# Dynamic View-Dependent Simplification for Polygonal Models

Julie C. Xia Amitabh Varshney

State University of New York at Stony Brook

## Abstract

We present an algorithm for performing view-dependent simplifications of a triangulated polygonal model in real-time. The simplifications are dependent on viewing direction, lighting, and visibility and are performed by taking advantage of image-space, object-space, and frame-to-frame coherences. A continuous level-of-detail representation for an object is first constructed off-line. This representation is then used at run-time to guide the selection of appropriate triangles for display. The list of displayed triangles is updated incrementally from one frame to the next. Our approach is more effective than the current level-of-detail-based rendering approaches for most scientific visualization applications where there are a limited number of highly complex objects that stay relatively close to the viewer.

## 1 Introduction

The scientific visualization and virtual reality communities have always faced the problem that their "desirable" visualization dataset sizes are one or more orders of magnitude larger than what the hardware can display at interactive rates. Recent research on simplification of polygonal objects to generate multiresolution hierarchies has been motivated by attempts to bridge the gap between the desired and the actual hardware performance, through algorithmic and software techniques.

Research on simplification of general three-dimensional polygonal objects (non-convex, non-terrain, possibly high genus) has spanned the entire gamut of highly local to global algorithms, with several approaches in between that have both local and global steps. Local algorithms work by applying a set of local rules, which primarily work under some definition of a *local neighborhood*, for simplifying an object. The local rules are iteratively applied under a set of constraints and the algorithm terminates when it is no longer possible to apply the local rule without violating some constraint. The global algorithms optimize the simplification process over the whole object, and are not necessarily limited to the small neighborhood regions on the object. Some of the local approaches have been – vertex deletion by Schroeder *et al* [21], vertex collapsing by Rossignac and Borrel [20], edge collapsing by Hoppe *et al* [17] and Guéziec [11], triangle collapsing by Hamann [12], and polygon merging by Hinker and Hansen [15]. Some of the global approaches have been – redistributing vertices over the surface by Turk [22], minimizing global energy functions by Hoppe *et al* [17], using simplification envelopes by Varshney [23] and Cohen *et al* [4], and wavelets by DeRose *et al* [7]. The issue of preservation or simplification of the genus of the object is independent of whether an algorithm uses local rules, or global rules, or both, to simplify. Recent work by He *et al* [13] provides a method to perform a controlled simplification of the genus of an object.

Simplification algorithms such as those mentioned above are iteratively applied to obtain a hierarchy of successively coarser approximations to the input object. Such multiresolution hierarchies have been used in level-of-detail-based rendering schemes to achieve higher frame update rates while maintaining good visual realism. These hierarchies usually have a number of distinct levels of detail, usually 5 to 10, for a given object. At run time, the perceptual importance of a given object in the scene is used to select its appropriate level of representation from the hierarchy [3, 5, 6, 9, 19, 18]. Thus, higher detail representations are used when the object is perceptually more important and lower detail representations are used when the object is perceptually less significant. Transitions from one level of detail to the next are typically based on simple image-space metrics such as the ratio of the image-space area of the object (usually implemented by using the projected area of the bounding box of the object) to the distance of the object from the viewer.

Previous work, as outlined above, is well-suited for virtual reality walkthroughs and flythroughs of large and complex structures with several thousands of objects. Examples of such environments include architectural buildings, airplane and submarine interiors, and factory layouts. However, for scientific visualization applications where the goal often is to visualize one or two highly detailed objects at close range, most of the previous work is not directly applicable. For instance, consider a biochemist visualizing the surface of a molecule or a physician inspecting the iso-surface of a human head extracted from a volume dataset. It is very likely during such a visualization session, that the object being visualized will not move adequately far away from the viewer to allow the rendering algorithm to switch to a lower level of detail. What is desirable in such a scenario is an algorithm that can allow several different levels of details to co-exist across different regions of the same object. Such a scheme needs to satisfy the following two important criteria:

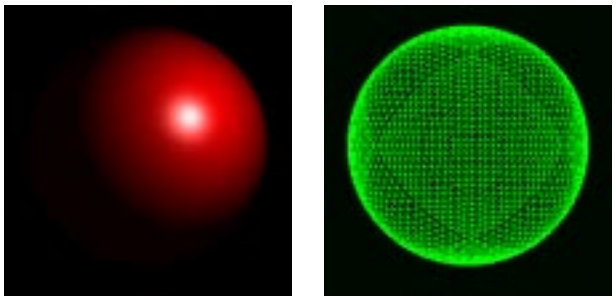
- It should be possible to select the appropriate levels of detail across different regions of the same object in real time.
- Different levels of detail in different regions across an object should merge seamlessly with one another without introducing any cracks and other discontinuities.

In this paper we present a general scheme that can construct such seamless and adaptive level-of-detail representations on-the-fly for polygonal objects. Since these representations are view-dependent, they take advantage of view-dependent illumination, visibility, and frame-to-frame coherence to maximize visual realism and minimize the time taken to construct and draw such objects. An example of using our approach is shown in Figure 1.

