Appearance-Preserving View-Dependent Visualization (Paper 367)

Justin Jang William Ribarsky Christopher Shaw Peter Wonka
GVU Center, Georgia Institute of Technology

ABSTRACT

In this paper a new quadric-based view-dependent simplification scheme is presented. The scheme provides a method to connect mesh simplification controlled by a quadric error metric with a level-of-detail hierarchy that is accessed continuously and efficiently based on current view parameters. A variety of methods for determining the screen-space metric for the view calculation are implemented and evaluated, including an appearance-preserving method that has both geometry- and texture-preserving aspects. Results are presented and compared for a variety of models.

Keywords: view-dependent, level of detail, simplification, appearance-preserving, multiresolution models.

1 INTRODUCTION

We are entering an era where 3D models from diverse sources are achieving unprecedented scale and detail. These include urban models that may contain extended streetscapes or large collections of detailed buildings. Some of these models are reconstructed from range data and imagery [Neu03, Fru01] while others are constructed using advanced CAD or procedural methods [Won03]. In addition there are highly complex single models of varying sizes that must be dealt with using view-dependent techniques [Lin03, Lev00]. Many of the models in these two categories are textured or have other strong appearance attributes.

There have concurrently been developments in level-of-detail (LOD) management and visualization methods. Many of these methods have been applied to compact but highly detailed models [Hop97, Gar97] while others have been applied to extended large scale models, such as terrain, where out-of-core management is necessary [Dav98, Lin03, Fau00]. Ultimately, multiresolution methods of sufficient flexibility are needed to provide good quality visualizations at minimal cost for all these different types of models. In addition, optimal interactive visualization is in general obtained when local resolutions within and among models are chosen dynamically based on the current viewpoint. Also, multiresolution models, when properly organized, can provide efficient, incremental access to data that may reside out-of-core or in networked archives.

This paper presents the following new results that are useful in attacking these diverse models.

- A new, view-dependent method is provided based on the quadric-error approach that has general appearance-preserving attributes.
- A multiresolution hierarchy is developed that efficiently encodes a succession of quadric-based simplifications permitting traversal from original highly detailed models to final, constrained models.
- It is shown how geometry and/or texture-preserving metrics can be used to produce view-dependent simplifications. A variety of metrics are developed and evaluated.
- The quality of this view-dependent method is evaluated for a range of architectural and non-architectural models.

Because it is based on the quadric error approach [Gar97, Gar98], our view-dependent method provides flexible, good quality shape-preserving simplification that applies to both topological and non-topological geometry. Our view-dependent mesh collapse or expansion is also monotonic. The methods presented here can fit into a general program attacking both structured (e.g., buildings) and natural models in a scalable geospatial framework [Fau00, Rib03]. In addition, the quadric approach handles boundary preservation in a general way. This permits the transition to simple textured objects that have been used successfully in interactively navigable large scale collections of buildings in urban environments [Dav99, Jep96].

2 RELEVANT RESEARCH

A lot of work has been done on geometric model simplification, and effective methods have been developed that apply to models with consistent meshes. In this section we will concentrate on work most relevant to the research described here. Some methods enact a series of topology-preserving edge collapses to produce a desired level of simplification, such as Hoppe's progressive mesh (PM) [Hop96, Hop97]. Other methods do not require topological consistency nor preserve topology. These include vertex clustering methods [Ros93] and methods that either remove vertices [Sch92] or combine vertices at new locations (not necessarily along edges) [Gar97, ElS99]. There are also methods based on regular meshes [Lin96, Paj01, Lin03, Fau00], which are usually obtained by resampling. These have the advantage of a compact representation, a simplification hierarchy that is straightforward to set up, and extension to efficient out-of-core management of large scale data [Fau00, Lin03]. However, these methods may not represent certain irregular surface features (say, a mountain ridge on terrain) as efficiently as irregular methods. Most of the regular methods, except for [Lin03], have been applied mainly to terrain height fields.

Garland and Heckbert [Gar97] present the quadric-based approach to polygonal simplification, which creates simpler approximations of the input mesh by performing a sequence of vertex merges. The algorithm always chooses the merge that would result in the lowest quadric error as the next merge to perform. The quadric error metric measures surface deviation and curvature by concisely encoding any number of plane equations of faces in the local neighborhood of a simplified point and its predecessors. However, this lowest error merge is achieved with the complication that the merged vertex can be anywhere in the 3D region near the original vertices rather than, say, along the connecting edge. The basic approach has been extended [Gar98, Hop99] to account for appearance attributes, including vertex colors, vertex normals, and texture coordinates. While these approaches yield nice results, it is not clear how the appearance attribute error relates to the geometric error in this metric, or how to bound appearance error in the rendered image. Erikson et. al. [Eri99] extend the quadric approach to support the joining of unconnected pieces of the mesh beyond just the initial threshold pairs. Their approach produces high quality drastic simplifications of potentially non-manifold models of arbitrary topology and fits well under a hierarchical level of details (HLOD) approach [Eri98]. The HLOD approach works best for scene graphs where

there exist many logically or actually separated objects, as opposed to expansive contiguous meshes.

The above methods can produce good models at a target LOD. In addition some of the methods above and others can produce either simplified or more complex models dynamically based on changing viewing parameters [Hop97, Lin96, Lin03, Paj01, Fau00, ElS99, Xia97, San01]. The latter capability is of significant importance for free navigation among highly detailed or large scale models where one may zoom in for a close-up look or back away for an overview. With these methods, views of large scale models, such as terrain, can be reduced by a factor of a hundred or more in polygon count without noticeable reduction in image quality [Lin96]. The view-dependent methods differ in the details of their approaches. Most of them use a merge hierarchy of some sort that is traversed at run-time to produce the current view, such as the *merge tree* [Xia97] or the *view-dependence tree* [ElS99]. Like [ElS99], Hoppe's approach [Hop97] imposes dependencies (restrictions to preserve mesh consistency) on the run-time simplification; however, they are looser and generally allow more optimally adaptive view-dependent simplifications. Luebke [Lue97] describes a framework for view-dependent simplification of arbitrary polygonal environments based on a vertex clustering-based tight-octree hierarchy. In this sense the method is similar to that of [Lin03]. The algorithm uses the screen-space projection of vertex deviation bounding spheres as the view-dependent simplification metric, which can be a very conservative bound due to the mismatch between the box-shaped clustering cells as well as lack of consideration for appearance attributes.

