

Compression of Polygon Meshes of Engineering Models using Discovery of Repeating Geometric Features

Dinesh Shikhare and S. P. Mudur

National Centre for Software Technology, Gulmohar Cross Rd. 9, Juhu, Mumbai 400049, India.

{dinesh|mudur}@ncst.ernet.in

Abstract

We present a new geometry compression technique particularly suitable for 3D mesh models of engineering class – architectural models, machine plants, factories, etc. We observe that such models have a number of repeating features at various levels of granularity. In most of the widely available models in this class, the geometric description of such features are also repeated, of course in their positions and orientations.

The technique described in this paper automatically detects the repetition of features and also repetition of groups of features and then compactly encodes the geometry using a “master geometry – instance transform” heirarchy. The master geometry itself can be encoded using the most suitable geometry compression algorithm developed earlier and thus our work is complementary to all the earlier work in this field.

We have implemented this technique and carried out tests on a number of representative large 3D models, including many that are freely available on the net. The results are excellent giving us high compression ratios for these models.

1 Introduction

Very large 3D models documenting architectural designs, heritage monuments, chemical plants and mechanical CAD designs are being increasingly deployed in various applications involving interactive visualization and Internet based access. Very often the complexity of these 3D models far exceeds the limits of what can be quickly downloaded at popular connection speeds and what can be easily stored and rendered for interactive exploration on personal desktops. Compression schemes for such large 3D geometric models have thus become a subject of intense study in recent years. A brief review of these schemes is presented in the next section. A new 3D geometry compression technique, devised by the authors is presented after that and this is followed by the results from a straight forward implementation of this new technique.

While a number of mathematical representations such as quadrics, bi-polynomials, NURBS, sweeps, Boolean operations etc. may be used during the initial creation of the 3D models, the final evaluated and stored representation is usually in the form of polygonal meshes, most commonly, triangle meshes. Many compression techniques have therefore focused on such triangle mesh representations. Quite naturally, specific compression schemes are best suited to specific classes of such complex 3D models with well distinguishable shape characteristics.

There have been a few attempts to classify 3D models. These are normally based on some formulation for measuring the complexity of the shapes and their representations. An early attempt is by Forrest [11] in which he classifies geometric models along three axes: geometric complexity (lines, planes, curves, surfaces, etc.), combinatorial complexity (number of components, edges, faces, etc.) and dimensional complexity (2D, 2.5D, 3D). More recent attempts have tried to relate polygonal representations to *shape factors* [39, 2, 12, 19, 26]. The shape factor addresses the complexity measure along the geometric complexity dimension. For the work being reported in this paper, Forrest’s formulation is more applicable. This is because earlier methods address compression of the inherent shape complexity in singleton surfaces, where as our scheme addresses compression of geometric models with high combinatorial complexity. Some of the large 3D geometric models presently being worked upon in various computer graphics research laboratories around the world are shown in Figure 1 positioned in Forrest’s model classification space.

The new 3D geometry compression technique that we present in this paper is particularly suitable for large digital 3D models of engineering class – architectural complexes, heritage monuments, factories, plants, etc. Unlike digital models of natural shapes such as terrains, anatomies, sculptures and so on, such engineering models invariably have a number of repeated components, e.g., windows, pillars, fixtures, etc. in different orientations and positions. Typically such models

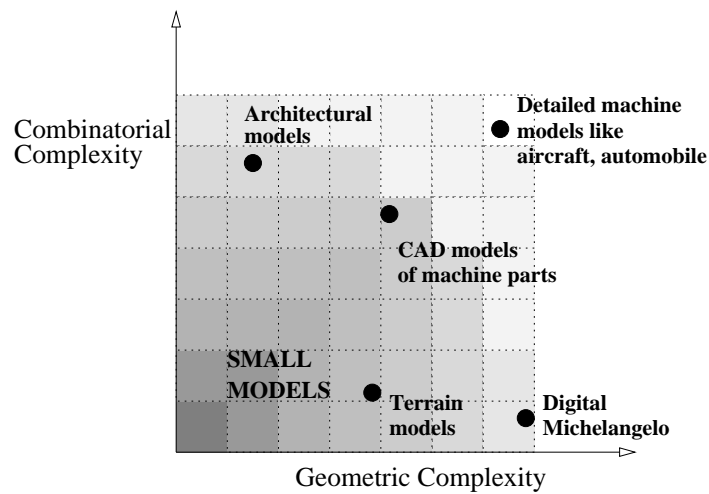


Figure 1: Classification of 3D models: Geometric complexity versus Combinatorial complexity

consist of a large number of small polygonal meshes, each consisting of up to a few hundred triangles. In Figure 1, these models would be typically positioned in the upper right triangular region.

Our technique seeks to detect repeating shape features in the model so that multiple instances of a shape or a group of shapes need not be stored in all details multiple times. Although most modeling languages like VRML [7] and modeling tools like 3DStudio MAX [21] have facilities to model repeating geometries as instances of a prototype, in practice, one does in fact encounter huge models having no instance structure information. Re-engineering such instancing relationships manually is a very laborious, expensive and error prone task. The fundamental component of our technique is the automatic identification of repetitive geometric features in a large geometric model at three different levels of granularity – features within and across component shapes, features at the granularity of component shapes and also at a macro-level of groups of features.

The model is then compactly encoded using a “master geometry – instance transform” hierarchy. The master geometry itself can be encoded using the most suitable geometry compression algorithms developed earlier and thus our work is complementary to all of the earlier work in this field. To the best of our knowledge, ours is the first significant attempt to address compression of geometric models with high combinatorial complexity.

1.1 Summary of Contributions

Our work represents the following significant contributions:

1. In this paper we pioneer the idea of compressing geometry of 3D polygonal mesh models by removing the redundancy in the representation of repeating geometric feature patterns.
2. We present innovative use of pattern matching techniques for the purpose of automatic discovery of repeating feature patterns in 3D polygonal mesh models.
3. Some simple ideas have been introduced for efficient implementation of our algorithm for compression using discovery of repeating feature patterns.
4. Our compression scheme can incorporate the best results achieved in the area of connectivity compression of polygon meshes, thereby always achieving compression performances at least as good as those reported in the literature [22, 25, 15]. In fact for engineering models, our technique gives large benefits which cannot be realised by simply using the connectivity compression algorithms.

The rest of this paper is organized as follows. In the next section, we briefly outline the previous art in the area of compression of 3D polygon mesh models and describe how the previously developed techniques do not fully exploit the redundancies

in engineering models. In section 3, we describe in detail the different schemes we have designed for the discovery of geometric feature patterns that repeat. We also analyse the benefits of using the discovery of repeating features for compression of large 3D models. In the following section 4 we present the results of our experiments with some representative engineering models. Finally we conclude with other potential uses of the techniques for discovery of repeating geometric features in polygon mesh models.

2 Previous Work

This section provides a brief review of the different schemes devised for compressing 3D polygonal geometric models and their applicability to large models of the engineering class.

2.1 Some Preliminary Definitions

Polygon mesh models are typically defined in terms of (a) *geometry* – the coordinate values of vertices of the meshes making up the model, (b) *connectivity* – the relationship among the vertices which defines the polygonal faces of the mesh (also called as *topology* of the mesh), and (c) *attributes* - such as color, normal and texture coordinates at the vertices.

Other information such as texture images and material properties are also commonly present as a part of the models, usually associated with meshes or groups of meshes.

