3D Compression of Engineering Models for Cooperative Computing Applications

S. P. Mudur Dinesh Shikhare

National Centre for Software Technology
Gulmohar Cross Rd. 9, Juhu, Mumbai 400049, India.
{*mudur*|*dinesh*}*@ncst.ernet.in*

Abstract

With wide spread access to Internet-based computing, increasing communication bandwidths and powerful desktop graphics facilities becoming available, cooperative and collaborative computing is gaining popularity in a wide range of application domains such as CAD/CAM/CAE, Virtual Reality, E-entertainment, E-Commerce, E-education, Telemedicine and many other scientific applications. 3D Geometric models are important components in many of these applications. The level of detail in which such models are represented varies with the application, but it is a known fact that model sizes are increasing and far exceed what can be easily downloaded at acceptable speeds for collaborative computing purposes. Compression of such 3D models is essential. As a result in the recent years, 3D compression has emerged as a new branch of the general field of data compression aimed at the use of large 3D models. In this paper we will first present an overview of the state of the art in 3D compression and then describe a new compression scheme that has been developed by the authors. The new scheme is specially suited to models in the engineering class and results in much higher compression than all earlier known techniques.

1. Introduction

With pervasive network connectivity there are major changes in the computational paradigm in a large number of applications, particularly high performance computing applications, which usually require collaboration amongst many distributed users [21, 24, 2]. Until recently, use of the network for collaborative computing in such applications has been restricted to local environments. Further, use of proprietary standards have required sophistication and effort on the part of the user to exploit distributed computing technology. The Internet transcends this traditional model, providing native support for the global computer web, and thus making it relatively easier for users and computers distributed geographically to cooperate. To date, however, it has been used primarily to publish textual information over the World-Wide Web. Cooperative computing represents the next step in exploiting wide-scale network connectivity wherein distributed users and computers globally cooperate on common computing problems. With the development of highly innovative compression techniques for other media types like images, music and video, the Internet is beginning to be used for more sophisticated collaborative computing applications.

Applications such as CAD/CAM/CAE, Virtual Environments, E-entertainment, E-commerce, E-education, Telemedicine and many other such scientific applications are based on the shared use of large and complex 3D representations. Collaboration between designers/players/users is an increasingly important aspect in complex design/gaming/visualization situations, as exemplified in the above domains. 3D geometric models at different levels of detail are used as representations in the above application domains, and thus 3D geometry is emerging as a new media type to be dealt with. Typical 3D data are often very large in size, ranging from several hundred kilobytes to several dozen megabytes. Developing interactive real-time applications with such data assumes, implicitly or explicitly, that the entire data can be loaded into main memory for efficient run-time processing. This places enormous burden on storage space as well as transmission bandwidth. One way to alleviate this problem is to store compressed representations. As a result in the recent years, 3D compression has emerged as a new branch of the general field of data compression aimed at the use of large 3D models.

Any compression technique designed for cooperative use of 3D models must include the following features:

(a) *Wide applicability*: Must compress 3D models for the most popular representation schemes.

(b) *High compression ratio*: This is possible by effective exploitation of data redundancy exhibited by large 3D models.

(c) *Control over lossiness*: Compression may be lossy or lossless. In case of lossy compression, the amount of loss that is acceptable will depend very much on the application

on hand. Hence user control over this is very important.
(d) *Ubiquitous access*: simple and fast decompression, causing minimum overhead during runtime reconstruction so that 3D models can be accessed from the desktops and in future even from mobile computing devices.
(e) *Minimum latency in reconstruction*: Since users are geographically distributed, 3D models must be visualisable with minimum latency. It should not be required that a large part of the compressed model be available before the decompression process can commence.
(f) *Multi-resolution representation*: is highly desirable. This offers the basis for LOD (Level of Detail) processing of compressed data.
(g) *Selective component-wise compression*: In many such applications, it is more effective to selectively compress a dataset component-wise rather than the entire dataset in totality. It is very desirable that a compression scheme includes this selective compression capability in its encoding algorithm for better compression.

This paper presents a new 3D compression scheme that includes a number of the above features and can be used in a large number of cooperative computing applications that are based on shared access to 3D data over the net. The new scheme is specially suited to models in the engineering class and results in much higher compression than all earlier known techniques when applied to models in this class. The rest of this paper is organized as follows: In Section 2, we provide the definitions and other background concepts, including a brief review of earlier techniques in 3D compression. In Section 3, we give the details of our new compression scheme. In Section 4, we present implementation results and complexity analysis of our algorithms. In Section 5, we discuss merits of our compression scheme in the context of cooperative computing. Finally, we present conclusions and possible extensions in Section 6.

2. Background

2.1. Polygon Mesh Models

A polygon mesh model is typically defined in terms of (i) *geometry* – coordinate values of vertices making up the model, (ii) *connectivity* – adjacency relationship among the vertices which defines the polygonal faces of the model (also called as *topology*), and (iii) *non-geometric attributes* – vertex/polygon colour, vertex normals, texture, material properties, etc.

Polygon mesh model: A *polygon mesh model* O consists of a set S of polygon meshes and associated non-geometric attributes. A *polygon mesh* $s \in S$ consists of a set V of vertices, a set E of edges and a set P of polygons. Each vertex corresponds to a point position from the set $X = \{x_i \in \mathbf{R^3}\}$. A *triangle mesh* is a special case with all polygons being triangles.