Cohen et. al. [Coh98] present an algorithm for appearance-preserving simplification. The approach involves a representation conversion whereby normal maps replace normals and texture maps are used for colors. This allows the algorithm to use a texture-deviation metric alone to guarantee appearance quality. The approach is able to generate low-polygon-count approximations of the original model while still preserving appearance, but it operates as a static simplification algorithm. While it is theoretically capable of generating single path simplification sequences as in a PM, it cannot be directly applied to an adaptive view-dependent simplification. Sander [San01] presents an approach for texture mapping progressive meshes that seeks to minimize texture stretch. The approach is mainly concerned with creating stretch minimizing texture atlases such that the entire PM simplification sequence can use the same map. Like [Coh98], our approach is concerned with bounding the texture deviation, whatever the parameterization.

3. HIERARCHICAL DATA STRUCTURE

As with most view-dependent simplification approaches [ElS99, Lin96, Hop97, Lue97, Paj01], our approach consists of two phases: an offline pre-processing phase and a run-time view-dependent meshing phase. The pre-processing phase generates a hierarchy that encodes all possible selectively refinable meshes attainable during the subsequent run-time simplification phase. As with [ElS99], we generate a vertex hierarchy of vertex-pair collapses from the bottom-up. However, instead of using the cubic-spline distance metric, we use an area-weighted quadric error metric with boundary preservation quadrics [Gar97] to determine the simplification sequence. During this pre-processing phase, we also calculate texture coordinates and incremental bounds on the texture deviation. These texture-deviation bounds are later used during run-time to select the appropriate LOD approximation within a user-specified screen-space error bound. (See Sec. 4 for a description of geometry and texture deviations.)

Recently, Lindstrom has developed a view-dependent quadric-based approach [Lin03] that uses a regular resampling of the original surface tesselation. A significant difference between this approach and ours is that we retain the original tesselation. In addition we consider view-dependent error metrics that depend more generally on appearance attributes (geometry, texture, etc.) whereas Lindstrom only considers geometry. Garland has considered color and texture in a quadric approach [Gar98], but without view-dependence. Although Lindstrom finds that resampling has little effect on the quality of the simplification for the models he considers [Lin03], it is still possible that for certain models important details may either be lost or require excessively detailed resampling to retain them. In addition, the resampling grid must be chosen for each model, which introduces an additional complication to the modeling process. Our approach has neither of these potential drawbacks. Ultimately it may be that the two approaches can be combined to take advantage of both the powerful out-of-core capabilities of Lindstrom's method and the precision detail-handling of our approach.

We thus make the following contributions. Our view-dependent method uses a quadric-based structure that produces better visual quality than ElSana's method [Els99] and is built on the original surface rather than the resampled surface, as in Lindstrom's method [Lin03]. It also depends on appearance attributes rather than geometric attributes alone (as in Lindstrom). In addition, as we show in Sec. 5, the structure is fast to build and traverse, as opposed to Hoppe's algorithm, which takes very long to preprocess [Hop97, Hop99].

Constructing the Quadric-Based Tree

We have chosen the quadric approach because it quickly produces good quality simplifications of polygonal models by contracting arbitrary vertex pairs, not just edges. This procedure can produce better quality approximations than those restricted to edge collapses and is more general (in particular, it is useful for non-manifold models encountered in urban, architectural, or other reality-based visualization). The quadric error metric measures surface deviation and curvature by concisely encoding any number of plane equations of faces in the local neighborhood of a simplified point and its predecessors [Gar97]. A quadric matrix (or simply, *quadric*) $\mathbf{Q}$ is the sum of any number of fundamental error quadrics $\mathbf{K_p} = \mathbf{p}\mathbf{p}^T$, where $\mathbf{p} = [a\ b\ c\ d]^T$ represents the plane defined by $ax + by + cz + d = 0$, where $a^2 + b^2 + c^2 = 1$. The quadric error $\Delta(\mathbf{v}) = \Delta([v_x\ v_y\ v_z\ 1]^T) = \mathbf{v}^T\ \mathbf{Q}\ \mathbf{v}$, is the sum of squared distances from a point (v_x,v_y,v_z) to all the planes encoded in $\mathbf{Q}$.

The basic quadric approach can be extended to preserve boundaries [Gar97]. For every edge on the boundary, we can construct a plane parallel to that edge and perpendicular to the face. We can compute the quadric for this plane (called a *border quadric*) and add it to the quadric of the face. For higher boundary preservation, the border quadric is multiplied by a weighting factor (we use a default of 1000) before being added in. We use these border quadrics to constrain the simplification process so it produces a particular lowest approximation. This is especially useful for simplifying collections of, say, buildings and other objects in an urban environment. Here, one needs to both move in for close-ups and navigate to an overview in the visualization [Dav99, Jep96]. To support the overviews, the simplification should converge consistently to a collection of simple textured objects (such as a polygon for an extended façade or a box for many buildings).

Our approach builds a binary tree of vertices from the bottom up via a sequence of vertex merges. We begin with all the vertices of the original mesh M_0, which will be the leaves of the eventual tree. We use the vertex-pair collapse sequence $\{vcol_0, \ldots, vcol_k\}$ of the quadric simplification algorithm to determine the order of vertex merges and the positions of the merged vertices. For $vcol_i$, when merging two vertices V_a and $V_b \in M_i$, we create a new vertex $V_c \in M_{i+1}$ to be the parent of V_a and V_b in the tree. Pointers to the faces removed by this merge are stored as the subfaces of V_c and each subface retains a *residence index*, the index of V_c. This information will be used during run-time to update the mesh. The algorithm proceeds until there is one vertex, the root of the tree. Note that we could also stop when the last face is decimated, when the error of the most recent vertex merge has passed some threshold, or when the above reference polygon. The result would then be a forest of binary trees [Hop97, Lue97].