A *polygonal mesh model* O consists of a set S of polygon meshes and associated non-geometric properties. 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 $X = \{x_i \in R^3\}$. An edge e is represented as a pair (v_1, v_2) of references into the list of vertices. A polygon p is represented as a sequence (v_1, v_2, \dots, v_k) of references into the list of vertices. A *triangle mesh* is a special case having all triangular polygons. Generally in the actual representation, edges are implied and not explicitly stored.

A mesh having each edge shared by at most two polygons and at least one polygon forms a *manifold*. A mesh having an edge shared by more than two polygons represents a *non-manifold*. A mesh model representing a manifold surface is *closed* if all edges are shared by 2 polygons, otherwise it is *open*.

In a mesh model O , we call two polygons as *neighbouring polygons* if they share an edge. There exists a *path* between polygons p_i and p_j if there is a sequence of neighbouring polygons $p_i, p_1, p_2, \dots, p_j$. A 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 simple labelling algorithm.

When deriving a compact encoding for 3D models, there are three processes that work on a polygonal mesh model O and are in some sense related: compression, simplification and progressive disclosure.

Compression is the process of deriving a new encoding for a given polygonal mesh model O such that the number of bits needed in the new encoding is much smaller than the number of bits needed for the uncompressed representation. If $U(O)$ and $C(O)$ denote the number of bits for the uncompressed and the compressed versions respectively, then the *compression ratio* is defined as

$$CR = \frac{U(O) - C(O)}{U(O)}.$$

Compression is said to be *lossy* if the decompression process cannot give back the original polygon mesh model exactly.

Simplification is the process of deriving a new representation O' , often referred to as an imposter, for a given polygonal mesh model O by which O' becomes simpler to deal with normally for rendering purposes. This is usually in the form of lesser number of meshes or polygons or vertices and the elimination of fine detail in geometry and associated information, etc. Clearly simplification is a kind of lossy compression. A comprehensive review of simplification techniques may be found in [18].

Progressive disclosure is the process of deriving a new representation of a given polygonal mesh model O , which enables one to transmit a coarse representation of the model first and subsequently transmit the details to refine it. Compression strategies have been used in the progressive representation of models to also make the entire transmission data compact. Many groups have developed compression schemes incorporating progressive disclosure capabilities [6, 32, 35, 36]. The schemes that combine compression and progressive disclosure always have a trade-off between compression and the additional data needed for progressive representation of the models.

2.2 Geometry Compression Strategies

Most of the geometry compression strategies have two distinct components:

1. *Compression of topology or the connectivity information.* This is usually done by reducing repeated references to vertices that are shared by many polygons/triangles.
2. *Compression of geometric data.* This is achieved by reducing the precision with which coordinate values, normal vector components, etc. are represented and by removing highly detailed features using techniques from signal processing.

2.2.1 Connectivity Compression

A large body of geometry compression research has concentrated on clever encoding of the connectivity among the vertices in the mesh. Typically, the number of triangles in a mesh is roughly twice the number of vertices and each vertex is referenced in 5 to 7 triangles, which means that large part of the representation of the model is the definition of connectivity among the vertices. Schemes that minimize repeated references to vertices obviously result in compression.

Almost all of the connectivity compression techniques [16, 22, 33, 37, 38] encode triangle/polygon connectivity in a lossless manner. Triangle-triangle adjacency is exploited by arriving at a suitable ordering of the triangles, such as triangle strips and triangle fans [42]. Recent techniques [38, 16] construct spiraling triangle spanning trees in which vertices are labeled in the order in which they are first encountered in the triangle tree. By defining a small set of operators to encode the traversal over the triangles high compression ratios have been achieved for connectivity encoding, typically a few bits per vertex [33].

Most of the above referenced compression schemes concentrate on triangle mesh models representing manifold surfaces. A number of researchers have devised and extended compression algorithms to handle non-manifold surfaces and more general polygon mesh models [22, 25, 15].

2.2.2 Geometric Data Compression

Deering [9] used quantization of coordinate values from 32 bits to 16 bits, a 9-bit index into a global list of values for vertex normals and a 15-bit representation of color values. Although this strategy of reducing data is blatantly lossy, it is usable as long as the results for interactive visualization are not deteriorated.

Use of Predictors: Quantization of coordinates and color values leads to an indiscriminate truncation of the accuracy regardless of the features in the geometric models. Predictive encoders perform much better by taking advantage of the coherence in the data to predict the value of the next element in the sequence. Predictors are designed so that most of the error values are small and only a few are large. These error values can then be quantized effectively and then entropy coded for compact representation. Taubin and Rossignac [37] use linear predictors; Touma and Gotsman [38] use the *parallelogram rule* to predict subsequent vertex positions in triangle sequences; and Pajarola and Rossignac [32] have introduced a *butterfly predictor* scheme.

Signal Processing Techniques: Signal processing based techniques which have been in use for surface fairing [34] and multiresolution analysis [17] are based on the construction of a set of basis functions for decomposition of a triangle mesh into signals. The low frequency components in the signals correspond to smooth features and high frequency components correspond to discontinuities such as creases, folds and corners. Retaining just a few frequency components suffices to capture the overall perceptual shape of the model. This compression scheme is analogous to the JPEG image compression algorithm [29]. The *spectral compression* effort of Karni and Gotsman [23] and wavelet based technique of Khodakovsky et al [24] are examples of this.

2.3 Applicability to Large Engineering 3D Models

Almost all of the above research has concentrated on the compression of large models having a few connected components – often a single mesh of a complex surface with continuously varying curvature – formed by a dense collection of a large number of small polygons (mostly triangles). The graph traversal techniques are best suited when we have very long traversals. Longer the traversal sequence, smaller is the average cost of representation. In most models such as architectural, urban planning, entertainment, etc., meshes have associated texture maps. In order to capture different texture mapping

coordinates at the shared vertices, the vertices are repeated. Due to this condition of the data, reducing repeated references to vertices and creating long graph traversals does not generally become possible.

Bajaj et al [3] present compression of CAD models. However, their technique is based on encoding the connectivity using breadth-first traversal. Their approach is not much different from the other techniques and will have the same limitations as described earlier.

We must also note here, that both predictive encoding and the signal processing approach work best for smooth surfaces with dense meshes of small triangles. Signal processing techniques also rely heavily on the specific connectivity of the mesh for deriving the basis functions [24]. Very large 3D geometric models of the engineering class usually have a large number of meshes, with small numbers of large triangles, often with arbitrary connectivity. The architectural and mechanical CAD models typically have many non-smooth meshes making these methods not the best suited.

3 Discovery of Repeating Feature Patterns for Compression

The basic approach in our compression scheme is to look for redundancy in a geometric model in the form of repeating geometry of near identical shape features and then eliminate this redundancy by suitably encoding the component shapes and their repeating instances.

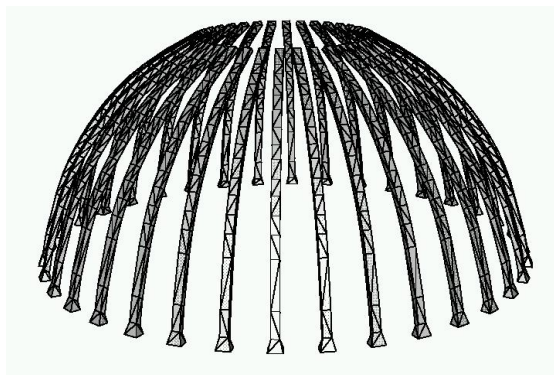


Figure 2: Part of the Capitol building model having 36 instances of a shape feature consisting of 68 vertices and 132 triangles. Each vertex is defined with x, y, z coordinates, a vertex normal and texture coordinates.

