HAL
open science

Reverse engineering for industrial-environment cad models

Rodrigo de Toledo, Bruno Levy, Jean-Claude Paul

To cite this version:

Rodrigo de Toledo, Bruno Levy, Jean-Claude Paul. Reverse engineering for industrial-environment cad models. International Symposium on Tools and Methods of Competitive Engineering, TMCE 2008, Apr 2008, Izmir, Turkey. inria-00202465

HAL Id: inria-00202465

https://inria.hal.science/inria-00202465v1

Submitted on 7 Jan 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Proceedings of TMCE 2008, April 21–25, 2008, Kusadasi, Turkey, edited by I. Horváth and S. Engin Kilic
© Organizing Committee of TMCE 2008, ISBN —

REVERSE ENGINEERING FOR INDUSTRIAL-ENVIRONMENT CAD MODELS

Rodrigo de Toledo
Tecgraf – PUC-Rio
rtoledo@tecgraf.puc-rio.br

Bruno Lévy
ALICE - LORIA - INRIA
levy@inria.fr

Jean-Claude Paul
Institute of Computer Aided Design - Tsinghua University
Paul@tsinghua.edu.cn

ABSTRACT

Industrial-environment CAD models are commonly represented by triangular meshes, which do not preserve original information about implicit surfaces used during design. The reverse-engineering algorithms presented in this paper focus on reconstructing implicit information, recovering original data. We propose two different approaches, a numerical one and an original topological approach. We explore specificities found in CAD meshes to achieve high effectiveness, reconstructing 90% of information from massive models (with millions of triangles) after few minutes of processing.

KEYWORDS

reverse engineering, CAD models, shape retrieval, surface recovery, model reconstruction, topological algorithms

1. INTRODUCTION

Modeling and simulation of complex industrial environments are difficult tasks for designers and engineers. There are several different domains in this context: oil industries (platforms and refineries), transport industries (airplanes and ships), industrial plant architecture, and so forth. Even though they involve similar kinds of tasks, each domain is quite different, and as a consequence there is a multitude of software applications.

Industrial-environment models are mainly composed of multiple sequences of pipes, cable trays and other technical networks, which are basically combinations of simple primitives (e.g., cylinders, cones and torus slices). Each application has its own internal data format and different ways to manipulate the primitives. For instance, even a simple cylinder can be stored in different ways: as two points and a radius, as a circle extrusion, as a segment revolution or even as a tessellated approximation. The most common way to export data is through a triangular mesh (actually, the only one found in all software). One of the reasons for using triangles as data format is rendering, since current graphics cards are optimized for triangle rendering (which has been the case for the last ten years).

However, representing simple primitives by tessellated approximations is clearly not the best solution for many reasons:

- Even using a very high resolution, tessellation is always an approximation. Losing the original information about the primitives may result in inaccurate simulation and coarse reconstruction. For example, during visualization, there will always be a detailed zoom at which one can see a faceted silhouette instead of a smooth one.
- It causes much higher memory consumption: while simple cylinder can be stored with 7 floating-point numbers (i.e., the 3D coordinates of two points and the radius) when it is tessellated with 8 vertices in both extremities it takes up about 400 bytes (to store topological information, vertex position and normal).
- Another issue is an ambiguity problem; for example, an 8-sided cylinder could be a representation of a circular cylinder or a pipe with really 8 faces.

Tessellation presents a clear tradeoff between quality and simplicity. The more quality is demanded, the larger number of triangles required for the rep-

resentation. However, an excessive number of triangles should be avoided for multiple reasons: memory consumption for storage, CPU/GPU transference overloading and performance downgrading for visualization.

On the other hand, if we have the original information on the primitives, we can generate the discretization only when necessary. Moreover, it is possible to control the discretization level, which is useful for visualization and numerical simulations (e.g., stress analysis).

In this paper we present an algorithm to retrieve higher-order geometric primitives from a tessellated database by means of a *reverse-engineering* method. We start by showing that classical numerical methods are not well suited for this kind of data (Section 3), even after some improvement (Section 3.2), and then we propose a novel topological method (Section 4). Our implementation is highly efficient, recovering 90% of original polygons in some massive industrial models after a few minutes of automatic processing (see results in Section 5).

2. RELATED WORK AND CONTEXT

Several applications make use of *Implicit Surface Recovery*: surface reconstruction, geometry enhancement, reverse engineering, model-based recognition and geometry simplification. For this reason, implicit surface recovery has been the subject of multiple research work in the last years (Petitjean, 2002; Werghi et al., 1999; Y.Y. et al., 1996; Lukacs et al., 1997; Wu & Kobbelt, 2005; Ohtake et al., 2003; Ohtake et al., 2005). Typically, the input consists of a dense 3D point set obtained after a three-dimensional scanning of an object. However, some methods consider polygon meshes as input information, which is our case.

In our application, we are only concerned with manufactured objects for our reverse-engineering process. Usually, objects composed by mechanical pieces can be well described by patches of planes, cones, spheres, cylinders and toroidal surfaces (Thompson et al., 1999). According to Nourse et al. (Nourse et al., 1980) and Requicha et al. (Requicha & Voelcker, 1982), 95% of industrial objects can be described by these implicit patches (or 85% if toroidal patches are not allowed). Thus, our recovery algorithm focuses on finding the equations of the original implicit surfaces used to describe the object.

Petitjean (Petitjean, 2002) lists the four main tasks usually found in recovery algorithms:

estimation: computes the local surface geometry using differential parameters such as normal and curvature;

segmentation: responsible for dividing the original data into subsets, each one probably forming a unique geometric primitive;

classification: this step decides in which surface type a subset should be included (cylinder, torus, cone or some other); and

reconstruction: finds the surface parameters to correctly fit the input data.