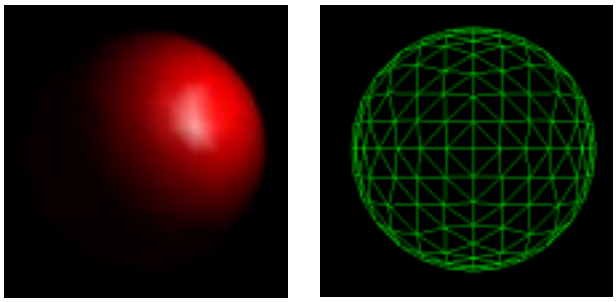
## 2 Previous Work

Adaptive levels of detail have been used in terrains by Gross *et al* [10] by using a wavelet decomposition of the input data samples. Wavelet space filters are defined that allow changes to the quality of the surface approximations in locally-defined regions. Thus, the level of detail around any region can adaptively refine in real-time. This work provides a very elegant solution for terrains and other datasets that are defined on a regular grid.

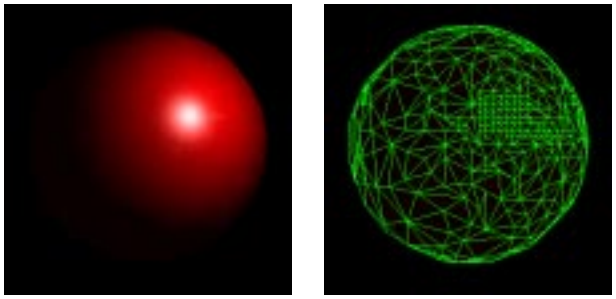
Contact address: Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, Email: varshney@cs.sunysb.edu



(a) Sphere with 8192 triangles (uniform LOD)



(b) Sphere with 512 triangles (uniform LOD)



(c) Sphere with 537 triangles (adaptive LOD)

Figure 1: Uniform and Adaptive Levels of Detail

Some of the previous work in the area of general surface simplification has addressed the issue of adaptive approximation of general polygonal objects. Turk [22] and Hamann [12] have proposed curvature-guided adaptive simplification with lesser simplification in the areas of higher surface curvature. In [23, 4], adaptive surface approximation is proposed with different amounts of approximation over different regions of the object. Guéziec [11] proposes adaptive approximation by changing the tolerance volume in different regions of the object. However in all of these cases, once the level of approximation has been fixed for a given region of the object, a discrete level of detail corresponding to such an approximation is statically generated. No methods have been proposed there that allow free intermixing of different levels of detail across an object in real time in response to changing viewing directions.

Work on surface simplification using wavelets [7, 8] and progressive meshes [16] goes a step further. These methods produce a continuous level-of-detail representation for an object in contrast to a set of discrete number of levels of detail. In particular, Hoppe [16] outlines a method for selective refinement – i.e. refinement of a particular region of the object based upon view frustum, silhouette edges, and projected screen-space area of the faces. Since the work on progressive meshes by Hoppe [16] is somewhat similar to our work we overview his method next and discuss how our method extends it.

Progressive meshes offer an elegant solution for a continuous resolution representation of polygonal meshes. A polygonal mesh $\hat{M} = M^k$ is simplified into successively coarser meshes $M^i$ by applying a sequence of edge collapses. An edge collapse transformation and its dual, the vertex split transformation, is shown in Figure 2.

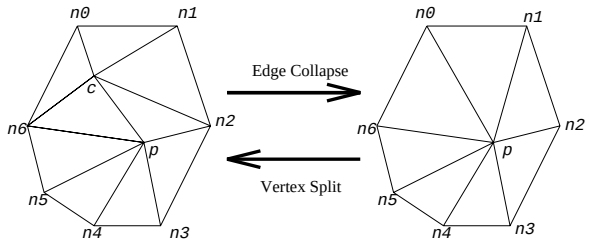



Figure 2: Edge Collapse and Vertex Split

Thus, a sequence of $k$ successive edge collapse transformations yields a sequence of successively simpler meshes:

$$M^k \overset{collapse_{k-1}}{\rightarrow} M^{k-1} \overset{collapse_{k-2}}{\rightarrow} \ldots M^1 \overset{collapse_0}{\rightarrow} M^0 \quad (1)$$

We can retrieve the successively higher detail meshes from the simplest mesh $M^0$ by using a sequence of vertex-split transformations that are dual to the corresponding edge collapse transformations:

$$M^0 \overset{split_0}{\rightarrow} M^1 \overset{split_1}{\rightarrow} \ldots M^{k-1} \overset{split_{k-1}}{\rightarrow} (\hat{M} = M^k) \quad (2)$$

