Geometric simplification and compression

Jarek Rossignac

GVU Center and College of Computing
Georgia Institute of Technology

One third of the workstation business is driven by industrial or scientific applications that heavily depend on the ability to interactively render 3D models whose complexity is increasing far more rapidly than the performance of the graphics subsystems. Simple scenes in 3D video-games may only need a few hundred textured polygons, but models used for mechanical CAD, scientific, geo-science, and medical applications involve scenes with millions of faces, sometimes grouped to form the boundaries of polyhedral solids of widely varying complexity. The successful exploitation of such large volumes of scientific and engineering three-dimensional data hinges on users' ability to access the data through phone lines or network connections and to manipulate significant portions of these 3D models interactively on the screen for scientific discovery, teaching, collaborative design, or engineering analysis. Currently available high-end graphics hardware is often insufficient to render the models at sufficient frame rates to support direct manipulation and interactive camera control.

The abundance and importance of complex 3D data bases in major industry segments, the affordability of interactive 3D rendering for office and consumer use, and the exploitation of the internet to distribute and share 3D data have exacerbated the need for an effective 3D geometric compression technique that would significantly reduce the time required to transmit 3D models over digital communication channels and the amount of memory or disk space required to store the models. Because the prevalent representation for 3Dshapes for graphics purposes is polyhedral and because polyhedra are in general triangulated for rendering, it is important to focus on the compression and decompression of complex triangulated models.

Although we anticipate further development of hardware acceleration graphics engines and new commercial explorations of hardware compression/decompression chips, we discuss software advances on both fronts, hoping that with maturity they will fuel hardware advances and impact data exchange standards.

Software techniques, which eliminate unnecessary or unessential rendering steps, may lead to dramatic performance improvements and hence reduce hardware costs for graphics. Many of these techniques require complex algorithmic pre-processing and a compromise on the quality and accuracy of the images. The relative impact of software techniques for accelerating the rendering of 3D scenes depends on the complexity and characteristics of the model, on the lighting model, on the image resolution, and on the hardware configuration. Acceleration techniques include hierarchical culling, memory management, visibility computation, reduced resolution or accuracy, model simplification, use of images, textures, or perturbation maps, and the optimization of the rendering library.

We first discuss a simple model of the rendering cost and review the impact of various acceleration techniques on the different cost factors and stress the need to combine the various techniques. We then focus on 3D model simplification, a preprocessing step that generates a series of 3D models (sometimes called "impostors" or "impersonators"), which trade resemblance to the original model for fidelity. Such decreasing levels-of-detail (LOD) involve less faces and vertices, and hence require less memory and less geometry processing at rendering time. Using a lower level of detail when displaying small, distant, or background objects improves graphic performance without a significant loss of perceptual information, and thus enables real-time inspection of highly complex scenes. Original models are used for rendering objects close to the viewpoint and during navigation pauses for full precision static images.

Simplification is an automatic process that takes a polyhedral surface model S and produces a model S' that resembles S, but has significantly less vertices and faces. We discuss both geometric and visual measures of the discrepancy between S and S' and review the general principles for computing and evaluating such simplifications. We briefly review several simplification techniques based on curved-surface tessellation, space or surface sampling, and bounding hierarchies. We investigate in more details the variants of vertex clustering and face merging techniques, which include edge collapsing and triangle or vertex decimation. We illustrate these techniques through several approaches: Kalvin and Taylor aggregate nearly co-linear facets into connected regions, simplify their boundaries, and triangulate the regions; Ronfard and Rossignac expand on the edge-collapsing work of Hoppe et al. by maintaining an efficient point-plane distance criteria for estimating the errors associated with each candidate edge; Gueziec preserves the volume of the model and uses spheres as error bounds; Rossignac and Borrel use vertex quantization (integer rounding on vertex coordinates) to efficiently

compute the clusters. The latter simplification process is more efficient, more robust, and simpler to implement than alternative approaches, because it does not require building and maintaining a topological face-vertex incidence graph and thus does not impose any topological integrity restrictions on the original model. Although it is not adaptive and does not produce optimal simplifications for a given complexity reduction, it appears particularly well suited for mechanical and architectural CAD models, because it automatically groups and simplifies features that are geometrically close, but need not be topologically close nor even be part of a single connected component.

These techniques produce discrete simplifications, i.e. models that are uniformly simplified to different resolutions. They work well for scenes with many small objects uniformly distributed through space. Large objects may require an adaptive simplification model, where the resolution of the approximation is automatically adapted to the location of the view point (and decreases with the distance to the viewer). We illustrate this approach with Hoppe's Progressive Meshes and with the adaptive terrain models developed at the Georgia Tech's GVU center.

Simplification reduces the triangle counts, but does not address the problem of data compression. In fact, one needs to store the original model plus its simplified versions. 3D compression techniques may be applied to the original and the simplified models independently. They do not reduce the triangle count, but reduce the number of bits required to code the vertex coordinates (geometry), the triangles' references to their supporting vertices (incidence), and the normals and color attributes (photometry). We discuss Deering's extensions of the triangle strips encoding to more general triangular meshes, and Taubin and Rossignac's "topological surgery" approach, which captures the incidence data in less than 2 bits per triangle (instead of the naive 126) and uses vertex quantization, geometric prediction, and entropy coding to compress geometry and photometry with controllable loss. Finally, we expand on Hoppe's Progressive Meshes to lay the ground for further research towards the unification of simplification and compression and towards the use of such scaleable models for internet-based 3D data access and 3D collaborative applications.

CR Categories and Subject Descriptors: I.3.3 **Computer Graphics**: Picture/Image Generation -- display algorithms;
I.3.5 **Computer Graphics**: Computational Geometry and Object Modeling -- curve, surface, solid, and object representations;
I.3.7 **Computer Graphics**: Three Dimensional Graphics and Realism.
General Terms: 3D Polyhedron Simplification, Levels-of-detail, 3D compression, Algorithms, Graphics, Performance acceleration.

Introduction

Computer graphics is an effective means for communicating three-dimensional concepts, for inspecting CAD models, and for interacting with education and entertainment software. Although photo-realistic images and video sequences, typically computed off-line, have their own merits, interactive graphics is required when direct manipulation is important. Direct manipulation increases ease-of-use and enhances productivity by exploiting our natural ability to use visual cues when controlling our gestures. The closed loop involving the hand, the modeling or animation software, the graphics, and the human vision is only effective if graphic feedback is instantaneous. Delays between the gestures and the resulting image, lead to overshooting, reduce the feeling of control, and thus make the user less productive.

Interactive manipulation of 3D models and interactive inspection of 3D scenes cannot be effective when the graphics feedback requires more than a fraction of a second. Although graphics performance, has significantly increased in the recent years (benefiting from faster CPUs, leaner APIs, and dedicated graphics subsystems), it is lagging behind the galloping complexity of CAD models found in industrial and consumer applications and of scientific datasets [Rossignac94], 3D models of typical mechanical assemblies (appliances, engines, cars, aircrafts...) contain thousands of curved surfaces. Accurate polyhedral approximations of these exact models, typically constructed to interface with popular graphics APIs, involve millions of polygonal faces. Such complexity exceeds by one or two orders of magnitude the rendering performance of commercially available graphics adapters. The solution is not likely to come from hardware development alone. Algorithmic solutions must be employed to reduce the amount of redundant or unessential computation.

We define *redundant* computation as the processing steps that could be omitted without affecting the resulting images. *Unessential* computation is defined as the steps which, when omitted, affect the resulting image only to a moderate degree, so that the visual discrepancies between the correct image and the one produced do not hinder the user's perception and understanding of the scene. The elimination of all redundant and unessential computations remains an open research issue, because the computational cost associated with the identification of what is redundant and what is not may often offset the benefits of eliminating redundancy. Several techniques are reviewed below.

To better identify redundant and unessential steps for rendering polyhedral scenes, we use an over-simplified model of the rendering cost. The scene is described by its geometry (the location of the polygons and the location and characteristics of the view) and by the associated photometric properties (light sources, surface colors, surface properties, textures). Rendering techniques compute, for each pixel, the amount of light reflected towards the eye by the objects in the scene. This computation may be arranged in various ways to cater to the desired degrees of photo-realism and to take advantage of the available graphics hardware or libraries. We focus here on rasterization techniques, which visit each polygon in the scene and combine the results in a frame buffer using a z-buffer for hidden surface elimination. This choice is dictated by the prevalence and performance of affordable 3D graphics hardware accelerators and rasterizers, and by the success of associated APIs.

Assume that the geometry of our scene is composed of T triangles. We use triangle counts in these arguments, instead of polygons, for simplicity and because polygons are typically converted into triangles, either during preprocessing [Rofard94] or during rendering, and are an accepted unit of graphics performance. Procedures for triangulating models bounded by curved parametric surfaces also exist.

The cost of rendering a scene comprising T triangles depends on a number of factors, such as the window size, the lighting model, the amount of memory paging involved, or how the triangles project onto the screen.

Rendering T triangles involves several costs:

- **F:** the cost of fetching the necessary triangles into cache memory
- **X**: the cost of transforming their vertices and lighting them using associated normals
- **C**: the cost of clipping the triangles and computing slopes
- **R**: the cost of rasterizing them

The overall rendering cost (i.e. performance) depends on the particular architecture of the 3D graphics subsystem. Three families of uniprocessor architectures are popular for graphics:

- all software rendering
- hardware rasterization with software geometric processing
- all hardware rendering

For purely software rendering architecture, the total cost is the sum of all the costs: **F+X+C+R**. For an architecture based on a hardware rasterizer, the total cost is the maximum of **max(F+X+C,hR**), where h provides the boost factor of the graphics rasterizer hardware. Because the hardware or the software may be bottlenecks, we use max instead of a sum when combining the individual costs. Similarly, because an all hardware rendering subsystem is pipelined, its cost is the maximum of the costs at each stage, i.e.: **max(F,kX+kC,hR**), where k provides the boost factor of the hardware-supported transformation, lighting, and clipping. Note that this simplified formula does not take into account the statistics effects of data buffers and load balancing in parallel architectures.