The *estimation* step can be seen as a preprocessing stage for segmentation and classification. In this step, local properties are computed for the whole original data set. Using topological information from the input data, different methods are capable of computing these properties. In general, the properties we are interested in are: normal (n), principal curvature directions (e_1, e_2) and principal curvature values (k_1, k_2). Based on *differential geometry* principles (do Carmo, 1976), the estimation methods vary significantly, and there is no consensus on the most appropriate way to approximate geometric local properties (Petitjean, 2002).

Segmentation and *classification* steps are closely related. In an ideal case, the classification step would clearly benefit from a correct segmentation output. In this case, when trying to find which surface best represents the *segment*, the algorithm would not suffer from disturbing neighboring data. But, unfortunately, both tasks are rarely executed in a perfect sequence. In general, they are executed simultaneously rather than sequentially.

The *segmentation* step is commonly executed using region-based approaches. There are different region-based strategies found in the literature: *region growing* (Lukacs et al., 1997), *split-and-merge* (Parvin & Medioni, 1986) and *region clustering* (Wu & Kobbelt, 2005). In some of these strategies, segmentation is done in combination with the classification step (Werghi et al., 1999).

Classification is the last difficult task in recovery algorithms, since *reconstruction* is almost a straightforward procedure, provided all the previous steps were well executed.

In this paper, we introduce two recovery strategies used in our application. The first one, described in

Section 3, is more similar to conventional surface-recovery approaches. The second one, in Section 4, is more industrial-CAD data driven, and is deeply based on specificities found in our input data.

In order to understand the characteristics present in industrial-environment data, we enumerate the details and important points of this data type.

2.1. Industrial-Environment CAD Models

Ideally, the tessellated representation should never be the only option to access industrial data sets. In practice, however, this is not the case, and triangle meshes are frequently the only data representation available. One reason is that triangle meshes are currently the *lingua franca* in computer visualization and mesh representation. The multitude of software in the numerous CAD areas also contributes to the lack of another common representation or data type.

Reverse engineering typically uses scanned data as input. In this kind of data, there is a dense set of points laid down on objects surfaces. However, the data we are dealing with presents a different situation, since they were generated without any scanning or capturing step. Although both types of data are composed by point sets (and additional topology information), we can list some different characteristics found in industrial tessellated software-generated data (or *CAD data*):

1. CAD data have sparser samples in the point set when compared to scanned data. For example, a cylinder patch can contain only 12 vertices in its border.

2. The vertices are regularly distributed in tessellated CAD data. For instance, in most cases, the vertices of a tubular structure will be located on a set of circles (or *rings*), like a backbone and vertebrae. This regularity can also be observed in the topology; for example, in a cylinder, edges are aligned with the axis.

3. Besides regularity, the vertices also have a precise location. This means, for example, that if a set of vertices is expected to be lying on the same plane, then this will be the case, except for a very small threshold.

4. There is a partial segmentation, with some objects separated from one another. For instance, two different pipelines are separated objects in the data, although each one is still composed by different patches (cylinders, cones and tori) that were not yet split.

This last characteristic is a significant advantage of our data set. However, characteristic (1) is a drawback that disturbs traditional recovery algorithms. In Section 3 we explain a *numerical approach* for surface recovery that tries to overcome this problem. On the other hand, characteristics (2) and (3) are an advantage that we have explored in our *topological method* described in Section 4.

3. NUMERICAL METHOD

In this section we approach the *classification* step, which is the third step in conventional recovery algorithms (*estimation*, *segmentation*, *classification* and *reconstruction*). In Subsection 3.2 we focus on how to overcome the sparse characteristic of CAD industrial data that disturbs conventional *classification*.

Let us consider that the normal $\vec{n_i}$ of each vertex v_i was already computed in the *estimation* step and that the *segmentation* step was already successfully executed. Now we have a set of surface meshes, each one corresponding to a higher-order geometric primitive. In the *classification* step we want to verify which of these meshes correspond to quadric surfaces. As explained in (Werghi et al., 1999) we can execute a least square fitting to find out the coefficients that determine the quadric equation.

Although some authors suggest a specific search for each quadric surface (Fitzgibbon et al., 1997; Petitjean, 2002), we adopted the strategy of starting by fitting the 10 coefficients (9 in practice) of generic quadrics and then classifying each surface as one of the expected types (cylinder, cone, sphere, etc.).

Our quadric recovery procedure follows the sequence:

Step I quadric fitting (by least-square fitting)

Step II quadric classification (by evaluating the quadric coefficients)

Step III reconstruction (by extracting main directions and by projecting input vertices)

3.1. Quadric Fitting (Step I)

We want to find the quadric surface that best fits the input point-set already segmented. In standard

quadric fitting, the adequacy is measured by evaluating the minimum distance between the point set and the quadric. The goal is to find the quadric surface Q parameterized by 10 coefficients (see Equation 2) that best describes the point set. For a point set with n 3D points $p_i = (x_i, y_i, z_i)$, we want the solution of the following system of linear equations:

$$f(x_i, y_i, z_i) = 0, \quad 0 > i \geq n \tag{1}$$

$$\begin{aligned} f(x,y,z) &= Ax^2 + 2Bxy + 2Cxz + 2Dx + \\ &\quad Ey^2 + 2Fyz + 2Gy + \\ &\quad Hz^2 + 2Iz + J = 0 \end{aligned} \tag{2}$$

or using 4×4 symmetric matrix Q:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^T \begin{bmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{bmatrix}_Q \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0 \tag{3}$$

For dense point clouds, n is much larger than the number of degrees of freedom (9 for quadrics) and the above system is overdetermined. The system is solved by using least-squares fitting to globally minimize the distance between the points and the surface.