Data Structures

We have developed several structures, based on the work of [Hop97] and [Lue97], to make the run-time traversal of the above trees efficient. Our approach extends [Hop97] to general meshes without requiring the use of dependencies, while being able to update the mesh more efficiently than [Lue97] due to the binary structure of the tree. *ListNode* is a doubly linked list structure used to string together faces in the active triangle list and vertices in the active vertex list. An index is used as a unique identifier as well as an index to locate the respective face or vertex being linked. *Face* consists of references to the three original vertices as well as the three current vertices of a triangle. The *residence_index* refers to the index of the vertex node in which the face becomes a subface. *Vertex* consists of a 3D point location, a 2D texture coordinate, refinement information, binary tree id and depth, adjacent face and subface lists, and pointers to the parent and two child nodes. *RefineInfo* depends on the selective refinement approach used. It includes a bounding sphere radius for frustum culling as well as information that defines a deviation space to be projected into screen space in order to make a refinement decision on the node. Below is a listing of the structural organization.

```
struct ListNode {
    long index;                 // unique identifier
    ListNode *next;
    ListNode *prev;
};
struct Face {
    ListNode active;            // list stringing active faces
    Vertex *vertices[3];        // the original vertices
    Vertex *proxies[3];         // the current vertices
    Long residence_index;       // index of the vertex where this
                                // face is a subface
};
struct Vertex {
    ListNode active;            // list stringing active boundary vertices
    NodeStatus status;          // inactive, active, or active boundary
    3-Vector pos;               // point location
    3-Vector normal;            // normal vector
    2-Vector texture_pos;       // texture coordinate
    RefineInfo refine_info;     // selective refinement info
    FaceNode *faces;            // head of linked list of pointers to Face
    FaceNode *subfaces;         // faces collapsed in this node
    BitVector tree_id;          // binary tree id; root is 1, child nodes
                                // are id*2 and id*2+1
    int depth;                  // depth in the binary tree
    Vertex *parent;             // parent node to collapse into
    Vertex *vt, *vu;            // child nodes to refine to

};
```

Iterative Updates. We need efficient per-vertex handling for merges and splits that are made at run-time in the view-dependent simplification. On a merge, the approach is to deactivate the subfaces, move all other adjacent faces to the parent node, and update the corner references of the faces. The approach on a split is to activate the subfaces, distribute the adjacent triangles (and the subfaces) to the appropriate child node, and update the corner references. Our algorithm differs from [Lue97] in two ways: we maintain adjacent triangle lists for each vertex in the active mesh and we leverage the binary tree structure to minimize calls to the routine that finds the lowest active ancestor of a node. This is desirable since this routine is the most computationally expensive part of the inner loop of these routines The pseudocode for collapse of v (the merge of its two child nodes) is as follows.

```
collapseVertex(Vertex *v)
    for each subface s of v
        removeAdjacency(s->proxies[3], s);
        deactivateFace(s);
    for each face f of v->vt
        if f->residence_index = v->active.index then
            removeAdjacency(v->vt, f);
    for each face f of v->vu
        if f->residence_index = v->active.index then
            removeAdjacency(v->vu, f);
  linkLists(v->vt->faces, v->vu->faces, v->faces);
  activateVertex(v);
  deactivateVertex(v->vt);
  deactivateVertex(v->vu);

splitVertex(Vertex *v)
    for each subface s of v
        activateFace(s);
        lowestActiveAncestor(s->proxies[3], s);
        for each corner c of {1, 2, 3}
            addAdjacency(s->proxies[c], s);
    for each face f of v
        if childIsLeft(f, v) then
            addAdjacency(v->vt, f);
        else
            addAdjacency(v->vu, f);
    for each face f of v->vt
        for each corner c of {1, 2, 3}
            if f->proxies[c] = v then
                f->proxies[c] := vt;
    for each face f of v->vu
        for each corner c of {1, 2, 3}
            if f->proxies[c] = v then
                f->proxies[c] := vu;
  clearList(v->faces);
  activateVertex(v->vt);
  activateVertex(v->vu);
  deactivateVertex(v);
```

In the pseudocode, activate… adds faces or vertices to the active lists, and deactivate… removes vertices or faces; addAdjacency(*v*, *s*) and removeAdjacency(*v*, *s*) add/remove face *s* to/from the adjacent face list of vertex *v*; linkLists(*a*, *b*, *c*) concatenates lists *a* and *b* and moves the resulting list to *c*. In addition, lowestActiveAncestor(*v*, *s*) replaces the proxy of vertex v of face s with the lowest active ancestor of vertex *v* in the tree; childIsLeft(*f*, *v*) uses the depth of vertex *v* and the *tree_id* of the corresponding proxy of face *f* to determine whether the face belongs in the adjacent faces list of the left or right child of *v*; clearList(*a*) clears linked list *a*. Note that we always store *vt* and *vu*, the child nodes of the residence node of the face, in corner indices 1 and 2, thus *s->proxies*[3] refers to the other vertex involved. The implementation of clearList() is trivial

(set list to NULL) since the nodes of the adjacent face list of *v* are moved into the child node lists, thus emptying the list of *v*.

4 VIEW-DEPENDENT METRICS AND MESH UPDATES

We now describe the details of the deviation metrics that are used to select a particular LOD (depending on the screen-space projection of the metric), the run-time meshing algorithm based on the structures in Sec. 3, and the particulars of the view-dependent simplification process. We will then have a complete algorithm for efficiently visualizing complex models.

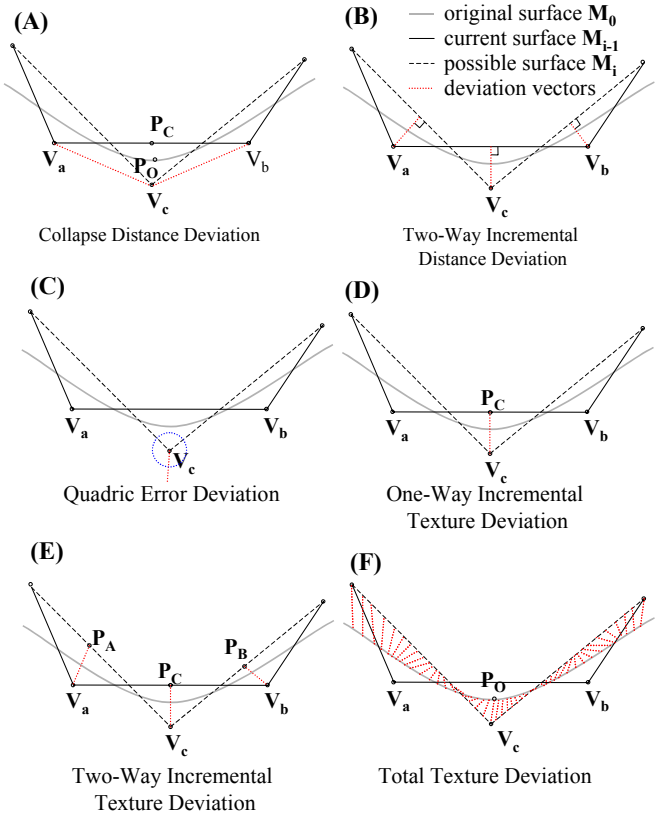

Fig. 1 Several metrics for view-dependent simplification.