Review of graphics acceleration techniques

The performance enhancing techniques reviewed in this section help reduce the different aspects of the overall rendering cost. None of these techniques is usually sufficient to address the graphics performance problem and most systems exploit combinations of several techniques simultaneously.

Meshing or storing transformed vertices

In a polyhedral scene, the number **T** of triangles is roughly twice the number **V** of vertices. For example, for a triangulated manifold polyhedron with no holes or handles, the Euler-Poincarre formula becomes: **T-E+V=2,** where E is the number of edges. Notice that **2E=3T,** because there are three edges for each triangle, but each edge of a manifold is shared by two adjacent triangles. Combining these two equalities yields: **T-3T/2+V=2**, and finally: **T=2V-4.**

Therefore, independently processing the vertices of each triangle may unnecessarily increase **X**, and maybe even **F**, by a factor of 6, since on average a vertex is processed six times (3 times per triangle and there are twice more triangles than vertices).

To reduce this computational redundancy, 3D graphics adapters and associated APIs support triangle meshes, where each vertex is used in conjunction with two of the recently processed vertices to define the next triangle. If very long meshes were used, the majority of vertices, and associated normals would only be processed twice, reducing by three the geometric cost of transforming and lighting the vertices. Note however that half of this cost is still redundant. Furthermore, constructing long meshes is algorithmically expensive.

In graphics architectures where the geometric transformation and lighting is performed in software, all the vertices could be transformed only once and the resulting coordinates stored for clipping and rasterization, as needed for the triangles. Similarly, normals may be transformed and the associated colors computed and stored (depending on the lighting model, colors may be defined by normals only or by combinations of normals and vertices). For compatibility reasons with hardware graphics adapters, this possibility is not systematically exploited by popular APIs.

Recent progress attempting to generalize the notion of a triangle strip to cover triangular meshes of arbitrary topology with minimal vertex duplication may be found in the context of geometric compression [Taubin96, Deering95, Hoppe96].

Frustum culling

Graphics performance may in general be significantly improved by avoiding to process the geometry of objects that do not project on the screen. Such objects may lie behind the viewpoint or outside of the viewing frustum. Simple bounds, such as min-max boxes or tight spheres around the objects may be easily pre-computed, updated during model editing or animation, and used for efficient culling. For scenes involving a large number of objects or for scenes where individual objects are relatively large and involve large numbers of triangles, pre-computed hierarchical spatial directories (see [Airey90, Clark76, Teller91, Naylor95]) are used to quickly prune portions of the scene outside the viewing frustum. These early culling techniques may have no effect when the entire models is examined and fits in the viewing frustum. In average however, they may significantly affect F, because one needs not fetch models whose bounds is outside the frustum, and of course also **X** and **C**. They do not affect R, because these triangles would have been rejected through clipping anyway. For example, culling may reduce **X** and **C** by a factor of if the viewpoint is in the center of the scene and if the viewing frustum spans a 90 degrees angle.

Smart caching and pre-fetching

Arranging the data so that contiguous memory locations are accessed by the graphics subsystem and using secondary processors for pre-fetching data may eliminate the delays caused by page faults and reduce **F**. This is particularly effective if the size of the model that is not culled out exceeds the available memory [Funkhouser93].

Pre-computed visibility

Culling objects or individual triangles that intersect the frustum but are invisible in the current view, because hidden by other objects, would directly impact the overall cost for all three types of architectures. Visibility information may be pre-computed for a given granularity of the space and of the models. For example, [Teller92] uses the model's geometry to subdivide space into cells, associating with each cell **c** the list of other cells visible from at least one point in **c**. Cells may correspond to a natural, semantic, partition of the scene (floors, rooms, corridors) or to more general partitions induced by planes that contain the faces of the model. The preprocessing is slow and requires storing vast amounts of visibility information, unless the number of cells and the number of objects in the scene is small. To improve performance, portals (i.e, holes, such as doors and windows, in the boundary of cells) may be approximated by an enclosing axis aligned 2D box in image space, which provides only a necessary condition for visibility, but reduces visibility tests to clipping against the intersection of such rectangles [Luebke95], A different approach was used in [Greene93], where a hierarchical quadtree representation of the z-buffer generated after displaying the front most objects is used to quickly cull hidden objects. Front most objects candidates are computed from the previous frame. Culling compares a hierarchical bound system of the 3D scene against the z-buffer quadtree. Both techniques are effective under the appropriate conditions, but are of little help in scenes, such as factories or exterior views, where very few objects are completely hidden.

Use of images

A distant or small group of objects may be replaced with a "3D sprite" or with a few textured polygons [Beigbeder91] showing the approximate image of the object(s) from the current viewpoint. A hierarchical clustering approach where groups of objects were displayed using such images was proposed in [Maciel95]. Previously rendered images with the associated z-buffer information provide an approximation of the model as seen from a specific location. They may be warped and reused to produce views from nearby locations on an inexpensive graphics system. Because the new view may reveal details hidden in the previous view, a more powerful graphic server may identify the discrepancies and send the missing information to the low-end client [Mann97].

Levels of detail

Using simplified models with a lower triangle count instead of the original models may in certain cases reduce paging **F**, and will in general reduce **X** and **C** significantly [Funkhouser93]. Lower levels of detail are usually appropriate for rendering distant features that appear small on the screen during camera motions or scene animations [Crow82]. During navigation pauses, the system will start computing the best quality image and, if not interrupted by the user, will automatically display it [Bergman86]. A review of the early techniques may be found in [Erikson96]. Two main approaches to the construction and use of such levels of detail (LOD), multi-resolution models have been investigated by a large number of researchers:

- Pre-compute a series of **static** view-independent simplifications for each subset of the scene (solid, group) which approximate the original model to decreasing levels of accuracy and select during navigation the appropriate level of details for each subset, depending on the viewing conditions and on the desired performance/accuracy trade-off.
- Pre-compute a unique **adaptive** model for the entire scene and during navigation adapt it to the current viewing conditions to produce a graphics model which approximates features close to the viewer with higher accuracy than distant features, which appear small on the screen.

A comparative discussion of some of the early techniques [Blake87] for computing static simplifications may be found in [Erikson96, Heckbert94]. We summarize here several trends to provide the context for the rest of the paper, which focuses on vertex clustering and edge collapsing techniques.

Polyhedral models are often constructed to approximate curved shapes through **surface tessellation**. Polyhedral approximations with fewer vertices and faces can thus in principle be produced by using coarser tessellation parameters. Emerging high-end graphic architectures support adaptive tessellation for trimmed NURBS surfaces [Rockwood89], However, the simplification process should be made independent of the design history and should be automatic [Crow82]. These surface tessellation techniques are of little help when trying to simplify complex shapes that involve thousands of surface matches, because they can at best simplify each surface patch to a few triangles, but cannot merge the triangulations of adjacent patches into much simpler models.

Surface fitting techniques to regularly spaced scanner data points [Schmitt86, Algorri96] may also be used for producing levels of detail polyhedral approximations [DeHaemer91]. These techniques have been recently extended to unorganized data points [Hoppe92, DeRose92] and to new sparse points automatically distributed over an existing triangulated surface (either evenly or according to curvature) [Turk92], These techniques suffer from expensive preprocessing. but yield highly optimized results.

Techniques for constructing approximating **inner and outer bounds** (or offset surfaces) for polyhedra have been extended to create simplified models that separate the inner and the outer offsets [Varshney94, Cohen96, Mitchell95]. The creation of valid, intersection-free offset surfaces still poses some challenges, although techniques based on extended octree representations of the interior and of the boundary of the polyhedron provide inner and outer bounds [Andujar96].

The rest of this paper focuses on **polygon count reduction** techniques that exploit an original triangular mesh and derive simplified models by eliminating vertices or triangles, by collapsing edges, or by merging adjacent faces.

Vertex clustering, the simplest to implement and most efficient approach, groups vertices into clusters by coordinate quantization (round-off), computes a representative vertex for each cluster, and removes degenerate triangles which have at least two of their vertices in the same cluster [Rossignac93]. All vertices whose coordinates round-off to the same value are merged. The accuracy of the simplification is thus controlled by the quantization parameters (see Figure 3).

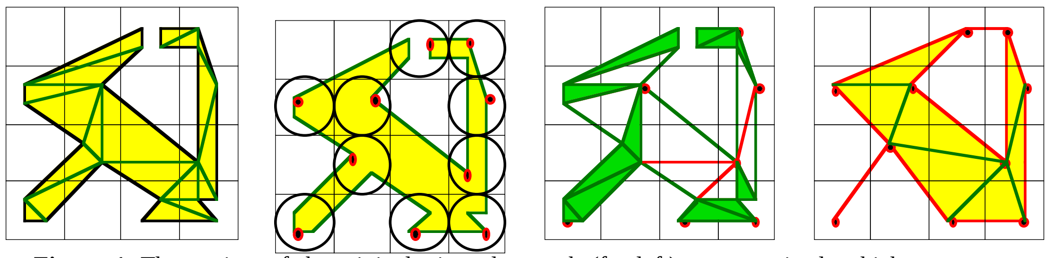

Figure 1: The vertices of the original triangular mesh (far left) are quantized, which amounts to associating each vertex with a single cell of a regular subdivision of a box then encloses the object. Cells which contain one or more vertices are marked with a circle (center left). All the vertices that lie in a given cell form a cluster. A representative vertex is chosen for each cluster. It is indicated with a filled dot. The other vertices of each cluster are collapsed into one and placed at the representative vertex for the cluster (far right). Triangles having more than one vertex in a cluster collapse during this simplification process (shaded triangles center right). They will be removed to accelerate the rendering of approximated versions of the object while other neighboring triangles may expand. The resulting model (far right) has fewer vertices and triangles, but has a different topology. Notice that a thin area collapsed into a single line, while a gap was bridged by a different line segment.

