PERGAMON Mathematical and Computer Modelling 38 (2003) 945–960

MATHEMATICAL AND COMPUTER MODELLING

www.elsevier.com/locate/mcm

Analysis of STL Files

M. SZILVÁSI-NAGY
Department of Geometry
Mathematical Institute
Budapest University of Technology
P.O. Box 91, H-1521 Budapest, Hungary
szilvasi@math.bme.hu

GY. MÁTYÁSI
Department of Manufacturing Engineering
Faculty of Mechanical Engineering
Budapest University of Technology
H-1521 Budapest, Hungary
matyasi@manuf.bme.hu

Abstract—STL files produced by 3D modelling systems contain triangular facet representation of surfaces and have become standard data inputs of rapid prototyping and manufacturing systems. In rapid prototyping technology, physical objects are produced layer by layer, each layer a 2D cross-section of the 3D mesh in STL format. In current CAD systems, the surface tessellation frequently ends with errors in the data structure as gaps and holes leading to open loops in the cross-sections that cannot be manufactured as layers.

In this paper, we present a method to detect defects in the surface representation and to analyse the shape of the approximated surface by constructing a polyhedral data structure from the STL file.

Keywords—Rapid prototyping, STL file, Polyhedral data structure.

1. INTRODUCTION

Computer-aided design and manufacturing (CAD and CAM) have significantly improved the traditional production design and manufacturing by developing rapid prototyping methods as *stereolithography* (SLA), *selective laser sintering* (SLS), *laminating object manufacturing* (LOM), *three-dimensional printing* (3D printing), etc. [1]. These technologies are capable of directly generating physical objects from CAD databases. They have a common important feature: the prototype is produced by adding materials rather then removing materials. This simplifies the 3D object producing process to 2D layer adding processes such that an object can be produced directly from its computer model.

An object is first designed by a geometric modeller such as solid modeller. The built-in tessellation algorithm creates a simple boundary representation that covers the surface of the solid with triangles. Each triangle is described by an outward normal and the coordinates of three ordered points. Such triangular meshes stored in STL format are used as definitions of geometry of real

This research was supported by the Hungarian National Scientific Foundation OTKA No. T 25613.

0895-7177/03/$ - see front matter Typeset by AMS-TEX
doi: 10.1016/S0895-7177(03)00296-6

solids for several industrial applications, and also for rapid prototyping and manufacturing. A mesh is then sliced into a series of parallel cross-section layers. The layers can be manufactured by different techniques. The SLA machine creates the prototype by tracing layer cross-sections on the surface of a liquid photopolymer pool with a laser beam. SLS uses powder instead of liquid and the laser beam heats up the powder to the sintering temperature so that the powder in a cross-section scanned by the laser is bounded. The LOM processes produce prototypes from bonded paper or plastic sheets. The cross-section layers are cut by a laser beam and glued together. In this way, the model is built layer by layer from bottom to top.

Triangle meshes provide maximum flexibility and efficiency in several other applications and visualization. However, two major difficulties arise with this representation. One difficulty is that many current solid modellers do not create topologically correct tessellations. Errors in the triangulation like gaps and multiple facets result in incorrect cross-sections which cannot be manufactured as layers. The other problem is the high complexity of such meshes. Polygonal models with millions of triangles often require more than available storage and long processing time. Reducing the complexity of surface meshes is therefore a must to be able to handle such model representations. Many research groups have worked in the last decade in surface simplification and published many papers on this topic. Mesh decimation algorithms reduce the number of triangles in a given model while minimally changing its geometric shape. As a byproduct, such algorithms also generate hierarchical representations for highly detailed meshes like *level of detail* (LOD) and *multiresolution* representations. Due to numerical problems, undesirable noise can also arise in the mesh during tessellation. Therefore, developing *discrete fairing* algorithms for triangular meshes to smooth the shape is another central problem. Such algorithms are using results and techniques from differential geometry, variational surface design, and numerical analysis and find their applications in many areas.

However, very few papers have dealt with the detection and correction of topological errors and with the construction of appropriate data structures over the triangular mesh which also contain topological information.

In this paper, a short review is given about the main research issues connected with STL files, the collection of which forms the topic of *discrete surface modelling*. In Section 6, a polyhedral datastructure is presented for detecting topological defects and for shape analysis.