The shape features we seek are at different levels of granularity:

- *Connected components in polygon meshes*: Engineering models typically have many components repeating in the form of identical meshes appearing in different positions and orientations. For example, in a machine plant model, nuts, bolts, fasteners, etc. repeat many times; in architectural models it is common to see structures having many identical parts (see example in Figure 2). We denote the set of such features discovered in a model as features of type F_1 .
- *Subsets of connected component polygon meshes*: Often these models have features repeating within or across meshes. For example, in a mechanical CAD model, a mesh representing a gear has many teeth. Each of the teeth corresponds to a feature of this kind. We denote the set of such features discovered in a model as features of type F_2 . (See example in Figure 3).
- *Aggregates of repeating features*: Groups of disjoint features are also found to repeat in many 3D models. Our algorithm discovers such macro-level aggregate features which may be composed from features of type F_1 and F_2 . Examples of such macro-level features are often found in architectural models where there are many pillars, where each pillar is made of multiple features.

Our algorithm automatically detects sets of features F (which may be of type F_1, F_2 and their aggregates) that repeat with a rigid body transformations within the model O . We also derive the transformation $\tau : R^3 \rightarrow R^3$, for each repeating instance of a feature to enable compact encoding in the form of “master geometry – instance transform” hierarchy.

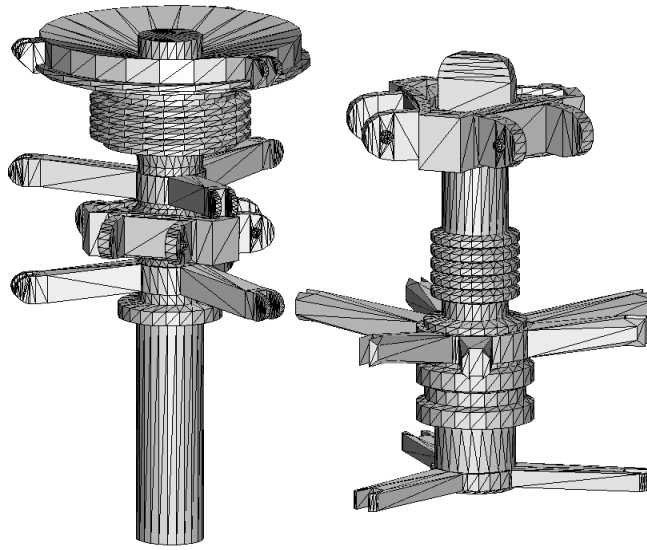


Figure 3: Repeating feature patterns in mechanical CAD models. Features repeat at the granularity of subsets of meshes. These features repeat within a mesh and also across the meshes.

In a given scene, one does not know which repeating feature patterns to expect. Hence, the traditional approach of matching 3D objects [43, 4, 40, 31, 30] by maintaining a dictionary of features and then looking for those features in the given model is not applicable for our work. Our goal is to *discover* repeating feature patterns in a polygon mesh model automatically without using a knowledge base of known features. We have created a simple and generic algorithm for discovery of repeated features.

Here is a brief sketch of our algorithm for compression of 3D models using discovery of repeating geometric patterns:

1. *Normalization*: Since the geometric features may occur in the model in any position and at any orientation, we obtain a rotation and translation invariant¹ representation for facilitating matching of shape features. A desirable characteristic of normalization process is that it should be able generate an index into a hash table for immediate grouping of the identical features. Hashing provides equivalence classes of features of the model which can be good starting points for discovery of complete features that repeat.
2. *Growth of repeating features*: Within each equivalence class obtained as mentioned above, a simultaneous growth of features is attempted around the neighbourhood of each member of a class. With every step in growth, a geometric match is carried out to *verify* the pattern. During verification we also obtain the transformation τ for instance transform representation and subsequent reconstruction of the original position and orientation of the feature pattern.
3. *Construction of macro-level features*: A set of disconnected features that repeat across the model have the same transformation associated across the elements of the set. Aggregation of features having identical transformations gives us macro-level features that repeat. This aggregation can be carried out at multiple levels.
4. *Compact encoding*: Once hierarchical relationships across the repeating features are found, a “master geometry – instance transform” information is encoded compactly.

This overall scheme is very generic in nature and can cater to various kinds of models. The specific use of this scheme can be customized to capture special features in a given class of models. In the following subsections, we describe in detail the techniques used for detecting repeating feature patterns.

3.1 Feature Patterns at the Level of Connected Components

In most engineering models we observed that there are a large number of small to medium sized connected components, each having upto a few hundred polygons on an average. The examples in Figure 4 show various models having many

¹Note that it is possible to extend this invariance to non-uniform scaling, reflection and shear.

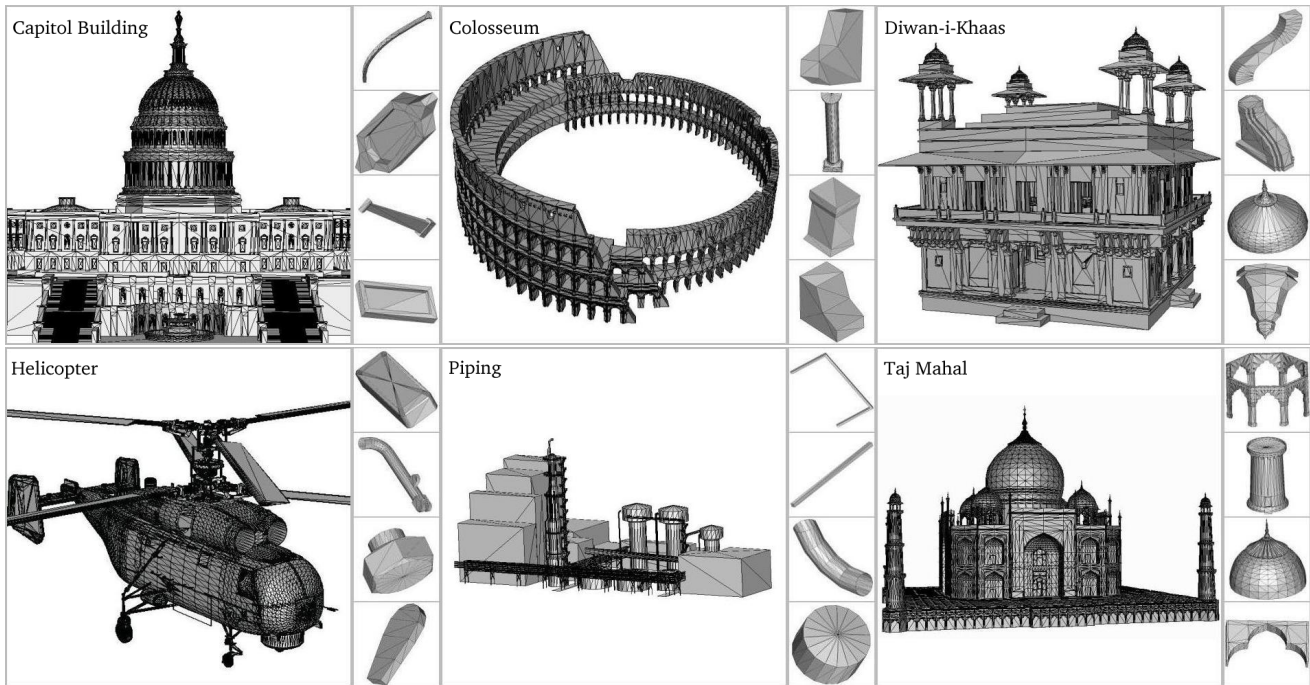


Figure 4: Repeating feature patterns in large engineering models. These models have multiple repeating features, many occurring at the granularity of connected components.

connected components repeating in different positions and orientations. For the purpose of compression, it is best to detect redundancies at a granularity as large as possible. Hence as a heuristic decision derived especially for the engineering models, we carry out discovery of repeated feature patterns at the level of connected components in the given model.