A different technique, **edge collapsing,** merges the two end points of the edges of the polyhedron one edge at a time [Hoppe93, Ronfard96].. Each edge collapse operation removes two triangles (see Figure 2). The accuracy of the resulting model is estimated as the error cumulated by the sequence of edge-collapses. This incremental process permits to achieve the desired accuracy or the desired triangle count and is well suited for optimization (selection of the sequence of edge collapsing operations which results in the lowest error). Early version of edge collapsing techniques are more complex to implement than vertex clustering approaches (they require maintaining a complete incidence graph), are significantly slower, and impose strong topological constraints on the input polyhedra. Edge collapsing is in fact a restricted form of vertex clustering, since an edge collapse operation merges two clusters that are linked by an edge of the polyhedron. Both techniques may be combined [Hoppe97].

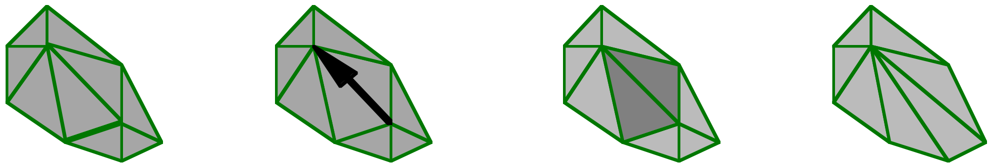

Figure 2: From left to right: the original triangle mesh, a selected edge showing the direction of collapse, the two triangles that will be eliminated during the collapse, and the resulting simpler mesh.

The **decimation** of a nearly-flat vertex [Schroeder92] is equivalent to an edge collapsing operation, possibly combined with edge-flips [Lawson72] (Fig. 3), and thus results in a hierarchical clustering of vertices. Its generalization [Kalvin91, Kalvin96] re-triangulates nearly flat regions constructed by incrementally merging triangles. Re-triangulating a region amounts to clustering all the internal vertices into a single "star" vertex, the apex of the new triangulation for the region.

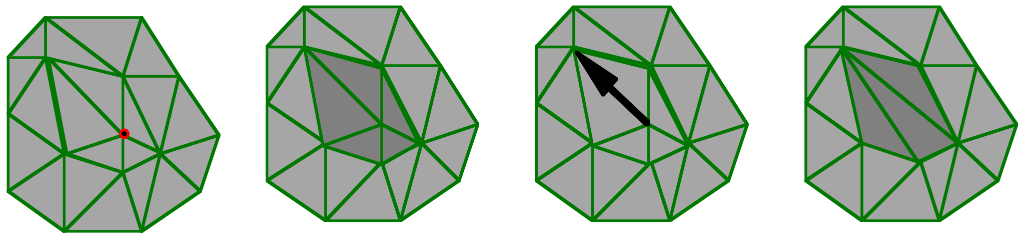

Figure 3: Decimating the highlighted vertex (far left) will affect the faces marked (center left). The affected region will be retriangulated (far right). The transformation corresponds to the edge collapse indicated by the arrow (center right).

Static simplifications are effective for **large and complex objects** only when these objects are viewed from far. Inspecting the details of a local feature on such a large complex object requires using the highest resolution for the entire object, which imposes that full resolution be used for the distant parts of the objects that could otherwise be displayed at much lower resolution. It is not uncommon that the complexity of a single object significantly exceeds what the graphics subsystem can render at interactive rates. An **adaptive multi-resolution** model is better suited for such situations [Xia96]. In fact, an adaptive model may be computed for the entire scene. An hierarchical version of the vertex clustering approach of [Rossignac93] is the basis of a new adaptive scheme [Luebke96], where octree nodes [Samet90] correspond to the hierarchy of vertex clusters. The main research challenges in the perfection of adaptive multiresolution models lie in: (1) the rapid generation of triangle strips for each view dependent simplification., (2) a fast decision process updating the levels of resolution at each new frame, and (3) the estimation of a tight error bound. Although it may be expensive to convert arbitrary triangular meshes into subdivision surfaces, wavelet-based hierarchical models offer an attractive and elegant solution to this problem [Eck95, Lounsbery94].

Adaptive level-of-detail for **terrain models** have been studied extensively [Garland95], because a terrain model is typically a single complex surface that must be rendered with non-uniform resolution and partly because error estimations for terrain models is simpler than for arbitrary 3D polyhedra. For example, hierarchical triangulated irregular network (HTIN), were proposed [DeFloriani92] as an adaptive resolution model for topographic surfaces. Researchers at Georgia Tech's GVU center have developed an adaptive multi-resolution model for the realtime visualization of complex terrain data over a regular grid [Lindstrom96]. Their approach uses a recursive decision tree to indicate which edges may be collapsed and also to set preconditions: an edge may not be collapsed unless its children have been collapsed.

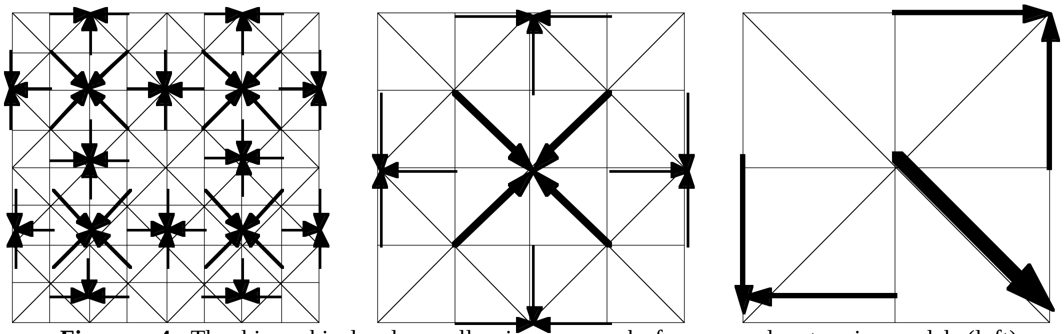

Figure 4: The hierarchical edge collapsing approach for a regular terrain model (left) produces a similar lower-resolution models by a sequence of vertical and then diagonal edge collapses (center). The process may be repeated iteratively (right). Not all edges at a given level need to collapse.

Review of 3D compression techniques

In comparison to image and video compression, little attention has been devoted to the compression of 3D shapes, both from the research community and from 3D data exchange standards committees. This situation is likely to change rapidly for three reasons: (1) the exploding complexity of industrial CAD models raises significantly the cost of the memory and auxiliary storage required by these models, (2) the distribution of 3D models over networks for collaborative design, gaming, rapid prototyping, or virtual interactions is seriously limited by the available bandwidth, and (3) the graphics performance of high level hardware adapters is limited by insufficient on-board memory to store the entire model or by a data transfer bottleneck.

We focus our study of 3D compression on triangular meshes which are shells of pairwise adjacent triangles. We chose triangle-based representations, because more general polygonal faces may be efficiently triangulated [Ronfard94], and because triangles provide a common denominator for most representation schemes. Furthermore, the number of triangles is a convenient measure of a model's complexity, which is important for comparing various compression techniques. We will mostly focus here on simply connected manifold meshes, where triangles are mutually disjoint (except at their edges and vertices), where each edge is adjacent to exactly two incident triangles, and where each vertex is adjacent to exactly one cone of incident triangles. Furthermore, we will assume for simplicity that the mesh is a connected surface with zero handles. These restrictions are made purely for the sake of simplicity, and most of the compression schemes reviewed here work or may be expanded to cope with more general meshes.

A triangular mesh is defined by the position of its vertices (geometry), by the association between each triangle and its sustaining vertices (incidence), and by color, normal, and texture information (photometry), which does not affect the 3D geometry, but influences the way the triangle is shaded.. What is the minimum number of bits required to encode a triangle mesh of **T** triangles and **V** vertices? Because for our simple meshes there are roughly twice more triangles than vertices, we will assume that **T=2V** and use **V** as a measure of the complexity of the mesh. For uniformity, we will assume that a unique normal (photometry) is associated with each vertex and that these normals are different for all vertices. To illustrate the storage requirement, we will pick **V** to be 2^{16} (i.e. 64K), which is a reasonable compromise between an average of about 500 vertices per solid in mechanical assemblies and significantly larger vertex counts for large architectural objects, terrain models, or medical datasets.

We compare below the storage requirements for several representation schemes. Of course other general purpose loss-less compression schemes [Pennebakerl93] may be applied to the bit stream resulting from these approaches, they will not be considered in our comparison.

An **array of triangles** is the simplest representation of a triangle mesh, It represents each triangle independently by the list of its vertex and normal coordinates, each represented by a 4 byte floating point number. Hence the number of bits per vertex for this simple representation of a triangle mesh of **V** vertices is **1152,** the product of the following terms:

- 2 triangles per vertex
- 3 vertex uses per triangle
- 2 vectors (vertex location and normal) per vertex use
- 3 coordinates per vector
- 4 bytes per coordinate
- 8 bits per byte

Note that a vertex is on average adjacent to 6 triangles and therefore, there are 6 vertex uses (i.e., descriptions of the same vertex (geometry and photometry) stored in the simple representation above, which does not need to encode explicitly the triangle-vertex incidence relation, since it is dictated by the place of the vertices in the data stream.