2. SURFACE REPRESENTATION IN STL FORMAT

A triangular mesh generated by tessellation of the boundary surface of a 3D solid is a very simple surface representation. For engineering applications, the mesh must also contain the information of which side of a triangle contains the interior of the solid. The STL file is a triangle facet representation where the vertices of each triangle are ordered to satisfy this requirement. Data of such triangular meshes are suitable for 3D model description, and the STL format has become standard for data input of all types of rapid prototyping systems.

The content of an STL file is a set of oriented triangles, each described by the following numerical data (Figure 1):

$$\begin{array}{ccc} n_x^{(i)} & n_y^{(i)} & n_z^{(i)} \\ p_{1x}^{(i)} & p_{1y}^{(i)} & p_{1z}^{(i)} \\ p_{2x}^{(i)} & p_{2y}^{(i)} & p_{2z}^{(i)} \\ p_{3x}^{(i)} & p_{3y}^{(i)} & p_{3z}^{(i)} \end{array}$$

where $(n_x^{(i)}, n_y^{(i)}, n_z^{(i)})$ are the coordinates of the outward normal of the i^{th} facet and $(p_{kx}^{(i)}, p_{ky}^{(i)}, p_{kz}^{(i)})$ are the coordinates of the k^{th} vertex of the i^{th} triangle, $i = 1, \ldots, N$ (N is the number of triangles). For example, an STL file reduced to three facets is as follows.

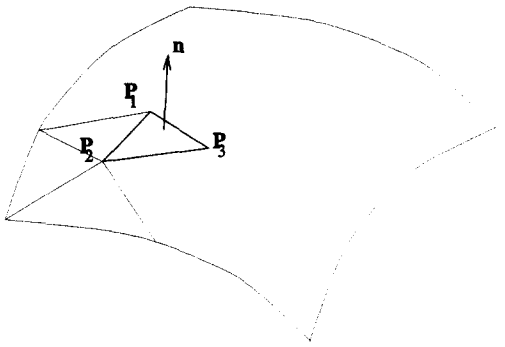

Figure 1. Face in STL format.

```
solid ascii
facet normal  −2.902847e − 01  −9.569403e − 01  0.000000e + 00
outer loop
vertex  6.173166e + 00  7.612047e − 00  0.000000e + 00
vertex  8.049097e + 00  1.921472e − 01  2.000000e + 01
vertex  6.173166e + 00  7.612047e − 01  2.000000e + 01
endloop
endfacet
facet normal  −2.902847e − 01  −9.569403e − 01  0.000000e + 00
outer loop
vertex  6.173166e + 00  7.612047e − 01  0.000000e + 00
vertex  8.049097e + 00  1.921472e − 01  0.000000e + 00
vertex  8.049097e + 00  1.921472e − 01  2.000000e + 01
endloop
endfacet
facet normal  9.951847e − 01  9.801714e − 02  −0.000000e + 00
outer loop
vertex  2.000000e + 01  1.000000e + 01  0.000000e + 00
vertex  1.980785e + 01  1.195090e + 01  0.000000e + 00
vertex  1.980785e + 01  1.195090e + 01  2.000000e + 01
endloop
endfacet
end solid
```

The file does not contain any topological information like links, pointers to another element, or proximity. Each vertex is written by its coordinates in the file as many times as it occurs in the mesh. This fact may cause problems with vertex identification depending on the numerical representation of real numbers. The STL file does not show whether the mesh is the result of a correct triangulation.

We say that a surface triangulation is a connected, knot-to-knot triangulation (shortly correct) if it is a finite set of triangles satisfying the following conditions [2].

- Each edge in the triangulation is shared by at most two triangles.
- A vertex in the triangulation can be shared by any number of triangles.
- Each triangle has at least one point in common with another triangle (connectivity; requires at least two triangles).
- If a vertex of a triangle is shared by a second triangle, then this point is also a vertex of the second triangle (knot-to-knot property).
- No triangle has an intersection with the interior of any other triangle (no piercing, no overlapping).

A correct triangulation represents a so-called oriented two-manifold, where each surface point has an open neighbourhood which is homeomorph to an open disk of the Euclidean plane.