For models having repeating feature patterns at the granularity of connected components, we use the following steps for compression:

1. split the model into connected components and pre-process the the model to remove any erroneously replicated geometry,
2. carry out discovery of repeating feature patterns at the connected component level (described in detail in 3.1.1),
3. build a hierarchy of “master geometry – instance transform” among the first occurrences and repeating occurrences of the feature patterns.
4. across all the first occurrences of the patterns at the level of connected components, carry out sub-mesh level discovery of repeating feature patterns for compact encoding of features at a smaller granularity (as described in 3.2).
5. construct aggregates of features by grouping repeated occurrences of features having identical transformations (described in 3.3).

3.1.1 Detecting Repeating Connected Components

Matching polygon meshes has been an active area of research in the recent years [43, 8, 4, 40, 31, 30]. Many sophisticated algorithms have been developed for robust matching of similar shapes. In our implementation we have used a simple technique based on principal component analysis (PCA) that uses Hotelling transform [13] to determine a *normalized orientation* of a given polygonal mesh for matching geometry. We have extended PCA to overcome the limitation of ambiguous result in cases of completely symmetric mass distribution. While this is very simple technique, it has given us very good results as we will see later in this paper.

Pre-processing: In an actual implementation of this scheme, we also need a preprocessing stage that heals the geometry in the model. Healing involves removal of erroneous and invalid geometric entities, say, spurious or duplicated vertices, edges and faces. This is required because many models have these problems in data, resulting in avoidable inefficiency and

errors in further analysis and use of these models. This is a one time operation and need not be considered as part of the compression scheme.

The previous work on healing, also sometimes referred to as CAD-data repair [5, 10, 27] addresses the mismatch between the geometry created or exported by modelers and the specific requirements imposed by the end-applications. Almost all geometric models created using any of the standard design/modeling tools require healing to handle this mismatch.

Removal of Erroneously Repeated Geometry: In almost all the models that we have used for testing our implementation, we have found that the models had avoidable duplicates or multiples of geometric data, for example, many vertices with identical triples (xyz -coordinates, vertex-normal, texture-coordinates). In our pre-processing stage, we remove such redundant vertices in the data. PCA is very sensitive to spuriously duplicated vertices. Hence this stage of pre-processing proved extremely important for the models we worked with. We have also found in some of the models, overlapping polygons, dangling edges and missing textures. Identification and repair of such artifacts is a complex problem and more automated tools need to be developed to heal such complex problems.

Splitting Meshes into Connected Components: The source geometry has meshes which may contain multiple connected components or parts of multiple connected components. At times the source polygonal mesh may contain only a part of a connected component. This happens because most modellers tend to group all the polygons having same attributes like material and texture into a single mesh. Hence it is common to come across an architectural model, for example, with a few pillars and some unconnected walls in the model forming a single mesh. We reorganize the total set of polygons in the source model into a new set of meshes such that each mesh in the reorganized model is a connected component. This is done using a straightforward recursive labelling algorithm of $O(n)$ complexity.

While this intermediate representation may be sometimes slightly verbose as compared to the original collection of polygons, it is necessary in the subsequent steps of our technique for detecting near identical components. Splitting the source geometric description into smaller connected components also often provides a finer granularity for detection of repeating components.

Normalized Orientation: We first obtain a normalized orientation for a given mesh. An orthonormal basis in 3-space that describes the eccentricities of the mesh irrespective of the orientation is computed using the Hotelling transformation. Using this basis as a pure rotation matrix, a mesh is brought to a normalized orientation.

The Hotelling transformation is based on statistical properties of vector representations. We take the list of vertices defining the mesh as a cluster of points $X = \{x_1, \dots, x_n\}$ for the following steps.

1. Determine the centroid of the cluster of the points:

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

2. Obtain the covariance matrix, C , as:

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

This is a symmetric matrix and its eigenvectors are mutually orthogonal.

3. Determine eigenvectors and corresponding eigenvalues of the covariance matrix. Sort the eigenvectors in increasing order of eigenvalues. Normalize the eigenvectors.
4. Use the three normalized eigenvectors to construct a pure rotation matrix R .

R is the orthonormal basis that describes the eccentricity of the mesh. If T_{-m} represents a translation by vector $-m$ computed in Step 1 above, the composite transform $T_{-m} \circ R$ places the mesh in a normalized orientation at the origin ².

The orthonormal basis determined using this method is quite sensitive to any stray geometry that may be present in the model. Hence we have found it necessary to clean up the meshes of any unused vertices and duplicated triangles.

Our Implementation for Matching Components: We use the simple technique described below to determine the extent of match between two connected components.

²It has been used earlier by Gottschalk et al [14] in their work to obtain an oriented bounding box (OBB) for a given set of points. On obtaining an oriented basis for a given mesh, the extremal vertices along each of the vectors in the basis give the size of the oriented bounding box.

1. Obtain centroids, orthonormal basis for the two components and the dimensions of their OBBs as described in the previous subsection. Let us denote the components as O_{c1} , O_{c2} , their centroids as m_1 , m_2 and the orthogonal bases representing their respective eccentricities as R_1 , R_2 .
2. If the dimensions of the OBBs corresponding to the two meshes do not match then there is no match possible.
3. Match the number of vertices and number of triangles in each of the components. If the difference is greater than 5%, then there is no match. We do not seek exact equality in number of vertices and triangles across components. This is to accommodate the existence of commonly found stray geometry that might escape the pre-processing phase of our scheme.
4. We obtain the transformation needed to align mesh O_{c1} with mesh O_{c2} as a composition of the following sequence of transformations:
 - (a) translate by $-m_1$
 - (b) rotate by using the transformation:

$$Rot := R_2 \circ R_1^{-1}$$
 - (c) translate by m_2

Using this transformation, we align the two meshes for matching geometry.

5. We then carry out a fuzzy comparison of positions of vertices and edges across the meshes. If the geometry so aligned matches again up to 95%, then we declare mesh O_{c2} to be an instance of mesh 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} .

Matching vertices implies matching their positions, texture coordinates and vertex normals, if they are defined for the given model.

This method of matching meshes is tolerant to small topological and geometrical differences between meshes being tested. Also it can handle manifolds and non-manifolds uniformly and is not restricted to triangle meshes.

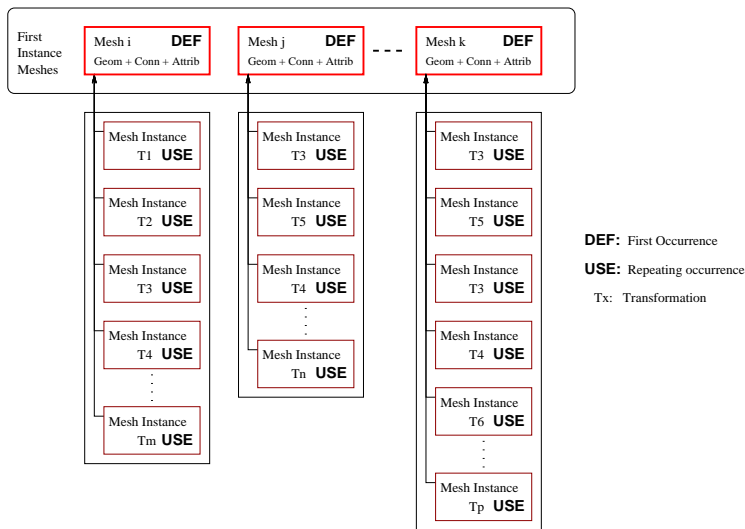


Figure 5: Creation of equivalence classes of features.