Deviation Metrics

We consider several deviation metrics that encompass either geometric or texture error measures, as shown in Fig. 1. In Sec. 5, we will evaluate and compare these metrics, using them on different models.

For the merge of vertices V_a and $V_b \in M_i$ to vertex $V_c \in M_{i+1}$, we define measures for the geometric deviation incurred. In Fig. 1, each of these deviations is indicated by red dotted lines. These include the collapse distance deviation vectors G_{CDV} : $G_{ab} = V_b - V_a$, $G_{ac} = V_c - V_a$ (Fig. 1A), and the incremental surface distance deviation vectors G_{ISV}: $G_c = V_c - H_i(V_c)$, $G_a = H_{i+1}(V_a) - V_a$, $G_b = H_{i+1}(V_b) - V_b$ (Fig. 1B) where $H_j(X)$ is the 3D point on mesh M_j closest to point X. The quadric error vector $G_{QV} = \Delta(V_c)\,\hat{n}$ (Fig. 1C), simplistically computed by scaling the surface normal $\hat{n}$ (which is the direction of highest deviation from the surface) by the quadric error at V_C [Lin03], provides a better alternative. In practice, this measure uses the quadric error to scale a careful characterization of the actual normals involved, which is derived from the same quadric matrices. Alternatively, a simpler but less precise formulation is to use the quadric error as the radius of a bounding sphere, which results in a conservative bound. Notably, G_{CDV} and G_{ISV} measure incremental errors, which are in general non-monotonic, while the quadric metrics measure errors from the original mesh, which are monotonic. In principle, the quadric metric should be the most accurate measure of geometric deviation.

Geometric deviation gives an incomplete measure of the actual appearance deviation. We must also track texture deviation, which is the measure of how far a point V_i on a surface M_i has deviated from the point V_j on another surface M_j that has the same texture coordinate as V_i [Coh98]. Using Cohen's notation, we can map between 3D object space and 2D texture space. The function, $F_j(X)$: $M_j \rightarrow P$, maps point, X, on the surface, M_j, to point, x, in the 2D texture domain, P.[1]. The inverse function, $F^{-1}{}_i(x)$: $P \rightarrow M_i$, maps point x in the texture domain P to a point X on surface M_i. We now define a one-way incremental texture deviation vector $G_{T1V} = V_c - P_c$, where $P_c = F^{-1}{}_i(F_{i+1}(V_c))$,and a set of two-way incremental texture deviation vectors G_{T2V}: (G_{T1V}, $V_a - P_a$, $V_b - P_b$), where $P_a = F^{-1}{}_{i+1}(F_i(V_a))$ and $P_b = F^{-1}{}_{i+1}(F_i(V_b))$. (Fig. 1D and Fig. 1E illustrate G_{T1V} and G_{T2V}, respectively.) The length of G_{T1V} or the max length of the vectors in G_{T2V} can also be used as the radius of a bounding sphere. Since this radius is non-monotonic (as is the case with all the other incremental metrics), we calculate the bounding sphere radius $r(V_c) = \|G\| + \max(r(V_a), r(V_b))$, where G is the deviation vector of choice, be it geometric deviation or texture deviation. The difference between one-way and two-way deviations is that the former calculates only the deviation from M_i due to V_c, while the latter calculates this deviation plus the deviation from M_{i+1} due to V_a and V_b. The two-way incremental deviation will thus provide a better bound.

Texture coordinates for V_c are calculated by using the texture coordinate of the point closest to V_c in mesh M_i. That is, $F_{i+1}(X) = F_i(H_i(X))$. Note that the mapping is potentially not one-to-one. Furthermore, for the two-way bounds, we seek the texture coordinate for V_c that results in the smallest r. Therefore, the approach examines only the local neighborhood $N_{i,Vc}$ of V_c and looks for the texture coordinate from the closest points to the faces in N_{Vc} that minimizes the max two-way texture devation.

Fig. 1F is provided for completeness. It describes the total texture deviation between the merged surface and the original surface. In this case deviations from all affected texture coordinates must be included [Coh98]. This total deviation is too complex to consider for interactive view-dependent simplification, hence we devise approximate metrics to bound it.

Mesh Updates

The run-time meshing algorithm resembles that of [Hop97] and is also similar to [Lue97] and [ElS99]. It maintains a linked list of active boundary vertices and a list of active triangles. A vertex may be active or inactive, and the active vertices may be on the boundary or interior. (See Fig. 2.) The boundary vertices are all the leaf nodes of the sub-tree of all active vertices. These boundary vertices (referred to as a *front* in the tree) comprise all the vertices of the current selectively refined mesh, the list of active triangles. A simplification pass (Fig. 3) consists of the traversal of the vertex front during which view-dependent simplification criteria are applied to decide whether to collapse, keep, or split a vertex node. A collapse removes a pair of vertices

[1] Capital letters (e.g., X) refer to points in 3D, while lower case letters (e.g., x) refer to points in the texture domain.

and adds their parent vertex to the active vertex list while a split replaces a vertex with its two child vertices. Depending on the mesh update information stored at the node, a split/collapse may also result in the introduction/removal of t triangles from the active triangle list. For [Hop97] t is always 2, while our approach permits zero or more, allowing it to support arbitrary meshes as in [Lue97]. Unlike the vertex-clustering tree of [Lue97], it is able to exploit the binary tree structure to perform less work during this update. In contrast to [Hop97] and [ElS99], it neither stores, updates, nor enforces dependencies.