Mathematical errors in the triangulation and also the limited precision of number representations can lead to topological defects in the mesh. In composite models the degree of discretization of different surfaces can be different. In models, e.g., with blending surfaces the description of surface boundaries may lead to differences in the coordinates of the same vertex shared by triangles lying on different surfaces (Figure 2). Similarly, the computation of the line of intersection of two surfaces can lead to errors in knot-to-knot connection (Figure 3). Raising the accuracy of the computation here does not help. In contrast, due to higher tessellating resolution, smaller triangles are generated, and triangles may collapse into a line whenever one of its sides becomes too small. At points of high curvature oscillation a hole may arise (Figure 4). High degree edges can be generated and shared by more triangles particularly at tangential positions (Figure 5). A corrupted disk around a vertex can arise (Figure 6), and reversed oriented normals can occur if too long and narrow triangles are generated. All these errors affect the shape of the layers computed as plane sections of the mesh, and distort the manufactured object.

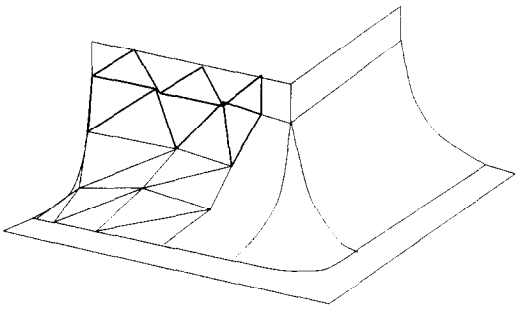

Figure 2. Triangulation of blending surface.

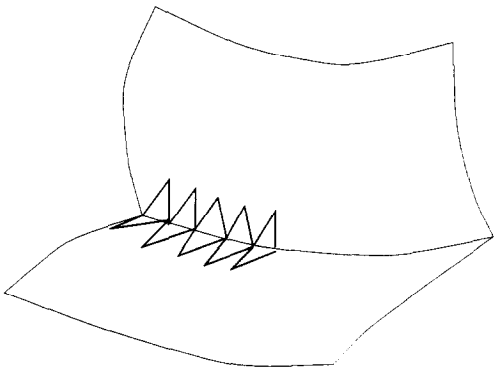

Figure 3. Gaps along a line of intersection.

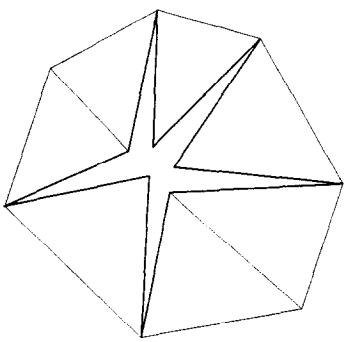

Figure 4. Hole at a vertex.

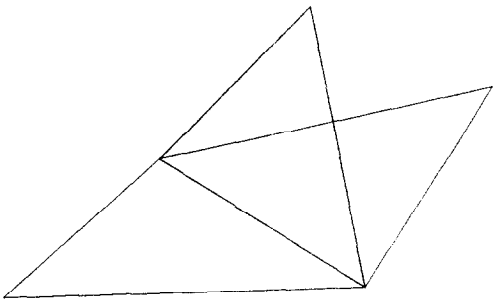

Figure 5. Invalid edge.

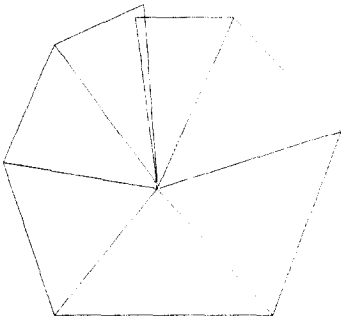

Figure 6. Corrupted disk around a vertex.

High degree edges and incorrect disks are detected and characterized in [3], and interactive methods are suggested for correcting such topological defects. The investigations show that no generic algorithm can be developed for the resolution of nonmanifold edges, and in many cases remodelling a valid solid and new tessellation are necessary.

A technique for filling gaps by adding triangles to the mesh is described in [4] without changing the structure of the original tessellation of the input parts.

3. MESH SIMPLIFICATION TECHNIQUES

Surface tessellations and many other engineering and computer graphics applications routinely generate geometric models consisting of large numbers of triangles. The direct use of high density meshes leads to long processing time and to inefficient manipulations. Therefore, simplification operations are required to reduce the number of triangles such that the coarser mesh preserves the original topology of the mesh and forms a good geometric approximation to the original mesh. In general, there is no one ideal solution to the problem of mesh simplification. Rather, it is a user-specified compromise between conciseness of the representation and goodness of shape description. For different applications a large number of data reduction algorithms have been developed and published in the last decade.