We can write the system above (Equations 1 and 2) in matrix form ($\mathrm{A}x = b$):

$$\begin{bmatrix} {x_1}^2 & {x_2}^2 & \cdots & {x_n}^2 \\ 2x_1y_1 & 2x_2y_2 & \cdots & 2x_ny_n \\ 2x_1z_1 & 2x_2z_2 & \cdots & 2x_nz_n \\ 2x_1 & 2x_2 & \cdots & 2x_n \\ {y_1}^2 & {y_2}^2 & \cdots & {y_n}^2 \\ 2y_1z_1 & 2y_2z_2 & \cdots & 2y_nz_n \\ 2y_1 & 2y_2 & \cdots & 2y_n \\ {z_1}^2 & {z_2}^2 & \cdots & {z_n}^2 \\ 2z_1 & 2z_2 & \cdots & 2z_n \end{bmatrix}^T \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

We want to minimize the residual error $r = \mathrm{A}x - b$, so we solve x for the least-squares equation $[\mathrm{A}^T\mathrm{A}]x = [\mathrm{A}^T b]$. This can be done using the Cholesky factorization. (Note that the 10th quadric coefficient (J) is actually a constant so setting the right hand to 1 is equivalent to fixing $J = -1$.)

Unfortunately, in our application the point cloud is not dense. For example, a cylinder candidate segment can have about 12 vertices with multiple alignments, yielding an underdetermined system of equations with many possible solutions. For instance, either a cylinder or a cone can fit the vertices of a typical CAD cylinder (see Figure 1). Therefore, we need more equations to provide our system with more independent information.

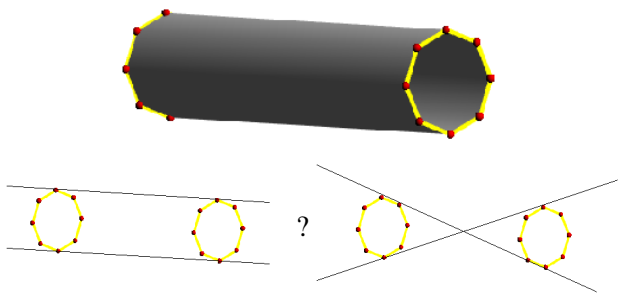

Figure 1 *Top:* The 16 vertices (in red) of a cylinder in a CAD model are used for fitting a quadric in the reverse-engineering process. That is not much information, and may result in ambiguity. For example, the vertices can be fitted by a cylinder (*bottom left*) or by a cone (*bottom right*).

3.2. Improving Numerical Fitting

In order to feed the linear system in Equation 1 with additional information, we have used the vertices normal vector. To do so, we have added extra equations to fit the quadric surface normal to the normal $\vec{n_i}$ of each vertex. Thus, for each vertex, three new equations (corresponding to x, y and z components of the normal) were inserted in the system respecting the rule:

$$\begin{aligned} \vec{n} &= \left[\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}\right] \\ \frac{df}{dx} &= 2Ax + 2By + 2Cz + 2D = \vec{n_i}.x \\ \frac{df}{dy} &= 2Bx + 2Ey + 2Fz + 2G = \vec{n_i}.y \\ \frac{df}{dz} &= 2Cx + 2Fy + 2Hz + 2I = \vec{n_i}.z \end{aligned} \tag{4}$$

With this improvement, the number of equations is $4n$ instead of n and we can converge to a single best solution (in least-squares sense). Figure 2 shows the final system of equations. Once again, we want to minimize the residual error of this new system of equations. This is done by solving $[\mathrm{A}^T\mathrm{A}]x =$

$$
\begin{bmatrix}
{x_1}^2 & 2x_1y_1 & 2x_1z_1 & 2x_1 & {y_1}^2 & 2y_1z_1 & 2y_1 & {z_1}^2 & 2z_1 \\
2x_1 & 2y_1 & 2z_1 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 2x_1 & 0 & 0 & 2y_1 & 2z_1 & 2 & 0 & 0 \\
0 & 0 & 2x_1 & 0 & 0 & 2y_1 & 0 & 2z_1 & 2 \\
{x_2}^2 & 2x_2y_2 & 2x_2z_2 & 2x_2 & {y_2}^2 & 2y_2z_2 & 2y_2 & {z_2}^2 & 2z_2 \\
2x_2 & 2y_2 & 2z_2 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 2x_2 & 0 & 0 & 2y_2 & 2z_2 & 2 & 0 & 0 \\
0 & 0 & 2x_2 & 0 & 0 & 2y_2 & 0 & 2z_2 & 2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
{x_n}^2 & 2x_ny_n & 2x_nz_n & 2x_n & {y_n}^2 & 2y_nz_n & 2y_n & {z_n}^2 & 2z_n \\
2x_n & 2y_n & 2z_n & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 2x_n & 0 & 0 & 2y_n & 2z_n & 2 & 0 & 0 \\
0 & 0 & 2x_n & 0 & 0 & 2y_n & 0 & 2z_n & 2
\end{bmatrix}_{4n\times 9}
\begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \end{bmatrix}_9
=
\begin{bmatrix}
1 \\ \vec{n_1}.x \\ \vec{n_1}.y \\ \vec{n_1}.z \\ 1 \\ \vec{n_2}.x \\ \vec{n_2}.y \\ \vec{n_2}.z \\ \vdots \\ 1 \\ \vec{n_n}.x \\ \vec{n_n}.y \\ \vec{n_n}.z
\end{bmatrix}_{4n}
$$

Figure 2 Improving numerical fitting. For each vertex, three new equations corresponding to normal components were inserted in the system of equations.

$\left[\mathrm{A}^T b\right]$ with the Cholesky factorization.

3.3. Classification (Step II)

Based on the quadric coefficients $[A, \cdots, J]$ computed in the previous step, we can find out what kind of quadric surface these coefficients determine. In our case, we are searching for simple surfaces such as cylinders, cones and spheres. Our classification is partially based on the quadric geometric invariants (Y.Y. et al., 1996). For this purpose, we need to evaluate the upper-left three-by-three matrix R extracted from the quadric matrix Q:

$$
R = \begin{bmatrix} A & B & C \\ B & E & F \\ C & F & H \end{bmatrix}, \text{where } R_{i,j} = Q_{i,j\,(1<i,j\leq 3)}
$$

The eigenvalues and the determinant of R are fundamental for quadric classification. The eigenvectors will be also explored in the next step, *reconstruction*.