Connected components: In a mesh model O, we call two polygons as *adjacent polygons* if they share an edge. There exists a *path* between polygons p_i and p_j if there is a sequence of adjacent polygons $p_i, p_1, p_2, ..., p_j$. A maximal subset O_c of the mesh model O is called a *connected component* if there exists a path between any two polygons in O_c. Note that a given mesh model may have multiple connected components. A mesh can be trivially decomposed into its connected components using a simple labelling algorithm based on breadth-first or depth-first traversal of complexity $O(n)$.

A popular data structure for representing and storing polygon meshes is to use a shared list of vertex coordinates to store the geometry and a list of vertex indices for each face to store mesh connectivity. For triangle meshes of v vertices, this requires approximately $3v$ space to store the vertex coordinates and $3t \log_2 v$ space to store connectivity among the vertices for t triangles, where t nearly equals $2v$. Using a four-byte precision for the coordinates, a model of 1000 vertices would require 20K bytes of storage and more than 3-5 seconds of transmission time at typical modem speeds. The transmission costs increase non-linearly as the number of vertices increase [15].

2.2. Characteristics of Large 3D Models

Natural and Sculptured Objects: The digital models of natural shapes such as terrains, anatomies, sculptures, etc. are typically acquired using semi-automatic techniques like 3D scanning [28, 5, 16], satellite imagery, stereogrammetry or by construction of iso-surface boundary in a volumetric image [18]. Such techniques enable accurate capture of the complex curvatures in such natural shapes. Models in this class usually have a small number of connected components (often a single component) having a large number of densely distributed polygons. Terrain models represent natural surfaces of the form $z = f(x, y)$. This simple graph surface structure has been exploited by many special techniques developed for their simplification and rendering [17].

Architectural/Heritage Monuments: Three-dimensional models of architectural and heritage monuments are also extremely detailed and rich in texture and material property specifications. These models have a much larger number of connected components. Often, most of the connected components are simple geometric shapes and have a sparse distribution of points and polygons. Further many vertices are repeated in their geometric positions to capture different textures associated with the different faces meeting at such vertices. Due to this, many polygons are adjacent only geometrically but not topologically. The vertices also have associated normals used while computing local and global illumination.

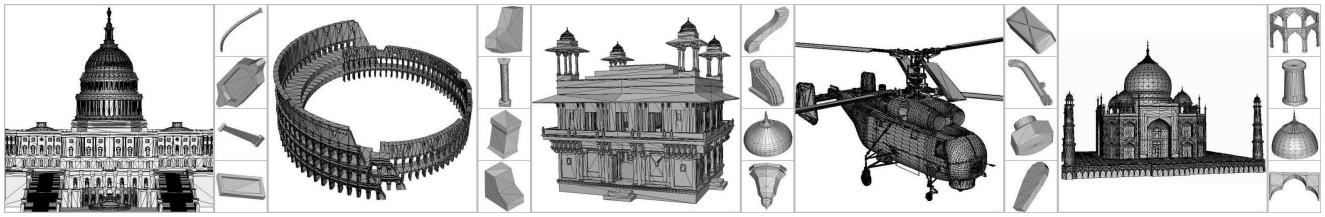


Figure 1. Architectural and engineering models with some examples of most frequently repeating features.

Engineering Designs: Detailed engineering models such as power plants, aircraft designs, automobile assemblies, etc. are also examples of large models, most significant for industrial applications such as virtual mock-ups, collaborative CAD software [21, 2], interactive inspection of 3D models, etc. While we seldom see engineering models with texture, they have attributes like material properties associated with meshes. Sometimes, these meshes also have colour values associated with the vertices of the model. These models often consist of large number of components, many of which may be geometrically more complex than in architectural/heritage models.

2.3. Distinctive Properties of Engineering Models

Clearly, architectural/heritage models and engineering designs, which we shall collectively refer to as engineering models, form a large class of 3D data used in industrial applications. We observe some common properties of models in this category:

High combinatorial complexity: These models have a large number of simple to moderately complex connected components – each having up to a few hundred vertices – forming combinatorially complex structures.

Repeating features at different levels of granularity: Many shape features at the granularity of connected component- and sub-component levels repeat in the model.

Multiplicity of representation: The repeating geometric shapes are multiply described[1]. For example, all teeth of a gear have the same geometric shape in a single mesh component of a mechanical assembly model, yet each tooth of the gear is described with complete geometric and topological details.

Arbitrary grouping of polygons across meshes: The organization of the geometry is in terms of meshes, where each mesh is often a collection of all polygons that share the same material property or texture map. This means that meshes in such models are likely to consist of multiple connected components.

Sharp discontinuities in geometric features: Unlike natural shapes, most geometric features in such models have sharp edges, corners and other discontinuities.

Modeling errors: Almost always, these models are manually created using 3D modeling tools, such as say, 3D Studio MAX [8]. As a result, such models often have representational defects like erroneous replication of geometry, invisible geometric shapes, etc.

[1] Although modeling tools and standard file formats have facilities for compact representation of such repetition of component level features, in practice almost all commonly available large models are in fully expanded format.