This redundancy can be eliminated by dissociating the representation of the vertices (location and normal) from the representation of the incidence relation, which requires 3 vertex references per triangle. Because a vertex reference often requires less bits than a vertex description, such schemes based on a **vertex and normal table** are more compact than the simple representations. In our example, if we store vertices in a table and reference them by an integer index, we need 16 bits for each vertex reference, since **$V=2^{16}$.** Consequently, the vertex and normal table representation requires **288** bits per vertex. **192** bits for the geometry and photometry:

- 2 vectors (vertex location and normal) per vertex
- 3 coordinates per vector
- 4 bytes per coordinate
- 8 bits per byte

and **96** bits for the incidence information:

- 2 triangles per vertex
- 3 vertex references per triangle
- 16 bits per vertex reference

However, such a compression scheme requires random access to the vertices, which is acceptable in graphics systems with software geometry processing, but not suitable for fully hardware graphics systems, where there is limited register memory for a few vertices and associated normals.

A representation based on **triangle strips**, supported by popular graphics APIs, such as OpenGL [Neider93], is used to provide a good compromise for graphics. It reduces the repeated use of vertices from an average of 6 to an average of 2.4 (assuming an average length 10 triangles per strip, which is not easily achieved) and is compatible with the graphics hardware constraints. Basically, in a triangle strip, a triangle is formed by combining a new vertex description with the descriptions of the two previously sent vertices, which are temporarily stored in two buffers. The first two vertices are the overhead for each strip, so it is desirable to build long strips, but the automation of this task remains a challenging problem [Evans96].With our assumptions, triangle strips require **461** bits, the product of the following terms:

- 2 triangles per vertex
- 1.2 vertex uses per triangle
- 2 vectors (vertex location and normal) per vertex use
- 3 coordinates per vector
- 4 bytes per coordinate
- 8 bits per byte

The absence of the swap operation in the OpenGL further increases this vertex-use redundancy, since the same vertex may be sent several times in a triangle strip to overcome the left-right-left-right patterns of vertices imposed by OpenGL. This latter constraint does not affect the suitability of triangle strips for data compression.

Because on average a vertex is used twice, either as part of the same triangle strip or of two different ones, the use of triangle strips requires sending most vertices multiple times. Deering's **generalized strips** [Deering95] extend the two registers used for OpenGL triangle strips to a 16 registers stack-buffer where previous vertices may be stored for later use. Deering generalizes the triangle strip syntax by providing more general control over how the next vertex is used and by allowing the temporary inclusion of the current vertex in the stack-buffer and the reuse of any one of the 16 vertices of the stack-buffer. One bit per vertex is used to indicate whether the vertex should be pushed onto the stack-buffer. Two bits per triangle are used to indicate how to continue the current strip. One bit per triangle indicates whether the next vertex shoul d be read from the input stream or retrieved from the stack. 4 bits of address are used for randomly selecting a vertex from the stack-buffer, each time an old vertex is reused. Assuming that each vertex is reused only once, the total cost for encoding the connectivity information is: 1+4 bits per vertex plus 2+1 bits per triangle. Assuming 2 triangles per vertex, this amounts to 11 bits per vertex. Algorithms for systematically creating good traversals of general meshes using Deering's generalized triangle mesh syntax are not available and naive traversal of arbitrary meshes may result in many isolated triangles or

small runs, implying that a significant portion (for example 20%) of the vertices will be sent more than once, and hence increasing the number of incidence bits per vertex to **13.** Deering also proposed to use coordinate and normal quantization and entropy encoding on the vertex coordinates expressed in a local coordinate system and on the normal coordinates to reduce the geometric and photometric storage costs. Basically, a minimax box around the mesh is used to define an optimal resolution coordinate system for the desired number of bits.
This quantization (which results from the integer rounding of the vertex coordinates to the units of the local coordinate system) is a lossy compression step. Deering then further compresses the coordinates by using an optimal variable length coding for the most frequent values of the discrete vertex coordinates and normal parameters. Because lossy compression schemes are difficult to compare, we will assume that a vertex-normal pair can be encoded using **48** bits per vertex for the geometry and photometry. Deering reports about 36 bits for the geometry alone. This yields a total of **61** bits per vertex.

Hoppe's **Progressive Meshes** [Hoppe96] define each vertex the identification of 2 incident edges in the previously constructed mesh, and a displacement vector for computing the location of the new vertex from the location of the common vertex to these two edges The construction is illustrated in Figure 5. Given that there are 3 times more edges than vertices, an index for identifying the first edge takes 18 bits. Given that there are on average 10 edges adjacent to an edge, we need 4 extra bits to identify the second edge The total connectivity storage cost per vertex would then be 22 bits. An entropy coding could further reduce this cost. Hoppe actually uses a vertex index to identify the shared vertex (which would require 16 bits in our example) and a combined 5 bit index identifying the two incident edges, resulting in **21** bits per vertex for the incidence.
Because Hoppe's corrective vectors are in general shorter than Deering's relative vectors, a Huffman coding should further improve the geometric and photometric compression. For simplicity, we will use the same cost for the geometry and photometry of **48** bits per vertex. The total reaches **69** bits per vertex, but the advantage of Hoppe's scheme lies not in its compression benefits, but in its suitability for scaleable model transmission, where a rough model is sent first and then a sequence of vertex insertion operations is sent to progressively refine the model. A progressive model is generated by storing in inverse order a sequence of simplifying edge collapses.

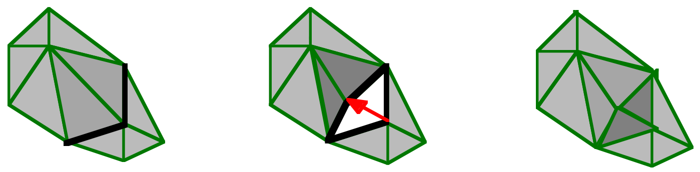

Figure 5: Two adjacent edges are identified (left). These edges are duplicated to produce a cut. The replicated instance of their common vertex is displaced the specified displacement vector (center). This creates a gap, which implicitly defines two new triangles.

Under the assumption that all vertex coordinates are available for random access during decompression, Taubin and Rossignac's **Topological Surgery** method [Taubin96] yields results comparable to Hoppe's Progressive Meshes for the geometric and photometric compression (**48** bits per vertex) and improves on Deering's approach for coding the incidence information by a factor of 4 (about between **2** and **4** bits, depending on the incidence graph). This yields a total of **52** bits. The approach is reviewed in more details in this report.

Estimating simplification errors

A simplification process takes a triangulated surface S and produces a simplified triangulated surface S'. The error that may be perceived when using S' instead of S in a scene depends on the shape of S and S' and also on the viewing conditions (size and orientation of the shape on the screen). It is vital to have a precise error evaluation or at least a tight and guaranteed upper bound on the error, otherwise, simplification schemes may produce simplified models that are not suitable substitutes for the original shape. The error bound will guide our selection of the appropriate level of detail that meets the desired graphics fidelity.

Although the image is defined only by the color components displayed at each pixel, it is useful to distinguish two types of errors: **geometric errors** (how far are the pixels from where they should be) and **color errors** (how do the pixels covered by the same feature of the shape differ in color), when evaluating the accuracy of a simplification technique. These two errors are discussed below.

Since, in general, we cannot predict the viewing conditions, a view-independent error estimation is usually computed during simplification and used during rendering to guide the choice of level of detail. To make this possible, one needs to compute an view-independent error representation that leads to a simple and efficient estimation of the corresponding view-dependent error. For example, if the a point on the surface was displaced along the normal to that surface by a vector **D**, the perceived error will depend on the projected size of **D** on the screen and on the resulting side effects (for instance the changes in the orientation of the surrounding faces). Hence, the worst viewing conditions for the geometric error may occur when the vector **D** is orthogonal to the viewing direction, while the worst viewing condition for the color error depends on the relative orientation of the light sources, but will typically be proportional to the area occupied on the screen by the faces whose shape was altered by the displacement of the point. On average, this area is the largest when **D** is parallel to the viewing direction. From this example, it should be clear that it is rather difficult to provide a tight bound on the color error and that it is useful to consider separately the tasks of estimating the two errors.

Hausdorff estimate of the geometric error

Several simplification techniques reported in the literature fail to provide a proper error bound and use instead heuristic estimates to guide the simplification process or the selection of the level of detail during rendering. Such strategies often results in simpler algorithms and sometimes to faster processing, and lead to impressive demonstrations. They may be well suited for many entertainment applications, but may not be appropriate for professional applications, where unexpectedly high errors may mislead the users.

Several techniques have been proposed for computing a provable upper bound on the geometric error. They cover a wide spectrum of compromises between the simplicity of the calculations, the performance of the preprocessing steps, and the tightness of the bound.

The geometric error may be measured using the Hausdorff distance between S and S', defined as **H**(S,S')=**max**(**dev**(S,S'),**dev**(S'S)), where **dev**(A,B)=**max**(**dist**(a,B)) for a∈A, and **dist**(a,B)=**min**(||a=b||) for b∈B. It measures the worst case distance that a point on one surfaces would have to travel to reach the other surface. Note that **H** is a symmetric version (i.e, **H**(A,B)=**H**(B,A)) of the deviation, **dev**(A,B), of A from B, and is not to be confused with the minimum distance, **dist**(A,B), between the two surfaces, as illustrated in Figure 6.

