Simplification and Compression of 3D Scenes

Jarek Rossignac

GVU Center, Georgia Institute of Technology, Atlanta, USA

The geometric complexity of 3D models used in scientific, industrial, medical, or military applications significantly exceeds the complexity of what can be rapidly downloaded over the internet and of what can be displayed at interactive rates on personal workstations. This situation is not likely to change, because the need for higher levels of accuracy in the 3D models and the complexity of the industrial and scientific 3D data sets increase at a much faster rate than network bandwidth, CPU speed, and graphics hardware performance. The solution requires intelligent techniques that compress the 3D scenes for fast transmission over the network or phone line and that use auxiliary datastructures and adaptive resolution techniques to avoid processing and displaying each geometric detail at each frame. This tutorial discusses 3D representation schemes for polyhedra, presents recent advances in 3D compression, reviews various graphics acceleration schemes, and teaches specific techniques for constructing and exploiting multi-resolution (LOD) models.

Table of content

1. Introduction

Computer graphics helps communicate 3D concepts, inspect medical or CAD models, and interact with education and entertainment software. Although canned video sequences of 3D animations or flythrough may be effective in some cases, interactive graphics is often needed to better understand the nature of the 3D scene, to locate areas of interest, and to inspect them in detail.

Interactive inspection is most effective when coupled with direct manipulation, where the user controls the position of the objects and of the view interactively by moving 2D or 3D input devices and by watching the effect of the motion on the screen in realtime. The term "direct manipulation" often implies that the user's gestures are interpreted in some natural metaphor, which makes it easy to predict their effect (for example, the user may change the view by dragging the projection of a selected point of the model on the screen). The realtime visual feedback is used to help with the precise adjustment of the positions and orientations of the manipulated objets or of the view. 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, distracts users and make them less productive. The effective exploitation of 3D models requires a graphic response time under 50 milliseconds. The response time combines the costs of processing the input signals, accessing the necessary geometry, rendering it, and displaying the resulting image on the screen. For highly complex scenes, the overall display cost is dominated by data access and surface rendering.

For graphic applications, surfaces of 3D objects are often approximated by polygons or triangles. The complexity (i.e., the number of triangles in the boundary of an object) varies greatly from object to object in a scene and the statistical distribution of this complexity in space and amongst objects vary from one application area to another. For example, some mechanical assemblies may involve large numbers of small components, each modeled with thousands of triangles that precisely approximate curved shapes or holes and matching pins. Architectural models on the other hand may involve very large objects with relatively simple flat surfaces requiring less triangles. Current generation models of vehicles, aircrafts, submarines, or city districts often involve over a hundred million triangles. These models are sufficiently precise for visual inspection of the design, but in general are to crude for the precise analysis and planning of assembly and manufacturing processes.

A naive approach to the rendering a scene requires to perform the following steps at each frame, although not necessarily in this order:

1. transform each vertex of each triangle of the entire scene so that it appears in the correct place in the viewer's coordinate system, as defined by the viewing transformation controlled by the user or the application,
2. transform the associated vertex normals by the same transformation,
3. compute the vertex colors for the current view from the vertex normals and from the position of the light sources,
4. clip the polygons to the interior of the viewing frustum so as to avoid overflow during integer scanconversion computations,
5. scan-convert the polygon to compute the color and surface depth at each one of the pixels covered by the projection of the clipped polygon. Each time a pixel is visited, the depth of the corresponding point on the polygon is compared to the depth stored in the z-buffer, which represents the smallest depth of a surface point encountered so far while scan-converting the initial subset of the faces in the scene. The color buffer and the z-buffer are updated if the new point is visible (i.e. lies in front of previously stored points at the pixel).

Using this naive approach to render a scene comprising 50 million triangular faces at 20 frames per second would require 3 orders of magnitude performance increase over current generation hardware. Although such an increase is anticipated over the next decade, the complexity of the scenes is likely to increase even faster in medical, construction, architecture, mechanical CAD, scientific, geo-science, and military simulation applications. Furthermore, the recent popularity of 3D graphics in entertainment and its potential impact on the next generation graphic user interfaces is challenging developers to support interactive 3D graphics capabilities with increasingly complex models on personal computers and portable devices.

Techniques that eliminate unnecessary or unessential rendering steps lead to dramatic performance improvements and hence reduce hardware costs for graphics. Many of these techniques require complex algorithmic pre-processing and sometimes 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. Several of these techniques must be combined to achieve the 3 orders of magnitude performance acceleration discussed above.

Among the various graphics acceleration approaches, **adaptive resolution** techniques recently gained popularity. Objects or features that appear small on the screen (because they are further away from the viewer under perspective projection) or that project onto the peripheral (lower-resolution) part of the human retina, away from the line of sight, may often be displayed using substitute models ("impersonators") with significantly fewer triangles but an overall similar appearance. Techniques which precompute such simplified models automatically are called simplification algorithms. They strive to minimize the number of triangles while preserving a prescribed geometric or visual accuracy. These techniques are reviewed in details in this tutorial.

The abundance and importance of complex 3D databases in major industry segments 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. Several studies of compression and decompression techniques for complex triangulated models have recently been published. Storage reduction rates achieved by current generation compression schemes reach between one and two orders of magnitude over standard formats. We anticipate considerable progress on this front in the near future.

The combination of adaptive resolution and compression techniques should eventually produce a new generation of 3D representation schemes suitable for scaleable access over the internet and for interactive rendering on a large spectrum of graphics hardware.

These notes are organized as follows.

1. We first present a simple datastructure for a triangulated model. We will use it to motivate and illustrate many of the concepts developed in these notes.
2. We review simple topological concepts which are important to characterize what kinds of shapes can be represented by such data structures. This formal characterization is essential to understand the limitations of simplification and compression algorithms. Readers familiar with these concepts may want to skip Section 3.
3. We list the various factors that limit the accuracy of polyhedral models and discuss error measures. These notions will guide us in the evaluation of bit-efficient representation schemes and of simplification algorithms.
4. We analyze the storage requirements for triangulated polyhedra and study several lossy and loss-less compression approaches.
5. We overview various graphics acceleration techniques.
6. We introduce a categorization of simplification algorithms and present several of them in detail.

2. A simple datastructure for triangulated meshes

Consider a simple data structure for triangular meshes. A vertex table, **V**, contains the three coordinates of each one of the **v** vertices of the model. Each coordinate (x_1, y_1, z_1, x_2, y_2, z_2...) is stored as a 4-byte floating-point number. A triangle table, **T**, contains the definition of **t** triangles, each represented by the 3 references to its supporting vertices. Each vertex reference (v_1, v_2, v_3, v_1, v_3, v_4,...) is stored as a 4-byte integer index into the vertex table.

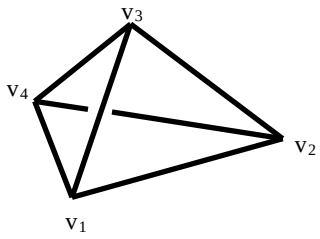
The example below shows how this data structure may be used to represent a tetrahedron.

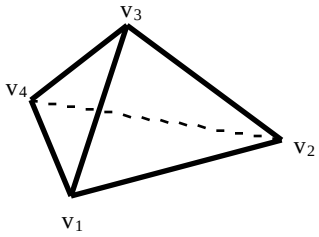
V

x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_3	z_3
x_4	y_4	z_4

T

v_1	v_2	v_3
v_1	v_3	v_4
v_1	v_4	v_2
v_2	v_4	v_3





Triangulated polyhedra may be conveniently represented using two arrays. The first one (top left) describes all the vertices by storing their x, y, and z coordinates as floating point numbers. The second one (top right) decribes all the triangles by storing for each triangle the indices of its three supporting vertices in the first array. These indices may be stored as integer numbers if the number of vertices in the polyhedron is less than 2^{32}. The interpretation of this datastructure (the semantics of the representation) defines the corresponding geometry. For example, the datastructure may represent a set of edges (bottom left) or the union of triangles (bottom right), or under certain conditions the solid bounded by such triangles.

Note that to define a geometric shape, the data structure must be associated with a semantic interpretation, which formally defines the corresponding pointset and guides the development of correct algorithms that perform computations on the corresponding pointset.

For example,. the above data structure may represent the union of the edges of all triangles (as depicted symbolically by the wireframe tetrahedron on the left) or may represent the union of the triangular faces or even the solid bounded by these faces (as depicted on the right by the engineering drawing of a solid with the hidden line dashed).

This tutorial should help the reader answer the following questions. What can and what cannot be represented using this and similar data structures? How many bits are necessary for representing an object made of **v** vertices and **t** triangles using the different datastructures and compression schemes? Is a particular datastructure suited for fast rendering?

3. Topological characterization of polyhedra

This section attempt to provide the reader with a few concepts necessary to discuss the validity and the domain of geometric representations and of the associated data structures. With these tools, the reader should be able to formulate precisely what kind of pointsets may be represented in a given modeling system and hence understand the limitations of that system.

The topological concepts pertinent to 3D modeling and graphics are introduced here at a very intuitive level. For more precise definitions, the reader should consult standard textbooks on geometry and topology [Alexandroff61].

Practitioners of solid modeling often distinguish between the topology and the geometry of a model. While a geometric representation captures the precise shape of each face or curve of a model, a topological representation focuses on properties that are invariant under continuous deformations, i.e. independent of the precise shape of the geometric components. An incidence graph is often used to capture the topological relations between the primitive entities (faces, edges, and vertices) used to model the object's boundary. The incidence graph may also contain ordering information, which accelerates applications that must traverse adjacent boundary elements.

3.1 Topological concepts and definitions

A **topological space** is a set **W** with a choice of a class of subsets of **W** (its open sets), each of which is called a neighborhood of its points, such that every point of **W** is in some neighborhood and that the intersection of any two neighborhoods of a point contains a neighborhood of that point. The three-dimensional Euclidean space, $\mathbf{E^3}$, in which the models discussed here are constructed, is so topologized.

For simplification, all the geometric entities considered here are well behaved. For example, surfaces or curves do not fill space.

3.1.1 Topological closure, interior, and boundary

The **interior** of a set S is the set of points having a neighborhood in S. (Intuitively the interior of a 3D (three-dimensional) set is the whole set except for points on its surface, or more precisely on its boundary, defined later. A set is **open** if, and only if, it is equal to its interior, or equivalently, if it contains a neighborhood for each one of its points.

Loosely speaking, a **sub-manifold** is a subset of $\mathbf{E^3}$, such as the whole space, a smooth surface, or a smooth portion of a curve. The **supporting manifold** (also called **carrier**) for a 3D solid is the whole space, $\mathbf{E^3}$. The supporting manifold for a face is a 2D (two-dimensional) surface that contains that face, although the surface needs not be flat and can live in $\mathbf{E^3}$. The supporting manifold for a curve segment (or edge) is a 1D (one-dimensional) curve that contains the edge. We will assume here that the supporting manifold for each face or curve is unique and known. (This is the case for most curves and surfaces used in modeling and graphics.)