2.4. Previous Work in 3D Compression

Connectivity compression: Compression of connectivity (or topology) information is achieved by defining and encoding a traversal of all the polygons/triangles in such a way as to minimise repeated references to vertices that are shared by multiple polygons/triangles [11, 22, 27, 14, 10]. All traversal encoding schemes we know encode the mesh topology in a lossless manner. A majority of the mesh compression techniques specialize in the compression of smooth manifold surfaces represented by triangle meshes [22, 26, 27].

Geometry compression: Compression of geometric data and attributes is achieved by reducing the precision with which coordinate values, normal vector components, etc. are represented [7, 26, 27, 20]. Geometric data is also compressed by suppressing highly detailed and redundant features [12, 13] using techniques from signal processing. By their nature, these techniques are necessarily lossy.

Attributes such as colour and material properties are also compressed by reducing the precision of their representation [7].

Progressive compression: In addition to achieving a compact encoding of the mesh model, some techniques also encode the model for progressive disclosure [3, 20, 25].

2.5. Compression Needs of Engineering Models

Almost all of the previous research has concentrated on the compression of large models having a few connected

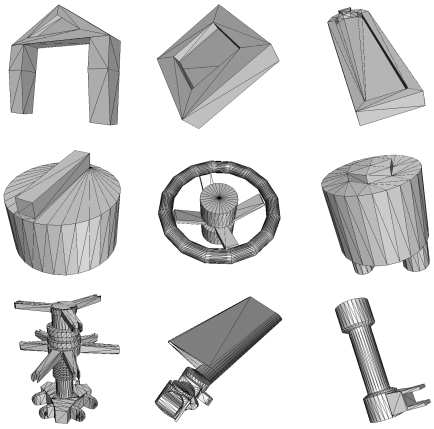


Figure 2. Mesh components that are parts of polygon mesh models of engineering designs. Note the sharp edges, corners, discontinuities, large triangles.

components – often a single mesh of a complex surface with smoothly varying curvature – formed by a dense collection of a large number of small polygons (mostly triangles). The traversal techniques are best suited when we have such very long traversals. Longer the traversal sequence, smaller is the average cost of representation. As already mentioned, very large 3D geometric models of the engineering class usually have a large number of non-smooth meshes, with small number of large triangles, often with arbitrary connectivity (see examples in Figure 2). Meshes have associated texture maps/material properties, and vertices need to be repeated. Hence, reducing repeated references to vertices and creating long traversal sequences does not generally become possible, making these methods not the best suited.

All the earlier developed techniques fail to exploit a fundamental property that is almost always present in engineering models - repeating occurrences of shape features as shown in Figure 2. Repeating features are repeatedly encoded by these compression algorithms. Such redundancy must be automatically discovered in a given model and specifically attacked. In the next section, we present our new technique for automatic discovery of repeating geometric feature patterns at different levels of granularity and a compression scheme that avoids repeated descriptions.

3. A New Compression Scheme

3.1. Steps in the Compression Technique

The compression technique proceeds in the following steps (as illustrated in Figure 3):

Uniform format: Models are available in 3D Studio binary file formats (having extensions `.3ds` or `.max`), Alias|Wavefront format (extension `.obj`) and so on. These models consist of mesh geometry, connectivity, vertex normals, material definition and sometimes, texture maps. Across various file formats, similar information is captured in different data formats and data structures. We translate these data into our native data structure capturing all these elements of the models into a single unified format.

Pre-processing: In this step, we eliminate replicated vertices that have identical triples formed by xyz-coordinates, vertex normals and texture coordinates. We then decompose the input model into its connected components.

Discovery of Repeating Features: Discovery of repeating features is a hard problem. The main difficulty lies in automatic partitioning of the meshes to cut out those parts that represent repeating features. Neither the sizes nor the descriptions of the repeating features are known to us a priori. Based on the characteristics of the models of engineering class, we have evolved heuristic algorithms to discover repeating features at three levels of granularity – (a) connected component features, (b) sub-component features and (c) aggregate assemblies of component and sub-component level features. The discovered repeating features are organized in a "master geometry – instance transform" hierarchy. The first instance of a feature is labeled as DEF instance and repeating instances of the feature are labeled as USE instances.

Compact encoding: After the repeating features are discovered, the USE instances and the aggregate features in the model are compactly encoded using references to the DEF instances and transformations required to reconstruct the original features. The DEF instances that represent the first instances of the geometric features are compressed using geometry and connectivity compression algorithms.

3.2. Discovery of Repeating Features

We consider a polygon mesh model as an undirected graph $G = (V, E)$ consisting of vertices V and edges E. The *topological structure* $\mathcal{T}(\mathcal{G}')$ of a sub-graph $G' = (V', E')$ of G is defined by the adjacency relationship among its vertices. The *geometric realization* $\mathcal{G}(\mathcal{G}')$ of the sub-graph is determined by the positions of the vertices of V' in three-space. We define a *geometric feature in a 3D polygon mesh model* to be a pair $(\mathcal{T}, \mathcal{G})$ corresponding to a sub-graph of 3D polygonal mesh model.