3.4. Reconstruction (Step III)

The input of this step is a mesh representing a surface segment already detected as one of the quadrics we are interested in. The expected output are the exact parameters that can be used to reconstruct this surface segment.

Let us take the cylinder as an example. We are interested in a higher-level representation of the cylinder consisting of the two extreme 3D points P_1 and P_2 and the cylinder radius r.

First of all, using the eigenvalues and eigenvectors it is possible to find the cylinder's main direction $\vec{v}$. Assuming that the quadric coefficients describe a cylinder, the matrix $R_{3\times 3}$ contains two identical eigenvalues and one null eigenvalue. The eigenvector associated with the null eigenvalue indicates the cylinder's main direction.

After obtaining cylinder direction, it is necessary to compute at least one point lying on the cylinder's main axis to completely describe it. To find this point we solve a very simple system of equations. Given the implicit function $c(P) = [P_xP_yP_z1]Q[P_xP_yP_z1]^T$, in which the zero-set defines a cylinder, the derivative of the function is null if and only if P is a point on the cylinder's main axis:

$$
c'(P) = [0, 0, 0], \text{ if and only if } P \in \text{cylinder axis.}
$$

So we can set the following system of equations:

$$
\frac{dc}{dx} = 0, \quad \frac{dc}{dy} = 0, \quad \frac{dc}{dz} = 0
$$

Any point P_0 satisfying the above system is on the cylinder's main axis.

So, up to this point, we have obtained P_0 and $\vec{v}$ defining the cylinder's axis. But we still need to find out the extreme points P_1 and P_2 and the radius r.

By projecting all the vertices v_i of the original mesh on the main axis parametrically defined by $P_0 + t\vec{v}$, we can find the extreme points P_1 and P_2:

$$P_1 = P_0 + \min_{i=1}^{N}(\vec{v} \cdot (v_i - P_0))$$
$$P_2 = P_0 + \max_{i=1}^{N}(\vec{v} \cdot (v_i - P_0))$$

and the radius r can be computed by:

$$r = \sum_{i=0}^{N} \frac{\| (\vec{v} \times (v_i - P_0)) \|}{N \| \vec{v} \|}.$$

Sections 5.1 and 5.2 show some results obtained with this numerical method.

3.5. Normal dependency

The use of normal equations has improved the results for recovering quadrics from CAD meshes. We have tested this algorithm with segmented pieces of industrial CAD models (see Section 5). However, for non-segmented data, a problem related to the normal computation may appear.

The normal at each vertex is obtained by computing the average normal of neighbor faces during the *estimation* phase. Therefore, a vertex shared by a cylinder and a torus will not be perpendicular to the cylinder axis as we should expect (see Figure 3)[1]. Even if the normal information is already available in the original data (as in the PowerPlant data set), the normals are not precise (see Figure 3(*right*)), and they were also probably computed using the average of normal faces. This problem disturbs our *classification* algorithm and it would also disturb standard *segmentation* algorithms, which are heavily normal dependent (Petitjean, 2002).

The next section shows a new approach, which is not based on numerical fitting.

4. TOPOLOGICAL METHOD

Drawbacks of the numerical approach (such as segmentation difficulties, heavy normal dependency and the complexity of tori fitting (Lukacs et al., 1997)) motivated us to search for alternative ideas about how to extract higher-order primitive information. The regularity found in the input data encouraged us to investigate the *topological* approach.

Our algorithm is exclusively designed to segment tubes composed of cylinders, truncated cones and

[1] In other words, the normal vectors of vertices forming circular sections (*rings*) should lie in the plane containing the ring. See Section 4 for more information about rings.

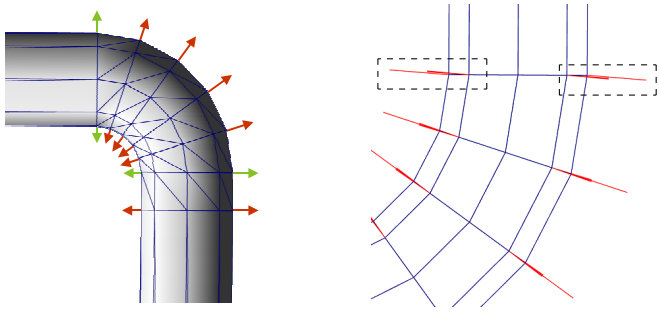

Figure 3 **Normal estimation problem.** *Left:* For a tubular mesh with fixed radius, one should expect the normals of vertices in rings (circular sections) to have a perpendicular direction to the ring's normal. Even for rings separating two different primitives (e.g., a torus and a cylinder) the normals (in green) should respect a perpendicular direction. *Right:* However, the normal estimation of a vertex uses the average of neighboring faces, resulting in normals not perpendicular to the ring's direction (in red).

"elbows" (elbows correspond to torus slices, shown in green in Figure 4). The basic idea is to find all circular regular sections (the *rings*) and their connectivity. By verifying consecutive rings we can trivially determine if they fit one of these three higher-order primitives. Speed is one of the advantages of this method over traditional numerical approaches for reverse engineering. Numerical approaches are very appropriate for rough data input (for example, obtained by scanner). In our case, however, we exploit the fact that the data are almost constantly regular and can be easily traversed by executing a simple cut-and-find algorithm.

Our method is based on a mesh traversal consisting of three steps described in the following subsections.

4.1. Ring detection

Rings are identified in the mesh as coplanar loops of six vertices or more, equidistant to their center. See Figure 4(a). By traversing mesh topology we identify them in sequence.