A set O is **relatively open** with respect to its supporting manifold, M, if around each point of O there exists an open ball in M with a strictly positive radius that is included in O. We will assume that (unless specified otherwise) the term "open", when used for an edge or face, means "relatively open with respect to the supporting manifold". If M is the entire space, $\mathbf{E^3}$, the open ball is a 3D sphere without its boundary. If M is a surface, the open ball looks like a disk without its boundary. As far as geometric modeling is concerned, a 3D set is open if it includes none of its bounding faces, edges, or vertices. A face F is open if F does not include its bounding edges and vertices. Similarly, an edge is open, if it does not include its bounding vertices (i.e. its end points). Note that an edge that forms a closed loop is open. Note that an edge without its endpoints is open relatively to its supporting curve.

The **relative interior,** iS of a set S (with respect to a supporting manifold M) is the largest relatively open subset of S. When speaking about edges and faces, we will assume that, unless specified otherwise, the term "interior" refers to the relative interior with respect to the supporting manifold. Practically, the interior of a solid (3D volume) is the solid without its bounding faces, edges, and vertices. The interior of a face is the face without its bounding edges and vertices. The interior of an edge is the edge without its bounding vertices. Note that the interior of a face or curve with respect to $\mathbf{E^3}$, rather than to its supporting manifold, is the empty set (no 3D open ball fits in the curve or surface). For example, the (relative) interior of a triangle is the triangular face without its bounding edges or vertices, while the interior of the same triangle relative to $\mathbf{E^3}$ is empty.

The **relative complement**, cS, of a set S in a supporting manifold M is the set of points of M that are not in S.

The **relative exterior,** eS, of set S is the relative interior of the relative complement of S.

The **relative boundary,** bS, of a set S is the set of points P such that any open ball around P in the supporting manifold of S contains points in S and in the complement of S. Basically, the boundary is adjacent to S and to its complement in the supporting manifold. The boundary of a triangle is the union of its edges and vertices. Note that the boundary of the triangle with respect to $\mathbf{E^3}$ is the union of the relative interior of the triangle with its relative boundary, because any 3D open ball centered around any point of the triangle intersects the triangle and its complement in

The **closure,** kS, of a set S is the union of S with its relative boundary. For example, the closure of an open face is the face plus its bounding edges and vertices. If S is an open disk without its center point, kS is the disk with its bounding edge and with the center point included. If a set S has a "crack", i.e., a missing internal face, the closure operator will "fill" that crack and hence remove it from the boundary of S, making it part of iS. The closure of a set is not relative.

3.1.2 Dimensional homogeneity

A subset of $\mathbf{E^3}$ is **full-dimensional** if it contains a three-dimensional part, i.e. a part that contains a 3D ball of positive radius. A subset of a surface is full-dimensional, relatively to the surface, if it contains a disk of positive radius. Note that non-empty open subsets are always full-dimensional with respect to their supporting manifolds. A set S is **homogeneously k-dimensional** if and only if all of its points lie in the closure of its interior with respect to a k-dimensional carrier.

For example, a set is homogeneously 3D if it does not contain any dangling faces, edges, or vertices that do not bound three dimensional parts. A disk with an edge sticking out is not homogeneously 2D.

3.1.3 Regularization and Boolean operations

Regularization is an operation that takes a pointset S and returns a regularized pointset rS that is closed and homogeneously k-dimensional. In 3D, regularization returns the closure of the interior of the solid (rS=kiS). A set S is **regular** if S=rS. Regularization not only removes dangling faces, edges, and isolated vertices, it also fills cracks and puts a tight skin around the object.

Many modeling systems restrict their representation domains to algebraic **r-set,** i.e., to bounded (not infinite), regular sets that have a semi-algebraic expression (i.e. that can be expressed as the unions of sets, each being defined as the locus of points whose coordinates satisfy a system of polynomial inequalities). For example, the union and the difference of two spheres are semi-algebraic sets.

Boolean set-theoretic operators may be used to combine arbitrary sets in $\mathbf{E^3}$. The most popular operators are the union ( ), the intersection (∩), and the difference (−). They are defined as follows: A B={p: p A OR p B}, A∩B={p: p A AND p B}, and A−B={p: p A AND p B}. Unfortunately, the set theoretic intersection and difference operators, when applied to r-sets do not always return regular sets. They may produce dangling faces or missing ones. A **regularized Boolean** version of these operators combines the set theoretic operators with a post-regularization operations. Specifically: A−*B = r(A−B) and A∩*B = r(A∩B).

The **boundary** of the result of a set-theoretic Boolean or regularized Boolean operation on two sets A and B is included in the union of the boundaries of A and B. The problem of computing the boundary of the result may be decomposed into a first phase of splitting the boundaries of A and B at all places where they intersect or where the topology of that intersection changes and a second phase where one selects the appropriate pieces of these split boundaries that correspond to the particular operation.

Note that, when applied to regular pointsets, the set-theoretic union operation always returns regular sets. This does not imply that the boundary of the union is simpler to compute than the boundary of the intersection or difference. Consider two cubes glued to each other through a common face. The set-theoretic intersection returns the common face which may be discarded by a regularization operation, since its interior is empty. When computing the boundary of the intersection, one can compute the intersection of the boundary of each object with the other object and then apply the regularization process to remove portions of the resulting boundary that are not bounding the full-dimensional intersection of the solids. Now consider the set-theoretic union of these two cubes. The interior of the common face is part of the interior of the result and thus is not on the boundary. It must be removed, even if we do not apply regularization, in order to obtain a precise representation of the boundary of the result. For graphics however, correct shaded images of the union of two or more solids may be produced by simply displaying the solids using a hidden surface elimination process [Rossignac86]. No boundary evaluation is necessary.

3.1.4 Connectedness, holes, and handles

A pointset S is **connected** if, and only if, for any two points in S there exists a curve segment in S that connects the points. Note that a solid, i.e. an r-set, needs not be connected.

Two sets are **disjoint** if their intersection is empty. The union of two disjoint sets may form a connected set. For example, consider the union of an open ball with its boundary.

A pointset is **interior-connected** if its interior is connected. For example, the union of two cubes touching at a vertex is not interior connected.

Two sets are **quasi-disjoint** if their interiors are disjoint, but their closures are not. For example, two cubes glued together at a common face are quasi-disjoint.

A **hole** in a bounded (i.e. non infinite) set S is a bounded connected component of the complement of S. In three dimensions, the term **hole** must be distinguished from the term **handle** (or a through-hole), which denotes a tunnel or way through the set (such as the handle of a tea pot). The number of holes through a set is important for assessing whether two sets are homeomorphic (i.e. may be deformed into each other through a smooth transformation).

A set is **simply-connected** if it does not have any handles**,** that is, if any closed curve inside the set may be contracted into a single point by performing a smooth deformation that maintains the curve inside the pointset. For example, a sphere is simply-connected, but a torus is not. Similarly, a face is simply-connected if it does not have a hole. Note that a volume with an interior hole, such as a ball with a hole at its center in the form of a smaller ball, is simply-connected, on the other hand, an infinite cylinder with a coaxial cylindrical hole is not simply connected.

3.1.5 Non-Manifold conditions

A set S is said to be a closed **manifold**, if around each point of the S one can construct an open ball of positive radius whose intersection with S is homeomorphic to (i.e smoothly deformable into) an open ball or to a half-ball, i.e. the intersection of an open ball with a linear half space (such as the set of points whose z-coordinate is less or equal to zero). For example, the union S of two closed cubes touching at a vertex is not a manifold set, because the any open ball centered at the common vertex will intersect S into two cones that are not homeomorphic to a ball or half-ball. Note that the interior of the above set S is a manifold—although not connected—solid.

The boundary bS of a manifold r-set forms a two-manifold surface, where around each point of bS one can construct an open ball of positive radius whose intersection with bS is homeomorphic to an open disk. Some authors say that a two-manifold, or manifold boundary, does not **self-intersect**.

3.1.6 Betti number and genus

The **zero Betti number**, b_0, denotes the number of connected components in a set.

The **first Betti number**, b_1, (also called 1-connectivity) specifies the number of handles in a 3D set. It may be defined as the maximum number of cuts through the set that can be made without disconnecting it (i.e. without producing two separate pieces). A cut through a set may be viewed as the surface swept by drawing a closed curve on the boundary of the solid and contracting it to a single point while maintaining it inside the set. For example the 1-connectivity is 0 for a ball, and 1 for a solid torus, and 2 for the surface of a torus (the zero Betti number for a closed surface is twice the zero Betti number for the solid bounded by the surface).

The **second Betti number**, b_2, denotes the number of holes.

The **genus** of a surface is the maximum number of closed curves (contained in it) that may be subtracted from it without disconnecting it. The genus, also denotes the number of handles, **H**. One closed surface may be mapped into another by a continuous bijection if they have the same genus. The genus of a closed surface is half its first Betti number and is also equal to the Betti number of the 3D set bounded by the surface.

3.1.7 Euler characteristic

The **Euler characteristic** of a 3D manifold CW-complex (a set bounded by a two-manifold surface) made of R three-dimensional cells (connected components of the interior of the complex), of F faces, of E edges, and of V vertices is a topological invariant independent of the subdivision and is equal to V-E+F-R.

Similarly, for a two-dimensional manifold closed surface without boundary made of F faces, E edges, and V vertices, the Euler characteristic (or Euler number) is equal to V - E + F.

The **Euler equation** states that the Euler characteristic is equal to the alternate sum of the first three Betti numbers: b_0-b_1+b_2.

Since b_0+b_2, the number of connected components plus the number of holes is the number of shells, S, in the surface bounding the 3D set, we have for a 3D set: V-E+F-R=S-H, where H is the number of handles through the entire 3-D set.

For a surface, b_0, the number of shells, is equal to the number of holes, b_2, and the maximum number of non-separating cuts in the surface is twice the number of handles (b_1=2H). Therefore, the **Euler equation** for a surface is: V-E+F= 2(S-H). Consequently, the Euler characteristic of a surface in 3-D is twice the Euler characteristic of the solid bounded by the surface.

Since V, E, F, R are readily available in the representation of a complex, and since S may be easily computed by separating connected components, the above formulae yield a practical means for computing H for each connected component. The formulae are restricted to manifold sets, although they can be extended to non-manifold CW complexes by incorporating the counts of various on-manifold situations, such as the additional cones of faces incident upon a vertex.

When the polyhedron is a triangulated manifold (i.e. all faces are split into triangles), there are three edges per face and each edge is shared by two adjacent triangles, and hence E=3F/2, and therefore, V=F/2+2(S-H). In other words, there is twice more triangles than vertices and each additional handle and shell introduces 2 additional vertices.

3.2 **Topological domains**

This subsection reviews several modeling schemes and compares their coverage, i.e., what type of geometries they can represent.

3.2.1 Simplicial and other complexes

A **k-simplex** is the convex hull of k+1 linearly independent points in E^3. An **m-face** of a k-simplex is the convex hull spanned by m of the k points of the k-simplex and is an m-simplex. All k-simplices are closed and homeomorphic to a closed k-ball. The boundary of a k-simplex is homeomorphic to a k-sphere.

A **simplicial complex** is a finite union of simplices glued together such that for any pair (A,B) of these simplices: either A and B are disjoint, or A and B share a common m-face, or A is an m-face of B, or B is an m-face of A (for some m).

The **polytope** of a simplicial complex is the union of the sets of all of its simplices.