A vector generated by a function of the kind $\mathcal{F} : (\mathcal{T}, \mathcal{G}) \rightarrow \mathbf{R}^k$ is called a *feature descriptor*. Here, $\mathcal{F}$ should be designed to have the following desirable properties:

- $\mathcal{F}$ must be efficiently computable.
- $\mathcal{F}$ should be *invariant* to some set of transformations $\mathbb{T}$ of the geometric realization $\mathcal{G}$ of the feature, that is, $\mathcal{F}(\mathcal{T}, \mathcal{G}) = \mathcal{F}(\mathbb{T}(\mathcal{T}, \mathcal{G}))$. This property is desirable

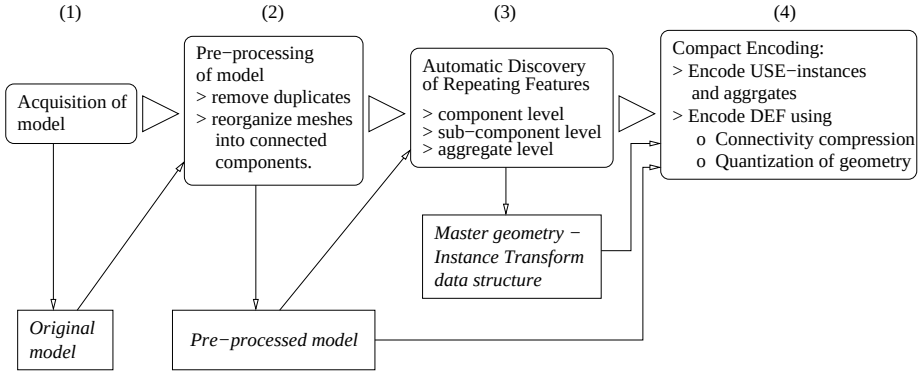


Figure 3. Steps in compression of large models: acquisition of the model, pre-processing, automatic discovery of repeating features and compact encoding.

to make the descriptor sensitive only to the shape and connectivity of the feature.

- $\mathcal{F}$ should be *descriptive* in the sense that the descriptors generated for identical features must correspond within some tolerance and on the other hand must be considerably different for non-identical features. For a "small" difference between two features the disparity between descriptors should be small, and for large difference between the features the disparity should have a large value.

- $\mathcal{F}$ should result in a descriptor of small dimensionality so that we have a compact description of the feature.

Matched Features: Two features f_1 and f_2 in a polygonal model O are said to be *matched* if $\|\mathcal{F}(f_1) - \mathcal{F}(f_2)\| \leq \epsilon$, where ϵ is scalar tolerance value. Note that these features could geometrically be in different positions, orientations and scale.

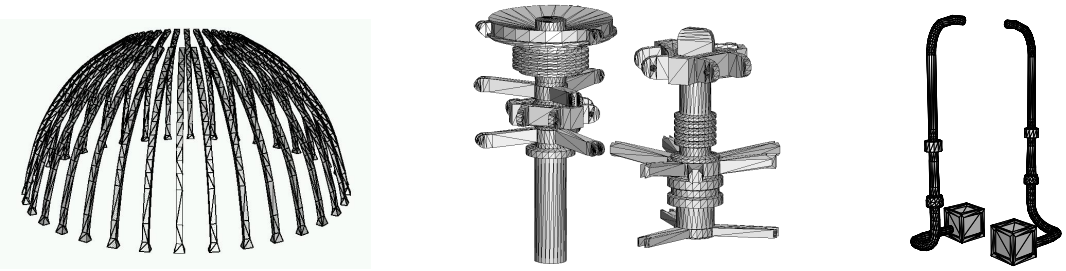
Features are viewed as sub-graphs of the model. Hence matching of features is treated as a *constrained graph isomorphism* problem. Constrained, since we seek to obtain vertex correspondence across graphs such that the geometric realization of the feature also matches under transformation $\mathbb{T}$. All algorithms we know for graph isomorphism are of exponential complexity [4]. However, constrained problems can be solved much faster in practice [6].

A brute force algorithm to discover all repeating features of all possible sizes greater than k vertices would be as follows. Consider the model O as a set of n vertices V. Construct the power set (the set of all subsets) of V having 2^n elements. From this set, remove all the elements which form infeasible features and those that have less than k vertices. Let the number of remaining elements be m. Carry out pairwise comparison and matching of the remaining elements to identify the features that repeat. This algorithm needs $O(2^n)$ operations to construct the power set and in the worst case, $O(m^2)$ operations to carry out pairwise matching! Each of these pairwise matching operations is again a very complex operation involving sub-graph isomorphism – an NP-complete problem.

3.3. Feature Discovery Algorithms

A practical solution to the problem of feature discovery cannot have exponential complexity if we are to deal with large models. Hence a heuristic algorithm must be designed to accelerate the process. Also, it is not practical to seek an optimal algorithm that will guarantee the discovery of all possible repeating features.