Hoppe [16] refers to $(M^0, \{split_0, split_1, \ldots, split_{k-1}\})$ as a *progressive mesh* representation. Progressive meshes present a novel approach to storing, rendering, and transmitting meshes by using a continuous-resolution representation. However we feel that there is some room for improvement in adapting them for performing selective refinement in an efficient manner. In particular, following issues have not yet been addressed by progressive meshes:

- The sequence of edge collapses is aimed at providing good approximations $M^i$ to $(\hat{M} = M^k)$. However, if a sequence of meshes $M^i$ are good approximations to $\hat{M}$ under some distance metric, it does not necessarily mean that they also provide a "good" sequence of edge collapse transformations for selective refinement. Let us consider a two-dimensional analogy of a simple polygon as shown in Figure 3. Assume that vertices $v_0, v_6, v_7$, and $v_8$ are "important" vertices (under say some perceptual criteria) and can not be deleted. An approach that generates approximations based on minimizing distances to the original polygon will collapse vertices in the order $v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_3 \rightarrow v_4, v_4 \rightarrow v_5, v_5 \rightarrow v_6$ to get a coarse polygon $(v_0, v_6, v_7, v_8)$. Then if selective refinement is desired around vertex $v_1$, vertices $v_6, v_5, v_4, v_3, v_2$ will need to be split in that order before one can get to vertex $v_1$. An approach that was more oriented towards selective refinement might have collapsed $v_1 \rightarrow v_2, v_3 \rightarrow v_4, v_5 \rightarrow v_6, v_2 \rightarrow v_4, v_4 \rightarrow v_6$ for better adaptive results, even though the successive approximations are not as good as the previous ones under the distance metric.

- Since the edge collapses are defined in a linear sequence, the total number of child links to be traversed before reaching the desired node is $O(n)$.

- No efficient method for incrementally updating the selective refinements from one frame to the next is given. The reverse problem of selective refinement – selective simplification too is not dealt with.

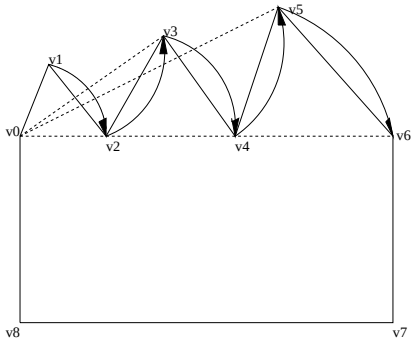



Figure 3: Good versus Efficient Selective Simplification

In this paper we provide a solution to the above issues with the aim of performing real-time adaptive simplifications and refinements. We define a criterion for performing edge collapses that permits adaptive refinement around any vertex. Instead of constructing a series of sequential edge collapses we construct a *merge tree* over the vertices of mesh $\hat{M}$ so that one can reach any child vertex in $O(\log n)$ links. We then describe how one can perform incremental updates within this tree to exploit frame-to-frame coherence, view-dependent illumination, and visibility computations.

# 3 Image-Space-Guided Simplification

Level-of-detail-based rendering has thus far emphasized object-space simplifications with minimal feedback from the image space. The feedback from the image space has been in the form of very crude heuristics such as the ratio of the screen-space area of the bounding box of the object to the distance of the object from the viewer. As a result, one witnesses coarse image-space artifacts such as the distracting "popping" effect when the object representation changes from one level of detail to the next [14]. Attempts such as alpha-blending between the old and the new levels of detail during such transitions serve to minimize the distraction at the cost of rendering two representations. However alpha blending is not the solution to this problem since it does not address the real cause – lack of sufficient image-space feedback to select the appropriate local level of detail in the object space; it merely tries to cover-up the distracting artifacts.

Increasing the feedback from the image space allows one to make better choices regarding the level of detail selection in the object-space. We next outline some of the ways in which image-space feedback can influence the level of detail selection in the object-space:

- **Local Illumination:** Increasing detail in a direction perpendicular to, and proportional to, the illumination gradient across the surface is a good heuristic [1]. This allows one to have more detail in the regions where the illumination changes sharply and therefore one can represent the highlights and the shadows well. Since surface normals play an important role in local illumination, one can take advantage of the coherence in the surface normals to build a hierarchy over a continuous resolution model that allows one to capture the local illumination effects well.

- **Screen-Space Projections:** Decision to keep or collapse an edge should depend upon the length of its screen-space projection instead of its object-space length. At a first glance this might seem very hard to accomplish in real-time since this could mean checking for the projected lengths of all edges at every frame. However, usually there is a significant coherence in the ratio of the image-space length to the object-space length of edges across the surface of an object and from one frame to the next. This makes it possible to take advantage of a hierarchy built upon the the object-space edge lengths for an object.

- **Visibility Culling:** During interactive display of any model there is usually a significant coherence between the visible regions from one frame to the next. This is especially true of the back-facing polygons that account for almost half the total number of polygons and do not contribute anything to the visual realism. A hierarchy over a continuous resolution representation of an object allows one to significantly simplify the invisible regions of an object, especially the back-facing ones.