4.2. Tube segmentation and classification

Rings are grouped into primitives (see Figure 4(c)) according to their radii and normal vector:

- cylinders: correspond to a pair of rings with the same normal and radius;

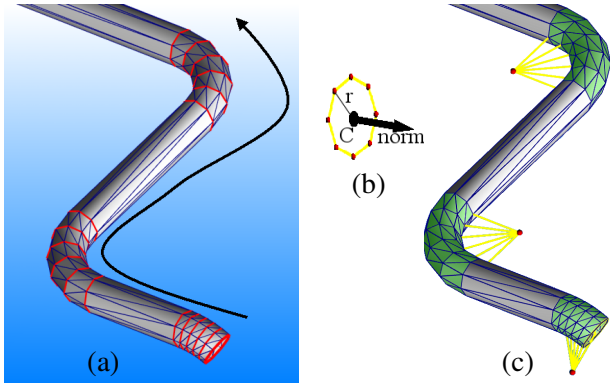

Figure 4 (a) Identification of coplanar loops (*rings*) by traversing the mesh. (b) We estimate the properties of each ring: center C, radius r and normal vector. (c) Segmentation and classification are done by evaluating consecutive rings. Reconstruction of the higher-order primitives is obtained by evaluating ring properties.

- cone sections: correspond to a pair of rings with the same normal and different radii;
- elbows: correspond to a set of rings with the same radius, and with co-normals radiating from the same point (see yellow lines in Figure 4(c)). For a consecutive pair of rings, R_1 and R_2, we first verify if their centers and their normals are coplanar; if this is not the case, it is not an elbow. Then, we find the radiation point (see red points in the same figure) as the intersection point between three planes: the plane just tested, the plane containing R_1 and the plane containing R_2. This way, the co-normal of a ring is the vector linking the radiation point and the center. (Note that we accept torus slices with up to 180 degrees, otherwise they will be cut into two.)

4.3. Primitive reconstruction

Once the set of rings corresponding to each primitive is determined, it is straightforward to find the parameters of the primitive from the loop centers and radii. The centers of the extreme sections (P_1 and P_2) of each primitive are simply the centers of the first and last corresponding rings. The radii are also determined from the loops. For elbows, the center C is the intersection of the co-normals (shown in yellow in Figure 4(c)). Each primitive is then completely determined by a small set of parameters:

- cylinder: P_1, P_2, r
- cone: P_1, P_2, r_1, r_2

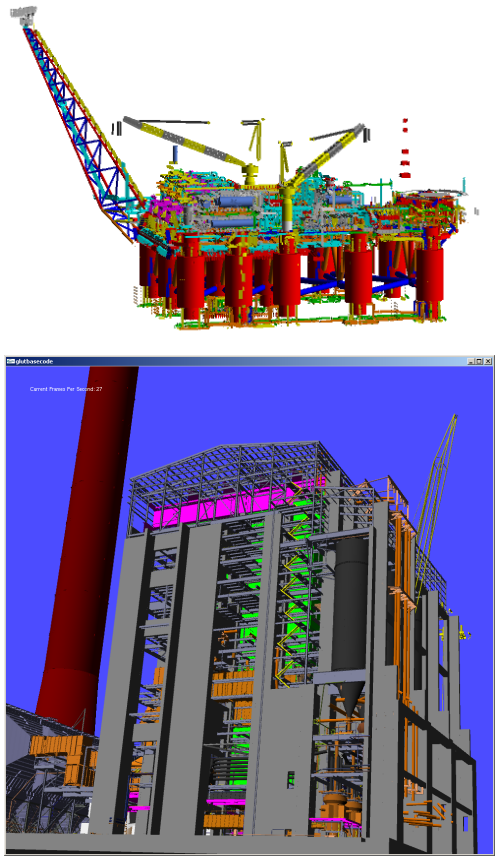

Figure 5 *Top:* P40 Oil-Platform model used on tests in Section 5.1. *Bottom:* PowerPlant model used on tests in Sections 5.2 and 5.3.

- elbow: P_1, P_2, C, r

After our reverse-engineering algorithm is applied to the database, the result is a set of higher-order primitives (cylinders, cones, elbows) and a set of polygons for the other objects.

5. RESULTS AND APPLICATIONS

We have used two data sets in our tests: PowerPlant (PP), with 13 million triangles, and Oil Platform P40 from Petrobras, with 27 million triangles (see Figure 5). We have tested both numerical and topological approaches. The first one has no cutting algorithm, so, in its case, we have restricted the tests to some examples where most of primitives were already separated (Sections 5.1 and 5.2).

The times were taken using an AMD Athlon(TM) XP 3800+ 2.41GHz with 2GB of memory.

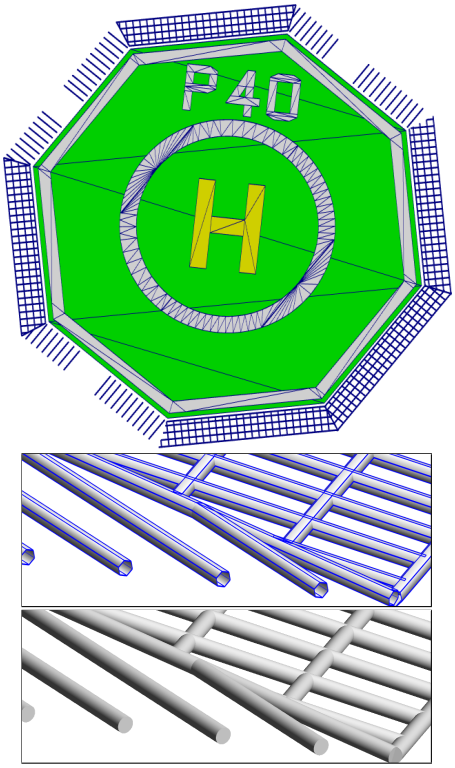

Figure 6 Part of an oil platform model made of 446 different meshes (*top*). There are several tessellated cylinders around the model sample (*middle*). Our numerical algorithm has recovered the higher-order information about the cylinders, so we can obtain their original parametric information (*bottom*).