We describe algorithms for discovering repeating shape features at three different levels of granularity:
Connected components in polygon meshes: For example, in a mechanical assembly, nuts, bolts, fasteners, etc. repeat many times; in architectural models it is common to see structures having many identical parts (see example in Figure 4(a)). Connected component features are the largest connected sub-graphs in the given model.
Sub-component level structures: Many small features repeat within or across connected components in a model. For example, in a mechanical CAD model, a component representing a gear has many teeth, each corresponding to a sub-component level feature. Further, many engineering models are simplified for the purpose of visualization. During simplification, multiple components are merged together using boolean operations or vertex clustering [23] and a new triangulated composite mesh is generated giving a single connected component. Such a component has many repeating features at sub-component level (see example in Figure 4(b)).
Aggregates of repeating features: Groups of disjoint features are also found to repeat in many 3D models. Our technique discovers such macro-level aggregate features which



(a) (b) (c)

Figure 4. Features repeat at different granularities: (a) connected component: part of an architectural model has 36 instances of a feature having 68 vertices and 132 triangles, (b) sub-component: parts of a mechanical CAD model, and (c) aggregate: each assembly adds up to 18 connected components, 1112 triangles and 560 vertices.

may be composed from features of the above two types. Figure 4(c) shows an example.

3.3.1 Component Level Features

We first carry out discovery of repeated feature patterns at the level of connected components using the following steps: (a) reorganize the total set of polygons in the source model into a set of connected components, and (b) discovery of repeating component level features by pairwise matching and build a "master geometry – instance transform" hierarchy.

For pair-wise matching, we have used a simple and efficient technique based on principal component analysis (PCA) with suitable extensions. We compute an orthonormal basis in 3-space that describes the eccentricities of the connected component using the Hotelling transformation [9]. This basis is used as a pure rotation matrix to bring a component to a normalized (or canonical) orientation.

We take the list of vertices defining the mesh as a cluster of points $X = \{x_1, ..., x_n\}$ in $\mathbf{R}^3$ to obtain the mean

$$m = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and the covariance matrix,

$$C = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T - mm^T.$$

We then find eigenvectors and corresponding eigenvalues of C. The three normalized eigenvectors are used to construct a pure rotation matrix R and also the the oriented bounding box (OBB).

Let us denote the components as O_{c1}, O_{c2}, their mean values as m_1, m_2 and the orthogonal bases representing the respective eccentricities as R_1, R_2. If the two components do no match in their number of vertices and the dimensions of their OBBs then no further matching is attempted. Otherwise, we carry out a fuzzy (i.e. using numerical tolerances) comparison of positions of vertices across the components. If the geometry so aligned matches in at least 99.9% of the vertices, then we declare O_{c2} to be an instance of O_{c1} and also record the transformation composited as $T_{-m_1} \circ$ *Rot* $\circ T_{m_2}$ required to reconstruct O_{c2} from O_{c1}, where *Rot* $= R_2 \circ R_1^{-1}$. Matching vertices implies matching their positions, texture coordinates and vertex normals, if they are defined for the given model.

Overcoming limitations of PCA: While attempting to match two components using PCA, ambiguity can arise when the components have a completely symmetrical mass distribution, because the eigenvectors will be similar. Examples of such a case are: a cylinder, which has a symmetrical mass distribution about an axis and a sphere, which has symmetrical mass distribution about the centre of mass. This limitation of PCA is overcome by a minimization procedure reported by Novotni and Klein [19] to obtain the best rotation transformation for matching of components. We consider all possible rotations around the axis or centre point of symmetry and choose the rotation corresponding to the maximum match. For a single axis of symmetry, R is given by

$$R = \max_{R(\alpha)}(M(R(\alpha), O_{c1}, O_{c2}) | \phi \in [0, 2\pi])$$

where α denotes the angle of rotation around the axis of symmetry, $R(\alpha)$ the rotation and $M(R(\alpha), O_{c1}, O_{c2})$ the measure of match between the *aligned* components O_{c1} and O_{c2}. In case of spherical symmetry, the best alignment is obtained as

$$R = \max_{R(\phi,\alpha,\psi)}(M(R(\phi, \alpha, \psi), O_{c1}, O_{c2}))$$

where the angles (ϕ, α, ψ) denote Euler angles for parameterization of rotations in 3D and $\phi \in [0, 2\pi], \theta \in [0, \pi], \psi \in [0, 2\pi]$.

The rotation space is discretized uniformly. For objects where the mass is symmetrical around one of the principal axes (say, a cylinder), this is an efficient approach, but not in cases of spherical symmetry. The latter situation occurs rarely in practice.

3.3.2 Sub-component level features

We have developed a heuristic technique for discovering repeating sub-component features which uses the strategy of "growing" patterns bottom up from features at the lowest level of granularity – vertices.
We begin the growth of features bottom up, starting from identical vertices in the model. To obtain these starting points, we analyse the neighbourhood of each vertex and group those vertices that have topologically and geometrically identical neighbourhoods. We carry out an identical simultaneous breadth-first traversal around these starting points while verifying that the features being grown are indeed identical.

To enable effective grouping of vertices having identical neighbourhoods in a given model, we associate a *footprint*[2] with each vertex in the model, composed of the vertex properties invariant to rigid-body transformation. The footprint consists of the following four entities: (a) *Density*: $|N_v|$, the cardinality of N_v, the set of vertices connected to v. The elements of N_v are also called the *neighbourhood* of v and $|N_v|$ is also called the *degree* of v; (b) *Size*: L_v, average of the lengths of the edges connected to v; (c) *Curvature*[3]: D_v, the average of the dihedral angles of the faces meeting at the edges connected to vertex v; and (d) *A second order descriptor*, D_v^2, given by $\frac{1}{|N_v|} \sum D_j$, $j \in N_v$. A simple inner product distance measure has been constructed for determining the similarity or disparity between two given features in the space defined by the footprints. For engineering models this composite footprint has performed very well.