Data reduction can be made by deleting vertices or edges from the mesh or substituting triangles by points, etc. Each of these operations changes the topology and the geometry of the mesh in the neighbourhood of the deleted element. Most algorithms operate locally, inside well-defined regions without propagating through the mesh. As a result, the chain effects so difficult to track are substituted with local transformations of the triangulation. Then, the process repeats, until some termination condition is met. Usually, the termination criterion is specified as a percent of reduction of the original mesh, or as some decimation value, or is controlled by the user interactively.

A widely used operation in preprocessing, or between iteration steps, is edge swapping which inverts the diagonal of a quadrilateral in order to minimize small oscillations that might be present in the input mesh. The swapping of an edge proceeds as follows [5]. Two triangles sharing the

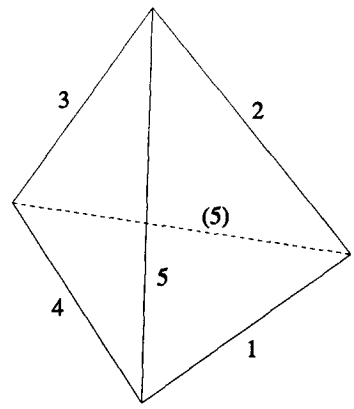

Figure 7. Checking five dihedral angles for edge flipping.

edge and forming a quadrilateral are considered. Five dihedral angle variations are measured between the pairs of adjacent outward normal vectors for the quadrilateral (Figure 7). The edge is accepted for swap if the mean variation is reduced when swapping the edge.

For characterization of the mesh geometry and for edge classification *feature edges* are used in many algorithms. An edge is called a feature edge, if the dihedral angle measured with the outward normals of the two triangles sharing this edge is larger than a specified planarity-threshold value [6]. A set of connected feature edges forms a characteristic line (or feature line).

A mesh simplification algorithm based on edge operations also classifies the vertices with the number of feature edges surrounding the vertex [6]. A vertex is lying on a feature line if two feature edges are adjacent to it and is a corner if three or more feature edges meet in it. The retriangulation of planar or nearly planar regions is performed by collapsing the edges under a specified planarity threshold within well-defined disjunct regions. A collapsed edge is substituted by a vertex and the surrounding polygon is retriangulated (Figure 8). However, the validation of the retriangulation requires sophisticated topological checks. In each step of the iteration, feature lines are also simplified according to linearity thresholds. The simplification algorithm starts with strict threshold values that only very planar regions and very straight edges will be simplified. Then, the threshold values are slowly relaxed as the iterations proceed.

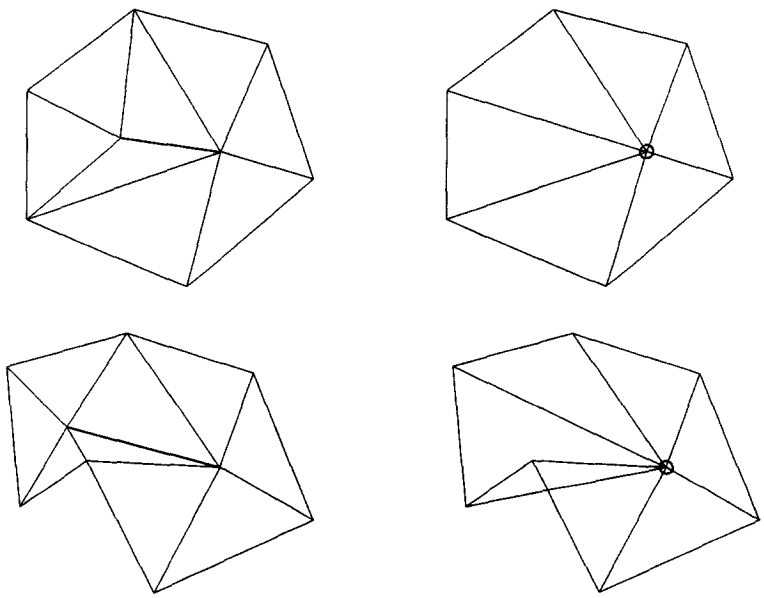

Figure 8. Valid and invalid triangulations after edge collapse.