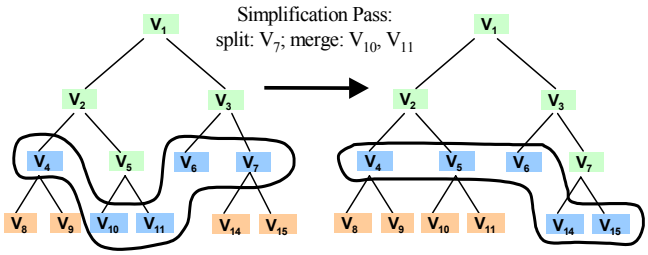

Fig. 2 The vertex front is circled. Green nodes are active-interior, blue nodes are active-boundary, and orange nodes are inactive. Here, vertex V_7 is split and vertices V_{10} and V_{11} are merged.

Dependencies. Both [Hop97] and [ElS99] enforce dependencies on the view-dependent simplification of the mesh to preserve some aspect of mesh validity or coherence including foldover-prevention and local adjacency information. However, this requires the algorithm to perform a few additional comparisons for every refinement evaluation as well as update the dependencies after every vertex split or merge occurs. In addition, to perform a desired vertex split, it may be necessary to split neighboring vertices and their neighbors therein (that is, to recursively evaluate a *chain of dependencies*) just to respect the dependencies. Because a long and expensive recursion might result, ElSana et. al. [ElS99] ignore the need to recursively activate secondary display vertices and instead opt for a lazy approach, waiting for vertices to split during later frames. They report that this is reasonable for slowly changing view-parameters. However, with our quadric-based tree, even slow navigation can result in a simplified mesh that is very visually inadequate for a long time, or it may never activate some visually critical nodes. Fig. 4 demonstrates this phenomenon. In 4a, the sphere was approached from the right and zoomed in. In 4b, the entire sphere was brought into view all at once, allowing the supporting vertices to be present, followed by zooming in to the same view. The tessellation inside the viewing frustum of 4a is inadequate. (It should resemble that of 4b.) In Fig. 4b, there is not much simplification enough outside the view frustum. The dependencies in 4b are overly restrictive due to a chain of dependencies.

This phenomenon is more severe with our quadric-based tree than with the tree of [ElS99] which is based on a spline-distance metric. The spline-distance metric is an indication of deviation across the surface. With this metric, a vertex pair collapse on the surface is highly likely to increase the error of the potential subsequent collapse between the newly picked vertex and its neighbors. The quadric metric is an indication of deviation orthogonal to the surface, so a vertex pair collapse does not necessarily increase the quadric error of the new vertex with respect to its neighbors, particularly in flat or common curvature regions. Thus, a tree built with the quadric metric is much more likely to result in chains of dependencies as it is more likely to nest neighbors as ancestors or descendents or each other as opposed to across the tree horizontally as cousins.

Note that the algorithm of [Hop97] imposes less restrictive dependencies. Even though this would reduce the chance of inadequate refinement when taking the lazy approach (as in [Els99]), their algorithm opts for correctness and evaluates chains of dependencies anyway.

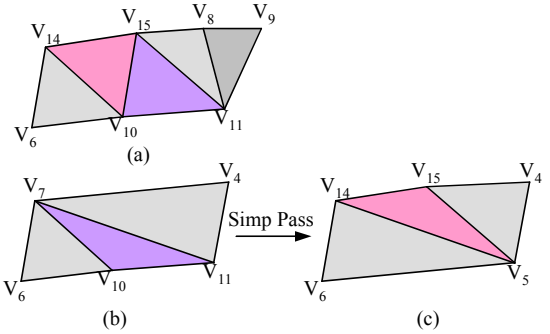

Fig. 3 The pink, purple, and dark gray triangles are subfaces of V_7, V_5, and V_4, respectively in Fig. 2. (a) Full mesh. (b) Tree on left of Fig. 2. (c) Tree on right.

In our approach, no run-time dependencies are enforced. The [Hop97] method requires manifold surfaces, which is too narrow for our case, and the [ElS99] method is too restrictive in terms of the run-time simplifications it will permit. Instead, we allow the view-dependent simplification criteria alone to determine the mesh from all those encoded in the tree structure. Not enforcing supplementary dependencies allows for maximally adaptive simplification and also speeds up computation of the active vertex front. Ignoring dependencies means that there may be a chance for mesh inconsistencies, such as fold-overs, during run-time simplification. However, these hardly ever occur in practice, although they are somewhat more likely for artificial meshes (such as meshes for flat or nearly flat walls). For textured surfaces, our screen-space appearance metric bounds texture deviation. So any visual artifacts due to fold-overs on texture-mapped surfaces with no additional surface-dependent visual ornamentation, e.g. specular highlights, have negligible visible impact (when the texturing is applied to both sides of the polygons). In practice we have found that not only do foldovers occur infrequently, but also that visual artifacts due to foldovers are not noticeable. This is consistent with our approach to focus on preserving appearance attributes rather than on mesh consistency. This is reasonable since the mesh itself often has no fundamental value (as in models acquired from laser range data or even in some constructed models) and sometimes is not even consistent, as in non-topological models.

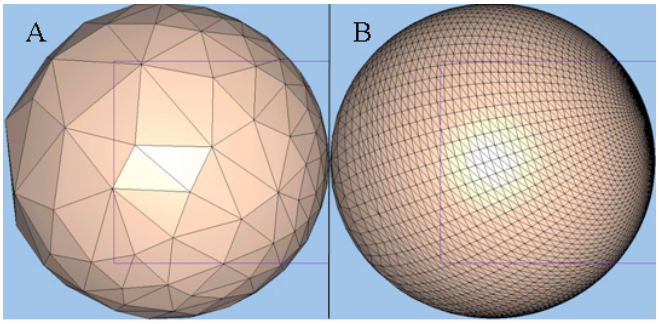

Fig. 4 Effect of dependencies on two different navigation paths.

Screen-Space Deviation. To determine the maximum screen-space deviation, we select one of the above deviation metrics and project the deviation bounding sphere to screen space.