Once we have detected similarity of the different meshes in the geometric model, we can partition the entire model into equivalence classes as mentioned earlier. All similar meshes belong to a single class. Figure 5 illustrates the construction of equivalence classes of meshes having similar geometries as USE-instances of the first occurrence of the geometry which is tagged as the DEF-instance. A USE-instance is represented as a name reference to the DEF-instance and the rigid body transformation required to reconstruct the original mesh. The transformation is represented in the form of a pure rotation encoded as a quaternion and a position vector corresponding to the location of centroid of the USE-instance. The DEF-instances are represented in full details including geometry, connectivity and attributes.

Overcoming limitations of PCA: While attempting to match two components using PCA ambiguity can arise. If the eigenvectors of the correlation matrix are very different, it is easy to find the correct correspondence across two components being matched. But when the components have a completely symmetrical mass distribution, 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 to obtain the best rotation transformation for matching of components. This procedure is similar to that proposed by Novotni and Klein [30]. Consider all possible rotations around the axis or centre point of symmetry choose the rotation corresponding to the maximum match. Thus, 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, $R(\alpha)$ denotes the rotation and $M(R(\alpha), O_{c1}, O_{c2})$ denotes the degree of match between the components O_{c1} and O_{c2} when O_{c2} is aligned with O_{c1} as described earlier. In case of spherical symmetry the correct alignment is obtained as

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

where the angles (ϕ, α, ψ) denote Euler angles for parameterization of three-dimensional rotations.

The rotation space is discretized uniformly to consider uniformly distributed samples. For a cylindrical symmetry (where the mass is symmetrical around one of the principal axes), this is an efficient approach. However in cases of spherical symmetry this approach is impractical due to the large number of configurations. Fortunately, components having spherical symmetry are very rare in practice.

3.1.2 Acceleration Techniques

A naive implementation of the above scheme of discovering identical connected component level features in the model has a complexity of $O(n^2)$, where n is the number of connected components in the given model. In most engineering models, n tends to be large. It is very inefficient to compare each component level feature with every other component and carry out a geometric match. A method that allows an access to smaller class of components for this match is incorporated in our implementation. Our technique is in the spirit of what is known as geometric hashing and indexing in computer vision literature [20, 41]. While the worst case complexity even after hashing is $O(n^2)$, for most practical situations the speedup achieved is substantial.

To achieve hashing of components such that candidates for detailed geometric matching are put into the same bucket, we need some invariant descriptors. These descriptors must be invariant to the orientation and position of the components but must have identical values for identical feature components. Some examples of such invariant properties are:

- dimensions of the oriented bounding box of the component,
- surface area of the component (this can be trivially computed by adding up the areas of the triangles of the component),
or
- a simple combinatoric descriptor such as number of vertices and triangles in the component.

In our implementation, we carry out hashing using the number of vertices and triangles for hashing, and before performing a detailed we check if the dimensions of the OBBS also match.

Matching two aligned components A and B having n vertices each is a procedure of $O(n^2)$ complexity. It is so because the correspondence between vertices of the two components is not known. Hence each vertex of mesh A must be compared with every unmatched vertex of mesh B to determine the correspondence and also the count of matching vertices. This procedure is easily accelerated by discretizing the 3D bounding box of the feature components into uniform cells and classifying the vertices across these cells. The test for correspondence is then performed only among the vertices of the two components that lie within a cell. The worst case complexity of this process also tends to be $O(n^2)$ for the rare pathological cases where almost all their vertices are clustered in a few regions of their bounding boxes.

3.2 Sub-mesh Level Feature Patterns

The problem of discovering repeating patterns within or across polygon meshes representing connected components is difficult. The difficulty lies in automatic partitioning of the meshes to cut out some parts that represent repeating patterns. We approach this problem by our technique of “growing” patterns bottom up from the granularity of vertices. To enable effective classification of vertices in a given model, we assign *invariant footprints*³ to each vertex in the model. A footprint can be a scalar or vector value which must be invariant under rotations and translations, and should be “descriptive” in the sense that two vertices should be said to represent the same feature if the structure formed by their respective neighbourhoods is mostly identical. Also, the footprints must enable us to effectively discriminate between two vertices if they belong to different features in the local neighbourhood.

More formally, to form a footprint we must select invariant properties $I(f)$ over the features $f \in F$ in the model such that under a linear transformation $X' = TX$ of coordinates, $I(f) = I(T(f))$. In our implementation we have sought invariance to rigid-body transformation τ .

Examples of invariant properties that can become footprints for a vertex v in a polygon mesh are: (a) $|N_v|$, the number of vertices connected to v , where N_v is the set of vertices connected to v . The elements of N_v are also called the neighbourhood of v , (b) L_v , the average of the lengths of the connected edges, (c) D_v , the average of the dihedral angles of the faces meeting at the edges connected to vertex v . These properties are descriptive of the local features: $|N_v|$ describes the density at the vertex v , L_v describes the size of the feature, and D_v approximates the surface curvature at vertex v . In addition to these invariant properties based on the first order neighbourhood of the vertices, we can also use footprints computed over higher order neighbourhoods. We have experienced that the higher order invariant quantities help in distinguishing between features. In our implementation we use a vector footprint consisting of $|N_v|$, L_v , D_v and a second order footprint element in the form of average of D_j where $j \in N_v$. A simple 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 describe our algorithm of discovering repeating sub-mesh level feature patterns as follows:

1. Computation of Footprints:

- (a) Compute the footprints for each vertex v in the model.
- (b) Create equivalence classes of vertices having near-identical footprints. Let us denote equivalence classes as C_i , and the set of equivalence classes as C . The vertices having identical footprints represent matching local features and are likely to form good starting points for the “growth” of identical patterns around them.
- (c) Mark all the vertices as *not visited*.

2. **Growth Phase:** This phase attempts a simultaneous growth of the patterns around the vertices in the equivalence classes (seeds). The connectivity in the polygon mesh is considered as a graph with vertices as the nodes of the graph and the sides of the polygons connecting vertices making up the edges in the graph. Considering one equivalence class at a time, a breadth-first growth is carried out simultaneously around each seed, while geometrically verifying if the patterns match after each step in traversal.

This phase is applied to each equivalence class C_i :

- (a) Associate a pattern container with each vertex in the current equivalence class. A pattern container is a structure consisting of *current_seed*, *current_neighbourhood*, a queue of breadth-first traversal *bf_q*, a list of vertices making up the pattern *pattern*. Initialize the queues of each of the patterns by inserting the corresponding seed vertex.
- (b) While the queues *bf_q* are not empty:
 - i. For each pattern, *current_seed* = head of the queue *bf_q*.
 - ii. For each pattern, *current_neighbourhood* = $N_{current_seed}$. The elements of the neighbourhood are sorted in the non-decreasing order of the euclidean distance of the seed from the neighbour. This neighbourhood contains only those vertices which are not marked as visited. If *current_neighbourhood* is non-empty for less than two pattern containers, skip this iteration in the loop.

³We have borrowed the term *Footprint* from the work of Barequet and Sharir on Partial Surface and Volume Matching in Three Dimensions [4]. However our definition and use of footprints are significantly different.