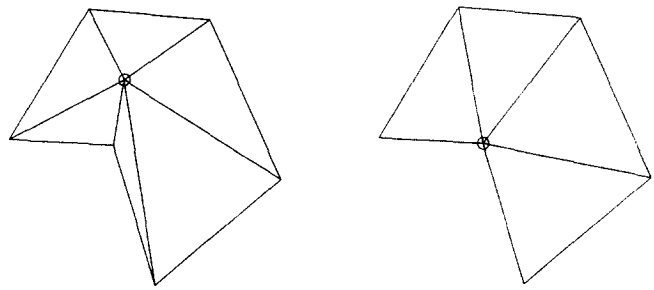

Figure 9. Retriangulation after deleting a vertex.

A mesh decimation algorithm is described in [7] by removing vertices that pass a distance criterion. For each vertex the distance to an average plane of the triangles surrounding the vertex is computed. If a vertex is not lying on feature edges and is within a specified distance to the average plane, it is deleted, and the created hole must be triangulated (Figure 9).

The criterion used in [8] to select the candidate vertices for deletion is based on discrete approximation of the Gaussian curvature around a vertex. The basic curvature approximation at a vertex is given by

$$K_1 = 2\pi - \sum_i \gamma_i,$$

where γ_i are the angles of the faces meeting at the vertex (Figure 10). The second approximation takes into account the areas A_i of the faces meeting at the vertex

$$K_2 = \frac{K_1}{(1/3)\sum_i A_i}.$$

The third approximation takes into account the shape differences of the faces meeting at the vertex.

$$K_3 = K_1 \frac{\sum_i A_i}{(1/2)\sum_i A_i - (1/8)\sum_i \cot g\gamma_i l_i},$$

where l_i is the length of the edge of the i^{th} face opposite to the vertex. All the vertices in the mesh are sorted from the smallest absolute value of the Gaussian curvature up to the highest. This allows us to remove first the vertices within plane or approximate plane regions. For identification of regions, error zones are defined for each vertex, and a semiglobal strategy is given to control the restoration of the geometry.

Removing of triangles is proposed in [2] as local operation on a triangle mesh. Each triangle of the mesh is weighted according the discrete principal curvatures at its vertices combined with an angle weight. The iterative algorithm removes the triangle with the lowest weight and retriangulates the affected region in each step. Thus, the local density of triangles reflects the surface's

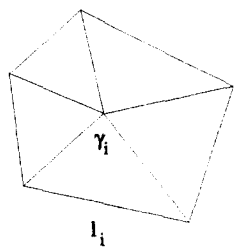

Figure 10. Data for curvature approximation.

curvature. At the end, surface regions with low curvature are represented with relatively larger triangles. The angle weight σ_i of a triangle T_i is given by

$$\sigma_i = 2\left(\left(\sum_{j=1}^{3}\cos\alpha_j\right) - 1\right),$$

where α_j, $j = 1, 2, 3$ are the interior angles of T_i. It is shown that $\sigma_i \in [0, 1]$, and its use helps to avoid long and narrow triangles. The curvature weight ρ_i of a triangle T_i is defined by the sum of the absolute curvatures at its vertices,

$$\rho_i = \sum_{j=1}^{3}\left|\kappa_1^j\right| + \left|\kappa_2^j\right|.$$

Finally, the triangle weight of the i^{th} triangle is defined as the product

$$\omega_i = \sigma_i\rho_i.$$

The point replacing the removed triangle and the principal curvatures at this point are computed on a quadratic surface constructed as least-squares polynomial approximation to the vertices of the surrounding triangles. The local coordinate system for the construction of such a bivariate function is chosen in a heuristic way considering the local topology of the mesh.

