can have a *model* node as its local root, which means the hierarchy underneath represents a skeleton. A skeleton sub-graph does not use separators to built a hierarchy, but *joint* nodes. A joint node stores the associated transformation plus references to at most one rigid and at most one deformable region. For each of these regions, polygon caches exist to avoid re-extraction of the polygonal model for every frame.

The controller node refers to a proprietary object storing the multi-resolution *model* node. This object stores the original polygonal data as well as the multiresolution sequence for each region. The cache for the currently selected LOD must be stored in the controller node to allow multiple instances of the same multi-resolution model at different LODs and even different deformation states.

Finally, the *allocator* node sits at the root of the scene graph. Its tasks is to allocate the polygon budget to *all* objects by performing the computation outlined in section 3.

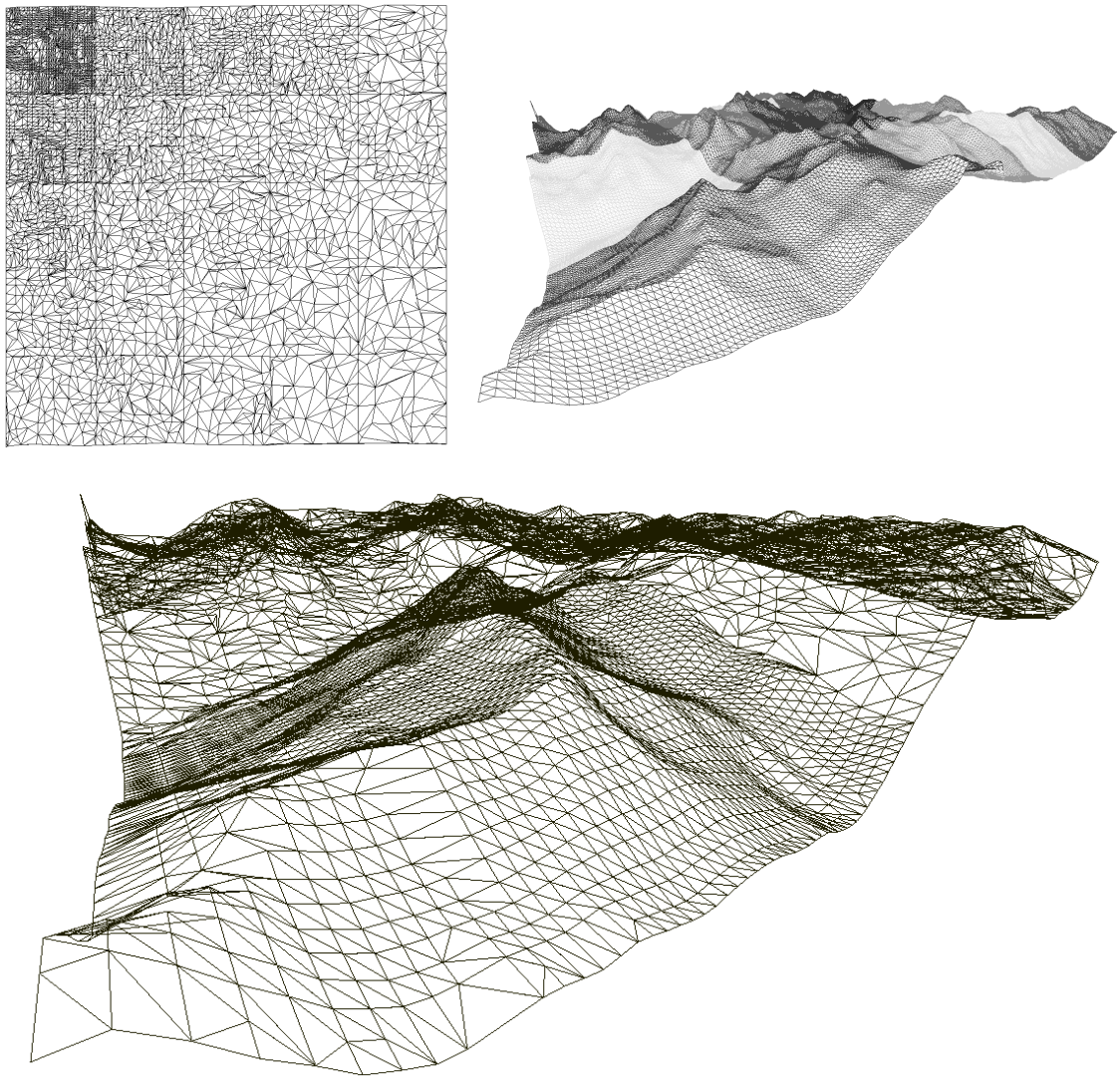

Figure 8: "Silvretta" mountain with coarse view-dependent rendering - (top right) original terrain model with 135200 triangles, (bottom) coarse-view dependent model with 10K triangles seen from corner, (d) coarse view-dependent model seen from top. Note how the terrain's upper left corner - the one closest to the viewpoint - has been assigned significantly more triangles.