A **CW complex** is a finite union of mutually disjoint relatively open cells, each being homeomorphic to an open ball (of some dimension) and having for boundary the union of the sets of other cells in the complex. The intersection of the closure of two cells is either empty or is the union of other cells in the complex. In 3D CW complexes generalize the notion of simplicial complexes because their cells are not restricted to simplices (i.e. points, line segments, triangles, and tetrahedra), but may include relatively open sets of arbitrary shape and of an arbitrary finite number of bounding simplices (k-faces), provided that the 2-D cells have no holes and that the 3-D cells have no holes or handles.

A serious limitation of simplicial and CW complexes is their lack of closure under Boolean operations. For example, the difference between two CW complexes, a triangle and a point inside the triangle, cannot be represented as a CW complex.

The popular geometric primitives, such as cylinders or cones, cannot be represented directly as simplicial complexes, nor even as CW complexes, if their faces, edges, and vertices are not simply connected. Artificial bridge-edges and cut-faces are sometimes introduced, but lead to inconsistencies in the semantic interpretation of the associated datastructures.

Geometric Complexes, introduced by Rossignac and O'Connor, [Rossignac89] generalize the concept of CW complexes allowing cells to be open sets of arbitrary genus (rather than being restricted to be homeomorphic to open balls). For example, a torus may be represented as a geometric complex by only two cells: its 3-D interior and its 2-D boundary.

Simplicial complexes, CW complexes, and Geometric Complexes are closed, i.e.: they contain the boundaries of all of their cells.

A **Selective Geometric Complex** (**SGC**) [Rossignac89] further extends the notion of a geometric complex by associating with each cell an attribute stating whether the cell is active (i.e. contributes to the final set) or not.

For example, an open sphere without its center point may be modeled by an SGC with three cells: the 3-D interior without the point, the 2-D bounding sphere, and the central vertex. Only the interior is active.

Structured Topological Complexes (**STC**s) [Rossignac97] further extend the SGC concept by replacing the SGC attribute by a color vector (one color per view). Each **view** divides the entire space into **features**, where a feature is the union of all cells that have the same color in the view. STCs provide a natural and general representation for composite objects with regions of different materials. It also provides multiple coherent views on how the same object (and its complement) may be partitioned into solid or surface features relevant to the various application areas.

Two distinct k-cells of a (simplicial, CW, or geometric) complex are **adjacent** if they share one or more bounding cells. A (k+1)-cell c is **incident** on a k-cell b if b is a bounding cell of c.

3.2.2 Space decomposition

Any set of geometric primitives may be used to impose a decomposition of the underlying three-dimensional Euclidean space into cells of a geometric complex from which one can select a subset of interest (the active cells of an SGC or a particular feature for a view of an STC). We describe here informally several ways of defining such a decompositions.

A single primitive P decomposes space into four parts: the interior, the boundary of P contained in P, the boundary of P not contained in P, and the exterior of P. The interior and exterior are open (and hence full-dimensional) sets and may be further decomposed into their connected components (although it may be difficult to distinguish these components).

The boundary in P and the boundary in cP may be decomposed into singularity free, dimensionally homogeneous subsets by identifying and extracting their non-manifold and singular components (cusps and self-crossings where some geometric continuity or topological manifold properties of the supporting geometry vanish) and by considering the connected components of the rest of these sets. The sets of singular points may be further (recursively) decomposed in this manner into dimensionally homogeneous sets. Finally, the maximally connected components of this decomposition may be identified.

For example, a truncated solid cone primitive may decompose space into: the complement of the cone, the three-dimensional interior of the cone, the circular edge at the base, the disk-like base-face (without its bounding circular edge), the apex (tip of the cone), and the conical face (without the bounding edge nor the apex).

The atomic entities defined by this decomposition process correspond to the cells of a geometric complex as discussed above. They are connected relatively open subsets of some n-dimensional manifold (the 3-D space or a smooth portion of a surface or of a curve). Any two different cells of such a decomposition are mutually disjoint. The boundary of the set of a cell C lying in a manifold M is the union of other cells in the decomposition, which are either entirely in the manifold M, or entirely out if it. For example the boundary of the conical face of a cone primitive is composed of a circular curve (part of the manifold surface supporting the face) and of the apex (singular point, not in the manifold supporting the face).

When several primitives overlap, the space decompositions induced by each primitive must be combined into a single (finer) decomposition. An algorithm for performing such a merging operation is called subdivision (or refinement) and is outlined in [Rossignac89]. It basically requires that all intersections between cells of one decomposition and cells of the other be computed, further decomposed into dimensionally homogeneous singularity-free connected components, and inserted in both decompositions to make them compatible.

The space partition induced by a set of planes is a simple example of decomposition. It cells are:

- points where three or more planes meet,
- relatively open (and possibly unbounded) line segments, defined as the connected components of the difference between a line of intersection of two or more planes and the union of all the other planes that each cross the line at an isolated point,
- relatively open convex polygonal faces defined as the connected components of the difference between each plane and all other non-parallel planes,
- and the open convex polyhedra (maximally connected components of the complement of the union of all the planes).

3.3 **What is a polygon?**

The naive question: “What is a valid polygon?” should lead to a mathematical definition of validity for polygonal representations accepted, processed, or produced by a particular application or algorithm. Numerous definitions

have been published. The reader may be puzzled by the diversity of such definitions and struggle when asked to establish if two definitions are equivalent, if a particular definition is complete, or even if a specific case satisfies a particular definition. It is helpful to decompose this questions as follows:

1. What is a polygon?
2. What representation scheme (abstract data structure) are used to store models of polygons?
3. How do we semantically interpret the representation (for example when testing whether a point lies inside the represented polygon?
4. Under which condition is a model expressed using that representation scheme valid (i.e., corresponds to a polygon according to the chosen definition)?

3.3.1 A definition of polygons as open sets

The author's definition of a polygon, provided below, is restrictive, but leads naturally to a simple and canonical (i.e. unique) representation scheme.

A set P is a **polygon** if and only if P satisfies the following three conditions:

1. P is a connected and bounded subset of a plane,
2. P is equal to the interior of its closure,
3. the boundary of P is contained in a finite union of lines.

The boundary of a polygon may be decomposed into a finite union of mutually disjoint **cells**. Cells are either **crossings** (i.e., non-manifold points), **vertices** (i.e. non-smooth manifold points that are not crossings), or **edges** (i.e., the connected components of the difference between the boundary of the set and its vertices and crossings). Edges are relatively open connected line segments free of crossings.

We have chosen to define polygons as open (rather than as closed sets that include their boundary), because this definition leads to a simpler definition of the loops of a polygon and to a simpler canonical representation. However, one can opt for a complementary definition where polygons are equal to the closure of their interior. The major difference between two definitions lies in the fact that the open definition (favored by the author) treats polygons that share one or several vertices as separate polygons, while the closed definition treats them as a single non-manifold polygon, since it is connected.

3.3.2 Properties of polygons

Our definition leads to the following properties for polygons:

1. A polygon is a 2D region, not a polygonal line. This definition is compatible with the use of the term polygon in graphics.
2. A polygon is (relatively) open, and thus does not contain its boundary. Note that the boundary of a polygon exists and is well defined. In fact the boundary may often be used to represent the polygon.
3. A polygon may have holes, but no islands (which would be separate polygons).
4. A polygon has no dangling edges, no isolated vertices, no interior cracks, and no missing points.
5. The boundary of a polygon needs not be a one-manifold (which is important since even manifold polyhedra may exhibit faces with non-manifold boundaries).
6. The vertices, crossings, and edges of a polygon are pairwise disjoint.
7. The vertices, crossings, and edges of any given polygon are always unambiguously defined.
8. Each edge of a polygon separates the relative interior of the polygon from its relative exterior.
9. The orientation of each edge (following the convention that the relative interior is on the left) is thus well defined.
10. Each edge is incident on exactly two points (vertices or crossings) and its orientation defines which is the start point and which is the end point.
11. Each vertex (not a crossing) is the start point of exactly one edge and the end point of exactly one other edge.
12. Each crossing has 2k (k positive integer) edges incident on it. It is the start point of exactly k of these.

13. The successor of an edge E having vertex V as its end point is defined as the only edge having V as its start point.
14. The successor of an edge E having crossing C as its end point is defined as the first edge that has C as its start point as we circle C clockwise in its immediate vicinity starting from E.
15. The successor of an edge is uniquely defined.
16. A loops of a polygon are the maximally connected components of the boundary of the polygon.
17. The loops of a polygon are uniquely defined and pairwise disjoint.
18. Each edge, each vertex, and each crossing belongs to exactly one loop.
19. An edge belongs to the same loop as its successor.
20. The successor operator (which returns the successor of the argument edge) induces a unique cyclic ordering for all the edges in a loop.
21. Edges may be uniquely defined by the references to their start and end points.
22. Since the end point of an edge is equal to the start point of its successor, one reference per edge in a loop suffices for defining the edges. Thus, a loop is completely represented by an ordered circular list of references to the start points of its edges
23. Each loop has at least 3 references to different points. Otherwise, it would be a degenerate loop with the geometry of an edge or point.
24. The starting points of two consecutive edges in a loop are always different (i.e. consecutive point-references in a loop must be different).
25. All the edges of a loop are different (i.e., have different pairs of vertices).

3.3.3 Canonical representation of polygons

These properties lead to a simple and canonical representation for pointsets that are polygons.

Any polygon may be represented by the set of its points (vertices and crossings) and the set of its loops. Each point may be represented explicitly by its coordinates or implicitly, for example as the intersection of three planes. Each loop is represented by a circularly ordered list of references to its points.

Note that given any polygon (i.e. a pointset that meets the proposed definition), its representation in the above scheme (data structure) is unique. (We do not take into account the possible permutations and various representations for sets and list, which can be addresses by imposing the appropriate lexicographic orderings.) For processing convenience, one could also require that the outer loop (which is also always well defined in the plane) be explicitly identified in the data structure.

Imposing additional constraints on the representation further restricts the set of representable polygons. For example, if each point is used only once in a loop, the polygon has a **manifold** boundary (no crossings). If the polygon has a single loop, it is **simply connected**.

3.3.4 Validity of a model

A data set organized as described above (set of loops, each loop being defined as a sequence of references to points) does not necessarily correspond to the loops, edges, vertices, and crossings of a the boundary of a polygon.

Validity violations may be of different nature: geometric, ordering, or topological. Geometric violations correspond to the wrong choice of point coordinates (for example, a point may coincide with another point or with an edge, or two edges may intersect). Ordering violations may simply correspond to the wrong orientation of the edges in a loop or to the wrong branch taken at a crossing (inconsistent with the definition of a successor). Topological violations may correspond to empty edges (two consecutive references to the same point), to degenerate loops (less than three point references), or to loops with non-manifold parts (for example, the multiple use of an edge).

3.3.5 Other semantic interpretations of a polygonal representation

It is possible to define semantics that guarantee the validity of arbitrary representations stored using this datastructure and that define a unique polygon for each dataset represented by a set of loops. For example, let L be the set defined as the closure of the XOR combination of all the relative interiors of the edges of all the loops. (A

point belongs to an XOR combination of a number of pointsets if and only if it belongs to an odd number of these sets.) The difference between the supporting plane and L is composed of connected open sets, one of which is unbounded. All the other sets are polygons. There exists a unique subset Q of these polygons, such that L is the boundary of the interior of the closure of the union of all the polygons in Q. An acceptable semantics for the loop-based representation would be to consider that the loops define a unique polygonal region which is the union of all the polygons in the set Q.