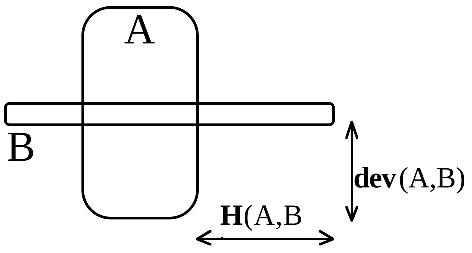

Figure 6: In this example, **dist**(A,B)=0 because the two sets intersect. The Hausdorff distance **H**(A,B) between the two sets is equal to the deviation, **dev**(B,A), but not to the deviation **dev**(A,B).

The Hausdorff distance provides a bound on the maximal geometric deviation between the two shapes and hence provides the maximum distance between a point on the profile (i.e. visible silhouette) of one object and the profile of another. We use the term profile to refer to visible edges that are adjacent to at least one front facing and one back facing face. Note that although the profile is a notion that depends on the view, the Hausdorff distance is not. The Hausdorff distance captures the worst case situation, where the line joining a point on one surface and its closest counterpart on the other surface is orthogonal to the viewing direction.

Although the Hausdorff distance of zero between two sets indicates that the sets are identical, in general, the Hausdorff distance is not a good measure of shape similarity, nor of proximity of surface orientation between two shapes (see Fig. 7 for a counterexample).

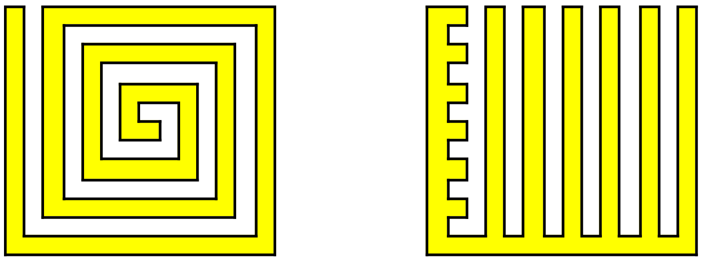

Figure 7: When translated for perfect alignment, these two shapes will have a relatively small Hausdorff distance, yet they differ significantly in their shape and structure.

The cost of computing the Hausdorff distance between two polyhedra is significant, because it does not suffice to check the distances between all the vertices of one set and the other set. One must also be able to detect configurations where the Hausdorff distance is realized at points that lie in the middle of faces. Figure 8 illustrates this concept in two dimensions. Consequently, simpler, although less precise bounds are often used.

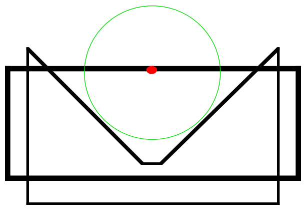

Figure 8: The Hausdorff distance between the two polygons is realized at a point on and edge, not at a vertex.

A geometric error bound based on vertex displacement

Because triangles are linear convex combinations of vertices, if no vertex of a triangle **T** has moved by more than a distance **D** then no interior point in **T** has either. Consequently, if we keep track of how much we move the original vertices, we will have an upper bound on the total error. Both vertex clustering and edge collapses may be viewed as a two step process:

1. Move selected vertices without changing the incidence graph,
2. Change the incidence graph to remove degenerate triangles without changing the geometry.

Voxels or octrees that contain the boundary may also be used to define tolerance zones for safe simplification steps within a prescribed tolerance [Airey90].

When the vertices of a cluster are collapsed into a single representative vertex, the maximum error is the **maximum distance** between the original vertices of the cluster and their representative vertex. This computation requires only one pass through the vertices of the cluster after the representative vertex was chosen. An even less expensive although more pessimistic error bound may be obtained by using half the length of the diagonal of a cell used for vertex quantization.

Although the same approach may be used for the first round of edge collapses, as clusters of vertices resulting from an previous edge collapses are merged into larger clusters, it becomes more expensive to keep track of all the vertices and to check their distance from the representative vertex. A pessimistic, yet simple estimate would be to use the maximum of the sum of all the length of the path from the representative vertex to the leaves of an edge collapse hierarchy.

A geometric error bound based on distances to supporting planes

Vertex displacement provides a rather pessimistic error bound. Imagine for example a simplification of a dense triangulation of a flat portion of a terrain. Although vertices may move during edge collapse or vertex clustering operations, the represented geometry remains flat, and therefore the error is zero. Ronfard and Rossignac have introduced an error estimator which measures the displacement of vertices in the direction orthogonal to their incident faces [Ronfard96]. Their estimator computes the maximum distance from the new location **P** of the vertex to all the supporting planes of the vertices that have collapsed into **P**. This estimator works well for flat and nearly flat regions, but may not provide an upper bound close to sharp vertices, as shown in Figure 9. In the cases of sharp corners, the authors introduce an additional plane which suffices to guarantee a precise upper bound on the error.

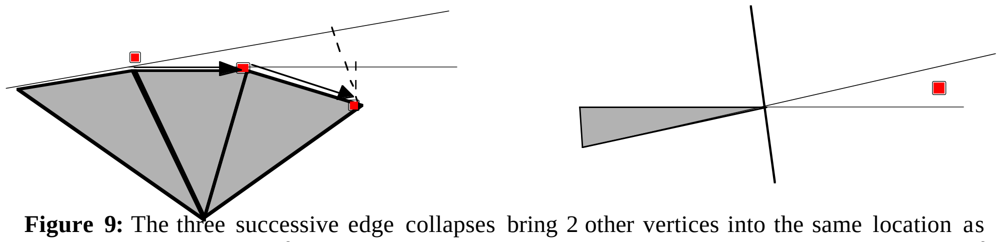

Figure 9: The three successive edge collapses bring 2 other vertices into the same location as a third vertex (marked left). The error is estimated by computing the maximum deviation of these three to the set of lines that support their incident edges. However, in sharp corners, the lines are almost parallel and the vertex could move far before the distance to the supporting lines for the incident edges in its initial position exceed the allowed threshold (right). Therefore, an additional line is introduced to limit this excursion.

Minimizing color errors

The color error is somewhat more subjective, because it requires computing distances in color space, and also harder to control, because slight variation in face orientations may result in considerable variation in specular reflections. Such color errors are best addressed by using photometric properties that are not affected by surface orientation, such as texture maps.

Note that using for the simplified model the normals of the original model may produce excellent results for faces that are roughly orthogonal to the viewing directions, but may produced undesirable black spots in situations where the face is still facing the viewer, but the normal inherited from the original model is not, end hence leads to a black color for the corresponding vertex.

Rossignac and Borrel’s vertex quantization

The geometric simplification introduced in [Rossignac93] is aimed at very complex and fairly irregular CAD models of mechanical parts. It operates on boundary representations of an arbitrary polyhedron and generates a series of simplified models with a decreasing number of faces and vertices. The resulting models do not necessarily form valid boundaries of 3D regions--for example, an elongated solid may be approximated by a curve segment. However, the error introduced by the simplification is bounded (in the Hausdorff distance sense) by a user-controlled accuracy factor and the resulting shapes exhibit a remarkable visual fidelity considering the data-reduction ratios, the simplicity of the approach, and the performance and robustness of the implementation.

The original model of each object is represented by a vertex table containing vertex coordinates and a face table containing references to the vertex table, sorted and organized according to the edge-loops bounding the face. The simplification involves the following processing steps:

1. grading of vertices (assigning weights) as to their visual importance
2. triangulation of faces
3. clustering of vertices
4. synthesis of the representative vertex for each cluster
5. elimination of degenerate triangles and of redundant edges and vertices
6. adjustment of normals
7. construction of new triangle strips for faster graphics performance

Grading

A weight is computed for each vertex. The weight defines the subjective perceptual importance of the vertex. We favor vertices that have a higher probability of lying on the object's silhouettes from an arbitrary viewing direction and vertices that bound large faces that should not be affected by the removal of small details. The first factor may be efficiently estimated using the inverse of the maximum angle between all pairs of incident edges on the candidate vertex. The second factor may be estimated using the face area.

Note that in both cases, these inexpensive estimations are dependent on the particular tessellation. For example, subdividing the faces incident upon a vertex will alter its weight although the actual shape remains constant. Similarly, replacing a sharp vertex with a very small rounded sphere will reduce the weight of the corresponding vertices, although the global shape has not changed. A better approach would be to consider the local morphology of the model (estimate of the curvature near the vertex and estimate of the area of flat faces incident upon the vertex (see [Gross95] for recent progress in this direction).

Triangulation

Each face is decomposed into triangles supported by its original vertices. Because CAD models typically contain faces bounded by a large number of edges, a very efficient, yet simple triangulation technique is used [Ronfard94]. The resulting table of triangles contains 3 vertex-indices per triangle.

Note that attempting to simplify non-triangulated faces will most probably result in non-flat polygons. On the other hand, it may be beneficial to remove very small internal edge loops from large faces prior to triangulation. This approach will significantly reduce the triangle count in mechanical CAD models, where holes for fasteners are responsible for a major part of the model’s complexity. Simplifying triangulated models without removing holes will not create cracks in the surface of the solid and will not separate connected components, Removing holes prior to simplification may result in separation of connected components or in the creation of visible cracks.

Clustering

The vertices are grouped into clusters, based on geometric proximity. With each vertex, we associate the corresponding cluster's id. Although a variety of clustering techniques was envisioned, we have opted for a simple clustering process based on the truncation (quantization) of vertex coordinates. A box, or other bound, containing the object is uniformly subdivided into cells. After truncation of coordinates, the vertices falling within a cell will have equal coordinates. A cell, and hence its cluster is uniquely identified by its three coordinates. The clustering procedure takes as parameters the box in which the clustering should occur and the maximum number of cells along each dimension. The solid's bounding box or a common box for the entire scene may be used. The number of cells in each dimension is computed so as to achieve the desired level of simplification. A particular choice may take into account the geometric complexity of the object, its size relative size and importance in the scene, and the desired reduction in triangle count. The result of this computation is a table (parallel to the vertex table) which associates vertices with cluster indices (computed by concatenating cluster integer coordinates).