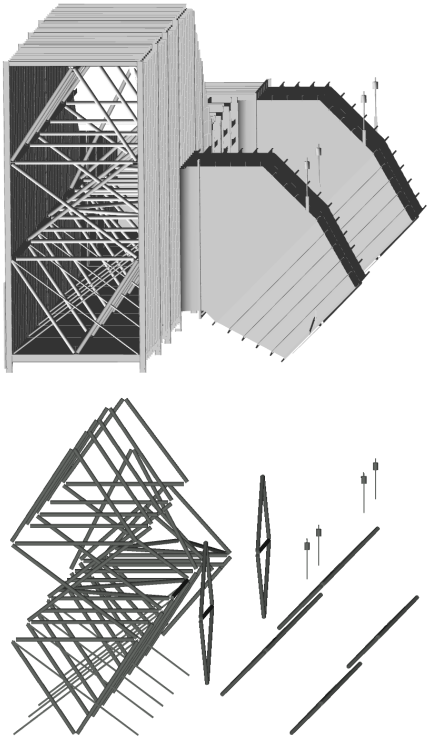

Figure 7 PowerPlant piece of data (Section 17, Part b, g1). 118 cylinders are found (*bottom*) from the original data (*top*). In this piece, there are mainly structural cylinders instead of pipes. As a consequence, they are already segmented.

5.1. Numerical approach for Helicopter Landing on P40 Oil-Platform

In the example shown in Figure 6, there are 446 separated meshes. Some of them define the interior of the sample model, which contains symbols ("P 40" and "H") and other geometric forms; and most of the meshes around the model are the tessellated cylinders we are interested in.

With the numerical approach for reverse engineering described in Section 3 we have successfully recovered the 436 cylinders in this model. The execution time was 6.11 seconds for the entire recovery process, which includes *fitting*, *classification* and *reconstruction*.

5.2. Numerical and Topological approaches for Sections 17 and 18 on PowerPlant

In Section 17 almost all the cylinders are disjointed meshes in the original data. They appear in the model more as structural objects than as pipes (see Figure 7). This is an important issue because our numerical algorithm does not segment the model.

Table 1 shows the results. Compared with the topological method, the numerical approach misses some cylinders (there are 901 found with the topological method against 871 with the numerical approach). They are probably some not-segmented meshes that are undetectable by our numerical approach.

Section 18 of PowerPlant is exclusively composed by cylinders and tori. These data were completely recovered by the topological reverse engineering. However, the numerical approach faced two problems: there is no torus detection and the not-segmented cylinders could not be found. Another significant difference between both methods is efficiency. While the numerical method executed the reverse engineering in almost two minutes, the topological one took less than two seconds.

Data set (triangles Δ)	PP – Section 17 (251,184)		PP – Section 18 (167,012)	
method	numer.	topo.	numer.	topo.
cylinders found	871	901	2214	2674
tori found	0	0	0	858
recovery time	9.53s	2.84s	110.62s	1.43s
unrecognized Δ	236,864	203,958	129,540	0
effectiveness	5.7%	18.8%	22.43%	100%

Table 1 Numerical and topological reverse-engineering approaches for Sections 17 and 18 of the PowerPlant. The topological approach has better performance and effectiveness.

5.3. Topological approach for the entire PowerPlant

We have applied the topological algorithm, based on ring traversal, to the PowerPlant data. The data are distributed in 21 sections and Section 1 is the largest, containing about 3.5 million triangles corresponding to 30% of the model.

We have tested the reverse-engineering method over the entire PowerPlant. Table 2 shows the numbers for each data section. For each one we have measured:

Original Data: the number of triangles and the size of a `ply` file that records the data.

Reverse Engineering: the execution time for the topological reverse-engineering method.

Unrecovered Data: the same kind of information as in *Original Data*. The unrecovered data are mainly box walls and floors and very few primitives not found (see *effectiveness* below).

Recovered Primitives: the number of the three basic primitives (cylinder, cone and torus) and the memory space computed as: $4 * (7 * \text{cylinders} + 8 * (cones) + 10 * (tori))$.

There are some specific but relevant information about the sections:

- Sections 1, 2, 15, 18, 19 and 20 were exclusively composed by tubes. As a consequence, they were 100% recovered. See Table 2.
- The number of triangles in Section 12 does not match the number available in the UNC homepage from where the PowerPlant was downloaded. There might be an error either in our loading algorithm or in their homepage.

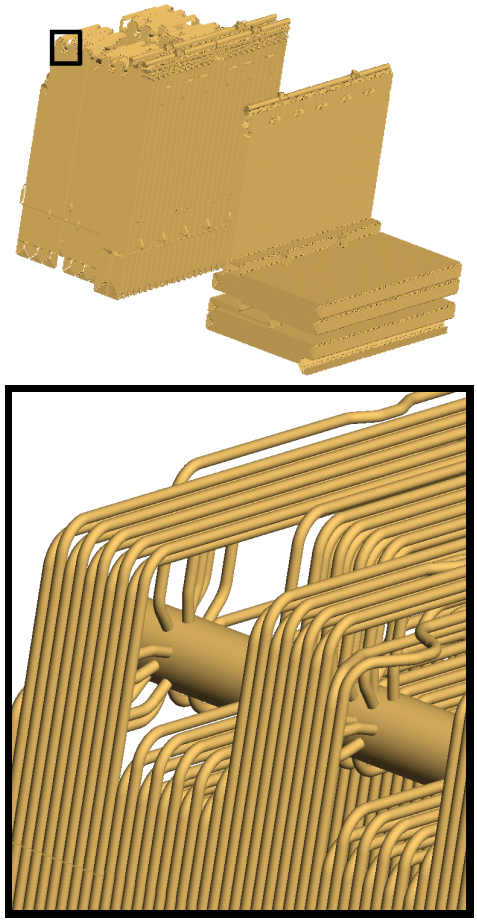

Figure 8 PowerPlant Section 1, with about 60,000 primitives recovered from 3,500,000 triangles.