This formal definition corresponds to the intuitive notion of the **parity rule,** which states that a point **p** lies inside a polygonal region defined by a set of loops if a ray constructed from **p** to infinity and not intersecting any vertex of the loops intersects an odd number of the edges of their loops. This simple rule does not cover points that lie on the loops, but can be extended to cover such points.

Note that polygonal regions defined by the parity rule need not be connected. Furthermore, their loop-based representation (with this extended semantics) is not unique. However, the interior of the polygonal region may be decomposed into polygons (as defined by the author), each one having a canonical representation in terms of its natural loops, as defined by the author.

3.4 What is a polyhedron?

The definitions discussed for polygons may be naturally extended to polyhedra (which are r-sets, but need not be manifold).

The author defines a polyhedron P as a set with the following properties:

1. P is a bounded, connected subset of the Euclidean space, E^3,
2. P is equal to the interior of its closure,
3. the boundary of P is contained in a finite union of planes.

Faces of polyhedra are identified as the connected components of the boundary from which we have removed all singularities (manifold and non-manifold edges and vertices).

The concept of loops introduced for polygons is replaced in three dimensions by the concept of shells. A **shell** is a connected subset of the boundary of a polyhedron.

Each face of a polyhedron is separating the interior of the polyhedron from its exterior. Therefore, each face may be oriented so that its normal points towards the exterior of the polyhedron.

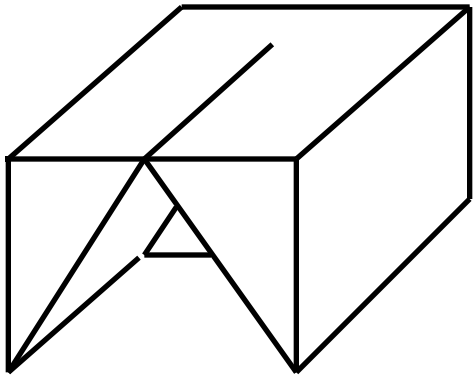
Given a face F and one of its bounding edges E in a polyhedron P, there is a uniquely defined successor to F around E for P. This successor is the first face encountered by walking on the interior side of F and crossing E.

The successor operator is symmetric: if F' is the successor of F around E, the F is the successor of F' around E.

A polyhedron may therefore be represented by a set of its shells, each shell being defined as the closure of the union of its faces. To facilitate traversal of the boundary from one face to an adjacent face (which is often used to exploit coherence in algorithms the process polyhedral representations), the successor of each face around each one of its edges may be stored explicitly.

Notice however that a shell may be composed of several **sub-shells** such that one can walk from any face to any other face in a sub-shell by following a sequence of successor operators, while such a sequence does not exist for faces of different sub-shells. (Two sub-shells are only touching by non-manifold vertices.)

Also note that the faces of a polyhedron are not always polygons. Consider for example the top face of the non-manifold polyhedron in the figure below. It is an open subset of the plane, but is not equal to the interior of its closure. In fact it has a crack (boundary edge that is adjacent to the face on both sides).



A non-manifold polyhedron may have faces that are not polygons, because their boundary contains crack-edges that are adjacent to the face on both sides. Such models may be captured in datastructures designed for manifold models bu suppressing the edge or by representing it twice in a loop. These violations of the validity rules of a polyhedral representation may lead to considerable difficulties in the development of algorithms that process such representations.

The crack-edge could be removed from the boundary of the face by a regularization process, which destroys the property that faces and edges are disjoint. Alternatively, it could be modeled using two coincident edges (one with each orientation), which contradicts the assumption that edges are pairwise disjoint. Both solutions lead to considerable complications for many algorithms that for example triangulate the faces or compute properties of the polyhedron. A better approach is to define faces of non-manifold polyhedra as a superset of polygons.

A set P is a **polygonal face** if and only if:

1. P is an open bounded, connected subset of a plane,
2. the boundary of P is contained in a finite union of lines.

According to this definition, polygonal faces exhibit the same topological domain as the 2-cells of geometric complexes.

3.4.1 Representing restricted classes of polyhedra

Although a simple enumeration of a solid's faces suffices to unambiguously define the solid, most boundary representation schemes store additional information to accelerate the traversal and processing of the boundary and combine the description of adjacent faces in order to eliminate the redundant descriptions of their common vertices. These data structures often capture the incidence relations between a face and its bounding edges and vertices, and between an edge and its bounding vertices. Many data structures have been studied to achieve desired compromises between:

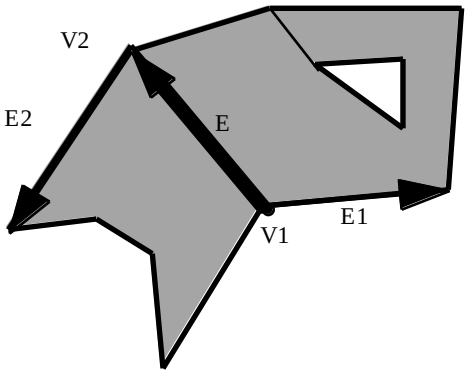
1. the domain (or coverage) of the modeler,
2. the simplicity, regularity, and compactness of the data structure,
3. the efficiency of the algorithms that process the representation,
4. the complexity of the software for building and processing such representations.

Polyhedral modelers whose domain is restricted to **manifolds** may use datastructures for their solids (i.e. r-sets) where each edge is bounding exactly two faces and where each vertex is adjacent to a single cone of faces.

Another popular restriction for polyhedral modellers is to require that **all faces be simply connected** (which does not imply that the solids are simply-connected). With this restriction, each face may be represented by a simple bounding loop of edges, which may in turn be implicitly defined by the cycle of vertices in the loop. The choice in the order of the vertices may be used to indicate the direction of the outward pointing normal of the face with respect to the solid it bounds. Such simplified models for faces are popular in graphic.

As an example, we describe with the help of the following figure a simple data structure for manifold polyhedral objects. It is a simplified version of the winged-edge representation introduced in [Baumgart72]. The vertices of

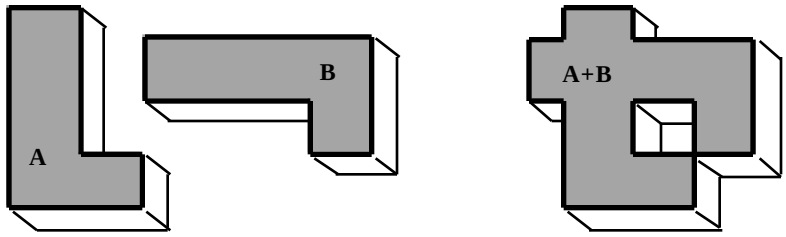
the object are stored in a table with the associated coordinates. Edges are represented by references to vertices and to adjacent "successor" edges. Note that this data structure defines the geometry of the edges and of the loops which bound the faces, but does not explicitly capture the relations between the different loops of the same face. For example, the face on the right of the figure below has an outer loop that encloses a smaller, triangular, inner loop of edges. Some boundary representations have separate nodes for faces, and store explicitly the face and loop relationships. Alternatively, faces with holes may be converted into simply-connected faces by introducing artificial "bridge" edges that each merge two loops of a face.



Edge E points to its starting and ending vertices V1 and V2. The order of vertex-references defines an orientation for the edge (indiated by the direction of the arrow). The node corresponding to E also contains references to edges E1 and E2. E1 is the next edge around V1 for the face on the right of E, and E2 is the next edge around V2 for the face on the left of E. The terms "next edge", "on the left", and "on the right" are unambiguously defined by considering the outward-pointing normals to the faces and the orientation of E. The bridge edge which links the outer loop of the right face with its inner loop is indicated by the double line segment through the face.

3.4.2 Non-manifold boundary representations

Some modelers cater only to r-sets that are manifolds. A manifold model has a boundary whose edges are shared by precisely two faces, and whose vertices are adjacent to a set of faces that forms a single cone with apex at the vertex. Unfortunately, the domain of manifold r-sets is not closed under some of the fundamental modeling operations. For example, the union of the two L-shaped manifold solids (see below) is a non-manifold solid. Hence, users of such restricted systems must carefully avoid creating non-manifold shapes.



The two L-shaped solids A and B (left), are positioned so that an edge of A coincides with an edge of B. Their union, A+B, is a non-manifold solid (right).

The radial-edge data-structure [Weiler87] was introduced to provide an extension of the earlier manifold boundary representations to support non-manifold solids. It was however extended to represent non-manifold pointsets that combine faces and edges and that do not necessarily form valid boundaries of any solid. In the solid modeling

community, the term “non-manifold boundaries” traditional referred to valid boundaries of solids. However, the terms “non-manifold geometric boundary” and “non-manifold topology” often refer to a more general scheme that includes dimensionally inhomogeneous pointsets.

The pointsets defined in this manner are restricted to be quasi-disjoint enumerations of closed bounded pointsets of dimension one and two.

When these pointsets are the union of one or several shells, i.e. face cycles that enclose a connected region of space, one can use them to represent 3D solids by delimitation. However, a set of faces that contains, in addition of the shell, faces that lie inside the enclosed region cannot be considered to be the boundary of a 3D solid, because the solid is closed and the faces inside the enclosed region are in the interior of the solid and thus are not part of its boundary. When representing such objects (i.e. closed sets with internal faces) by schemes such as those proposed in [Weiler87], one in fact uses the faces of the outside shell to define a 3D region. The interior faces are not part of the boundary, but are modeled together with the solid through superposition.

The alternative is to consider open 3D cells, such as the cells of an SGC, which is a non-regularized inhomogeneous pointset represented through enumeration as the union of mutually disjoint connected open cells [Rossignac89]. Note that because SGC cells are open, they are dimensionally-homogeneous. Furthermore, cells are required to be connected and are singularity-free sub-manifold of algebraic varieties. SGC cells are represented through delimitation. Therefore, cells that constitute the boundary of a cell are also part of the SGC. For example, an SGC solid-cell in 3D is represented by an SGC decomposition of its boundary; an SGC face-cell is represented by a supporting surface and its bounding edge-cells and vertex-cells; and so on. Treated as a disjoint enumeration of all its cells, the SGC would always represent a closed pointset. However, in addition to their capabilities for modeling inhomogeneous pointsets with internal structures, SGCs can also be used to represent non-closed pointsets with cracks. This generalization is achieved through the SGC mechanism for selective subdivision: cells in the enumeration are marked as active or inactive; the pointset represented by an SGC is the union of the pointsets represented by its active cells. Using inactive cells, SGCs may also define a subdivision of the complement of the pointset they represent. A simplification mechanism which removes unnecessary subdivisions or unnecessary inactive cells while maintaining the validity of the SGC representation exists. Its results are unique.

3.4.3 Boundaries of curved objects

The faces in a boundary representation (BRep) of a curved object may be represented as parametric patches, which are the images of a unit square or of a triangle by a polynomial mapping from a 2D parameter space into the 3D modeling space. Alternatively, a face may be represented as a trimmed surface by a reference to the supporting surface on which it lies, and by its boundary in that surface. The supporting surface may itself be a larger parametric patch or an implicit surface. The boundary is usually represented by a set of edges, which may be arranged into loops.