This approach does not permit to select the precise triangle reduction ratio. Instead, we use a non-linear estimator and an adaptive approach to achieve the desired complexity reduction ratios. For instance, given the size and complexity of a particular solid relative to the entire scene, we estimate the cell size that would yield the desired number of triangles, we run the simplification, and if the result is far from our estimate, we use it for a different level of detail and adjust the cell size appropriately for the next simplification level.

Synthesis

The vertex/cluster association is used to compute a vertex representative for each cluster. A good choice is the vertex closest to the weighted the average of the vertices of the cluster, where the results of grading are used as weights. Less ambitious choices, permits to compute the cluster's representative vertices without reading the input data twice, which leads to important performance improvements when the input vertex table is too large to fit in memory. Vertex/cluster correspondence yields a correspondence between the original vertices and the representative vertices of the simplified object. Thus, each triangle of the original object references three original vertices, which in turn reference three representative vertices. (Note that representative vertices are a subset of the original vertices, although a simple variation of this approach will support an optimization step that would compute new locations of the representative vertices.) The representative vertices define the geometry of the triangle in the simplified object.

The explicit association between the original vertices and the simplified ones permits to smoothly interpolate between the original model and the simplified one. The levels of detail may be computed in sequence, starting from the original and generating the first simplification, then starting with this simplified model and generating the next (more simplified) model and so on. This process will produce a hierarchy of vertex clusters, which may be used to smoothly interpolate between the transitions from one level to the next, and hence to avoid a distracting popping effect. We have experimented with such smooth transitions and concluded that, although visually pleasant, they benefit did not justify the additional interpolation and book-keeping costs. Indeed, during transition phases, the faces of a more detailed simplification must be used when the lower level of detail may suffice to meet the desired accuracy. For example, consider that simplification 2 contains 1000 triangles and corresponds to an error of 0.020, and that simplification 3 contains 100 triangles and corresponds to an error of 0.100. If the viewing conditions impose an error cap of 0.081, we could use simplification 2 alone and display only 100 triangles. If however we chose to use a smooth interpolation between consecutive levels in the transition zone for errors between 0.080 and 1.020, we would have not only to compute a new position for 500 vertices as a linear combination of two vertices, but we will have to display 1000 triangles. Consequently, the smooth interpolation will result in significant runtime processing costs and in an order of magnitude performance drop for this solid. Assuming uniform distribution, this penalty will be averaged amongst the various instances (only 40% of instances would be penalized at a given time). The total performance is degraded by a factor of **4.6**.

Also note that in order to prevent the accumulation of errors, when a level of detail is computed by simplifying another level of details, the cells for the two simplification processes should be aligned and the finer cells should be proper subdivisions of the coarser cells.

Elimination

Many triangles may have collapsed as the result of using representative vertices rather than the original ones. When, for a given triangle, all three representative vertices are equal, the triangle degenerates (collapses) into a point. When exactly two representative vertices are equal the triangle degenerates into an edge. Such edges and points, when they bound a triangle in the simplified object are eliminated. Otherwise, they are added to the geometry associated with the simplified model. Duplicated, triangles, edges, and vertices are eliminated during that process. Efficient techniques may be invoked, which use the best compromise between space and performance. When the number of vertices in the simplified model is small, a simple hashing scheme [Rossignac93] will yield an almost linear performance. When the number of vertices in the simplified model is large, duplicated geometries may be eliminated at the cost of sorting the various elements. The approach of [Rossignac93] has been re-engineered at IBM Research by Josh Mittleman and included in IBM's 3D Interaction Accelerator system [3DIX]. The implementation uses a few simple data-structures and sorting to achieve **O(V)** space and **O(V log V)** time complexities for a solid containing **V** vertices.

Adjustment of normals

This step computes new normals for all the triangles coordinates. It uses a heuristic to establish which edges are smooth. The process also computes triangle meshes. We use a face clustering heuristics which builds clusters of adjacent and nearly coplanar faces amongst all the incident faces of each vertex. An average normal is associated with the vertex-use for all the faces of a cluster.

Generation of new triangle strips

Because each simplification reduces the model significantly, it is not practical to exploit triangle strips computed on the original model. Instead, we re-compute new triangle strips for each simplified model.

Runtime level selection

Several levels of detail may be pre-computed for each object and used whenever appropriate to speed up graphics. In selecting the particular simplification level for a given object, it is important to take into accounts the architecture of the rendering subsystem so as not to oversimplify in situations where the rendering process is pixel bound. For example, the cost of rendering in software and in a large window an object that has a relatively low complexity but fills most of the screen is dominated by **R**. Consequently, simplification will have very little performance impact, and may reduce the image fidelity without benefit. On the other hand, displaying a scene with small, yet complex objects, via a software geometric processing on a fast hardware rasterizer will be significantly improved by simplification before the effects of using simplified models become noticeable.

Isolated edges, that result from the collapsing of some triangles may be displayed as simple edges whose width is adjusted taking into account their distance to the viewer.

Advantages and implementation

The process described above has several advantages over other simplification methods:

- The computation of the simplification does not require the construction of a topological adjacency graph between faces, edges, and vertices. It works of a simple array of vertices and of an array of triangles, each defined in terms of three vertex-indices.
- The algorithm for computing the simplification is very time efficient. In its simplest form, it needs to traverse the input data (vertex and triangle tables) only once.
- The tolerance (i.e. bound on the Hausdorff distance between the original and simplified model) may be arbitrarily increased reducing the triangle count by several orders of magnitude.
- To further reduce the triangle count, the simplification algorithm may produce non-regularized models. Particularly, when using the appropriate tolerance, thin plates may be simplified to dangling faces, long objects to isolated edges, and (groups of) small solids into isolated points.
- The approach is not restricted by topological adjacency constraints and may merge features that are geometrically close, but are not topologically adjacent. Particularly, an arbitrary number of small neighboring isolated objects may be merged and simplified into a single point.
- The simplification algorithm was combined with the data import modules of IBM's 3D Interaction Accelerator and exercised on hundreds of thousands of models of various complexity. It exhibits a remarkable performance characteristics, making it faster to re-compute simplifications than to read the equivalent ASCII files from disk. The algorithm pre-computes several levels of detail, which are then used at run time to accelerate graphics during interaction with models comprising millions of triangles. The particular simplification level of a given object is computed so as to match a user specified performance or quality target while allocating more geometric complexity (and thus more rendering cost) to objects which a higher visual importance.
- Our experience shows that typical CAD models of mechanical assemblies comprise dozens of thousands of objects. The relative size and complexity of the objects may vary greatly. A typical object may have a thousand triangles in its original boundary. The simplification process described here may be used to automatically reduce the triangle count by an average factor of 5 without impacting the overall shape and without hindering the users ability to identify the important features. Further simplifications lead to further reduction of the triangle count, all the way down to a single digit, while still preserving the overall shape of the object and making it recognizable in the scene. This simplification process has been further improved to yield a better fidelity/complexity ratio by incorporating topological and curvature considerations in the clustering process. However, these improvements only lead to additional computational costs and more complex code.

This approach was recently improved by Low and Tan [Low97], who suggest:

- a better grading of vertices (use **cos(a/2**) rather than **1/a**, where **a** is maximum angle between all pairs of incident edges),
- a floating-cell clustering where the highest weight vertex in a cell attracts vertices in its vicinity from immediately adjacent cells,
- shading edges that approximate elongated objects.

Ronfard and Rossignac’s edge collapsing

The principle of the edge-collapsing approaches is to iteratively collapse pairs of vertices that are connected by an edge of the polyhedron into a single new vertex that may be positioned at one of the original two vertices or in a new position, so as to minimize the error resulting from the transformation. The main differences between the various approaches lie in the techniques for estimating an error bound associated with each candidate edge and the optimization criteria for positioning the new vertex.

The technique developed by Ronfard and Rossignac at IBM Research [Ronfard96] associates with each vertex a compact description of the planes that support each of the incident triangles and possibly additional planes through sharp incident edges. These are used to define and evaluate approximation constraints. The user may wish for example to ensure that no vertex moves further away from each one of these planes than a prescribed distance. The maximum distance between a point and these planes is used as an error bound estimator. The advantage of this criteria for error estimation lies in the fact that it enables vertices to travel far away from their original location along nearly planar regions.

Initially, an error estimate is computed for each edge collapsing operation and the edges are sorted in a priority queue. At each step, the best edge is chosen, so as to minimize the total error estimate. When two vertex clusters are merged, their constraints (lists of planes) are merged and possibly pruned to keep the lists short.

Each collapsing operation only alters immediate neighbors and therefore, the a new error bound for each one of the neighboring edges may be quickly estimated and the priority queue updated.

Advantages and drawbacks

The method guarantees a tight error bound and permits to move vertices further away from their original location as long as they move along nearly parallel faces. Consequently, it produces much lower triangle counts than the vertex clustering method for the same error bound.

However, this method requires maintaining a face-edge-vertex incidence graph and special treatment to allow topological changes, such as the elimination of collapsed holes (see also [He96]).

Gueziec’s volume preserving simplification

Gueziec's algorithm, developed at IBM Research, preserves the volume of the original polyhedron during simplification [Gueziec96], It favors the creation of near-equilateral triangles. Gueziec computes an upper bound to the approximation error and reports this bound using a novel tool, the error volume, which is constructed by taking the union of balls centered on the surface, whose radii, the error values vary linearly between surface vertices. Gueziec guarantees that the error will be less than user specified tolerances, which can vary across the surface, as opposed to a single tolerance. An originality of his method is that errors and tolerances are defined with respect to the simplified surface, as opposed to the original surface.