As with most approaches, e.g. [Hop97, Paj01, Lin03], we opt for an approximate but more efficient evaluation instead of precise evaluation. We calculate the projected radius, p, of the sphere centered at v with radius r as follows.

$$\gamma = 2 * h / \phi$$
$$p = \gamma * r / (v - e) \bullet \vec{e}$$

where h is the vertical size in pixels of the viewport, ϕ is the (vertical) field of view angle, γ is an approximation of the pixels per view angle subtended, and $\vec{e}$ is the view direction vector, which is calculated once per view. A vertex is refined if its p is less than a pixel threshold τ.

This approximation underestimates the projection sizes away from the view center and overestimates those near the view center and a parallel projection assumption is made. Furthermore, the neighborhood of V on an adaptively simplified mesh may be different than the neighborhood of V for which the bounds were originally calculated during the build phase.

Hoppe [Hop97] estimates the Hausdorf distance between $N_{v,i+1}$, the local neighborhood of v after edge collapse i, and $N_{v,0}$, the corresponding local neighborhood on the original mesh, M_0, by analyzing the residual error vectors from a dense set of points on M_0. The distance bound obtained is used as the radius μ of the bounding sphere, which is used to bound error during view-dependent refinement. In addition, this approach can project a vector scaled in the direction of the normal that biases the refinement for preserving geometry on or near the silhouette. This approach yields nice results, but requires significant computation, especially if it were to be extended to account for texture deviation. Although the tree is built off-line, it is still desirable to have an efficient build phase, especially for applications such as urban modeling where data generation to visualization turn-around time is important.

model	verts	tris	initial pairs	load	init	T1V	ISV	T2V	tree height
sphere	10k	20k	30k	0.17	0.19	0.36	0.44	0.84	54
wave	103k	205k	308k	1.64	2.22	4.64	5.22	8.16	26
bunny	36k	69k	104k	0.49	0.77	1.48	1.70	1.77	22
buddha	150k	300k	450k	2.12	6.05	7.31	8.06	8.49	26
façade	49k	97k	146k	1.03	0.95	2.05	2.36	3.86	36
wall	3.6k	7.2k	36k	0.14	3.67	0.25	0.28	0.52	31

Table 1. All timings are in seconds.

Frustum and Backface Culling. Each node of the vertex tree stores a frustum culling bounding sphere radius that bounds all descendant vertices. Our implementation compares the sphere with the six planes of the view frustum. Nodes with frustum-culling bounding spheres that intersect the frustum are candidates for refinement. Alternatively, one can opt for a faster but more conservative evaluation of view frustum visibility as in the frustum cone of [Paj01].

For closed manifold models of objects, faces on the back side with respect to the viewpoint are not visible as long as the viewpoint is never located on the inside the model. Therefore, it makes sense to allow these faces to simplify as much as possible. Like [Hop97, Paj01], we bound the spread of normals of the adjacent faces of a vertex v and the descendants of v with a cone represented as the vertex normal, $\hat{n}_v$, and a cone angle, α_v. A vertex is considered unnecessary for supporting a front face if $\hat{n}_v \bullet (v - e) / \|v - e\| > \sin \alpha_v$ holds. The situation is more complicated, of course, for open or non-manifold models, as are sometimes encountered in urban visualization.

5 RESULTS AND DISCUSSION

We test our approach and evaluate the various metrics using a variety of models of different types (Figs. 5-8). The sphere, wall, and wave models are procedurally generated and include texture coordinates. The wall is comprised of 1800 separate components with arbitrarily connectivity. The wave is a height-field of sine waves that continuously vary in frequency. The bunny, the Buddha, and the building façade are models constructed from scanned data. The building façade [Fru01] includes texture information scanned concurrently with the geometry, and thus possesses an inherently correct parameterization. Table 1 gives basic information, including size, initial candidate simplification pairs, and tree height, for these various models and shows their respective tree construction times in seconds. (These timings are from our prototype implementation running on a 2.4GHz Pentium4 Win2k PC with 512MB of RDRAM and a NVIDIA Quadro4 900XGL graphics card.) Our build times are comparable with qslim [Gar97] since the quadric simplification approach is the foundation of our tree build algorithm. On top of the basic algorithm, we perform additional computation associated with the view-dependent structure, including linking the tree, computing texture coordinates, and computing run-time refinement information such as the error bounds. The two-way texture bounds (T2V) followed by the two-way geometric bounds (ISV are the most computationally expensive error bounds to compute, so those build times are listed separately. The total build times are significantly faster than for some other methods [Hop97].

For a fly through of the wave model, we achieve average frame rates of 20fps, where 54 percent of each frame is devoted to the simplification pass and the rest to rendering. We achieve simplification throughputs of over 60k triangles per second (tps) for collapses and over 50k tps for splits. Because we have concentrated on the new view-dependent structure and error metric implementation in this paper, our implementation is un-optimized for rendering. It traverses a linked-list for every refinement pass and traverses a linked-list to render every frame. We have not implemented optimizations, such as vertex arrays (as in [Els02]) and display lists (as in [Lin03]), or mesh update optimizations, such as prioritized traversal [ElS02], triangle-budged simplification [Lue97], asynchronous simplification [Lue97], etc. We anticipate that significant improvement in performance would result in incorporating any of the above, which is straightforward for most.

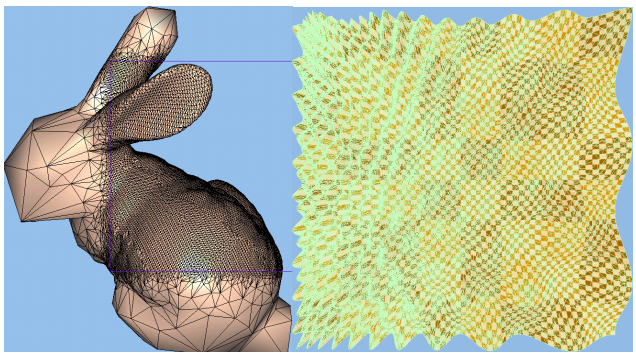

Fig. 5 View-dependent simplification of bunny and wave models.

Fig. 5 shows the view-dependent simplification in action. For the bunny model, notice the high fall-off in mesh tessellation density outside the view frustum (blue outline). The wave model (right) exhibits more simplification in areas of lower frequency content (towards the lower right of the image) and less simplification in areas of higher frequency content (towards the upper left). Here we use G_{T2V}. The G_{ISV}, G_{QV}, and G_{T1V} metrics behave similarly.