- iii. Match the neighbourhoods by comparing the footprints of the corresponding elements in *current_neighbourhood*. (Note: Within an equivalence class, it is possible to obtain multiple types of repeating patterns. Care is taken to do all the housekeeping to keep track of these multiple growths. To track the multiple patterns the pattern container has an element *ref* pointing to the next matching pattern so that repeating patterns can be tracked during the simultaneous growth.)
 - iv. To each partial pattern of current loop, apply the Verification Step (3) to verify that the patterns are instances of the same features obtainable by rigid-body transformations. If the patterns at this level of growth match then add the elements of the neighbourhood to the corresponding queues *bf-q*. Also add the matched vertices to the corresponding *pattern* lists.
 - v. The vertices that are added to patterns are marked as *visited*.
3. **Verification Step:** This phase has two purposes, firstly verify geometric match between the two features identified as similar based on their footprints and connectivities, secondly determine the transformation τ . Given two patterns represented as a two sequences of vertices having one to one correspondence, the patterns tested for a match:
- (a) select some three non-collinear vertices from one pattern and the corresponding set of three vertices from the other pattern,
 - (b) determine the transformation to align these two sets of vertices,
 - (c) transform the two patterns using this transformation and verify if the corresponding vertices in the two patterns match geometrically within a tolerance.

This algorithm returns a set of patterns which correspond 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.

We have chosen to associate the footprints with vertices and not with higher order elements like edges or polygons. This is because Euler’s relation for planar graphs [28] indicates that for triangle meshes the number of triangles is roughly twice the number of vertices, and the number of edges is roughly thrice the number of vertices.

This “master geometry – instance transform” hierarchy among the repeating feature patterns is encoded compactly as discussed below.

3.2.1 Analysis

Compression of Vertex List: The compressed representation of the list of vertices that constitute the repeating patterns is obtained using the following scheme. On obtaining the patterns, we renumber the vertices and also the indices in the polygon lists such that the sequences of the vertices belonging to the repeating patterns are grouped by the patterns and have a contiguous numbering within the groups. The vertex list for each pattern forms a node that represents either the first occurrence of the pattern or its repeated occurrence. If the pattern node is the first occurrence then it consists of an integer k indicating the number of vertices in the pattern followed by coordinates of the k vertices. If it is a repeated occurrence, then it consists of an integer reference to the first occurrence, a rigid-body transformation to reconstruct the original position and orientation and an integer indicating the number of elements of this occurrence that match with the first occurrence. A rigid body transformation can be represented in terms of a pure rotation and a translation. Since a pure rotation can be represented using a quaternion with four floats and a translation vector with three floats, the representation for transformation requires only seven floating point numbers.

Thus a repeated instance requires three integers and seven point numbers as opposed to $3k$ floating point numbers for representing a repeated occurrence of a pattern having k vertices. We note that for a pattern of four or more vertices this scheme of encoding begins to yield compression.

Complexity Analysis: 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 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 of growth of repeating features. Classification of a vertex is a constant time complexity operation.

The complexity of the Growth Phase varies with the characteristics of the given model. The worst case depends on characteristics such as the maximum $|N_v|$ in the model, number of equivalence classes of vertices, size of the largest equivalence class, number of repeating feature patterns and sizes of the feature patterns in term of number of vertices. In an equivalence

class of k vertices, the worst case complexity of the growth of features is $O(k^2)$. This case occurs when it is not possible to grow any repeating patterns starting from the seeds in the class.

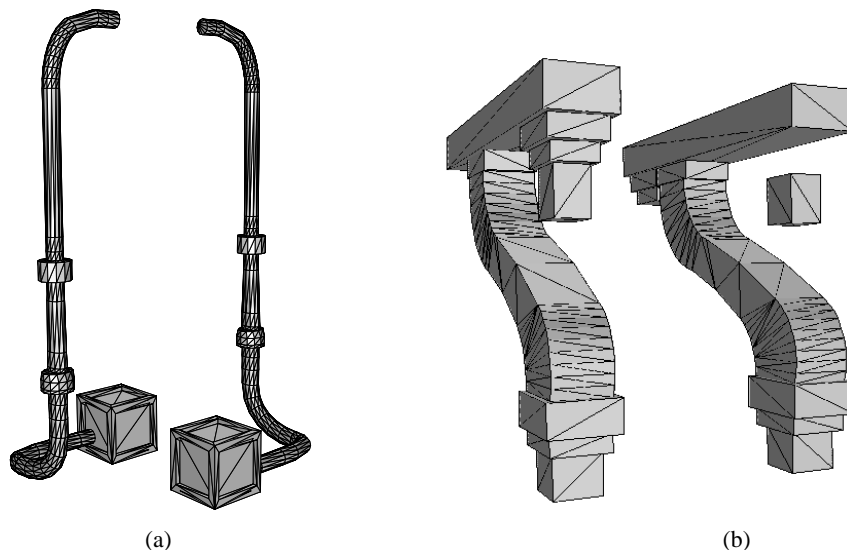


Figure 6: Examples of repeating structures: (a) each structure adds up to 18 meshes, 1112 triangles and 560 vertices; and (b) each structure adds up to 6 meshes, 212 triangles and 470 vertices (note that there could be gaps in automatic compositing of such macro structures).

3.3 Aggregate Features

Most large engineering plant models have many repeated instances not only of meshes but also of structures formed by groups of meshes. For example, a pillar in an architectural model is often formed using numerous small meshes and usually there are many such pillars across the model; in mechanical CAD models, many identical structures are formed using an assembly of parts which repeat in multiple locations.

Figure 6 shows two examples of repeating macro-level components consisting of multiple meshes. After carrying out mesh-level instance detection, many USE-instances of different meshes are found to have near identical rigid body transformations. This *iso-transformation* set enables us to infer *repeating macro-level component structures*. Figure 7 shows aggregation of iso-transformation instances (drawn in shaded envelopes) to identify macro structures in the model. Note that these macro structures may not always correspond to complete user identifiable component aggregates. There could be gaps in the automatic compositing of such macro structures, because of the need for numerical tolerances while matching the models (see Figure 6(b)). These gaps do not represent any shortcoming of this approach, since the goal is not to correctly reconstruct the complete structures but to obtain as large structures as possible to minimize the repeated representation of transformations.

As a side benefit of considerable value, our scheme also lets us automatically identify the common problem of erroneous duplication of meshes in large models. In Figure 7, we see that there are two instances of *Mesh-k* having the same rigid body transformation associated with them. This is an undesirable case of coinciding geometries in the model. It is almost impossible to identify such cases visually to heal the model to remove such artifacts. It must also be noted that a USE-instance having an identity transformation associated with it is also a duplicate mesh, and must be removed.

3.4 Structure of Compressed Storage

For compactly storing the master definition of components, we make use of the geometry and connectivity encoding schemes reported earlier. Figure 8 shows the elements of the compressed storage we use for our scheme. The storage structure consists of a file header; description of number of material nodes, number of DEF-meshes and number of macro-level component structures; a global list of representative normal directions; a list of DEF-meshes; and a list of macro-level component structures.

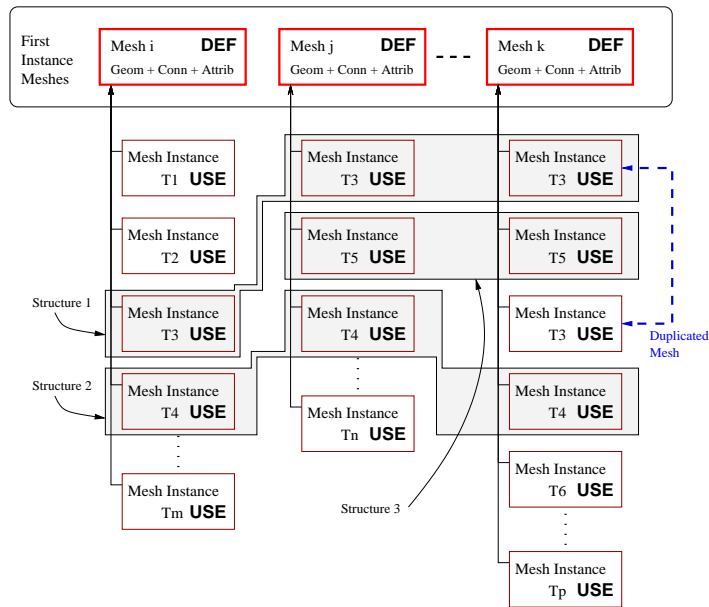


Figure 7: Creation of equivalence classes of geometric shapes: mesh-level tagging of instances and identification of *iso-transform* instances as repeating structures. Also note the identification of duplicate geometries.