Following the approach of [Ronfard96], the algorithm uses a greedy strategy based upon the edge collapsing operation. Edges are weighted and sorted in a priority queue. Before collapsing the edge with the lowest weight into a simplified vertex, tests determine whether the simplification is appropriate. The simplified vertex is positioned such that the volume enclosed by a closed surface will stay the same. Gueziec shows that the simplified vertex must lie on a specific plane. On that plane, the vertex position minimizes the sum of squared distances to the planes of the edge star, which corresponds to minimizing a sum of distances to lines in the plane. Once the optimum position of the simplified vertex is found, Gueziec verifies that the triangle orientations are not perturbed by a rotation exceeding a user specification. Also, he verifies that the minimum value for the triangle aspect ratios, as measured with the triangle compactness or ratio between area and perimeter, is not degraded in excess of a pre-specified factor.

Gueziec also determines the effect of the edge collapse on the overall approximation error, and tests whether it is tolerable. The error volume is initialized with error values equal to zero, or with positive error values reported by a previous simplification process. As the simplification progresses, the error values at the remaining vertices are gradually updated. In a manner similar to Russian Dolls, a hierarchy of nested error volumes is built such that each new volume is guaranteed to enclose the previous error volume. A linear optimization is performed to compute error estimates that minimize their overall error volume. The simplification stops in a particular region when the width of the error volume reaches the tolerance. Assuming that a bound to the maximum valence at a vertex is respected, Gueziec shows that the overall computational complexity of his method is sub-quadratic in the number of edges of the surface.

The algorithm has the following limitations: Since it deals with the problem of finding a sub optimal approximation with a given error bound, it cannot guarantee that a given triangle reduction can be obtained, nor can it guarantee that a it reaches the minimum number of triangles. The current implementation is computationally intensive, mainly due to a linear programming step used for maintaining dynamically the error volume. Finally, although a pre-processing step could assign vertex tolerances that would prevent surface self-intersections, the current program allows them where tolerance volumes overlap. The algorithm assumes that the topology of the input surface is that of a manifold; this property is then also verified for the output surface.

Kalvin and Taylor's face-merging

Although it is domain independent, the Superfaces algorithm has been originally developed by Alan Kalvin and Russ Taylor at IBM Research for simplifying polyhedral meshes [Kalvin96] that result from building iso-surfaces of medical data-sets. The simplification is based on a bounded approximation criterion and produces a simplified mesh that approximates the original one to within a pre-specified tolerance. The vertices in the simplified mesh are a proper subset of the original vertices, so the algorithm is well-suited for creating hierarchical representations of polyhedra.

The algorithm simplifies a mesh in three phases:

- Superface creation: A "greedy", bottom-up face-merging procedure partitions the original faces into superface patches.
- Border Straightening: The borders of the superfaces are simplified, by merging boundary edges into superedges,
- Superface Triangulation: Triangulation points for the superfaces are defined. In this phase, a single superface may be decomposed into many superfaces, each with its own boundary and triangulation point.

Advantages and drawbacks

The Superfaces algorithm has the following advantages:

- It uses a bounded approximation approach which guarantees that a simplified mesh approximates the original mesh to within a pre-specified tolerance. That is, every vertex **v** in the original mesh is guaranteed to lie within a user-specified distance of the simplified mesh. Also, the vertices in the simplified mesh are a subset of the original vertices, so there is zero error distance between the simplified vertices and the original surface.
- It is fast. The face-merging procedure is efficient and greedy-- that is, it does not backtrack or undo any merging once completed. Thus the algorithm is practical for simplifying very large meshes.
- It is a general-purpose, domain-independent method.

Disadvantages of the Superfaces algorithm are:

- It can produce simplified surfaces that self-intersect.
- The current triangulation procedure (that uses no Steiner points) can produce skinny triangles, that are not desirable for rendering.

Taubin and Rossignac's 3D compression by topological surgery

The Topological surgery method for compressing the incidence relation is based on the following observations:

- Given the simply connected polygon that bounds a triangle strip and the selection of a starting seed edge, the internal triangulation of the strip may be encoded using **one** bit per triangle. This bit simply states whether one should advance the left or the right vertex of a progressing edge that sweeps the entire strip starting at the seed edge and moving its vertices one at a time along the boundary of the strip. (Note that if the strip was a regular alternations of left and right moves, we would need zero bits.)
- Given a set of non-overlapping triangle strips that cover the surface of a polyhedron and union of their boundaries identifies a set of "cut" edges. Each cut edge is used exactly twice in the boundary of the same or of different strips.
- A vertex spanning tree formed by selecting the minimum number of edges of the polyhedron that connect all the vertices but do not create loops cuts the boundary a single shell, genus zero, manifold polyhedron into a simply connected topological polygon that may be decomposed into the union of triangle strips. These strips and their adjacency may be encoded as a binary spanning tree of the triangles of the mesh.
- Cutting strategies that produce vertex and triangle spanning trees with few nodes that have more than one child lead to very efficient schemes for compressing the trees. Although encoding general trees and graphs is much more expensive [Jacobson89], the expected cost of our encoding is less than a bit per vertex. One basically needs to encode the length of a run of consecutive nodes that have a single child and the overall structure of the tree (which requires 2 bits for each node with more than one child.)

- The handling of non-manifold polyhedra and of meshes with higher genus is possible through simple extensions of this approach.
- The order imposed on the vertices by the vertex spanning tree may be exploited for computing good estimates for the location of the next vertex from the location of its 3 or 4 ancestors in the tree. Entropy coding the differences between the actual location and the estimate results in better compression for the geometric information than coding the absolute coordinates, because the coordinates of the difference vector are smaller on average.

The details of this approach may be found in [Taubin96].

Turan has shown that the incidence of a simple mesh can be encoded using **12** bits per vertex [Turan84]. However, Turan's study focused on the problem of triangulating a labeled graph. Taubin and Rossignac use the order of the vertices (i.e. a permutation on the vertex labels) to capture some of the incidence relation, and hence are able to reduce further improve the compression by a factor of 3.

Scaleable models

In order to support remote viewing of highly complex models, the user must be able to download a low resolution model and start using it while more detailed model description is still being transferred. Hoppe's progressive meshes [Hoppe96], discussed earlier, are suitable for such an adaptive scheme.

Furthermore, to accelerate graphics, it is important to transfer the different parts of the scene at different resolutions (models close to the viewer should receive more details early on, while distant models may never need to be displayed I full detail. Although the progressive mesh representation supports multi-resolution adaptive simplification, the constraints imposed on the order in which vertices may be inserted make it ill suited for adaptive LOD generation. Indeed, the system may often be forced to expand distant edges simply to create vertices whose descendants are needed to expand nearby edges. A more localized approach may be more effective.

Conclusion

Although the performance of state of the art hardware graphics and the internet bandwidth are growing rapidly, price constraints and the growing needs of industrial customers call for algorithmic solutions that improve the speed of visualizing and transmitting complex 3D scenes. We have presented several techniques, which automatically computes one or several simplified graphics representations of each object. These representations may be used selectively in lieu of the original model to accelerate the display process while preserving the overall perceptual information content of the scene. The described methods clusters vertices of the model and produces an approximate model where original faces are approximated with fewer faces defined in terms of selected vertices. Several simplified representations with different simplification factors may be stored in addition to the original model. Actual viewing conditions are used to establish automatically for each object which representation should be used for graphics. We have also reviewed several techniques for compressing the geometry, the incidence, and the photometry of a triangulated model. The best results yield a 20:1 compression ratio over naive representation schemes.

Bibliography

[Airey90] J. Airey, J. Rohlf, and F. Brooks, Towards image realism with interactive update rates in complex virtual building environments, ACM Proc. Symposium on Interactive 3D Graphics, 24(2):41-50, 1990.

[Algorri96] M.-E Algorri, F. Schmitt, Surface reconstruction from unstructured 3D data, Computer Graphics Forum, 15(1):4760, March 1996.

[Andujar96] C. Andujar, D. Ayala, P. Brunet, R. Joan-Arinyo, J. Sole, Automatic generation of multi-resolution boundary representations, Computer-Graphics Forum (Proceedings of Eurographics'96), 15(3):87-96, 1996.

[Beigbeder91] M. Beigbeder and G. Jahami, Managing levels of detail with textured polygons, Compugraphics'91, Sesimbra, Portugal, pp. 479-489, 16-20 September, 1991.

[Bergman86] L. Bergman, H. Fuchs, E. Grant and S. Spach, Image Rendering by Adaptive Refinement, Computer Graphics (Proc. Siggraph'86), 20(4):29-37, Aug. 1986.

[Blake87] E. Blake, A Metric for Computing Adaptive Detail in Animated Scenes using Object-Oriented Programming, Proc. Eurographics`87, 295-307, Amsterdam, August1987.

[Borrel95] P.Borrel, K.S. Cheng, P. Darmon, P. Kirchner, J. Lipscomb, J. Menon, J. Mittleman, J. Rossignac, B.O. Schneider, and B. Wolfe, The IBM 3D Interaction Accelerator (3DIX), RC 20302, IBM Research, 1995.

[Haralick77] R. Haralick and L. Shapiro, Decomposition of polygonal shapes by clustering, IEEE Comput. Soc. Conf. Pattern Recognition Image Process, pp. 183-190, 1977.

[Cignoni95] P. Cignoni, E. Puppo and R. Scopigno, Representation and Visualization of Terrain Surfaces at Variable Resolution, Scientific Visualization 95, World Scientific, 50-68, 1995. http://miles.cnuce.cnr.it/cg/multiresTerrain.html#paper25.