Mesh simplification is regarded as an optimization problem with an energy function that penalizes meshes with a large number of vertices, and directly measures deviation of the final mesh from the original [9]. It does not penalize sharp dihedral angles in the mesh, in order to preserve feature lines and curvature variations. The energy function over the triangle mesh has three components,

$$E = E_{\text{dist}} + E_{\text{rep}} + E_{\text{spring}}.$$

The distance energy E_{dist} is equal to the sum of squared distances of the actual vertices to the original mesh. The representation energy E_{rep} is proportional to the number of vertices, and the spring energy E_{spring} is proportional to the sum of squared lengths of all edges. The optimization proceeds iteratively alternating between two subproblems. For solving the mesh decimation subproblem three elementary transformations, edge collapse, edge split, and edge swap, are defined as legal moves on the mesh. Then, a sequence of randomly selected legal moves is performed. A legal move is accepted if the energy function decreases. After achieving a local minimum the other subproblem, the optimization of vertex positions in the reduced mesh is solved as a linear least squares problem. In the evaluation of E_{dist} sophisticated methods are needed, e.g., the projection of actual vertices onto the original mesh and the selection of a nearby subset for each vertex. The topological validity of the resulting mesh is checked by the user *a posteriori*. This mesh simplification and optimization algorithm is extended for generating a set of nested meshes at various levels of detail in [10].

4. MESH REPRESENTATIONS

In the mesh simplification processes mentioned above, only one mesh is constructed, and intermediate meshes cannot be retrieved. In many applications to improve efficiency, a representation is needed that allows transformation between meshes with different *levels of detail* (LOD), also called levels of resolution.

In *progressive mesh* (PM) representation, which is one realization of multilevel structures, an arbitrary mesh $\hat{M}$ is stored as a much coarser mesh M^0 together with a sequence of n detail records that indicate how to incrementally refine M^0 exactly back into the original mesh $\hat{M} = M^n$ [10]. If the triangle mesh $\hat{M}$ is simplified through a sequence of n edge collapse transformations to

yield a much simpler base mesh M^0, then using the inverse transformation, called vertex split, the process can be reversed.

$$\left(\hat{M} = M^n\right) \xrightarrow{\text{ecol}_{n-1}} \cdots \xrightarrow{\text{ecol}_n} M^1 \xrightarrow{\text{ecol}_0} M^0$$

$$M^0 \xrightarrow{\text{vsplit}_0} M^1 \xrightarrow{\text{vsplit}_1} \cdots \xrightarrow{\text{vsplit}_{n-1}} \left(M^n = \hat{M}\right),$$

where ecol and vsplit stand for edge collapse and vertex split, respectively. The tuple (M^0, $\{\text{vsplit}_0, \ldots, \text{vsplit}_{n-1}\}$) forms a PM representation of $\hat{M}$. Of course, the vertex split must store the positions of the two split vertices as well as other attributes associated with the mesh. The resulting sequence of meshes $M^0, \ldots, M^n = \hat{M}$ can be quickly traversed with the help of a dependency graph. In this way, geometric information can be extracted at any level of detail. A mesh itself is described by a vertex-oriented data structure, where a structured cycle of adjacent faces is listed at each vertex. The data structure contains not only geometric data but also application specific attributes, like materials, textures, etc.

A different LOD representation is implemented for structured meshes in [11]. First, the construction of a structured mesh from the unstructured triangular mesh is needed. The structured mesh is in this case a quad tree with the coarsest level in its root. The nodes represent regions of the mesh, from which vertices are sampled uniformly. The regions are subdivided into four subregions until the finest level is achieved. To each node representing a region, weighted average values of the coordinates and other attributes of the sampled mesh points are attached. If the difference between these values and those of the four subregions is larger than an error threshold, then the region is subdivided and four subquadrants are created to the node. In this way, the vertices of the mesh are mapped to an adaptive hierarchical structure according to the density of the mesh vertices. When the process terminates, the error of each node is accumulated, re-estimated, and attached to the node. The level-of-detail extraction algorithm extracts one level from the multilevel model according to a given error tolerance. In rapid prototyping applications nodes at different z-values are extracted, and closed boundaries are created for producing valid layers. In this way, the number of layers can be reduced, while the shape is preserved at critical locations too.