- **Silhouette boundaries:** Silhouettes play a very important role in perception of detail. Screen-space projected lengths of silhouette edges (i.e., edges of whose adjacent triangles is visible and the other is invisible), can be used to very precisely quantify the amount of smoothness of the silhouette boundaries. Again, a hierarchy built upon a continuous-resolution representation of a object allows one to do this efficiently.

# 4 Construction of Merge Tree

We would like to create a hierarchy that provides us a continuous-resolution representation of an object and allows us to perform real-time adaptive simplifications over the surface of an object based upon the image-space feedback mechanisms mentioned in Section 3. Towards this end we implement a *merge tree* over the vertices of the original model. In our current implementation, the merge tree stores the edge collapses in a hierarchical manner. However, as we discuss in Section 7 the concept of a merge tree is a very general one and it can be used with other local simplification approaches as well. Note that the merge tree construction is done as an off-line preprocessing step before the interactive visualization.

## 4.1 Basic Approach

In Figure 2, the vertex $c$ is merged with the vertex $p$ as a result of collapsing the edge $(pc)$. Conversely, during a vertex split the vertex $c$ is created from the vertex $p$. We shall henceforth refer to $c$ as the child vertex of the parent vertex $p$. The merge tree is constructed upwards from the high-detail mesh $\hat{M}$ to a low-detail mesh $M^0$ by storing these parent-child relationships in a hierarchical manner over the surface of an object.

At each level $l$ of the tree we determine parent-child relationships amongst as many vertices at level $l$ as possible. In other words, we try to determine all vertices that can be safely merged based on criterion defined in Section 4.3. The vertices that are determined to be the children remain at level $l$ and all the other vertices at level $l$ are promoted to level $l+1$. Note that the vertices promoted to level $l+1$ are a proper superset of the parents of the children left behind at level $l$. This is because there are vertices at level $l$ that are neither parents nor children. We discuss this in greater detail in the context of *regions of influence* later in this section. We apply the above procedure recursively at every level until either (a) we are left with a user-specified minimum number of vertices, or (b) we cannot establish any parent-child relationships amongst the vertices at a given level. Case (b) can arise because in determining a parent-child relationship we are essentially collapsing an edge and not all edge collapses are considered legal. For a detailed discussion on legality of edge collapses the interested reader can refer to [17].

Since in an edge collapse only one vertex merges with another, our merge tree is currently implemented as a binary tree. In Figure 4, the three highest detail levels for a sphere are shown colored by the corresponding vertices in red, green, and yellow. The remaining levels are shown colored in blue.

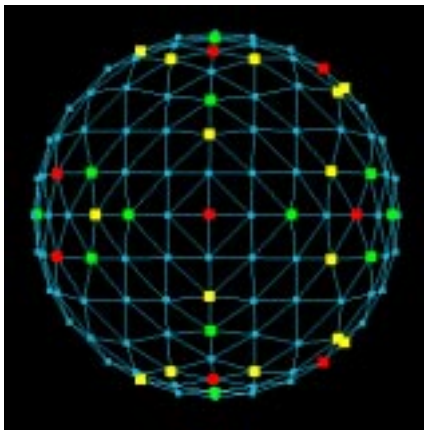


Figure 4: Three Levels in a Merge Tree for a Sphere

To construct a balanced merge tree we note that the effects of an edge collapse are local. Let us define the *region of influence* of an edge $(v_0, v_1)$ to be the union of triangles that are adjacent to either $v_0$ or $v_1$ or both. The region of influence of an edge is the set of triangles that can change as an edge is gradually collapsed to a vertex, for example, in a morphing. Thus, in Figure 2 as vertex $c$ merges to vertex $p$, (or $p$ splits to $c$), the changes to the mesh are all limited to within the region of influence of edge $(pc)$ enclosed by $n_0, n_1, \ldots n_6$. Note that all the triangles in region of influence will change if vertices $p$ and $c$ are merged to form an intermediate vertex, say $(p + c)/2$.

To create a reasonably balanced merge tree we try to collapse as many edges as possible at each level such that there are no common triangles in their respective regions of influence. Since this step involves only local checks, we can accomplish this step in time linear in the number of triangles at this level. If we assume that the average degree (i.e. the number of neighboring triangles) of a vertex is $6$, we can expect the number of triangles in an edge's region of influence to be $10$. After the collapse this number of triangles reduces to $8$. Thus the number of triangles can be expected to reduce roughly by a factor of $4/5$ from a higher-detail level to a lower-detail level. Thus the total time to build the tree is given by $n + \frac{4n}{5} + \frac{16n}{25} + \ldots = O(n)$.

To decide in what order to collapse the edges, we sort the edges by their edge lengths and collapse the shortest edges first. Collapsing an edge causes the neighboring edges to change their lengths. However as mentioned above, since changes are local we can maintain the sorted edge lengths in a heap for efficient updates.