7. Results

We demonstrate results for coarse view-dependent rendering and animation with examples for rigid and deformable models:

- The "Silvretta" model (an Austrian mountain, Figure 8) consists of 135200 triangles. It has been arbitrarily subdivided into a grid of 6 x 6 regions and rendered with a polygon budget of 10000 triangles to demonstrate the application of coarse-view dependence to large rigid models.
- The mannequin was created from a laser range scan and has 29799 triangles. It was decomposed into 19 regions (10 rigid, 9 deformable) and animated to perform a series of dancing steps. Various assemblies of mannequins were used to test how view-dependent LOD rendering, animation and deformation work in ensemble (Figure 10).

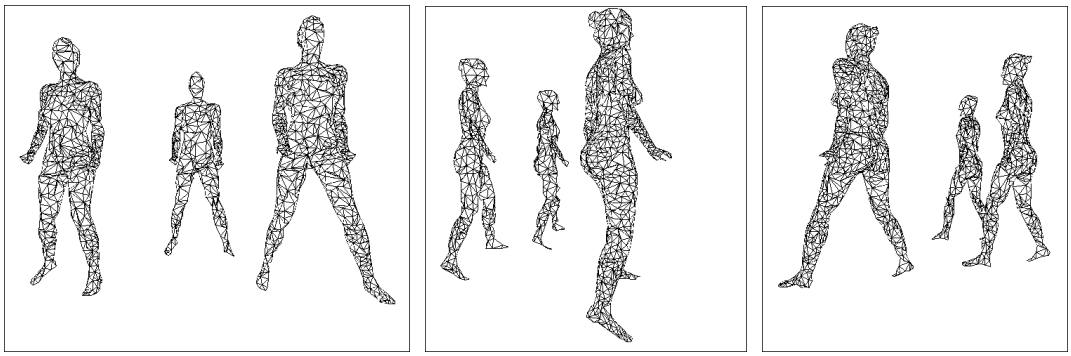

Figure 9: Three frames of the "aerobic class" animation sequence used for performance evaluation

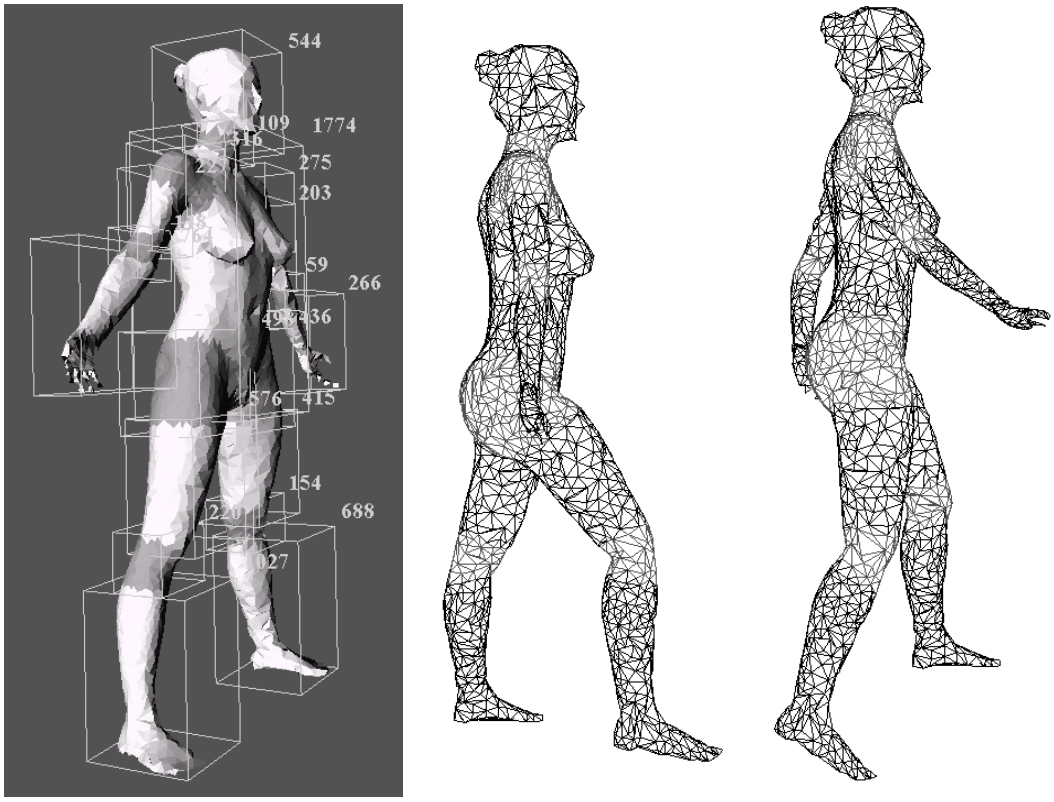

Figure 10: Left: shaded view of mannequin with overlaid bounding boxes of regions and associated polygon count, middle and right: two views from an animation sequence of the deformable model