We have chosen to associate footprints only with vertices and not with higher order elements like edges or polygons because in polygon meshes the number of polygons and the number of edges are in multiples of the number of vertices. While this choice does not affect the asymptotic complexity of the algorithm, the savings realized are significant when working with large models.

This algorithm returns a set of patterns (sub-graphs) which correspond to features that repeat in the given model. Since the process of discovering the repeating patterns uses mesh connectivity for traversal, the matching patterns are identical in coordinates of the vertices (in the sense of rigid-body transformation) as well as the connectivity.

[2]We have borrowed the term *footprint* from the work of Barequet and Sharir on Partial Surface and Volume Matching in Three Dimensions [1]. However our definition and use of footprints are significantly different.

[3]This is only an approximate descriptor of the curvature.

3.3.3 Aggregate Features

After carrying out feature instance detection, many USE-instances in different meshes are found to have identical rigid body transformations. This *iso-transformation* set enables us to infer *repeating macro-level structures, i.e., part assemblies*. In order to intuitively understand how grouping of iso-transformation features means capturing macro-level features, we take a simple example. Consider an architectural model having many identical pillars, each consisting of an assembly of multiple component features. The discovery of features at the level of components would construct a DEF-USE hierarchy among identical components across the pillars. If components of the first pillar are marked as DEF instances, the components of the remaining pillars would be marked as corresponding USE instances with respective transformations. All USE components belonging to a pillar would have identical transformation associated them. Thus grouping of iso-transformation USE components yields assemblies.

As a side benefit of considerable value, our scheme also lets us automatically identify the common problem of erroneous replication of components in large models - two instances of a feature having the same rigid body transformation. It must also be noted that a USE-instance having an identity transformation associated with it is also a duplicate component, and must be removed. These are undesirable cases of coinciding geometries in the model. It is almost impossible to identify such cases visually. However, they manifest in interactive walk-throughs of such models in the form of unnecessary overhead and also as flickering facets.

3.4. Reconstruction

The decompression procedure for reconstruction of the original model from the compressed representation is simple and efficient. While decoding the aggregate features, the following steps are taken for the reconstruction of each USE instance: (i) obtain the centroid m_1 of the DEF instance, (ii) obtain the rotation matrix R from the quaternion component of the transformation and the position vector m_2 of the USE instance, (iii) transform the vertex positions and normals of the DEF instance by the composite transformation $\mathbb{T} = T_{-m_1} \circ R \circ T_{m_2}$.

Since each vertex in the model being reconstructed is visited exactly once, the complexity of decompression procedure is a linear function of the number of vertices – much faster than the compression algorithm.

4. Implementation and Analysis

4.1. Experimental Results

Our heuristic of looking for repeating features at the level of connected components results in discovery of largest connected features and yields large savings in storage. Table 1 shows the results of component level discovery on some representative models that we experimented with. The table is partitioned into three sections – (a) name of the model, (b) the count of components, vertices and triangles in the original model, (c) the count of DEF instances at component level discovered.

	#cmp. (orig)	#vtx (orig)	#tri (orig)	#cmp. (DEF)	#vtx (DEF)	#tri (DEF)
Capitl	2662	52606	87258	93	10347	19944
Coloss	1129	69868	135159	20	18912	38103
D-i-K	3726	295695	162590	848	44363	46165
Helcpt	976	105079	187929	480	76231	136372
Taj	375	65323	126453	45	28427	55238

Table 1. Large number of repeating components are detected, thereby yielding savings in the representation of the model.

Consecutive steps in our compression scheme shown in Figure 3 result in successive improvements. Table 2 shows a comparison of file sizes of five representative large models (see Figure 1) on successive application of the different steps. It is clear that connectivity compression alone ("EB alone" column denotes the file size on application of Edgebreaker algorithm [22]) cannot achieve this level of compression. Further it may be noted that the connectivity compression algorithm is unusable on Helicopter and the Diwan-i-Khaas models because they contain non-manifold components. Note the high compression ratios[4] achieved on these models.

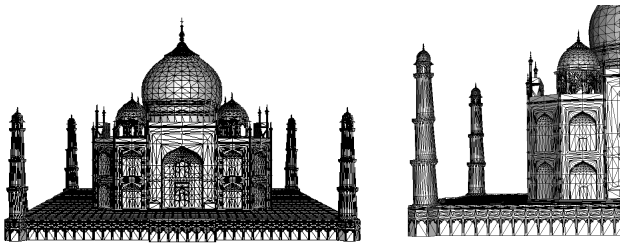
Discovery of aggregate features yields an incremental improvement. This is expected since the saving is only by avoiding the repeated representation of a rigid body transformation.