[Clark76] J. Clark, Hierarchical geometric models for visible surface algorithms, Communications of the ACM, 19(10):547-554, October 1976.

[Cohen96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agrawal, F. Brooks, W. Wright, Simplification Envelopes, Proc .ACM Siggraph’96, pp. 119-128, August 1996.

[Crow82] F. Crow, A more flexible image generation environment, Computer Graphics, 16(3):9-18, July 1982.

[Deering95] M. Deering,Geometry Compression, Computer Graphics, Proceedings Siggraph'95, 13-20, Augiust 1995.

[DeFloriani92] L. De Floriani, E. Puppo, A hierarchical triangle-based model for terrain description, in Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, Ed. A. Frank, Springer-Verlag, Berlin, pp. 36--251, 1992.

[DeHaemer91] M. DeHaemer and M. Zyda, Simplification of objects rendered by polygonal approximations, Computers and Graphics, 15(2):175-184, 1991.

[DeRose92] T, DeRose, H, Hoppe, J. McDonald, and W, Stuetzle, Fitting of surfaces to scattered data, In J. Warren, editor, SPIE Proc. Curves and Surfaces in Computer Vision and Graphics III, 1830:212-220, November 16-18, 1992.

[Eck95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery and W. Stuetzle, Multiresolution Analysis of Arbitrary Meshes, Proc. ACM SIGGRAPH'95, pp. 173-182, Aug. 1995.

[Erikson96] C. Erikson, Polygonal Simplification: An Overview, UNC Tech Report TR96-016, http://www.cs.unc.edu/~eriksonc/papers.html

[Evans96] F. Evans, S. Skiena, and A. Varshney, Optimizing Triangle Strips for Fast Rendering, Proceedings, IEEE Vizualization'96, pp. 319--326, 1996.

[Funkhouser93] T. Funkhouser, C. Sequin, Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments, Computer Graphics (Proc. SIGGRAPH '93), 247-254, August1993.

[Funkhouser93] T, Funkhouser, Database and Display Algorithms for Interactive Visualization of Architectural Models, PhD Thesis, CS Division, UC Berkeley, 1993.

[Garland95] M. Garland and P. Heckbert, Fast Polygonal Approximation of Terrains and Height Fields,
Research Report from CS Dept, Carnegie Mellon U, CMU-CS-95-181,
\URL{http://www.cs.cmu.edu/~garland/scape, Sept.1995.

[Greene93] N, Greene, M, Kass, and G, Miller, Hierarchical z-buffer visibility, ACM SIGGRAPH'93 Proceedings, pp:231-238, 1993.

[Gross95] M. Gross, R. Gatti and O. Staadt, Fast Multi-resolution surface meshing, Proc. IEEE Visualization’95, pp. 135-142, 1995.

[Gueziec96] A, Gueziec, Surface Simplification inside a tolerance volume, IBM Research Report RC20440, Mars 1996.

[He96] T. He, A. Varshney, and S. Wang, Controlled topology simplification, IEEE Transactions on Visualization and Computer Graphics, 1996.

[Heckbert94] P. Heckbert and M. Garland, Multiresolution modeling for fast rendering, Proc Graphics Interface'94, pp:43-50, May 1994.

[Hoppe92] H, Hoppe, T, DeRose, T, Duchamp, J. McDonald, and W, Stuetzle, Surface reconstruction from unorganized points, Computer Graphics (Proceedings SIGGRAPH'93), 26(2):71-78, July 1992.

[Hoppe93] H, Hoppe, T, DeRose, T, Duchamp, J, McDonald, and W, Stuetzle, Mesh optimization, Proceedings SIGGRAPH'93, pp:19-26, August 1993.

[Hoppe96] H, Hoppe, Progressive Meshes, Proceedings ACM SIGGRAPH'96, pp. 99-108, August 1996.

[Hoppe97] H, Hoppe, Progressive Simplicial Complexes, Proceedings ACM SIGGRAPH'97, August 1997.

[Jacobson89] G. Jacobson, Succinct Static Data Structures, PhD Thesis, Carnegie-Mellon, Tech Rep CMU-CS-89-112, January 1989.

[Kalvin91] A, Kalvin, C, Cutting, B. Haddad, and M, Noz, Constructing topologically connected surfaces for the comprehensive analysis of 3D medical structures, SPIE Image Processing, 1445:247-258, 1991.

[Kalvin96] AD, Kalvin, RH, Taylor, Superfaces: Polyhedral Approximation with Bounded Error, IEEE Computer Graphics \& Applications, 16(3):64-77, May 1996.

[Lawson72] C. Lawson, Transforming Triangulations, Discrete Math. 3:365-372, 1972.

[Lindstrom96] P. Lindstrom, D. Koller and W. Ribarsky and L. Hodges and N. Faust G. Turner, Real-Time, Continuous Level of Detail Rendering of Height Fields, SIGGRAPH '96, 109--118, Aug. 1996.

[Lounsbery94] M. Lounsbery, Multiresolution Analysis for Surfaces of Arbitrary Topological Type, PhD. Dissertation, Dept. of Computer Science and Engineering, U. of Washington, 1994.

[Low97] K-L. Low and T-S. Tan, Model Simplification using Vertex-Clustering.

[Luebke95] D, Luebke, C, George, Portals and Mirrors: Simple, fast evaluation of potentially visible sets, 1995 Symposium on Interactive 3D Graphics, ACM Press, pp. 105-106, April 1995.

[Luebke96] D. Luebke, Hierarchical structures for dynamic polygonal simplifications, TR 96-006, Dept. of Computer Science, University of North Carolina at Chapel Hill, 1996.

[Maciel95] P, Maciel, P, Shirley, Visual Navigation of Large Environments Using Textured Clusters, 1995 Symposium on Interactive 3D Graphics, ACM Press, pp. 95-102, April 1995.

[Mann97] Y. Mann and D. Cohen-Or, Selective Pixel Transmission for Navigation in Remote Environments, Proc. Eurographics'97, Budapest, Hungary, September 1997.

[Mitchell95] J.S.B, Mitchell, S, Suri, Separation and approximation of polyhedral objects, Computational Geometry: Theory and Applications, 5(2), pp. 95-114, September 1995.

[Neider93] J. Neider, T. Davis, and M. Woo, OpenGL Programming Guide, Addison-Wesley, 1993.

[Naylor95] B, Naylor, Interactive Playing with Large Synthetic Environments, 1995 Symposium on Interactive 3D Graphics, ACM Press, pp. 107-108, April 1995.

[Pennebaker93] B. Pennebaker and J. Mitchell, JPEG, Still Image Compression Standard, Van Nostrand Reinhold, 1993.

[Rockwood89] A, Rockwood, K Heaton, and T, Davis, Real-time Rendering of Trimmed Surfaces, Computer Graphics, 23(3):107-116, 1989.

[Ronfard94] R Ronfard, and J, Rossignac, Triangulating multiply-connected polygons: A simple, yet efficient algorithm, Proc. Eurographics'94, Oslo, Norway, Computer Graphics Forum, 13(3):C281-C292, 1994.

[Ronfard96] R. Ronfard and J, Rossignac, Full-range approximation of triangulated polyhedra, to appear in Proc. Eurographics'96 and in Computer Graphics Forum. IBM Research Report RC20432, 4/2/96.

[Rossignac93] J. Rossignac, and P. Borrel, Multi-resolution 3D approximations for rendering complex scenes, pp. 455-465, in Geometric Modeling in Computer Graphics, Springer Verlag, Eds. B. Falcidieno and T.L. Kunii, Genova, Italy, June 28-July 2, 1993.

[Rossignac94] J. Rossignac and M, Novak, Research Issues in Model-based Visualization of Complex Data Sets, IEEE Computer Graphics and Applications, 14(2):83-85, March 1994.

[Samet90] H. Samet, Applications of Spatial Data Structures, Reading, MA, Addison-Wesley, 1990.

[Scarlatos92] L. Scarlatos, and T, Pavlidis, Hierarchical triangulation using cartographic coherence, CVGIP: Graphical Models and Image Processing, 54(2):147-161, 1992.

[Schmitt86] F, Schmitt, B. Barsky, and W, Du, An adaptive subdivision method for surface-fitting from sampled data, Computer Graphics, 20(4):179-188, 1986.

[Schroeder92] W, Schroeder, J. Zarge, and W, Lorensen, Decimation of triangle meshes, Computer Graphics, 26(2):65-70, July 1992.

[Taubin96] G. Taubin and J, Rossignac, Geometric Compression through Topological Surgery, IBM Research Report RC-20340. January 1996. (http://www.watson.ibm.com:8080/PS/7990.ps.gz).

[Teller91] S.J. Teller and C.H, Sequin, Visibility Preprocessing for interactive walkthroughs, Computer Graphics, 25(4):61-69, July 1991.

[Teller92] S, Teller, Visibility Computations in Densely Occluded Polyhedral Environments, PhD Thesis, UCB/CSD-92-708, CS Division, UC Berkeley, October 1992.

[Turk92] G, Turk, Re-tiling polygonal surfaces, Computer Graphics, 26(2):55-64, July 1992.

[Turan84] G. Turan, Succinct representations of graphs, Discrete Applied Math, 8: 289-294, 1984.

[Varshney94] A, Varshney, Hierarchical Geometric Approximations, PhD Thesis, Dept. Computer Science, University of North Carolina at Chapel Hill, 1994.

[Xia96] J. Xia and A. Varshney, Dynamic view-dependent simplification for polygonal models, Proc. Vis'96, pp. 327-334, 1996.

[3DIX] IBM 3D Interaction Accelerator, Product Description, http://www.research.ibm.com/3dix. 1996.