The edges of a solid typically lie on the intersection curves between two surfaces, and sometimes on singular curves of a single surface. A simple edge, such as a line segment or a circular arc, may be represented by its type, parameters, and position in space. More complex edges are often approximated by piecewise-polynomial parametric curves, either in 3D, or in the 2D parameter space of the host surface. The latter case leads to redundant representations because each edge typically belongs to two surfaces. Redundant representations may conflict due to numeric round-off errors, and cause “cracks” in the boundary. Exact, closed-form parametric representations for the intersection of natural quadric surfaces were first derived in the late 1970s at the University of Rochester for the PADL-2 modeler [Brown82]. The intersections of these edges with implicit polynomial surfaces can be computed efficiently by substituting the parametric expressions, $(x(t),y(t),z(t))$, of a point on the curve into the implicit polynomial equation for the surface, $f(x,y,z)=0$ and solving for t using an efficient numeric polynomial root finder.

Edge loops may be insufficient to define a face unambiguously. For example, a circular edge on a spherical surface is the boundary of two complementary faces. These may be distinguished by storing information about which points in the neighborhood of the edge belong to the face. This neighborhood information can be encoded efficiently, as a single-bit “left” or “right” attribute, in terms of the orientation of the surface normal and the orientation of the curve.

4. Modeling approximations and error measures

Alternate representations for a 3D geometric model are generated to produce bit-efficient coding or to generate impostors that resemble the original shape, but may be displayed more efficiently. In both cases, it is important to measure precisely the error introduced when substituting the compressed or simplified model to the original.

There are many ways to measure this error. A few are discussed in this section. However, we first discuss the error introduced by the modeling process itself, so as to better understand the impact of additional errors introduced during compression or simplification.

4.1 Modeling is a sequence of approximations

The modeling process is the result of a sequence of abstractions and approximations (idealization, surface approximation, and digitization). Hence, the resulting models and the properties computed using these models approximate those of the corresponding real objects to different degrees of precision.

First the physical shape is abstracted into a perfect and homogeneous 3D point set, ignoring internal structures and boundary imperfections.

In a second idealization stage, the boundary of the shape is approximated by a relatively small number of faces, each face being a subset of a surface of a type supported by the modeling system. Much of the variation between solid modeling technologies stems from the different choices of the primitive surfaces they support. These surfaces essentially determine the geometric domain of a system, i.e., the objects that can be modeled exactly.

On one hand, planar face primitives, i.e., triangles or more general polygons, provide a poor approximation of the real shape, unless used in large quantities to define fine tessellations of highly curved geometries. Although algorithms for dealing with individual triangles are simple and relatively robust, compact data structures for representing very large numbers of triangles and efficient algorithms for processing or rendering them are complex to develop and to maintain.

On the other hand, a single parametric free-form surface patch [Farin90] which may be smoothly stitched with other patches in a Non-Uniform Rational B-spline Surface (NURBS), may provide a better fit to the desired geometry than a thousand triangles. However, detecting and computing intersections of such free-form surfaces involves elaborate mathematical techniques and algorithms that are significantly slower and less reliable than their counterparts for triangular geometries.

Natural quadric surfaces (plane, cylinder, cone, sphere) offer an attractive compromise, because they provide mathematically exact representations for the majority of faces found in manufactured objects, and lead to closed form expressions for their intersection curves and to low degree polynomial solutions for the computation of the points where 3 surfaces intersect. However, these surfaces cannot model precisely the numerous fillets and blends found in most manufactured parts [Rossignac84]. They also cannot model sculptured or free-form surfaces that appear in many objects, especially those that must satisfy esthetic requirements, such as car bodies.

The choice of the geometric domain of a modeler may affect the accuracy of the analysis results. For example, a cylindrical pin may freely rotate in a cylindrical hole of a slightly larger radius, if both surfaces are modeled using natural quadrics. Using faceted approximations for the pin and the hole may lead to the wrong conclusion that the pin cannot rotate or doesn't even fit.

In a third approximation stage, the numeric parameters that define the precise shape and position of the surfaces and their intersections are rounded to the nearest value representable in the digital format selected by the developer of the system. The most common formats are floating point and integer or rational [Ralston83]. Floating point representations cover a wider range of values, but their worst case round-off error grows with the distance to the origin. Integer numbers, when scaled and offset properly by a judicious choice of units and of the origin, provide a much denser and uniform coverage of a desired modeling range, and hence lead to lower and better-controlled round-off errors. In practice, floating point numbers are favored because they do not require prior knowledge of the range and can be used in a uniform way to represent the parameters of the model and the results of intermediate calculations. However, floating point calculations generate and propagate round-off errors. The developers of a modeling system must ensure that these round-off errors do not lead to logical errors, to software crashes, or to wrong design decisions. Exact arithmetic packages do not suffer from round-off problems, but are significantly slower and usually only effective for polyhedral geometries (see discussions in [Banerjee96, Agrawal94]).

4.2 Geometric and visual errors

A simplification process takes a triangulated surface S and produces a simplified triangulated surface S'. The error that may be perceived in graphics applications 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 differ in color between the two images), 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 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. This area is large when **D** is parallel to the viewing direction and when the incident faces are orthogonal 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.

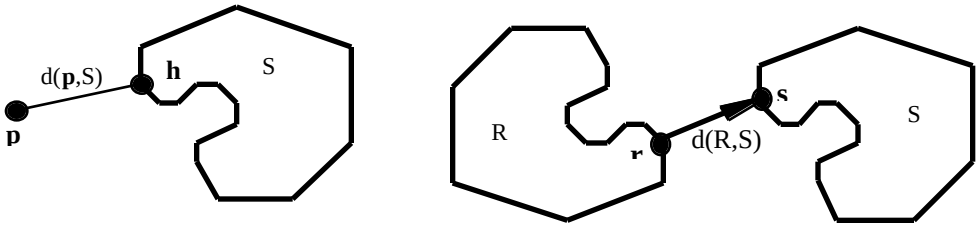
Numerous simplification techniques reported in the literature fail to provide a proper error bound and use instead heuristic estimates to guide the simplification process or the selection of the level of detail during rendering. Such strategies often results in simpler algorithms and sometimes to faster 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 into wrong conclusions about the model.

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.

4.2.1 Distances and deviations between sets

The **distance** d(**p**,S) between a point **p** and a set S is min(||**p-q**||) for **q** S. Here ||**p-q**|| stands for the norm of the vector from **q** to **p**.

The **closest set** Cl(**p**,S) to point **p** in set S is the set H = {**h** H : d(**p**,S)=||**h-p**||}. H may contain a single point, several isolated points, or a continuous set. For example, the closest set to **o** in a spherical surface centered around **o** is the whole surface.



The distance between a point **p** and a set S is achieved at point **h** (left). The distance between two points R and S is achieved (right) at a point of R and a point of S.

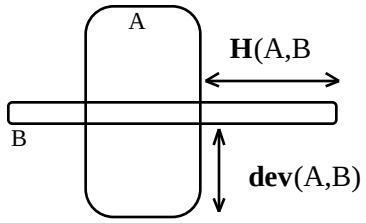
The **distance** d(R,S) between two sets R and S is min ||**r-s**|| for **r** R and **s** S.

The **shortest path** Sh(R,S) from set S to set R is the set of vectors **r-s** such that ||**r-s**||=d(R,S) for **r** R and **s** S. It represents the set of translations of minimal distance that bring the two sets in contact.

The **maximum deviation** De(R,S) from set R to set S is max d(**r**,S) for **r** R. It measures the largest distance from points of R to S. Note that this measure is not symmetric, in general De(R,S) ≠ De(S,R).

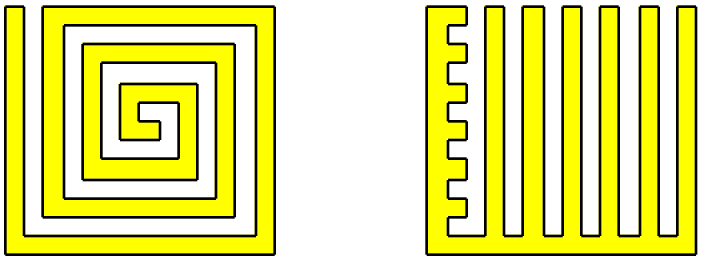
4.2.2 Hausdorff estimate of the geometric error

The geometric error between two sets S and S' may be measured using their Hausdorff distance, defined as **H**(S,S')=**max**(**De**(S,S'),**De**(S',S)). It measures the worst case distance that a point on any one set would have to travel to reach the other set. Note that **H** is a symmetric version (i.e, **H**(A,B)=**H(**B,A)) of the deviation **De**(A,B), as illustrated below.



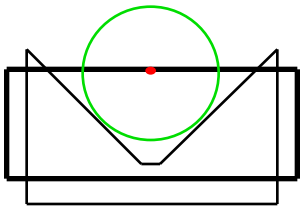
Because the two sets intersect, **d**(A,B)=0. The Hausdorff distance **H**(A,B) between the two sets is equal to the deviation, **dev**(B,A), but not to the deviation **dev**(A,B).

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 between two shapes (see the counterexample below).



When properly aligned, these two sets have a relatively small Hausdorff distance, yet they shapes differ significantly.

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. This situation is illustrated in 2D below.



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

A possible approach to computing the Hausdorff distance is to compute the intersection of the **skeleton** of one polyhedron with the boundary of the other and vice versa. This intersection contains points in the relative interior of the faces of the first polyhedron where the Hausdorff distance is realized. The skeleton of a polyhedron S is the set of points whose shortest path to the boundary of S has at least 3 distinct vectors.

Alternatively, simpler to compute, although less precise, bounds are often used.

Consider two sets, A and B such that H(A,B)=d. Let A_2 and B_2 be the projections of A and B onto any given plane. Then, $H(A_2,B_2) \leq d$. Consequently, the Hausdorff distance provides a view independent bound on the difference between the parallel projections of two shapes onto a plane. Furthermore, with some care, the Hausdorff distance may also be used to provide a bound for the difference between the same perspective projection oftwo shapes, and is hence suitable for bounding the deviation between two shapes in 3D graphics applications.

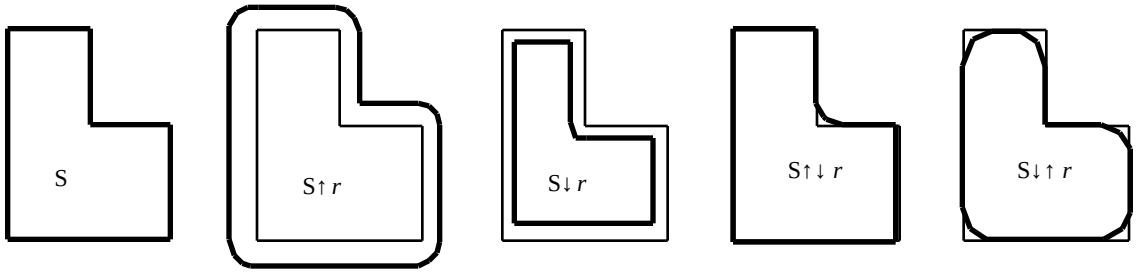
4.2.3 Offsets

The **expansion** $S\uparrow r$ of a set S by a distance r is the set of points **p** such that $d(\mathbf{p},S) \leq r$. $S\uparrow r$ is the union of balls of radius r and center in S. All points in the boundary of $S\uparrow r$ are exactly at a distance r from S, but the inverse is not always true.