## 4.2 Storing Subtree Attribute Ranges

To allow real-time refinement and simplification we can store at every parent node (i.e. a node that splits off a child vertex) of the merge tree, a range of scalar and vector attributes of the children in the subtree below it. Then image-space feedback can be used to determine if this range of scalar and vector attributes merits a refinement of this node or not. We explain this process of incremental refinement and simplification in greater details in Section 5.1.

In our current implementation every merge tree node $v$ stores only the Euclidean distances to its child and parent that determine when $v$'s child will merge into $v$ and when $v$ will merge into its parent. These distances are built up during the merge tree creation stage. However, we plan to extend this to store the surface normal and color ranges also, so that the incremental simplification and refinement stages can take advantage of the coherences across these vector attributes.

## 4.3 Merge Tree Dependencies

By using techniques outlined in Section 5.1, one can determine which subset of vertices is sufficient to reconstruct an adaptive level-of-detail for a given object. However, it is not simple to define a triangulation over these vertices and guarantee that the triangulation will not "fold back" on itself or otherwise represent a non-manifold surface (even when the original was not so). Figure 5 shows an example of how an undesirable folding in the adaptive mesh can arise even though all the edge collapses that were determined statically were correct. $A$ shows the initial state of the mesh. While constructing the merge tree, we first collapsed vertex $v_2$ to $v_1$ to get mesh $B$ and then collapsed vertex $v_3$ to $v_4$ to get mesh $C$. Now suppose at run-time we determined that we needed to display vertices $v_1, v_2$, and $v_4$ and could possibly collapse vertex $v_3$ to $v_4$. However, if we collapse $v_3$ to $v_4$ directly, as in mesh D, we get a mesh fold where there should have been none. Checks for such conditions are too expensive to be performed at run-time.

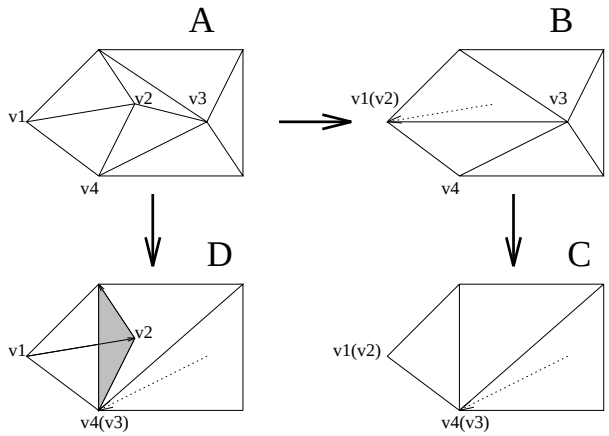



Figure 5: Mesh Folding Problem

To solve the above problem we introduce the notion of dependencies amongst the nodes of a merge tree. Thus, the collapse of an edge $e$ is permitted only when all the vertices defining the boundary of the region of influence of the edge $e$ exist and are adjacent to the edge $e$. As an example, consider Figure 2. Vertex $c$ can merge with vertex $p$ only when the vertices $n_0, n_1, \ldots, n_k$ exist and are adjacent to $p$ and $c$. From this we determine the following edge collapse dependencies, restricting the level difference between adjacent vertices:

1. $c$ can collapse to $p$, only when $n_0$, $n_1$, …, $n_k$ are present as neighbors of $p$ and $c$ for display.

2. $n_0, n_1, \ldots, n_k$ can not merge with other vertices, unless $c$ first merges with $p$.

Similarly, to make a safe split from $p$ to $p$ and $c$, we determine the following vertex split dependency:

1. $p$ can split to $c$ and $p$, only when $n_0, n_1, \ldots, n_k$ are present as neighbors of $p$ for display.

2. $n_0, n_1, \ldots, n_k$ can not split, unless $p$ first splits to $p$ and $c$.

The above dependencies are followed during each vertex-split or edge collapse during real-time simplification. These dependencies are easily identified and stored in the merge tree during its creation. Considering Figure 5 again, we can now see that collapse of vertex $v_3$ to $v_4$ depends upon the adjacency of vertex $v_1$ to $v_3$. If vertex $v_2$ is present then $v_1$ will not be adjacent to $v_3$ and therefore $v_3$ will not collapse to $v_4$. Although having dependencies might sometimes give lesser simplification than otherwise, it does have the advantage of eliminating the expensive floating-point run-time checks entirely. The basic idea behind merge tree dependencies has a strong resemblance to creating *balanced subdivisions* of quad-trees as presented by Baum *et al* in [2] where only a gradual change is permitted from regions of high simplifications to low simplifications. Details of how these merge tree dependencies are used during run-time are given in Section 5.1.

# 5 Real-Time Triangulation

Once the merge tree with dependencies has been constructed off-line it is easy to construct an adaptive level-of-detail mesh representation during run-time. Real-time adaptive mesh reconstruction involves two phases – determination of vertices that will be needed for reconstruction and determination of the triangulation amongst them. We shall refer to the vertices selected for display at a given frame as *display vertices* and triangles for display as *display triangles*. The phases for determination of display vertices and triangles are discussed next.