- The external part of the chimney in Section 16 was found after a manual cutting of its base. This is the only case of assisted recovering on PowerPlant.

Effectiveness

From the sum row (Σ) in Table 2 we can observe that 90% of the PowerPlant was recovered as cylinders, cones and tori $\left(\frac{\text{(unrecovered triangles)}}{\text{(original triangles)}} \approx 10\%\right)$.

Most of the unrecovered data were objects that cannot be directly described by implicit surfaces (for example, boxes and walls). However, our algorithm missed some surfaces that should be described implicitly. Figure 9 shows the most common cases. Unfortunately, in our implementation, if a tube contains one of the objects in Figure 9 the whole pipe is not recovered.

Section	Original Data		Rev. Eng.	Unrecovered Data		Recovered Primitives				Effect.
#	triangles	mem. (KB)	time (sec.)	triangles	mem. (KB)	cylinders	cones	tori	mem. (KB)	recovered rate (%)
1	3429528	119213	24,03	0	0	31372	20	26546	1895	100%
2	161568	5955	1,23	0	0	1795	0	1205	96	100%
3	382062	14406	3,75	41836	1125	4061	629	2038	210	89%
4	55178	2675	0,70	22136	1076	771	12	154	27	60%
5	222728	9196	2,09	28629	1691	2589	593	1021	129	87%
6	55300	2410	0,78	18945	790	804	96	80	28	66%
7	98592	5519	1,26	53075	3485	1391	0	2	38	46%
8	152586	6013	1,40	9270	525	1664	30	972	84	94%
9	121862	6253	2,40	35901	1434	2773	0	6	76	71%
10	197120	11472	3,09	161401	9862	636	20	0	18	18%
11	133398	8692	1,95	114352	7185	674	0	0	18	14%
12	360872	13755	3,93	38067	1392	3678	493	2034	195	89%
13	114334	4360	1,15	8616	239	1312	185	659	67	92%
14	248432	15748	3,73	212635	13127	507	2	18	14	14%
15	1141240	34777	9,00	1632	90	13656	0	8183	693	100%
16	365970	22739	5,34	291864	18271	970	2	68	29	20%
17	251184	15638	2,84	203958	12451	901	0	0	24	19%
18	167012	6465	1,43	0	0	2674	0	858	106	100%
19	2650680	89245	18,48	0	0	22433	6	19607	1379	100%
20	2415976	86556	17,51	0	0	22940	24	18908	1366	100%
21	17356	986	0,26	8702	58	300	0	0	8	50%
Σ	12742978	482073	106,35	1251019	72801	117901	2112	82359	6507	90.18%

Table 2 Reverse engineering over the power plant sections.

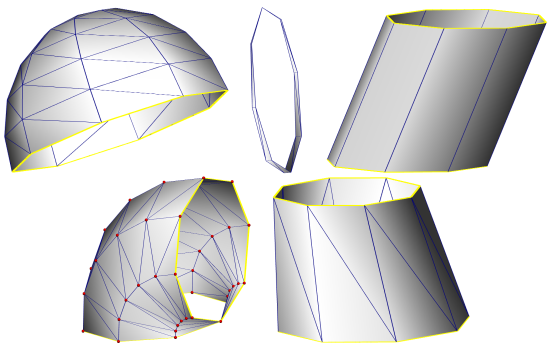

Figure 9 Some of the missing data in our recovery algorithm: half-sphere; very thin torus section; sheared cylinder section; torus section with a singular vertex (this happens when the torus radii have the same value, it is called *horn torus*); and sheared cone.

Efficiency

In our tests, before executing reverse engineering, the data was already in memory, thus we did not consider the time of loading and topology creation in the column *Rev. Eng.* of Table 2. We have measured the execution time including deleting the recovered meshes from original data. By deleting them, we are able to correctly save the unrecovered data after execution.

In Figure 10 we plot the relation (triangles×time) of all 21 sections (the *deleting* column in Table 2). As expected, the recovery algorithm's performance is linearly proportional to the number of triangles.

We were able to process more than 100 thousand triangles per second:

$$\frac{12,742,978\triangle}{106.35s} \approx 120,000\triangle/s.$$

Memory Reduction

The meshes that were completely converted to primitives represent 11,491,959 triangles in 409,272 KB. After recovering them to higher-order primitives they

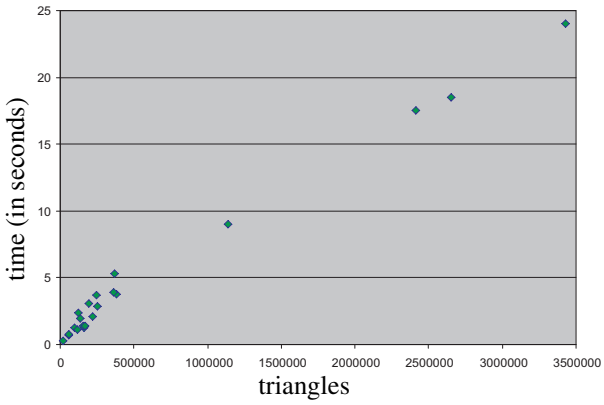

Figure 10 Performance for recovering the 21 sections of the PowerPlant. The algorithm's performance is linearly proportional to the number of triangles.

were reduced to 6,507 KB. The reduction factor is 98.41%.

$$1 - \frac{409,272}{6,507} = 0.9841$$

However, if we consider that 1,251,019 triangles were not converted and they still use 72,801 KB of memory, the reduction factor becomes 83.54%, which still is an expressive rate.

$$1 - \frac{482,073}{79,308} = 0.8354$$

6. CONCLUSION AND FUTURE WORK

The insertion of equations that fit the normals improved the numerical recovery process of sparse vertices. However, the results shown here were successful exclusively over disjoint sets of meshes. On the other hand, we have observed that regularity is a positive characteristic in this kind of input data. For this reason, we have developed the topological approach, which is well adapted to the specificity of such data.