	Orig size	gzip only	EB only	comp, subco level	Aggr level	Aggr +[EB] +gzip	CR
Cap	1.8M	875K	1.1M	403K	395K	134K	0.92
Col	2.5M	2.1M	1.3M	516K	497K	217K	0.91
DiK	10.0M	2.8M	—	1.9M	1.8M	603K	0.94
Hel	4.9M	1.9M	—	1.6M	1.6M	879K	0.82
Taj	764K	1.1M	1.1M	630K	629K	196K	0.74

Table 2. Comparison of performance of stages in the compression scheme on large models of architectural and engineering class.

[4]CR = 1 – (compressed size / original size)

In this encoding scheme, lossiness in reconstruction is introduced due to the following reasons: (a) the quantization of vertex normals, and (b) matching, with a numerical tolerance, of vertices across features. Hence features related by DEF-USE relationship are identical only within the tolerance value. One instance of lossiness due to tolerance is illustrated in Figure 5 below. This loss of information can be controlled by using finer tolerance values.



(a) (b)

Figure 5. Loss in reconstruction: (a) the reconstructed model of Taj Mahal, (b) zooming in near the towers in the left side reveals that they have been slightly distorted.

4.2. Complexity Analysis and Acceleration

Component level features: A naive implementation involving exhaustive pair wise matching for discovering identical connected component level features has a complexity of $O(n^2)$, where n is the number of connected components in the given model. In most engineering models, n is large enough to warrant the use of acceleration techniques. We have used using geometric hashing [29] of components using the number of vertices and triangles as the hash key for acelleration in our algorithm. Before performing detailed matching we also check if the dimensions of the OBBs match. While the worst case complexity even after hashing is $O(n^2)$, for most practical situations the benefit is large.

Matching two aligned components A and B, each having m vertices, is a procedure of $O(m)$ complexity. Finding the correspondence between the first pair of vertices across the components needs $O(m)$ effort. The correspondence between the rest of the vertices is established by carrying out identical spanning tree explorations in the connectivity graphs of the mesh components while verifying geometric match – another $O(m)$ step. The assumption used in this case is that identical components will have identical geometric shape as well as an identical connectivity. When it is not possible to hold this assumption, a more complex procedure must be used. In case of the models which have texture coordinates associated with vertices, we have seen that after alignment of two components, the test for matching is a process of $O(m^2)$ complexity. This is because in such models, to capture different texture mapping coordinates at the

shared vertex positions, the vertices are repeated. Hence, a single identical spanning tree traversal is not possible across the connectivities of the components.

Sub-component level features: During the discovery of sub-component level features, the computation of footprints is an $O(n)$ operation where n is the number of vertices in the model. On computation of a footprint, the vertex index is hashed into a table using the value of the computed footprint. The hashing function is based on two keys from the footprint vector: $|N_v|$ and L_v. The buckets formed by hashing the vertices give us the equivalence classes of vertices representing starting points for growth of repeating features.

The complexity of the growth phase varies with the characteristics of the given model. If an equivalence class of vertices has k elements, each forming a seed for growth of features, the analysis proceeds as follows: The neighbourhoods grown around each element must be matched to confirm a match or reject it. This process can take place up to $O(k^2)$ times. Matching any two neighbourhoods having m elements has a linear complexity.

The worst case situation occurs when all n vertices in the model are classified in the same equivalence class and each vertex has a degree of $O(n)$. In this case, the growth phase has a complexity of $O(n^3)$. Thus this algorithm is bounded by a polynomial time complexity. A configuration that comes closest to the worst case is a mesh having a uniform connectivity and geometry around each vertex, a uniformly tessellated sphere.

5. Applicability in Cooperative Computing

In comparison to all of the earlier 3D compression techniques our scheme is far more suited for cooperative computing applications. This is because of the following distinctive features:

Works on general polygonal models: As was mentioned earlier, polygonal mesh representation is the most popular modeling scheme that has maximum interoperability. Our scheme works extremely well on polygon mesh models. It can work on general polygonal models, and is not restricted to triangulated models or just to manifolds. Although the sub-component feature discovery algorithm uses traversal of mesh graphs, it does not expect connectivity representing manifold topology. Further, it has the unique added capability to handle modeling defects so commonly present in the engineering class of models.

Achieves high compression by exploiting redundancy present in engineering models: Our compression method not only uses special a priori knowledge of properties of the models of engineering class – repeating shapes, large triangles, sharp corners, edges etc., but also "learns" more through the discovery of repeating features at different levels of granularity. This special capability enables the method to achieve high compression, much higher than what can be achieved by the algorithms reported earlier. The ability to learn also makes it more widely applicable across many 3D models where features are likely to repeat.

Incorporates best developments in single component compression: Large engineering models such as aircraft designs, on one hand, have many repeating geometric features, and on the other hand, there are also some large and complex components that have no repeating features. Such components must be compressed using connectivity and geometry compression algorithms developed earlier. For all DEF instance encoding we can incorporate the best developments in the area of connectivity compression research and thus obtain far better compression ratios than those reported in the literature. Table 2 shows clear evidence of this.

Simple and fast decompression: Compression of the largest model we experimented with required about four minutes (234.82 seconds), including the time required to load the uncompressed representation. However decompression of the same model required only 2.23 seconds. This is so because of the linear time complexity of the decompression algorithm. Further, the reconstruction algorithm is very simple to implement and does not involve heavy memory or computation requirements. These properties make the scheme highly suitable for implementation on simple desktops and other computing devices, enabling ubiquitous access to shared repositories of large engineering models.