## 5.1 Determination of display vertices

We next describe how we take into account the screen-space projections in determining which regions of an object to simplify more and which ones less. The heuristics for determining lighting, visibility culling, and silhouette boundaries are similar. As mentioned earlier, every merge tree node $v$ stores a Euclidean distance for splitting a vertex to its child as well as the distance at which it will merge to its parent. The former is called the *downswitch distance* and the latter is called the *upswitch distance*. If the maximum possible screen-space projection of the downswitch distance at the vertex $v$ in the object space is greater than some pre-set threshold $T$, we permit refinement at $v$ and recursively check the children of $v$. However, if the maximum possible screen-space projection of the upswitch distance at $v$ in the object space is less than the threshold $T$, it means that this region occupies very little screen space and can be simplified, so we mark $v$ as *inactive* for display. We follow this procedure and select all those vertices for display that either (a) are leaf nodes and none of their parents have been marked as inactive, or (b) have their immediate child marked as inactive. This determines the initial list of vertices selected for display.

We then follow the merge dependencies from the initial list of display vertices to select the final set of display vertices in the following manner. If a vertex $v$ is in the initial list of display vertices and for it to be created (via a vertex split), the vertices $v_{d_0}, v_{d_1}, \ldots, v_{d_k}$ had to be present, we add the vertices $v_{d_0}, v_{d_1}, \ldots, v_{d_k}$ to the list of display vertices and recursively consider their dependencies. We continue this process until no new vertices are added.

When determining the vertices for display in frame $i+1$ we start from the vertex list for display used in frame $i$. We have found a substantial frame-to-frame coherence and the vertex display list does not change substantially from one frame to the next. There are minor local changes in the display list on account of vertices either refining or merging with other vertices. These are easily captured by either traversing the merge tree up or down from the current vertex position. The scalar and vector attribute ranges stored in merge tree nodes can be used to guide refinements if the difference in the display vertex lists from one frame to the next becomes non-local for any reason. We compute the list of display vertices for first frame by initializing the list of display vertices for frame $0$ to be all the vertices in the model and then proceeding as above.

## 5.2 Determination of display triangles

If the display triangles for frame $i$ are known, determination of the display triangles for frame $i+1$ proceeds in an interleaved fashion with the determination of display vertices for frame $i+1$ from frame $i$. Every time a display vertex of frame $i$ merges in frame $i+1$ we simply delete and add appropriate triangles to the list of display triangles as shown in Figure 6. The case where a display vertex in frame $i$ splits for frame $i+1$ is handled analogously. Incremental determination of display triangles in this manner is possible because of the dependency conditions mentioned in Section 4.3. The list of display triangles for the first frame is obtained by initializing the list for frame $0$ to be all the triangles in the model and then following the above procedure.

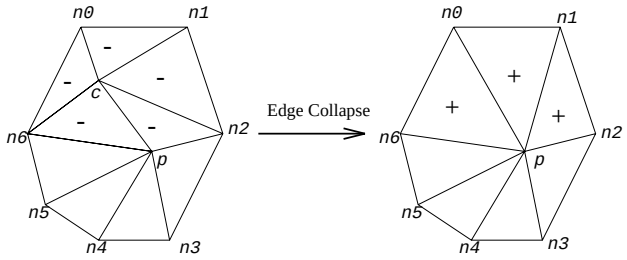



Figure 6: Display Triangle Determination

# 6 Results and Discussion

We have tried our implementation on several large triangulated models and have achieved encouraging results. These are summarized in Table 1 The images of sphere, crambin, and bunny models that were produced for the above times are shown in Figures 1, 7 and 8 respectively. All of the above timings are in milliseconds on a Silicon Graphics Indigo2 Impact with a 250 MHz R4400 processor and 128MB RAM. It is easy to see that the time to traverse the merge tree and construct the list of triangles to be displayed from frame to frame is fairly small. This is because of our incremental computations that exploit image-space, object-space, and frame-to-frame coherences. The above times hold as the user moves the model, or the lights around. The triangulation of the model changes dynamically to track the highlights as well as the screen-space projections of the faces.

As can be seen from the merge tree depths, the trees are not perfectly balanced. However, they are still within a small factor of the optimal depths. This factor is the price that has to be paid to incorporate dependencies and avoid the expensive run-time floating-point checks for ensuring good triangulations. For each dataset, we continued the merge tree construction till $8$ or fewer vertices were left. As expected, the factor by which the number of vertices decreases from one level to the next tapers off as we reach lower-detail levels since there are now fewer alternatives left to counter the remaining dependency constraints. As an example, for sphere, only $64$ vertices were present at level $30$ and it took another $12$ levels to bring down the number to $8$. If the tree depth becomes a concern one can stop sooner, trading-off the tree traversal time for the display time.