The **contraction** $S\downarrow r$ of a set S by a distance r is the set of points **q** such that $d(\mathbf{q},cS) > r$, where cS denotes the complement of S. $S\downarrow r$ is the set of points that are the center of all the balls of radius r contained in S. Note that, $S\downarrow r = c((cS)\uparrow r)$ and that $S\uparrow r = c((cS)\downarrow r)$.

The **closing**, $(S\uparrow r)\downarrow r$, also denoted $S\uparrow\downarrow r$, is the set of points that cannot be reached by an open ball lying outside of S.

The **opening**, $(S\downarrow r)\uparrow r$, also denoted $S\downarrow\uparrow r$, is the set of points that can be reached by a closed ball lying inside S.



A reference solid S (left) may be expanded ($S\uparrow r$) or contracted ($S\downarrow r$).The closing ($S\uparrow\downarrow r$) and the opening ($S\downarrow\uparrow r$) define areas that may be reached by a ball of radius r that remains outside (respectively inside) the set S.

The support of the two offset operators (expansion and contraction) in a CSG modeling scheme have been studied in 3D in [Rossignac86b]. They are a special case of a Minkowski operation between sets. Their applications to blending [Rossignac84], has been implemented through closing (filleting) and opening (rounding).

The difference $S\uparrow r - S\downarrow r$ defines a tolerance zone around the boundary of S. Note that $S\uparrow r - S\downarrow r = (bS)\uparrow r$.

The distance between points in this tolerance zone and boundary of S is less than r. Given two sets, S and S', if $bS \quad S'\uparrow r - S'\downarrow r$ and $bS' \quad S\uparrow r - S\downarrow r$, then the Hausdorff distance between the two boundaries is less than r.

A simple representation of a set T included in $(bS)\uparrow r$ may be used to control a continuous simplification process which generates the boundary of a set S' through a continuous deformation of the boundary of S, while keeping it at al times in T.

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

4.2.4 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 vertices of a polyhedron, we will have an upper bound on the total Hausdorff error.

Moving a single vertex **v** of a polyhedron S by a displacement vector **d** (i.e., replacing **v** by **v**+**d**) results in a Hausdorff distance between the original shape S and its modified version S' that is lower than the norm of **d**.

Quantizing the vertices of a polyhedron for compression (i.e., truncating their coordinates to a fixed length integer fixed-point format) introduces an error ε on each coordinate. The total Hausdorff error resulting from this approximation is lower than ε times square root of 3.

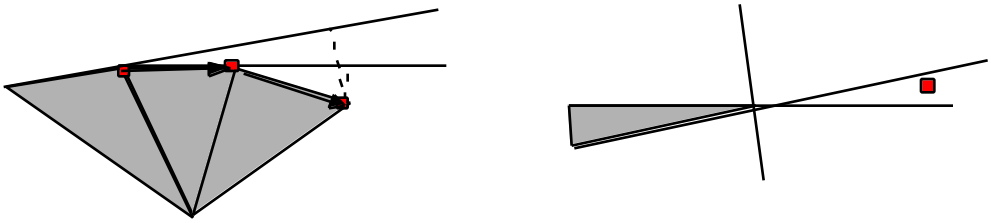
Similarly, simplification techniques which replace all the vertices that fall onto a single cell of a regular grid around the model cell by a single vertex representative also introduce a Hausdorff error of less than the diagonal of the cell.

Simplification techniques based on edge collapsing, where one vertex of an edge is merged with the other vertex somewhere along the edge introduce an Hausdorf error at each edge collapse that is bounded by the edge. As the vertex may be further moved by subsequent edge collapses, the total Hausdorff error may be bounded by the maximum of the sum of the length of all the edges along which a vertex has moved during edge collapses, or more precisely by the maximum of the total distance between the new and the original locations of the vertices. Nevertheless, such bounds computed from vertex displacements are in general pessimistic. For example, moving a vertex of a triangulation of a flat or nearly flat surface along that surface may result in a minor change of the object although the vertex displacement may be arbitrarily large.

4.2.5 Error bound based on distances to supporting planes

Vertex displacement provides a rather pessimistic error bound. Imagine for example a displacement of a vertex in a dense triangulation of a flat portion of a terrain. When vertices move along the plane of the terrain, the represented geometry remains flat, and therefore the Hausdorf error is zero, although the vertex displacements may be large.

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 edges or vertices, as shown below. In the cases of sharp corners, Ronfard and Rossignac introduce an additional plane that is orthogonal to the average normal to the incident triangles and suffices to guarantee a precise upper bound on the error.



The two successive edge collapses bring 2 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.

When a large number of vertices are clustered together and replaced by a single vertex, evaluating the Hausdorff distance approximation proposed in [Ronfard96] may require a large number of point-plane distance computations. Garland and Heckbert propose to use a least square distance to the supporting planes, rather than a maximum distance [Garland97]. The advantage of their approach is that the lease square distance may be computed for any point from a 4x4 matrix that is independent of the number of supporting planes. The drawback of their approach is that the least square distance does not provide an upper bound on the error, which limits its suitability for design inspection.

4.2.6 Minimizing color errors

The color error is somewhat more subjective, because it requires computing distances in color space, and also is harder to control. A slight variation in face orientations may result in considerable variation in specular reflections. Such color errors are best addressed by basing the estimation of the simplification error on photometric properties that may be view dependent. Two approaches are possible:

- the worst case color error will be computed for all possible viewing conditions (light position and viewpoint positions) and simplification or lossy-compression will be precomputed so as not to exceed a prescribed error bound
- the selection of the appropriate level of resolution for transmission or graphics will be performed at run time, based on the actual viewing and lighting conditions and some precomputed error characteristics

Note that shading the simplified model (i.e. a model with fewer faces) using the normals of the original model may produce excellent results for faces that are roughly orthogonal to the viewing direction, 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.

5. Compression approaches

5.1 The cost of storing polyhedral models

Relatively little attention has been devoted so far to the compression of 3D shapes. 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 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.

As discussed in Section 2, a triangular mesh may be defined by the position of its vertices (geometry) and by the association between each triangle and its sustaining vertices (incidence). One may wish to add color, normal, and texture information (photometry), which does not affect the 3D geometry, but influences the way the triangle is shaded. Although different normals and colors may be used for the various uses of the same vertex (one per incident triangle), we consider for simplicity that a unique normal is associated with each vertex and that a unique color is associated with each polyhedron. In fact, this simplifying assumption is often pessimistic, as in most scenes many vertices share the same normal. Note however that radiosity models require the storage of a different color with each vertex or triangle.

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 found in terrain models and 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.

5.1.1 Table of independent triangle descriptions

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 descriptions per triangle
- 2 vectors (vertex location and normal) per vertex used in the triangle
- 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 vertex description in the data stream.

5.1.2 Vertex and triangle tables

The redundancy of describing the same vertex 6 times in the above model 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 in general requires less bits than a vertex description, such schemes based on a **vertex and normal table** are more compact than the simple table of independent triangle descriptions representation discussed above. If we store, as described in Section 2, 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. This total includes **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

5.1.3 Triangle strips

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 high-end 3D graphics acceleration adapters, which perform geometric transformations, color calculations, triangle clipping, and triangle scanconversion and which have limited on board register memory for storing 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.

5.1.4 Deering's generalized strips

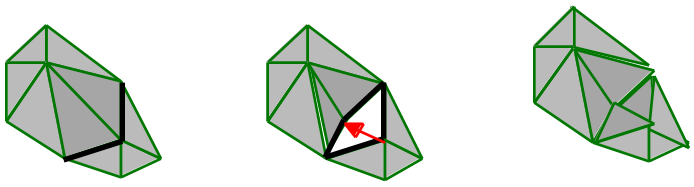
Because on average a vertex is used twice, either as part of the same triangle strip or of two different ones, the use of triangle strips requires sending most vertices multiple times. Deering's **generalized strips** [Deering95] extend the two registers used for OpenGL triangle strips to a 16 registers 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 should 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.

5.1.5 Hoppe's progressive meshes

Hoppe's **Progressive Meshes** [Hoppe96] specify a sequence of vertex insertion operations that construct the final mesh. Each vertex insertion is specified by 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 below.



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.

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.

5.1.6 Taubin and Rossignac's Topological Surgery

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 and Deering's compression for the geometry and photometry (**48** bits per vertex) and improves on both Hoppe's and Deering's approach by a factor of 4 for coding the incidence information (between **2** and **4** bits per vertex, depending on the incidence graph). This yields a total of **52** bits per vertex or less.

The Topological Surgery method for compressing the incidence relation is described in details in [Taubin96]. It starts with a simple representation of the triangles, maybe as described in Section 2. The method 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 of 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 was achieved 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. Variable length encoding of the differences between the actual and the estimate location of the vertices results in better a compression for the geometric information than coding the absolute coordinates, because the coordinates of the difference vector are smaller on average.

Turan has shown that the incidence of a simple mesh can be encoded using **1 2** 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 relations, and hence are able to further improve the compression by a factor of 3.

6. Graphic acceleration techniques

Interactive manipulation of 3D models and interactive inspection of 3D scenes cannot be effective when the graphics feedback requires more than a small 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 [Rossignac97b]. 3D models of typical mechanical assemblies (appliances, engines, cars, aircrafts...) contain thousands of curved surfaces. Accurate polyhedral approximations of these precise models, typically constructed to interface with popular graphics APIs, involve millions of polygonal faces. Such complexity exceeds by several 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.

6.1 Cost factors

To better identify the performance impact of 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. The primary ones are:

- **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 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 the software and the hardware costs: **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.

6.2 **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.

6.2.1 Meshing or storing transformed vertices

In a polyhedral scene, the number **T** of triangles is roughly twice the number **V** of vertices (**T=2V-4** for a single manifold shell with no handles).

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].

6.2.2 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].

6.2.3 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.

6.2.4 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.

6.2.5 Back-face culling

It is redundant to display faces that are not visible. A face may be invisible if it lies outside of the viewing frustum (efficient tests were discussed above) or when it is hidden by other faces (precomputing visibility is expensive as discussed in the previous paragraph. However, a large fraction of hidden faces may be detected directly without visibility tests when displaying models of solid and assuming that the viewpoint lies outside of all solids. Indeed, about half the faces (in fact a bit more under perspective viewing) are oriented so as to have incident 3D cells in the direction towards the viewpoint. These faces are called back-facing (or simply back faces). If the normals to the faces are consistently chosen to point towards the exterior of the solids, then testing whether a face is back facing amounts to computing the sign of the cross-product between the normal and the vector from the face to the viewer. (Note that many graphics libraries and hardware accelerators do not use this test, but rather use the assumption that vertices in the loops that bound the face are ordered anti-clockwise around the outward normal to the face.) Although the graphics sub-system can quickly reject such back faces if desired, it is still expensive to send them and to transform their vertices and normals. It is therefore beneficial to use a face organization that allows a rapid rejection of sets of back faces without having to process them at all. Clustering faces according to their normal may be used, but care must be taken to account for perspective deformations when rejecting a cluster. For example, a cluster of faces may be represented by an average normal, a bound on the maximum deviation angle α between this average normal and the normals of the faces in the cluster, and a bounding sphere around all the faces. The whole cluster will not be visible if the vectors from any point on the bounding sphere to the viewpoint form an angle with the average normal that is larger than $\Pi/2+\alpha$. Furthermore, breaking the faces into such orientation clusters may be incompatible with the construction of long triangle strips.