Several techniques have been suggested in the literature for multilevel structures and for criteria defining the levels of resolution. The common task of these multilevel mesh representations is to enable manufacturing of mixed rough and fine details effectively. A flexible multilevel representation based on a vertex decimation approach is proposed in [12]. A mesh simplification step includes the removal of a candidate vertex, the retriangulation of the created hole (described in detail), and the optimization of this triangulation by edge swaps. A local error is evaluated from distances between specified points of old and new facets in each step. To each facet a global error is attached which is incrementally updated during the simplification by choosing the maximum value of certain local errors. The criterion of vertex selection combines the local error attached to the actual step and the global errors on the affected faces. The multilevel representation finally is a set of all triangles that were generated during the whole decimation process and having two associated global errors: the starting value, when it was generated, and the end value, when it was removed. An effective structure for such a set is an interval tree. The extraction of a mesh at a given error retrieves all the triangles whose global error interval covers a user-defined precision (error value). The authors call their representation *multiresolution representation*, because the user can extract a mesh (similarly to multiresolution management) at the "best" level of detail in real time. However, in widely accepted terminology, only multilevel structures used for continuous representation are usually called multiresolution representation. With a common continuous multiresolution method, the level-of-detail curves are represented as B-spline curves, and the remainder sets (detail records) between the levels are represented by

transformation functions, frequently by wavelet curves. The second particular technique is called multiresolution analysis [11].

5. DISCRETE FAIRING

While working with triangle meshes, besides the former mentioned difficulties, another central problem arises, namely the *smoothness* of the mesh. Since each mesh is considered as one individual level of detail approximation of a smooth continuous surface, undesirable noise, and uneven edges have to be removed. The smoothing or *discrete fairing* of high-resolution meshes is based on the proper definition of discrete curvature and on appropriate filter operators using different energy functionals.

During the construction of a multiresolution representation a discrete fairing method is applied after each mesh decimation step (by edge collapses) on the intermediate triangle mesh in [13]. The method minimizes the most frequently used energy functionals, the membrane energy punishing large surface area

$$E_M(f) = \int_S \left(f_u^2 + f_v^2\right)\, dS,$$

and the thin plate energy punishing strong bendings

$$E_T(f) = \int_S \left(f_{uu}^2 + 2f_{uv}^2 + f_{vv}^2\right)\, dS,$$

where $f(u,v)$ is the bivariate function which describes the surface S. Note that in the case of isometric (u,v) parameterization the thin plate energy turns out to be equal to the total curvature

$$\int_S \left(\kappa_1^2 + \kappa_2^2\right)\, dS,$$

where κ_1 and κ_2 are the principal curvatures.

The variational calculus leads to the following simple characterizations of the minimum energy surfaces:

$$\Delta f = f_{uu} + f_{vv} = 0$$

or

$$\Delta^2 f = f_{uuuu} + 2f_{uuvv} + f_{vvvv} = 0,$$

respectively, called Euler-Lagrange equations. The discrete fairing approach replaces the difference operators with divided difference operators. To construct these operators, an appropriate local parameterization has to be given in the vicinity of each vertex. Here, the symmetric parameterization is chosen

$$(u_j, v_j) = \left(\cos\left(2\pi\frac{j}{n}\right), \sin\left(2\pi\frac{j}{n}\right)\right), \qquad j \in N_1(i),$$

where $n = \#N_1(i)$ is the valence of the center vertex $\mathbf{x}_i$ and $N_1(i)$ denotes its one-ring neighbors (Figure 11). In the case when the edges in the mesh are equal long (isometric mesh), with this special parameterization the discrete analogon of the Laplacian Δf is the umbrella operator

$$U(\mathbf{x}_i) = \frac{1}{n} \sum_{j \in N_1(i)} \mathbf{x}_j - \mathbf{x}_i,$$

where $\mathbf{x}_j$ are the direct neighbors of $\mathbf{x}_i$. The umbrella operator can be applied recursively leading to

$$U^2(\mathbf{x}_i) = \frac{1}{n} \sum_{j \in N_1(i)} U(\mathbf{x}_j) - U(\mathbf{x}_i),$$

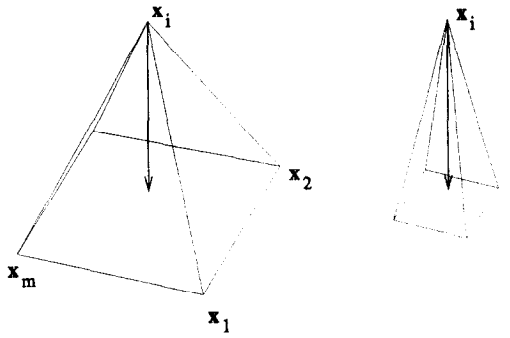

Figure 11. The same umbrella operator at two vertices.