| Dataset | Uniform Resolution | | Adaptive Resolution | | | | |
|---|---|---|---|---|---|---|---|
| | Number of Tris | Display Time (msec) | Number of Display Tris | Number of Tree Levels | Tree Traversal (msec) | Display Time (msec) | Total Time (msec) |
| Sphere | 8192 | 36 | 537 | 42 | 6 | 3 | 9 |
| Bunny | 69451 | 420 | 3615 | 65 | 90 | 20 | 110 |
| Crambin | 109955 | 530 | 7433 | 61 | 56 | 49 | 105 |
| Phone | 165963 | 1020 | 4351 | 63 | 29 | 36 | 65 |

Table 1: Adaptive Level of Detail Generation Times

An interesting aspect of allowing dependencies in the merge tree is that one can now influence the characteristics of the run-time triangulation based upon static edge-collapse decisions during preprocessing. As an example, we have implemented avoidance of slivery (long and thin) triangles in the run-time triangulation. As Guéziec [11], we quantify the quality of a triangle with area $a$ and lengths of the three sides $l_0$, $l_1$, and $l_2$ based on the following formula:

$$\text{Quality} = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2} \tag{3}$$

Using Equation 3 the quality of a degenerate triangle evaluates to $0$ and that of an equilateral triangle to $1$. We classify all edge collapses that result in slivery triangles to be invalid, trading-off quantity (amount of simplification) for quality.

## 7 Conclusions and Future Work

We have outlined a simple approach to maintain dynamically adaptive level of detail triangulations. Crucial to this approach is the notion of merge trees that are computed statically and are used during run-time to take advantage of the incremental changes in the triangulation. In our current implementation we are using the method of edge collapses. However the idea behind merge trees is pretty general and can be used in conjunction with other local heuristics for simplification such as vertex deletion and vertex collapsing. We plan to study some of these other heuristics in the future and compare them with our current implementation that uses edge collapses.

At present we do not store surface normal or color ranges at the nodes of the merge tree. Storing and using these should improve the quality of the visualizations produced using merge trees even more. Also of some interest will be techniques that create better balanced merge trees while still incorporating dependencies. We plan to investigate these issues further.

Of course, our approach also makes dynamically-specified manual simplifications possible, where the user can interactively specify the amounts of approximation desired at various regions of the object. Using this, certain parts of the object can be rendered at lower or higher details than otherwise. However, in this paper we have only considered automatic object simplifications during interactive display.

## Acknowledgements

We would like to acknowledge several useful discussions with Arie Kaufman and Greg Turk. We would like to thank Greg Turk and Marc Levoy for generously sharing their models of the phone and the bunny. We deeply appreciate the thorough review and detailed and insightful comments by the anonymous referees. This work has been supported in part by the National Science Foundation CAREER award CCR-9502239.

## References

[1] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, No. 2, pages 41–50, March 1990.

[2] D. R. Baum, Mann S., Smith K. P., and Winget J. M. Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. *Computer Graphics: Proceedings of SIGGRAPH'91*, 25, No. 4:51–60, 1991.

[3] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. V. Wright. Simplification envelopes. *Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, 1996. (to appear).

[5] M. Cosman and R. Schumacker. System strategies to optimize CIG image content. In *Proceedings of the Image II Conference*, Scottsdale, Arizona, June 10–12 1981.

[6] F. C. Crow. A more flexible image generation environment. In *Computer Graphics: Proceedings of SIGGRAPH'82*, volume 16, No. 3, pages 9–18. ACM SIGGRAPH, 1982.

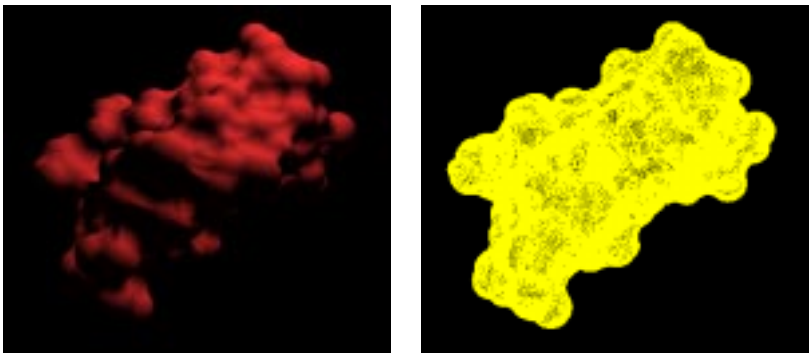
[7] T. D. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for surface of arbitrary topological type. Report 93-10-05, Department of Computer Science, University of Washington, Seattle, WA, 1993.

[8] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH 95 (Los Angeles, California, August 6–11, 1995)*, Computer Graphics Proceedings, Annual Conference Series, pages 173–182. ACM SIGGRAPH, August 1995.