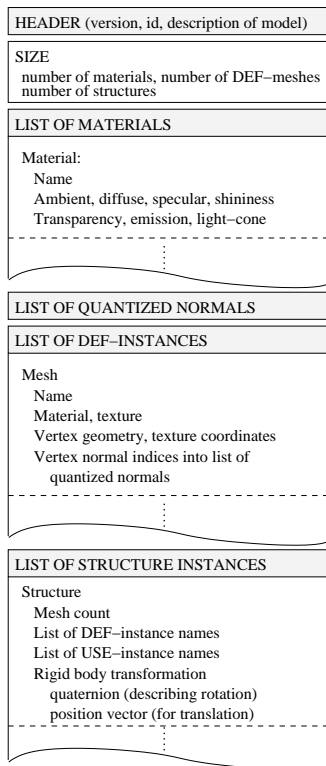


Figure 8: Structure of compressed storage

Quantization of vertex normals: In the source model, each vertex has an associated normal (3 floats). A simple lossy compression technique is to maintain a global list of normals for the scene. Using Deering’s strategy [9] we build a global list of normals by uniformly tessellating a unit sphere. We then represent vertex normals using indices to the closest entries in the list. More sophisticated global quantization methods can be used for encoding normals.

Encoding meshes as triangle-strips: In our current implementation, for encoding the connectivity information of DEF-meshes, we have implemented a simple algorithm for traversing vertices in the mesh and forming OpenGL type triangle strips. More sophisticated encoding schemes described in Section 2 can also be used for even more compact storage.

Representation of macro-level component structures: Macro-level structure components consist of one or more USE-instances represented by (i) a sequence of integer references to DEF-instances preceding the list of structures, (ii) a sequence of mesh names for the corresponding USE-instances, (iii) rigid body transformation consisting of a rotation and a translation, represented using a quaternion (4 floats) and a translation vector (3 floats).

3.4.1 Reconstruction of the Model

During reconstruction from the compressed representation of the model, the DEF-meshes are obtained from the file by straight-forward parsing of the name-value pairs. The vertex normals of these meshes are obtained from the global list of vertex normals.

While building the structure instance meshes, for the reconstruction of each USE-mesh the following steps are taken: (i) obtain the centroid m_1 of the DEF-mesh, (ii) obtain the rotation matrix R from the quaternion component of the transformation and the position vector m_2 of the USE-mesh, (iii) transform the vertex positions and normals of the DEF-mesh by the composite transformation $T' = T_{-m_1} \circ R \circ T_{m_2}$.

4 Results and Discussion

We carried out our tests on a large number of 3D models that are available for download on the net. We present some details of our results on six of these shown in Figure 4. The results demonstrate the effectiveness of the method on such models. On Diwan-i-Khaas model, for example, our algorithm gives us over 90% compression ratio. It is important to note that this performance has been achieved without using any special techniques for connectivity encoding of the DEF-meshes in the model.

4.1 Detection of Repeating Component Shapes

























The following table shows the results of our technique for detecting repeating component shapes in the six models. Clearly in such engineering models a small number of component shapes are repeated in different positions and orientations.

Model	Total No. of meshes in original	No. of DEF-meshes	No. of USE-meshes
Capitol	2662	93	2569
Colosseum	1129	20	1109
Diwan-i-Khaas	3726	848	2878
Helicopter	976	480	496
Piping	1051	122	929
Taj Mahal	375	45	330

The savings achieved in storing the data in terms of actual amount of geometry and connectivity can be gauged from the table below.

Model	No. of vertices (orig)	No. of triangles (orig)	No. of vertices (DEF)	No. of triangles (DEF)
Capitol	52606	87258	10347	19944
Colosseum	69868	135159	18912	38103
Diwan-i-Khaas	295695	162590	44363	46165
Helicopter	105079	187929	76231	136372
Piping	13103	20794	3753	6366
Taj Mahal	65323	126453	28427	55238

Another important observation that we have made has been that in most models a small number of shapes that repeat most often account for a large part of the model. The following table shows the four master component shapes that repeat most often in the six models and the corresponding savings in storage space. The number of bytes denoted in the compressed encoding column comprise of the space needed for the storage of the DEF-instance, integer references to the DEF-instance and the rigid body transformations associated with the USE-instances.

DEF mesh	No. of inst	No. of bytes per mesh	total # bytes	# bytes in compr encoding	No. of bytes saved	% saved
Capitol building:						
	220	824	181280	3233	178047	98.22
	199	1856	369344	4034	365310	98.91
	36	2192	78912	2577	76355	96.73
	18	3760	67680	3947	63733	94.17
Colosseum:						
	105	1072	112560	2216	110344	98.03
	96	624	59904	1669	58235	97.21
	92	736	67712	1737	65975	97.43
	36	10592	381312	10977	370335	97.12
Diwan-i-Khaas:						
	102	1136	115872	2247	113625	98.06
	36	14032	505152	14417	490735	97.15
	24	35552	853248	35805	817443	95.80
	24	9760	234240	10013	224227	95.73
Helicopter:						
	49	3648	178752	4176	174576	97.66
	49	1408	68992	1936	67056	97.12
	16	823	13312	988	12180	92.00
	8	5160	41280	5237	36043	87.31
Piping:						
	79	400	31600	1258	30342	96.02
	36	848	30528	1233	29295	95.96
	10	1792	17920	1891	16029	89.45
	4	4096	16384	4129	12255	74.80
Taj Mahal:						
	80	4248	339840	5117	334723	98.49
	8	39432	315456	39509	334723	87.48
	4	46376	185504	46409	139095	74.89
	4	23920	95680	23953	71727	74.97

4.2 Reduction in Storage Requirements

The following table shows reduction in file sizes (in bytes) from the original raw data to the compressed format.

Model	original	original (gzip)	compressed	compressed (gzip)	CR (%)
Capitol	1790767	875337	433922	211720	88.1
Colosseum	2503030	2090255	570931	401974	83.9
D'-Khaas	10072648	2850640	1978765	671828	93.3
Helicopter	4891936	1916270	1664592	899383	81.6
Piping	469891	172964	133329	35876	92.3
Taj Mahal	2329465	1122914	757002	467413	79.9

The compression achieved here is by avoiding detailed multiple descriptions of repeating meshes. Only DEF-instances are written out in detail using simple techniques for compressed single mesh encoding. For the DEF-instance mesh encoding, we have simply used `gzip`. This gives us an idea of the minimum compression performance we can expect. Clearly we can achieve even better performance by using better single mesh and instance transformation compression schemes.

5 Conclusion

We have presented a new 3D compression scheme particularly suited to very large engineering models, with repeating geometric shape features. Test results from a straight forward implementation of this scheme on a number of large models including those that are available on the net have shown excellent compression performance. The scheme uses Hotelling transform based registration technique and simple geometry matching procedure for identifying near identical meshes and then detects *iso-transformation* groups of USE-instances to determine repeating structures in the given model. Compression is achieved by representing only the master geometry in full details and avoiding the detailed description of the instances.