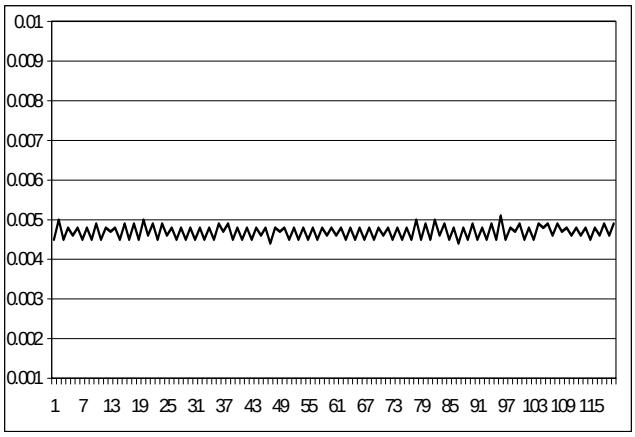

Figure 11: Rendering times for 120 frames of the "aerobic class" animation sequence - note how rendering time varies only little due to the constant polygon budget which is re-allocated as the scene changes

To evaluate performance, an "aerobic class" group of three mannequins was assembled and rendered with a constant budget of 5000 polygons. Figure 11 shows the rendering times for 120 frames of the animation sequence. The results demonstrate that rendering time is almost constant, although the allocation of polygons to the individual objects varies drastically over the sequence (Figure 9).

8. Conclusions and future work

We have presented ASLOD, a multi-resolution modeling and rendering system which opens up new directions towards integrating animation. The system allows the combination of progressive LOD models with conventional scene graphs. LODs can be selected for each object individually with a simple analytical run-time LOD selection strategy, which is able to allocate a given polygon budget over a large number of objects. We also propose to approximate the ideal view-dependent representation with coarse view-dependence that is based on a decomposition of large polygonal objects into multiple connected regions. This decomposition also enables multi-resolution rendering of organically deformable models animated with skeleton animation.

Future work will address several limitations:

- The current modeling procedure relies on external tools, which are far from ideal for the given task. An integrated modeling tools for region decomposition, skeleton construction, and vertex weight assignment is required to create more appealing models.
- An improved real-time deformation algorithm should be able to deal with more complex and appealing animations of deformable models
- Finally, our scheme of decomposition into rigid and deformable regions with precomputed multi-resolution models does not take into account the deformation itself. Consequently, deformations can only be moderate or the assumption regarding polygon size will fail as some polygons are enlarged and others shrunk or squeezed. A multi-resolution metric that can incorporate deformation is a challenging new research goal.

Acknowledgements

This work was sponsored by the Austrian Science Funds (*FWF*) under project no. P11392-MAT. Special thanks to Heinrich Hey and Erik Pojar for working on the implementation, to Gernot Schaufler and Michael Wimmer for valuable discussions, and to M. Eduard Gröller for proofreading. The mannequin model is courtesy of Cyberware.

Web information: http://www.cg.tuwien.ac.at/research/vr/aslods/

References

[Cignoni et al., 1998] P. Cignoni, C. Montani, R. Scopigno: A comparison of mesh simplification algorithms. Computers & Graphics, Vol. 22, No. 1, pp. 37-54, Feb. 1998.

[Cignoni et al., 1998b] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno: Zeta: A Resolution Modeling System. Graphical Models and Image Processing, Vol. 60, No.5, pp. 305-329, Sept. 1998.

[Clark, 1976] J. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. Communications of the ACM, 19(10), pp. 547-554, 1976.

[Floriani et al., 1997] L. De Floriani, P. Magillo, and E. Puppo. Building and Traversing a Surface at Variable Resolution. In Proceedings of IEEE Visualization'97, pp. 103-110. IEEE Press, 1997.

[Funkhouser & Sequin, 1993] Funkhouser T., C. Sequin. Adaptive Display Algorithm for Interactive Frame Rates During Visualisation of Complex Virtual Environments. Proceedings of SIGGRAPH'93, pp. 247-254, 1993.

[Garland & Heckbert, 1997] M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metrics. Proceedings of SIGGRAPH'97, pp. 209-215, 1997.