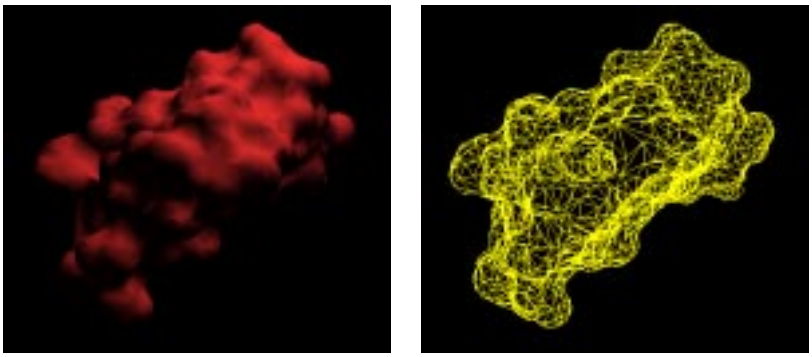
[9] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH 93 (Anaheim, California, August 1–6, 1993)*, Computer Graphics Proceedings, Annual Conference Series, pages 247–254. ACM SIGGRAPH, August 1993.

[10] M. H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In G. M. Nielson and D. Silver, editors, *IEEE Visualization '95 Proceedings*, pages 135–142, 1995.

[11] A. Guéziec. Surface simplification with variable tolerance. In *Proceedings of the Second International Symposium on Medical Robotics and Computer Assisted Surgery, MRCAS '95*, 1995.

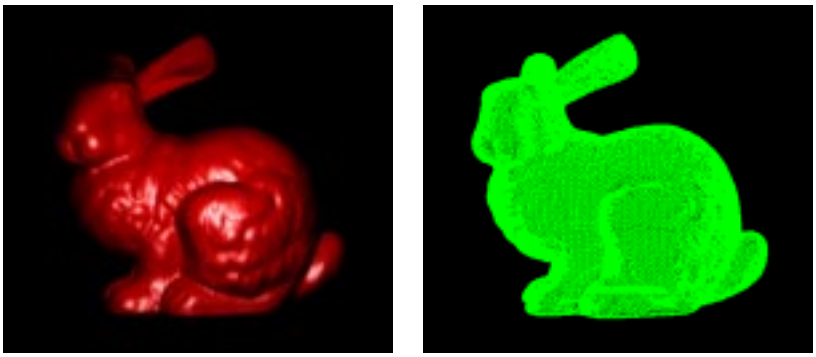


Original molecular surface (109955 triangles)

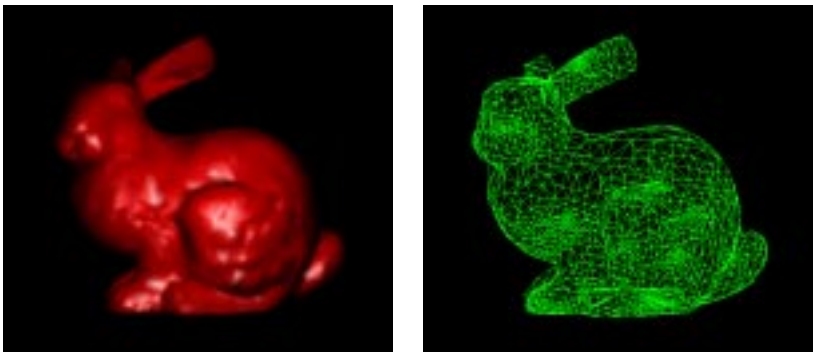


Dynamic adaptive simplification (7433 triangles)

Figure 7: Dynamic Adaptive Simplification for Crambin Surface



Original bunny model (69451 triangles)



Dynamic adaptive simplification (3615 triangles)

Figure 8: Dynamic Adaptive Simplification for Bunny Model

[12] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11:197–214, 1994.

[13] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 1996. (to appear).

[14] James Helman. Graphics techniques for walkthrough applications. In *Interactive Walkthrough of Large Geometric Databases, Course Notes 32, SIGGRAPH '95*, pages B1–B25, 1995.

[15] P. Hinker and C. Hansen. Geometric optimization. In Gregory M. Nielson and Dan Bergeron, editors, *Proceedings Visualization '93*, pages 189–195, October 1993.

[16] H. Hoppe. Progressive meshes. *Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, 1996. (to appear).

[17] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH 93 (Anaheim, California, August 1–6, 1993)*, Computer Graphics Proceedings, Annual Conference Series, pages 19–26. ACM SIGGRAPH, August 1993.

[18] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 Symposium on Interactive 3D Computer Graphics*, pages 95–102, 1995.

[19] J. Rohlf and J. Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, July 1994.

[20] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.

[21] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Computer Graphics: Proceedings SIGGRAPH '92*, volume 26, No. 2, pages 65–70. ACM SIGGRAPH, 1992.

[22] G. Turk. Re-tiling polygonal surfaces. In *Computer Graphics: Proceedings SIGGRAPH '92*, volume 26, No. 2, pages 55–64. ACM SIGGRAPH, 1992.

[23] A. Varshney. Hierarchical geometric approximations. Ph.D. Thesis TR-050-1994, Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175, 1994.