Low decompression latency: The compressed format has DEF instance features appearing before the corresponding USE instances. This enables the decompression procedure to make the elements of the 3D model accessible, without any delay, while it processes the compressed representation. Thus the latency of the decompression algorithm for our scheme is very low.

Selective compression capability: In a cooperative engineering design environment, design activities are most likely to be shared at component and sub-assembly levels. Our scheme addresses compression precisely at these levels and hence automatically enables selective compression.

Control over lossiness: As already mentioned, loss can be controlled by compromising compression ratio with a suitable increase in the tolerance factor to which feature matching is carried out.

6. Conclusions

We have presented a new 3D compression scheme for very large engineering models with automatic discovery of repeating features at its core. We have shown why such a scheme is highly suited for cooperative computing applications. The test results from our implementation of this scheme on a number of large models including those that

are available on the net have shown excellent compression performance.

The fundamental contribution of automatically discovering similar shape features in a large 3D model has ample scope to be applied to the other 3D geometry handling processes of healing, simplification and progressive transmission.

The process of discovering repeated features is computationally complex. Hence the execution time for compression is much longer than the decompression procedure which has a linear complexity. However, the compression algorithm is highly parallel in nature and hence in a cooperative computing situation, the computations can be easily distributed. Detailed matching of a pair of features is completely independent of matching some other pair. This independence of processing tasks is one major potential for parallelisation of the algorithm. The discovery of repeating features at a particular level of granularity can therefore be distributed across computers on the network. The speed-up of the parallel implementation would be linear.

Acknowledgements: The models of Capitol building, Colosseum and Taj Mahal were downloaded from www.3dcafe.com. Diwan-i-Khaas model is from Visions Multimedia Visualization Studio and the Helicopter model was downloaded from the Avalon 3D archive (avalon.viewpoint.com).

References

[1] G. Barequet and M. Sharir. Partial surface and volume matching in three dimensions. *IEEE PAMI*, 19(9):929–948, 1997.

[2] CoCreate. OneSpace: Virtual conference room for collaborative CAD. http://www.CoCreate.com, 2001.

[3] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangle meshes. In *IEEE Visualization 99*, pages 67–72, 1999.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[5] B. Curless and M. Levoy. A volumentric method for building complex models from range images. In *SIGGRAPH 96*, pages 303–312, August 1996.

[6] K. Deb. *Optimization for Engineering Design: Algorithms and Examples*. Prentice-Hall of India, New Delhi, 1996.

[7] M. Deering. Geometry compression. In *SIGGRAPH 95*, pages 13–22, 1995.

[8] Discreet (Autodesk Inc.). 3DStudio MAX R4. http://www.discreet.com, 2000.

[9] R. Gonzalez and R. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.

[10] A. Guiziec, F. Bossen, G. Taubin, and C. Silva. Efficient compression of non-manifold polygonal meshes. In *IEEE Visualization 1999*, pages 73–80, 1999.

[11] M. Isenbueg and J. Snoeyink. Face Fixer: Compressing polygon meshes with properties. In *SIGGRAPH 2000*, pages 263–270, 2000.

[12] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH 2000*, pages 279–286, 2000.

[13] A. Khodakovsky, P. Schroeder, and W. Sweldens. Progressive geometry compression. In *SIGGRAPH 2000*, pages 271–278, 2000.

[14] D. King and J. Rossignac. Connectivity compression irregular quadrilateral meshes. Technical Report GIT-GVU-99-36, GVU Center, Georgia Tech., Atlanta, USA, 1999.

[15] D. King and J. Rossignac. Optimal bit allocation in compressed 3D models. *Computational Geometry*, 14:91–118, 1999.

[16] M. Levoy, K. Pulli, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, B. Curless, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D scanning of large statues. In *SIGGRAPH 2000*, pages 131–144, 2000.

[17] P. Lindtrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH 96*, pages 109–118, 1996.

[18] W. E. Lorenson and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH 87*, pages 163–169, 1987.

[19] M. Novotni and R. Klein. A geometric approach to 3D object comparison. In *Proceedings of International Conference on Shape Modelling and Applications (SMI2001)*, May 2001.

[20] R. Pajarola and J. Rossignac. Compressed progressive meshes. Technical Report GIT-GVU-99-05, GVU Center, Georgia Tech., Atlanta, USA, 1999.

[21] M. A. Rosenman and J. S. Gero. Modelling multiple views of design objects in a collaborative CAD environment. *Computer-aided Design*, 28(3):193–205, 1996.

[22] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January-March 1998.

[23] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.

[24] A. Stork and U. Jasnoch. A collaborative engineering environment. In J. Rossignac, editor, *First GVU/NIST Workshop on Collaborative Design (TeamCAD'97)*. Georgia Institute of Technology, Atlanta, 1997.

[25] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH 98*, pages 123–132, 1998.

[26] G. Taubin and J. Rossignac. Geometry compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

[27] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceeding of Graphics Interface 98*, June 1998.

[28] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH 94*, pages 311–318, 1994.

[29] H. J. Wolfson and I. Rigoutsos. Geometric hashing: An introduction. *IEEE Computational Science & Engineering*, pages 10–21, October-December 1997.