Figure 6 shows the facade model simplified using each metric at one pixel screen-space deviation for 1024 x 768 pixel views. The blue box shows the viewport. The first pair (6A, 6B) is at full resolution; each subsequent pair is for a different metric. In each pair, the right-hand image shows the mesh explicitly. Note that the geometry-only metric G_{CDV} (6E, 6F) preserves appearance, but does not allow much simplification. Also note that the geometry-only metric G_{ISV} (6I, 6J) allows significant simplification, but fails to bound texture deviation. As shown in (6C, 6D), G_{T1V} bounds texture deviation at the vertices of the active mesh, but not in between. G_{T2V} (6G, 6H) not only bounds texture deviation at the vertices, but it also bounds deviation across the faces. The quadric sphere metric G_{QV} (6K, 6L) gives nice adaptive simplification, refining more in areas of high geometric detail, but guarantees no bounds on texture deviation. Furthermore, there is less of a fall-off in tesselation for portions of the model further from the viewpoint than with other approaches.

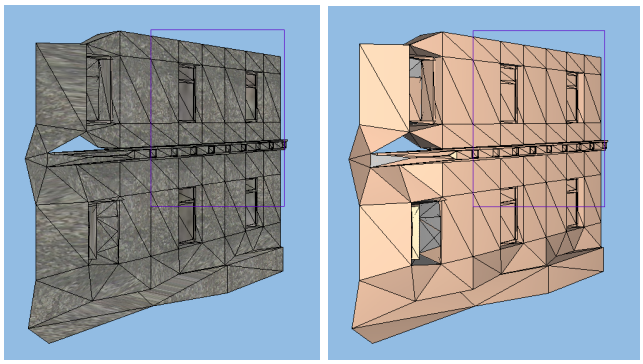

Fig. 7 Wall model with (left) and without texture.

Fig. 7 demonstrates that our scheme can preserve appearance even on piecemeal meshes such as the wall model. This an architectural model generated with a procedural technique where mesh topology is not enforced [Won03]. Notice how the mesh falls apart outside the view frustum, yet inside is virtually indistinguishable from the original. As it moves inside the frustum, the outside mesh also reforms consistently. Finally, Fig. 8 shows the Buddha model simplified with the texture deviation metric G_{T2V}, despite it not being given (nor does it compute) a texture parameterization. The G_{T2V} metric gracefully falls back to G_{ISV}.

6 CONCLUSIONS AND FUTURE WORK

We have presented a quadric-based approach for appearance-preserving, view-dependent visualization of triangulated models. We have described a method for quickly generating a visualization-ready hierarchy from an input model. This hierarchy can be efficiently traversed for view-dependent rendering. In addition the data structure accommodates different error metrics. We have characterized the relative merits of several metrics in determining the appropriate mesh for preserving appearance. We have presented results for several models that show the visual quality of our approach and the merits of the different error metrics.

There are a number of avenues for future work. Normal maps and vertex color information can be added to the formalism to efficiently improve the appearance-preserving character of non-texture mapped models. A formulation for boundary preservation using reference planes can be built on our approach to permit consistent transition to simple textured objects appropriate for overviews of collections of objects. In addition a more general approach could be developed for urban models based on architectural semantics [Won03] that would support interactive 3D planning. Finally large collections of models could be placed in a scalable structure for interactive visualization that ranges over all scales. Our approach has the flexibility to support all these avenues.

Acknowledgments.

We acknowledge the Stanford Computer Graphics Lab for the use of the bunny and Buddha models. This work is supported by the Department of Defense's MURI program, administered by the Army Research Office; it is also supported by a grant from the NSF Large Scientific and Software Data Visualization program.

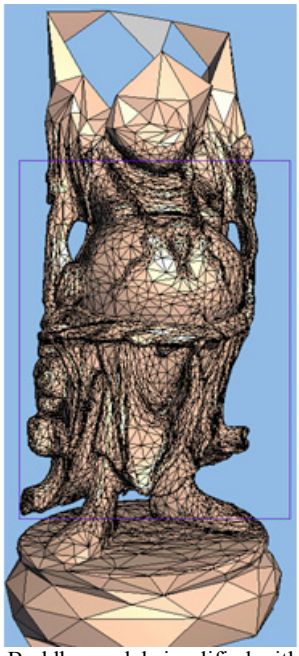

Fig. 8 Buddha model simplified with G_{T2V}.

REFERENCES

Coh98 J. Cohen, M. Olano, and D. Manocha. Appearance-Preserving Simplification of Polygonal Models. *Proc. SIGGRAPH 98*, pp. 115-122 (1998).

Dav98 Davis, D., Jiang, T.F., Ribarsky, W., Faust, N. Intent, Perception, and Out-of-Core Visualization Applied to Terrain. *Proc. IEEE Visualization '98*, pp. 455-458 (1998).

Dav99 Douglass Davis, William Ribarsky, T.Y. Jiang, Nickolas Faust, and Sean Ho. "Real-Time Visualization of Scalably Large Collections of Heterogeneous Objects," Report GIT-GVU-99-14, *IEEE Visualization '99* pp. 437-440, (1999).

ElS99 J. El-Sana and A. Varshney. Generalized View-Dependent Simplification. Computer Graphics Forum (*Eurographics '99*) 18(3), pp 83-94 (1999).

ElS02 J. El-Sana and E. Bachmat. Optimized View-Dependent Rendering for Large Polygonal Datasets. Proc. IEEE Visualization '02, pp. 77-84 (2002).

Eri98 Carl Erikson and Dinesh Manocha. Simplfication Culling of Static and Dynamic Scene Graphs. UNC Chapel Hill Computer Science Technical Report TR98-009 (1998).

Eri99 Carl Erikson and Dinesh Manocha. GAPS: General and Automatic Polygonal Simplification. Symposium on Interactive 3D Graphics, pp 79-88 (1999).

Fau00 N. Faust, W. Ribarsky, T.Y. Jiang, and T. Wasilewski. Real-Time Global Data Model for the Digital Earth. *Proc. INTERNATIONAL CONFERENCE ON DISCRETE GLOBAL GRIDS (2000).* An earlier version is in Rep. GIT-GVU-97-07.