6.2.6 Use of images and textures

A distant or small group of objects may be replaced with a "3D sprite" or with a few textured polygons [Perlin84, Kajiya85, 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 [Chen93]. 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].

6.2.7 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, Upstill90]. 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 [Blake87, Erikson96, Heckbert94]. More recent results are reviewed in [Heckbert97].

The next section presents several simplification techniques in detail.

7. Simplification algorithms

We focus in this section 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.

Simplification, in its simplest form, generates decreasing levels-of-detail (LOD), which involve decreasing amounts of 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 have discussed both geometric and visual measures of the discrepancy between S and S' in Section 4.

In this section we present 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.

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 (unless subdivided) 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).

7.1 Mesh reuse

The original mesh may be simplified by collapsing some of its edges or by clustering its vertices and then removing the degenerate triangles collapsed during these transformations. With these approaches, there is no need to fit a new mesh to the original geometry, because the new mesh is an alteration of the original one.

Alternatively, one may decide to generate a new mesh, with fewer triangles, to fit the original geometry.

Several compromises between the two approaches have been proposed, where subsets of the original mesh are removed and new triangulations fit to the holes.

Incremental approaches simplify the original mesh one step at a time. Their advantage lies in the fact that they can stop as soon as the desired triangle count reduction is achieved or the limit error is achieved for the desired level of resolution. Their drawback lies in the fact that one needs to maintain a consistent topological structure throughout the whole process. Topological changes during simplification must be detected and either prevented or fully supported.

The alternative is to simplify the geometry by merging vertices, and then to remove degenerate triangles. This approach is more efficient and more immune to topological inconsistencies in the original model.

7.2 Static versus adaptive resolution models

The objective of many automatic simplification algorithms is to produce a simplified model B of an original mesh A where B resembles A from all directions and where the number of triangles in B is significantly lower than in A.

A broader and more general objective is to produce a datastructure well suited for extracting in realtime a multiresolution model for A which would display the features of A that are closer to the viewer with more details than distant features.

If A is a relatively small and relatively simple object in a larger, more complex scene, then under most viewing conditions all the features of A would be at similar distances from the viewer and would hence require comparable

levels of resolutions. Consequently, it may not be necessary to compute an adaptive multiresolution model for A. Several constant-resolution models may be precomputed and used as dictated by the viewing conditions and the performance constraints imposed by the user or the application. When A appears small on the screen, a model of lower resolution, having fewer triangles, would be used to increase performance. As the view point approaches A, A will appear larger on the screen and a higher resolution model may be used.

To avoid the disconcerting and distracting effect of popping details as one switches from one resolution to the next, a smooth transition may be used to interpolate between the two levels, gradually revealing or hiding the details. During the transition phase, the higher resolution model must be used, although the desired accuracy may have been achieved with the lower resolution model. Furthermore, the interpolation must be computed at each frame. The effect of these additional costs may be reduced by reducing the transition period to a small interval around the cross-over resolution value.

When A is a large and complex model relatively to the scene, using a uniformly lower resolution model is only possible when A is sufficiently far away from the viewer. If the viewer is close to one part of A, the full resolution model must be used to ensure that the details close to the viewer are displayed properly. Consequently, features far from the viewer will be displayed with unnecessary resolution, yielding a performance penalty. A recent focus of much of the research in simplification is to produce adaptive models that provide higher resolution where it is needed and that smoothly adapt this resolution as the viewpoint is moved.

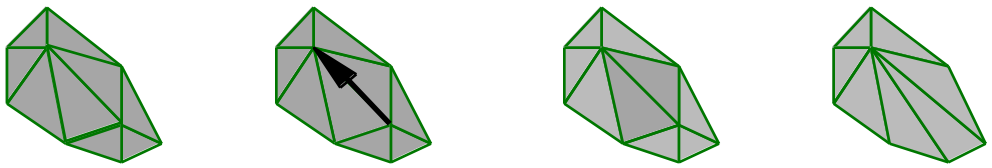
Note that a simple way to achieve adaptive resolution is to split A into smaller components for which uniform resolution simplifications have been computed and to decide on the particular resolution of each component. based on its distance from the viewpoint. When A is an assembly of smaller, numerous, and uniformly distributed solids, these may be used as the natural components and there is no need to perform geometric computations to split A into its components. Otherwise, A may be split so as to avoid increasing the complexity of its description and to ensure that the choice of different resolutions for neighboring components does not generate visible cracks at the junctions of their surfaces. An alternative is to compute an adaptive model for the entire set A. Proposed approaches include Hoppe's Progressive Meshes [Hoppe97], Luebke's octree based extension of Rossignac's and Borrel's vertex clustering technique [Luebke97], and Lindstrom's adaptive terrain models [Lindstrom96].

7.3 **Incremental simplification steps**

The atomic simplification steps simplifies a **feature** of the model. A feature may be the entire solid or a set of connected triangles. (Other transformations, such as merging disconnected vertex, swapping edges, or performing other topological changes are discussed separately, as they enable further simplification, but are not simplification steps by themselves.) We review several atomic simplification steps and discuss relations between these steps.

7.3.1 Edge collapse

An edge may be contracted so that its two vertices become coincident. This **edge collapse** operation eliminates all the triangles that were incident upon the edge. For manifold shells without boundary there are always two such triangles (see below).



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.

Sequences of edge collapses will simplify a manifold mesh, but can produce meshes with invalid embedding (the simplified mesh intersects itself or is no longer the boundary of a regularized manifold set) and with different topologies. Most schemes based on edge collapsing maintain a topological representation of the models, and only allow collapses of edges that preserve the desired validity conditions. Efficient techniques have been proposed [Ronfard96, Hoppe96] for always keeping track of the edge whose collapse would result in the lowest

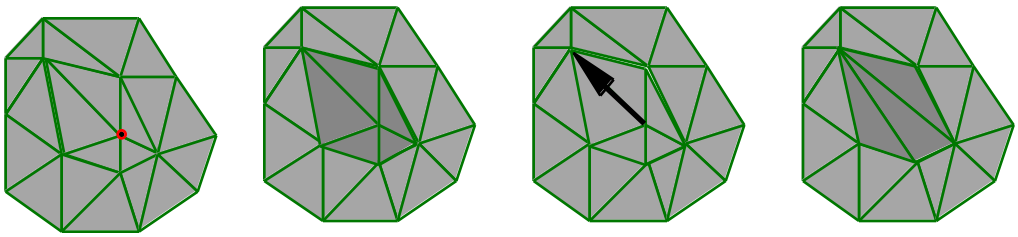
total error between the simplified model and the original one. These techniques maintain a priority queue and an incidence graph throughout the simplification process.

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). Edge collapsing techniques are complex to implement and slow, because they require maintaining a complete incidence graph. They also impose strong topological constraints on the input polyhedra.

7.3.2 Vertex, triangle, and flat-region decimation

The **decimation** of a nearly-flat vertex [Schroeder92] is equivalent to an edge collapsing operation, possibly combined with edge-flips [Lawson72], where the common edge between two adjacent triangles forming a diagonal of four points is replaced by the other diagonal. This operations corresponds to the addition or subtraction of a tetrahedron. The result of a sequence of such decimations results in a hierarchical clustering of vertices.

The approach was generalized to nearly flat sets of connected triangles (called superfaces) [Kalvin91, Kalvin96] that are constructed by incrementally merging triangles. Re-triangulating a superface amounts to clustering all the internal vertices into a single “star” vertex, the apex of the new triangulation for the region.



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).

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. Techniques that collapse edges and techniques that cluster disconnected vertices may be combined [Popovic97].

7.4 **Criteria and error estimates for a simplification step**

Several techniques have been applied to estimate the error resulting from a simplification step.

7.4.1 Hausdorff

We have already discussed the difficulties of computing a tight bound on the Hausdorff distance between two polyhedra and the possible divergence between two shapes separated by a small Hausdorff distance. Because of these difficulties, the precise Hausdorf distance has not been used in practice.

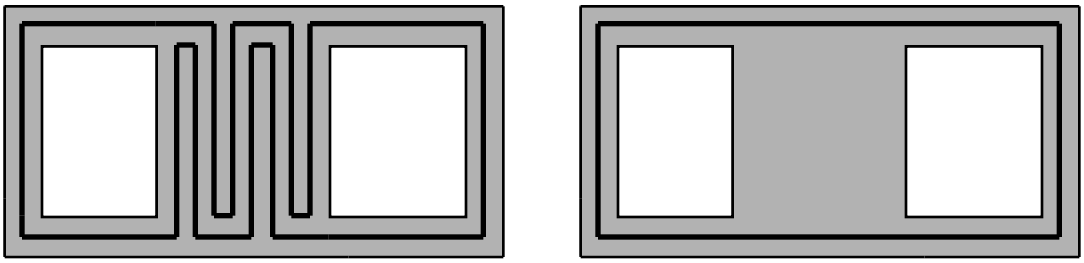
7.4.2 Local curvature

The local curvatures at a vertex or edge may be estimated by taking into account the neighboring vertices of the polyhedral surface. Some schemes will decimate vertices or triangles if the local curvature is low. This conservative approach works well when one tries to reduce the complexity of smooth surfaces that have been overtessellated (i.e. that are represented by far too many triangles). However, curvature based techniques are not capable of simplifying shapes with lots of small features that are not perceptible from far, but that have sharp angles. For example, if we consider a sphere tessellated with a triangles, most decimations schemes will produce suitable simplifications. However, if now the original sphere is perturbed by displacing each vertex randomly by a very small fraction of the sphere radius, decimation schemes based on local curvature will not simplify the model, unless we reduce the curvature error bound, which would result in a very poor approximation of the overall shape of the sphere.

7.4.3 Tolerance zone

The expanded version of the boundary of a polyhedron defines a tolerance region that can be exploited to drive the simplification process. The principle is to allow simplification steps that apply a continuous deformation while maintaining at all times the boundary of the evolving model inside this tolerance zone. Any sequence of such transformation that preserve this inclusion may be used, and this until no more simplification steps are possible. To get a cruder model, we augment the tolerance zone and resume the process.

In general, maintaining the boundary within the tolerance region does not guarantee a bounded Hausdorff distance. In fact, one can construct cases where the Hausdorf distance is arbitrarily large, as illustrated below.



The dark shaded polygon on the left has a tolerance region (shaded) that does not reflect the shape of the polygon. A smooth transformation may simplify the polygon (right) while maintaining it inside the tolerance zone. Yet the Hausdorff distance between the original and the simplified model excedes the tolerance.

7.4.4 Distance to planes

The placement of a single point **P** on a surface by a small vector **d** may be measured by the projection of **d** onto the normal to the surface at **P**. The lateral displacements slide the surface tangentially and do not change the shape. Therefore, the error may be estimated by measuring the surface displacement along the normal to the surface. A surface does not have a single well defined normal at sharp vertices. Therefore, one may use a normal to each face incident upon the vertex. Sharp vertices require additional normal directions that will measure the displacement away from the vertex as indicated below.

The displacement of the vertex on the left is measured using the distances to the lines that support the abutting edges. However, the vertex on the right requires an additional direction that will control how far the vertex is moved in a direction that is close to both lines.