The topological recovery procedure converts 90% of the original data of industrial models. The memory space of recovered data is reduced to at most 2% (98% of reduction). This is a consequence of compact implicit representation used for recovered primitives. If we consider the remaining 10% of unrecovered triangles, industrial models such as oil platforms and power plants can be stored in about 15% of their original data size. This expressive reduction in storage can contribute to rendering performance. Transferring information from CPU memory to GPU memory and then to vertex processing are expensive operations. Through special GPU primitives (see (Toledo & Levy, 2004)), these steps are alleviated, increasing the frame rate.

The tests done with the topological method have demonstrated its efficiency. The positive points of this method are simplicity and execution speed. The ring-traversal implementation is simple and robust for regular CAD meshes. Only a few minutes were necessary to entirely recover the high-order primitives in the PowerPlant data (13 million triangles), considering the model already loaded in memory.

A very important point in recovering CAD data is how to separate the higher-order recovered primitives from the rest. As shown in our results, for models with mixed data (pipes and boxes, for example), we were able to split both types of primitives (implicit surfaces and triangle meshes) without losing information.

However, there are some limitations to the topological approach. This method only extracts three types of higher-order primitives: cylinders, truncated cones and tori. Other implicit surfaces, such as spheres, although not as numerous as tubes and pipes, may also appear in industrial environments. We propose the use of the numerical approach, explained in Section 3, right after running the topological algorithm. This procedure enables the recovery of quadric surfaces that are already segmented inside the model and that were not found in topological step.

Another problem related to CAD mesh recovery is ambiguity. Without any additional information, we cannot ensure that a faceted piece is an approximation of a round one or an exact representation of the original piece. For example, in Figure 11, the octagonal piece on the left should not be represented by a perfect cylinder, but by an octagonal mesh. This is an intrinsic issue when recovering CAD models. In a perfect scenario, round implicit surfaces, such as cylinders, should not have the tessellated representation as their unique information, but unfortunately this is often the case.

We also suggest for future work extending our topological technique to other possible implicit primitives. For example, in our testes we have missed some *sheared cylinders* and *sheared cones*. *CSG primitives* (Constructive Solid Geometry) are complex primitives, formed by Boolean combination of simple ones, that can also be the subject of study.

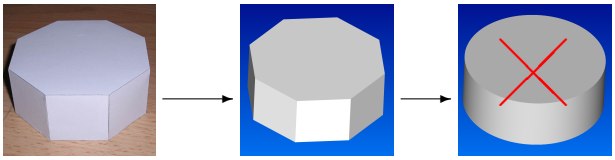

Figure 11 An octagonal piece (*left*) should be represented by an octagonal mesh (*center*). Ideally, the recovery algorithm should not substitute the tessellated model for a perfect cylinder (*right*), but it cannot identify it without extra input information. This is an intrinsic issue when recovering CAD data.

Finally, another open issue is the application of the topological recovery strategy to data sets different from the industrial-environment domain.

ACKNOWLEDGEMENTS

To Petrobras, for allowing the publication of this paper and for supporting part of the present work. To Paulo Ivson, for helping in tuning the algorithm for Petrobras platform models.

REFERENCES

do Carmo, M. (1976). Differential Geometry of Curves and Surfaces. Prentice Hall.

Fitzgibbon, A. W., Eggert, D. W., & Fisher, R. B. (1997). High-level CAD model acquisition from range images. Computer-aided Design, 29(4), pp. 321–330.

Lukacs, G., Marshall, A., & Martin, R. R. (1997). Geometric least-squares fitting of spheres, cylinders, cones and tori.

Nourse, B., Hakala, D. G., Hillyard, R., & Malraison, P. (1980). Natural quadrics in mechanical design. Autofact West, 1, pp. 363–378.

Ohtake, Y., Belyaev, A., & Alexa, M. (2005). Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In SGP '05: Proceedings of the 2005 Eurographics/ACM SIGGRAPH symposium on Geometry processing: Eurographics Association.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., & Seidel, H.-P. (2003). Multi-level partition of unity implicits. ACM Trans. Graph., 22(3), pp. 463–470.

Parvin, B. & Medioni, G. (1986). Segmentation of range images into planar surfaces by split and merge. In CVPR '86: In Proc. of International Conference on Computer Vision and Pattern Recognition (CVPR '86): pp. 415–417.

Petitjean, S. (2002). A survey of methods for recovering quadrics in triangle meshes. ACM Comput. Surv., 34(2), pp. 211–262.

Requicha, A. A. G. & Voelcker, H. B. (1982). Solid modeling: a historical summary and contemporary assessment. IEEE Computer Graphics and Applications, 2, pp. 9–22.

Thompson, W., Owen, J., de St. Germain, H., Stark, S., & Henderson, T. (1999). Feature-based reverse engineering of mechanical parts. pp.$\tilde{5}$7.

Toledo, R. & Levy, B. (2004). Extending the graphic pipeline with new gpu-accelerated primitives. In 24th International gOcad Meeting, Nancy, France. Also presented in Visgraf Seminar 2004, IMPA, Rio de Janeiro, Brazil.

Werghi, N., Fisher, R., Ashbrook, A., & Robertson, C. (1999). Faithful recovering of quadric surfaces from 3d range data.

Wu, J. & Kobbelt, L. (2005). Structure recovery via hybrid variational surface approximation. Computer Graphics Forum (Proceedings of Eurographics '04), (3), pp. 277–284.

Y.Y., C., A.Y.C., N., & H.T., L. (1996). Geometric feature detection for reverse engineering using range imaging. Journal of Visual Communication and Image Representation, 7, pp. 205–216.