Fru01 C. Früh and A. Zakhor. 3D model generation for cities using aerial photographs and ground level laser scans. *IEEE Computer Vision and Pattern Recognition Conference* (Kauai, Hawaii, December, 2001).

Gar97 M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metrics. *Proc. SIGGRAPH 97*, pp. 209-216 (1997).

Gar98 M. Garland and P. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. *Proc. IEEE Visualization 98*, pp. 263-269 (1998).

Hop96 H. Hoppe. Progressive Meshes. *Proc. SIGGRAPH 96*, pp. 99-108 (1996).

Hop97 Hoppe, Hugues. View-Dependent Refinement of Progressive Meshes. *Proc. SIGGRAPH 97*. pp. 189-198 (August 1997).

Hop99 H. Hoppe. New Quadric Metric for Simplifying Meshes with Appearance Attributes. *IEEE Visualization 99*, pp. 59-66 (1999).

Jep96 W. Jepson, R. Liggett, S. Friedman. Virtual Modeling of Urban Environments. *Presence*, 5(1), pp. 72-86 (1996).

Lev00 M. Levoy et. al. The Digital Michelangelo Project: 3D Scanning of Large Statues. *Proc. SIGGRAPH 00*, pp. 131-141 (2000).

Lin03 Peter Lindstrom. Out-of-Core Construction and Visualization of Multiresolution Surfaces. To be published, *ACM Symp. on Interactive 3D Graphics* (2003).

Lue97 David Luebke and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. *Proc. SIGGRAPH 97*, pp. 199-208 (1997).

Neu03 U. Neumann, S. You, J. Hu, B. Jiang, and J. Lee, "Augmented Virtual Environments (AVE): for Visualization of Dynamic Imagery", *IEEE Virtual Reality'03*, pp. 61-67 (2003).

Paj01 Renato Pajarola. FastMesh: Efficient View-dependent Meshing. *Proc. Pacific Graphics 2001*, pp. 22-30, (2001).

Rib03 William Ribarsky. Virtual Geographic Information Systems. To be published. The Visualization Handbook, Charles Hansen and Christopher Johnson, editors (Academic Press, New York, 2003).

San01 P. Sander, J. Snyder, S. Gortler, H. Hoppe. Texture Mapping Progressive Meshes. *Proc. SIGGRAPH 01*, pp. 409-416 (2001).

Sch92 W. Schroeder, J. Zarge, and W.E. Lorensen. Decimation of Triangle Meshes. *SIGGRAPH 92*, pp. 65-70 (1992).

Won03 Peter Wonka, Michael Wimmer, Francois Sillion, and William Ribarsky. Instant Architecture. To be published, *SIGGRAPH 2003* (2003).

Xia97 J.C. Xia, J. El-Sana, 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).

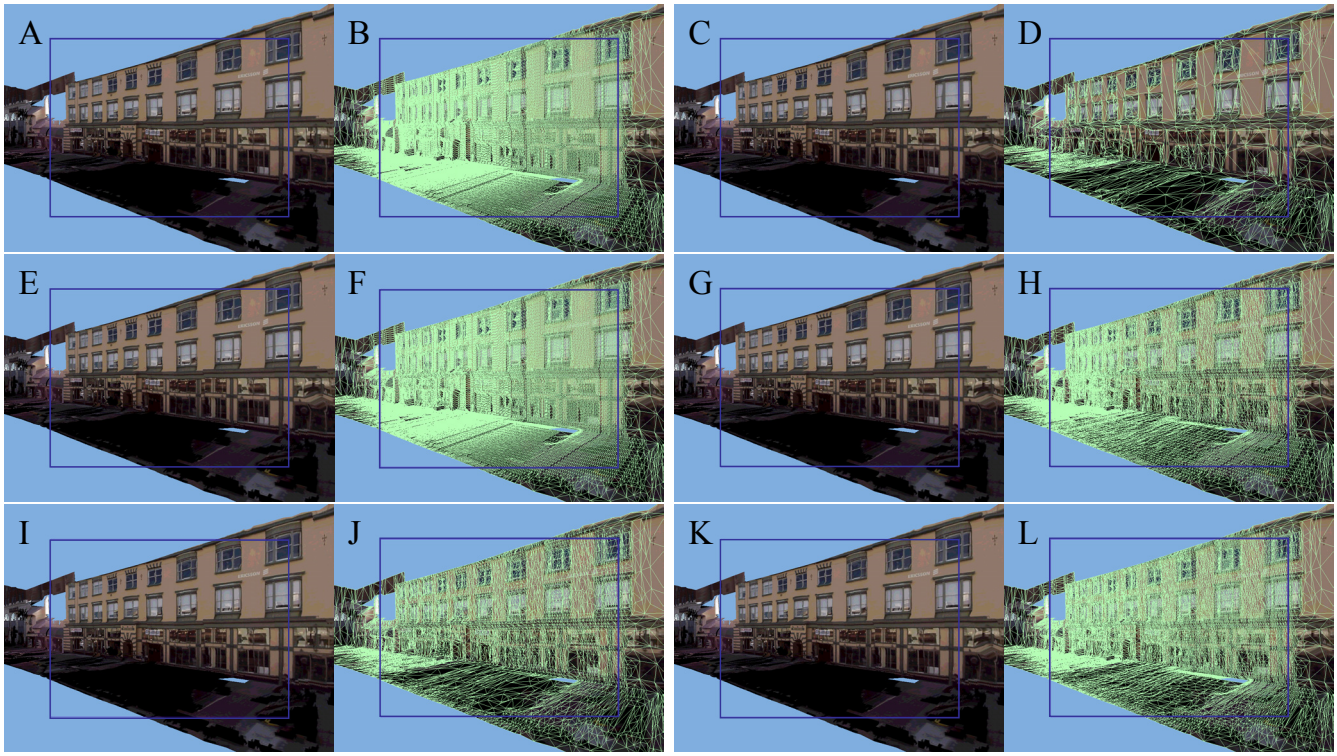

Fig. 6 Façade model (Fru01) comparing different metrics at a resolution of 1 pixel (except for 6A, which is full resolution).