These displacements indicate the distances to the planes supporting the triangles incident upon the vertex. If a vertex does not move more than a prescribed distance along any of these normals, then points in the triangles incident upon the vertex will not move by more. Consequently, the maximum excursion of a vertex along the normals provides a much better bound on the Hausdorff distance than other measures. Note that the bound needs to be adjusted since the normals to the incident planes and the additional normals are only a sampling of all

directions for which vertex displacement should be recorded. This estimator was introduced by Ronfard and Rossignac {Ronfard96].

When a vertex is moved and merged with another vertex, we need to combine the normals. Rossignac and Ronfard keep a list of all supporting planes. Garland proposes to replace the list of planes by a 4x4 matrix which suffices to evaluate the mean square distance [Garland97] to an unlimited set of planes, independently of their number. Although the least square measure is not appropriate for many applications, because it does not provide an upper bound, the solution is elegant and effective.

7.5 **Optimal vertex placement**

A sequence of edge collapses or equivalent operations reduces the complexity of the model. However, the resulting geometry may be adjusted so as to minimize the geometric error resulting from the simplification steps.

7.5.1 Minimize Hausdorf distance

It may be difficult to find the best placement for the vertices so that the Hausdorff distance is minimized. A sampling of the original surface was used to drive the vertex placement optimization [Hoppe93]. This process is slow and it may be difficult to guarantee optimality.

7.5.2 Minimize distance to plane

Given that the list of planes is available in the simplification approach proposed by Rossignac and Ronfard, they may be used to drive the optimization of vertex placement. The least square distance to a set of planes may be solved as a linear optimization problem. Minimizing the maximum distance requires more work.

7.5.3 Minimize volume change

For some applications, it may be important to maintain a constant volume during simplification. A vertex resulting from an edge collapse may be positioned so as to maintain the total volume of the enclosed solid [Gueziec96].

7.6 **Topological changes**

Although some authors initially praised simplification techniques that preserved topology, most agree today that simplification of complex parts must be accompanied with topological changes. These changes may be decomposed into the primitive operations discussed below.

7.6.1 Merging components

Disconnected components, such as 100 small spheres, may need to be merged into a single simplified shape when viewed from far. Distinct features of a single solid, such as a hair brush should also be merged, although they may not be adjacent in the incident graph of the solid. These transformations are best modeled by the merging of two or more vertices.

7.6.2 Removing handles

A wire fence may be viewed as a single solid with a large number of handles. Advanced levels of simplification can only be achieved if some of these handles are eliminated. Edge collapsing operations may reduce a handle to a single edge. A new simplification operation is needed to remove this edge and thus free its two vertices to slide along the two surfaces that were bridged by the edge.

7.6.3 Removing bridges

The dual of the hole made by a handle is a bridge (i.e. the handle itself). If the hole is large and the handle is thin, simplification steps will typically reduce the handle to a single poly-line, which could be eliminated, although we prefer to keep it to indicate the presence of material.

8. Examples of simplification approaches

We illustrate the simplification techniques discussed above with a few simplification approaches.

8.1 De Rose surface fitting

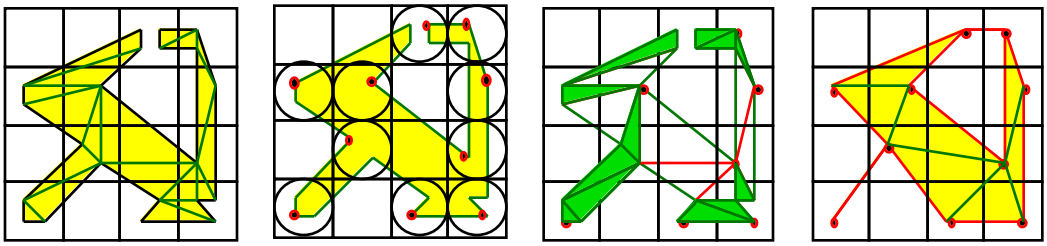
Surface fitting techniques to regularly spaced scanner data points [Schmitt86, Algorri96] may also be used for producing levels of detail polyhedral approximations [DeHaemer91]. These techniques have been recently extended to unorganized data points [Hoppe92, DeRose92] and to new sparse points automatically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

8.2 Varshney's envelopes

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].

8.3 Rossignac and Borrel's vertex quantization

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 attached figure).



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.

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

8.3.1 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).

8.3.2 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.

8.3.3 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.

8.3.4 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.

8.3.5 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.

8.3.6 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.

8.3.7 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.

8.3.8 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.

8.3.9 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.

8.4 **Low and Tan ‘s improvements**

Rossignac and Borrel’s 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.

8.5 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.

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]).

8.6 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.

8.7 **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.

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.

8.8 **Hoppe's 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.

9. Adaptive surface tesselation

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.

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].

9.1 Lindstrom's adaptive terrain models

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.

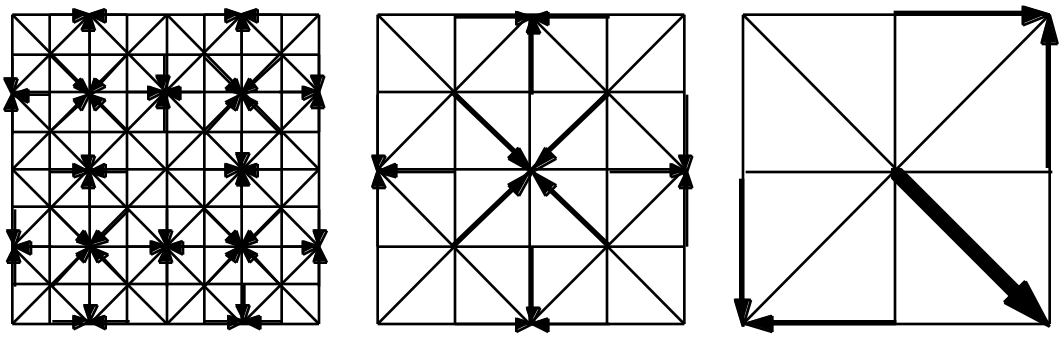


Figure 4: The hierarchical edge collapsing approach for a regular terrain model (left) produces a similar lower-resolution models by a sequence of vertical and then diagonal edge collapses (center). The process may be repeated iteratively (right). Not all edges at a given level need to collapse.

10. 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 for more precise and more complex 3D models call for algorithmic solutions that improve the speed of visualizing and transmitting complex 3D scenes.

We have 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.

We have also presented several techniques, which automatically compute 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 simplest and most efficient methods collapses clusters of vertices and removes degenerate triangles.

The next challenge is to combine adaptive multi-resolution models with compression schemes to provide fully scaleable models.

11. Bibliography

[Agrawal94] A. Agrawal and A. A. G. Requicha, "A paradigm for the robust design of algorithms for geometric modeling", Computer Graphics Forum, Vol. 13, No. 3, pp. 33-44, September 1994. (Proc. Eurographics '94.)

[Alexandroff61] P. Alexandroff, Elementary Concepts of Topology, Dover Publications, New York, NY, 1961.

[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.

[Banerjee96] R. Banerjee and J. Rossignac, Topologically exact evaluation of polyhedra defined in CSG with loose primitives, to appear in Computers Graphics Forum, Vol. 15, No. 4, pp. 205-217, 1996.

[Baumgart72] Winged Edge Polyhedron Representation, B. Baumgart, AIM-79, Stanford University Report STAN-CS-320, 1972.

[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.

[Brown82] C. Brown, PADL-2: A Technical Summary, IEEE Computer Graphics and Applications, 2(2):69-84, March 1982.

[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.

[Chen93] E. Chen and L. Williams, View interpolation for image synthesis, Proc. ACM Siggraph. pp. 279-288, 1993.

[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 and 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.

[Farin90] G. Farin, Curves and Surfaces for Computer-Aided Geometric Design, Second edition, Computer Science and Scientific Computing series, Academic Press, 1990.

[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. {http://www.cs.cmu.edu/~garland/scape). Sept.1995.

[Garland97] M. Garland and P. Heckbert, Surface simplification using quadric error metrics, Proc. ACM SIGGRAPH'97. To appear. 1997.

[Greene93] N, Greene, M, Kass, and G, Miller, Hierarchical z-buffer visibility, 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.

[Heckbert97] P. Heckbert and M. Garland, Survey of Polygonal Surface Simplification Algorithms, in *Multiresolution Surface Modeling Course*, ACM Siggraph Course notes, 1997.

[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, View Dependent Refinement of Progressive Meshes, 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.

[Kajiya85] J. Kajiya, Anisotropic reflection models, Computer Graphics, Proc. Siggraph, 19(3):15-21, July 1985.

[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.

[Luebke97] D. Luebke and C. Erikson, View-dependent simplification of arbitrary polygonal environments, Proc. Siggraph'97. To appear.

[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.

[Perlin84] K. Perlin, A unified texture/reflectance model. Siggraph'84, Advanced Image Synthesis Sminar, July 1984.

[Popovic97] J. Popovic and H. Hoppe, Progressive Simplicial Complexes, Proceedings ACM SIGGRAPH'97, August 1997.

[Ralston83] A. Ralston and E. Reilly, Editors, Encyclopedia od Computer Science and Engineering, second Edition, van Nostrand Reinhold Co., New York, pp .97-102, 1983.

[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.

[Rossignac84] J. Rossignac and A. Requicha, Constant-Radius Blending in Solid Modeling, ASME Computers in Mechanical Engineering (CIME), Vol. 3, pp. 65-73, 1984.

[Rossignac86] J. Rossignac and A. Requicha, Depth Buffering Display Techniques for Constructive Solid Geometry, IEEE Computer Graphics and Applications, 6(9):29-39, September 1986.

[Rossignac86b] J. Rossignac and A. Requicha, Offsetting Operations in Solid Modelling, Computer-Aided Geometric Design, Vol. 3, pp. 129-148, 1986.

[Rossignac89] J. Rossignac and M. O'Connor, SGC: A Dimension-independent Model for Pointsets with Internal Structures and Incomplete Boundaries, in Geometric Modeling for Product Engineering, Eds. M. Wosny, J. Turner, K. Preiss, North-Holland, pp. 145-180, 1989.

[Rossignac90] J. Rossignac, Issues on feature-based editing and interrogation of solid models, Computers&Graphics, Vol. 14, No. 2, pp. 149-172, 1990.

[Rossignac91] J. Rossignac, and A. Requicha, Constructive Non-Regularized Geometry, Computer-Aided Design, Vol. 23, No. 1, pp. 21-32, Jan./Feb. 1991.

[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.

[Rossignac97] J. Rossignac, Structured Topological Complexes: A feature-based API for non-manifold topologies", Proceedings of the ACM Symposium on Solid Modeing 97, pp.1-12, May 13-15, Atlanta, 1997.

[Rossignac97b] J. Rossignac, The 3D revolution: CAD access for all, International Conference on Shape Modeling and Applications, Aizu-Wakamatsu, Japan, IEEE Computer Society Press, pp. 64-70, 3-6 March 1997.

[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.

[Upstill90] S. Upstill, *The Renderman Companion.* Addison Wesley, Reading, MA, 1990.

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

[Weiler87] K. Weiler, Non-Manifold Geometric Boundary Modeling, ACM Siggraph, Tutorial on Advanced Solid Modeling, Anaheim, California, July 1987.

[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.