Our approach is significantly different from earlier approaches which have tried to compress large single mesh curvature continuous surfaces. Our scheme is somewhat analogous to dictionary-based algorithms [29] for compression of text; component shapes and groups of such components take the place of dictionary phrases.

The basic principle of detecting 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. Fuzzy matching can help us in detecting and correcting small geometric and topological errors. Simplification and LoDs of a DEF-instance mesh or a group of meshes automatically will serve many meshes in the model that are USE-instances. The observation that it is only a few components that account for a large part of the model is significant as it may lead to an optimal progressive transmission sequence.

Acknowledgements

For the test results reported in this paper, the models of Capitol building, Colosseum, Piping and Taj Mahal were downloaded from www.3dcafe.com. Diwan-i-Khaas model [1] is from Visions Multimedia Visualization Studio and the Helicopter model was downloaded from the Avalon 3D archive (avalon.viewpoint.com).

References

- [1] Fatehpur Sikri: An Epic in Red Sandstone. <http://www.visions-net.com>, Visions Multimedia Visualization Studio, Mumbai, India., 1999. (First public demonstration as a showcase application during the launch of Intel PIII processor in San Jose, USA.).
- [2] N. Amenta, M. Bern, and M. Kamvyselis. A New Voronoi-based Surface Reconstruction. In *SIGGRAPH 98*, pages 415–421, 1998.
- [3] C. Bajaj, V. Pascucci, and G. Zhuang. Compression and coding of large cad models, 1998.
- [4] Gill Barequet and Micha Sharir. Partial Surface and Volume Matching in Three Dimensions. *IEEE PAMI*, 19(9):929–948, 1997.

- [5] G. Butlin and C. Stops. CAD Data Repair. In *Proceedings of the 5th International Meshing Roundtable*, pages 7–12. Sandia National Laboratory, October 1996.
- [6] D. Cohen-Or, D. Levin, and O. Remez. Progressive Compression of Arbitrary Triangle Meshes. In *IEEE Visualization 99*, pages 67–72, 1999.
- [7] The Web3D Consortium. The Virtual Reality Modeling Language. <http://www.web3d.org>, ISO/IEC 14772-1, September 1997.
- [8] George Cybenko, Aditya Bhasin, and Kurt D. Cohen. Pattern Recognition of 3D CAD Objects: Towards Electronic Yellow Pages of mechanical Parts. In *Smart Engineering Systems Design, Volume I*, pages 1–13. 1997.
- [9] M. Deering. Geometry Compression. In *SIGGRAPH 95*, pages 13–22, 1995.
- [10] J. El-Sana and A. Varshney. Topology Simplification for Polygonal Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):133–144, April-June 1998.
- [11] A. R. Forrest. Computational geometry - achievements and problems. In R.E. Barnhill and R.F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 17–44. Academic Press, 1974.
- [12] M. Garland. *Quadric-based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, 1998.
- [13] R. Gonzalez and R. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.
- [14] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *SIGGRAPH 96*, pages 171–180, 1996.
- [15] A. Guizic, F. Bossen, G. Taubin, and C. Silva. Efficient Compression of Non-manifold Polygonal Meshes. In *IEEE Visualization 1999*, pages 73–80, 1999.
- [16] S. Gumhold and W. Strasser. Real-time Compression of Triangle Mesh Connectivity. In *SIGGRAPH 98*, pages 133–140, 1998.
- [17] I. Guskov, W. Sweldens, and P. Schroeder. Multiresolution Signal Processing for Meshes. In *SIGGRAPH 99*, pages 325–334, 1999.
- [18] P. Heckbert and M. Garland. Survey of polygonal simplification algorithms. In *Multi-resolution Surface Modeling Course*. ACM SIGGRAPH Course Notes, 1997.
- [19] P. Heckbert and M. Garland. Optimal Triangulation and Quadric-based Surface Simplification. *Computational Geometry*, 14:49–65, 1999.
- [20] Yaron C. Hecker and Rudd M. Rolle. On Geometric Hashing and the Generalized Hough Transform. *IEEE Transactions on Systems, Man and Cybernetics*, 24(9):1328–1338, September 1994.
- [21] Discreet (Autodesk Inc.). 3dstudio MAX R4. <http://www.discreet.com>, 2000.
- [22] M. Isenbueg and J. Snoeyink. Face Fixer: Compressing Polygon Meshes with Properties. In *SIGGRAPH 2000*, pages 263–270, 2000.
- [23] Z. Karni and C. Gotsman. Spectral Compression of Mesh Geometry. In *SIGGRAPH 2000*, pages 279–286, 2000.
- [24] A. Khodakovsky, P. Schroeder, and W. Sweldens. Progressive Geometry Compression. In *SIGGRAPH 2000*, pages 271–278, 2000.
- [25] D. King and J. Rossignac. Connectivity compression irregular quadrilateral meshes. Technical Report GIT-GVU-99-36, GVVU Center, Georgia Tech., Atlanta, USA, 1999.
- [26] D. King and J. Rossignac. Optimal Bit Allocation in Compressed 3D Models. *Computational Geometry*, 14:91–118, 1999.
- [27] FEGS Ltd. CADFix: A Product for CAD Data Exchange, Repair and Upgrade. <http://www.fegs.co.uk>, 1997.
- [28] M. E. Mortensen. *Geometric Modeling*. Wiley & Sons, New York, 1985.
- [29] M. Nelson. *Data Compression Handbook*. M & T Publishing, Inc., 1991.

- [30] Marcin Novotni and Reinhard Klein. A Geometric Approach to 3D Object Comparison. In *Proceedings of International Conference on Shape Modelling and Applications (SMI2001)*, May 2001.
- [31] Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Matching 3D Models with Shape Distributions. In *Proceedings of International Conference on Shape Modelling and Applications (SMI2001)*, May 2001.
- [32] R. Pajarola and J. Rossignac. Compressed Progressive Meshes. Technical Report GIT-GVU-99-05, GVU Center, Georgia Tech., Atlanta, USA, 1999.
- [33] J. Rossignac. Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January-March 1998.
- [34] G. Taubin. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95*, pages 351–358, 1995.
- [35] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive Forest Split Compression. In *SIGGRAPH 98*, pages 123–132, 1998.
- [36] G. Taubin, W. Horn, and P. Borrel. Compression and Transmission of Multi-resolution Clustered Meshes. Technical Report RC21398-(96630)-2FEB1999, IBM T.J. Watson Research Centre, New York, USA, 1999.
- [37] G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.
- [38] C. Touma and C. Gotsman. Triangle Mesh Compression. In *Proceeding of Graphics Interface 98*, June 1998.
- [39] G. Turk. Re-tiling Polygonal Surfaces. In *SIGGRAPH 92*, pages 55–64, 1992.
- [40] D. V. Vranic and D. Saupe. 3d Model Retrieval. In *Proceedings of Spring Conference on Computer Graphics and its Applications (SCCG2000)*, Budmerice Manor, Slovakia, pages 89–93, May 2000.
- [41] Haim J. Wolfson and Isidore Rigoutsos. Geometric Hashing: An Introduction. *IEEE Computational Science & Engineering*, pages 10–21, October-December 1997.
- [42] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 1996.
- [43] Andrew Zisserman, David Forsyth, Joe Mundy, Charlie Rothwell, Jane Liu, and Nic Pillow. 3D Object Recognition using Invariants. Technical Report OUEL 2027/94, Robotics Research Group, University of Oxford, November 1994.