[Gross et al., 1995] M. Gross, R. Gatti, and O. Staadt. Fast Multiresolution Surface Meshing. Proceedings of Visualization'95, pp. 135-142, 1995.

[Guziec et al., 1998] A. Guziec, F. Lazarus, G. Taubin, W. Horn: Surface partitions for progressive transmission and display and dynamic simplification of polygonal surfaces. In Proceedings VRML'98, Monterey CA, February 16-19, pp. 25-32, 1998.

[Hartman & Wernecke, 1996] J. Hartman, J. Wernecke. VRML 2.0 Handbook. Addison-Wesley, 1996.

[Heckbert & Garland, 1997] P. Heckbert, M. Garland: Survey of polygonal surface simplification. SIGGRAPH'97 course notes, 1997.

[Hesina & Schmalstieg, 1998] G. Hesina, D. Schmalstieg: A Network Architecture for Remote Rendering. Proceedings of 2nd International Workshop on Distributed Interactive Simulation and Real Time Applications (DIS-RT'98), pp. 88-91, Montreal, Canada, July 19-20, 1998. Available as technical report TR-186-2-98-02 from ftp://ftp.cg.tuwien.ac.at/pub/TR/98/TR-186-2-98-02Paper.pdf.

[Hey, 1998] H. Hey: Animated Smooth Levels of Detail. Master's thesis, Vienna University of Technology, Austria, Jan. 1998.

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

[Hoppe, 1997] H. Hoppe. View-Dependent Refinement of Progressive Meshes. Proceedings of SIGGRAPH'97, 1997.

[Hoppe, 1998] H. Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. Proceedings of IEEE Visualization'98, pp. 35-50, 1998.

[Klein et al., 1998] R. Klein: Multiresolution Representation for Surface Meshes Based on the Vertex Decimation Method. Computers & Graphics, Vol. 22. No. 1, pp. 13-26, 1998.

[Lau et al., 1998] R. Lau, M. Green, D. To, and J. Wong. Real-Time Continuous Multiresolution Method for Models of Arbitrary Topology. Presence, 7(1), pp. 22-35, 1998.

[Lindstrom et al., 1996] 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.

[Luebke & Erikson, 1997] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Objects. Proceedings of SIGGRAPH'97, pp. 199-208, 1997.

[Mason & Blake, 1997] Ashton E. W. Mason and Edwin Blake. Automatic Hierarchical Level of Detail Optimization in Computer Animation. Proceedings EUROGRAPHICS'97, pp. 191-200,.

[nvidia, 1999] GeForce 256 product literature. Available from nvidia's web site http://www.nvidia.com

[Rohlf & Helman, 1994] J. Rohlf and J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. Proceedings of SIGGRAPH'94, pp. 381-388, 1994.

[Rossignac, 1997] Simplification a. Compression of 3D Scenes, J. Rossignac, Eurographics Tutorial,1997

[Schmalstieg & Schaufler, 1997] Schmalstieg D., G. Schaufler: Smooth Levels of Detail. Proceedings of IEEE VRAIS'97, pp. 12-19, Albuquerque, New Mexico, March 1-5, 1997.

[Schmalstieg & Tobler, 1999] D. Schmalstieg, R. F. Tobler. Fast Projected Area Computation for 3D Bounding Boxes. To appear in: Journal of Graphics Tools, A. K. Peters, Ltd., 1999. Also available as technical report TR-186-2-99-05, Vienna University of Technology, 1999, from ftp://ftp.cg.tuwien.ac.at/pub/TR/99/TR-186-2-99-05Paper.pdf.

[Stalpers & Overveld, 1999] M. Stalpers and C. van Overveld: Deforming Geometric Models Based on a Polygonal Skeleton Mesh. ACM Journal of Graphics Tools, 2(3), 1999.

[Strauss & Carey, 1992] P. Strauss and R. Carey. An Object Oriented 3D Graphics Toolkit. Proceedings of SIGGRAPH'92, pp. 341-347, 1992.

[Wimmer & Schmalstieg, 1998] M. Wimmer, D. Schmalstieg: Load Balancing for Smooth Levels of Detail. Technical report TR-186-2-98-31, Vienna University of Technology, 1998. Available at ftp://ftp.cg.tuwien.ac.at/pub/TR/99/TR-186-2-98-31Paper.pdf

[Xia et al., 1997] J. Xia, J. El-Sana, and A. Varshney. Adaptive Real-Time Level-of-Detail Based Rendering for Polygonal Models. IEEE Transactions on Visualization and Computer Graphics, 3(2), pp. 171-183, 1997.

[Zorin et al., 1997] Interactive Multiresolution Mesh Editing. Proceedings of SIGGRAPH'97, pp. 259-268, 1997.