as a discretization of $\Delta^2 f$. When minimizing the discrete version of E_M under boundary conditions, a closed boundary polygon is fixed and the membrane spanned in between is computed. For the minimization of E_T, two rings of boundary vertices are used; i.e., a closed strip of triangles is fixed. All internal vertices can be moved freely to minimize the global energy. The properly chosen boundary conditions imply positive definiteness of the energy functional and guarantee the convergence of the iterative solving algorithm. The characteristic linear system for the corresponding unconstrained minimization problem has the rows $U(\mathbf{x}_i) = 0$ or $U^2(\mathbf{x}_i) = 0$, respectively, for the free vertices $\mathbf{x}_i$. An iterative solving method approaches the optimal solution by solving each row of the system separately and cycling through the list of free vertices until a stable solution is reached. In case of the membrane energy E_M, this leads to the local update rule

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + U(\mathbf{x}_i),$$

and accordingly for the thin plate energy E_T to

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - \frac{1}{\nu} U^2(\mathbf{x}_i),$$

with the "diagonal element"

$$\nu = 1 + \frac{1}{n} \sum_{j \in N_1(ij)} \frac{1}{m},$$

where $n = \#N_1(i)$ and $m = \#N_1(ij)$ are the valences of the center vertex $\mathbf{x}_i$ and its j^{th} neighbor, respectively.

In the multilevel smoothing process, an appropriate mesh reduction has to be implemented which generates intermediate meshes with strong coherence in the length of edges, and then the smoothing method above can be applied in each level.

However, the umbrella operator does not truly represent a Laplacian, and may lead to the same approximated Laplacian for different configurations (Figure 11). Its use may result in some unnatural deformations of the mesh. To improve the approximation, a scale-dependent umbrella operator is used for irregular meshes in [14]. The smoothing is implemented by a diffusion process

$$\frac{df}{dt} = \lambda \Delta f.$$

This differential equation on the mesh X with vertices $\mathbf{x}_i$ is written in the form

$$\frac{d\mathbf{x}_i}{dt} = \lambda L(\mathbf{x}_i) \qquad \text{or} \qquad \frac{\partial X}{\partial t} = \lambda L(X),$$

where L is the Laplacian operator and t is the time. If the Laplacian is linearly approximated by the umbrella operator then a sequence of meshes (X^n) can be constructed by integrating the diffusion equation with a simple explicit Euler scheme (forward Euler method), yielding

$$X^{n+1} = (I + \lambda\, dt\, U) X^n.$$

With the umbrella operator the stability criterion requires $\lambda\, dt < 1$ which needs a large number of integrations to produce noticeable smoothing. Instead of this an implicit integration (backward Euler method) is used

$$X^{n+1} = X^n + \lambda\, dt\, U\left(X^{n+1}\right).$$

This leads to the linear system

$$(I - \lambda\, dt\, U)X^{n+1} = X^n.$$

Consequent smoothing will then be obtained by increasing the value $\lambda\, dt$, and solving the linear system for the free mesh vertices $\mathbf{x}_i$. The matrix $A = I - \lambda\, dt\, U$ is sparse, namely, the i^{th} line contains $n = \#N_1(i)$ nonzero elements, and thus, the solution requires only linear time computation.

For irregular meshes a simple generalization of the umbrella operator is defined, the scale-dependent umbrella operator

$$\tilde{U}(\mathbf{x}_i) = \frac{2}{E} \sum_{j \in N_1(i)} \frac{\mathbf{x}_j - \mathbf{x}_i}{|e_{ij}|}, \qquad \text{with } E = \sum_{j \in N_1(i)} |e_{ij}|,$$

where $|e_{ij}|$ is the length of the edge $\mathbf{x}_j - \mathbf{x}_i$. This operator is not linear. But, during a typical smoothing, the length of the edges does not change very much. Therefore, the coefficients of the matrix $A = (I - \lambda\, dt\, \tilde{U})$ can stay constant during an integration step, and the same method can be applied as in the linear case. Note that the diffusion process leads to shrinking, and therefore, a volume preserving scaling is carried out on the mesh after smoothing.

Though the scale-depending umbrella operator does not distort the surface, applying the Laplacian operator leads to sliding of the mesh even in flat or nearly flat regions depending on the parameterization.

Curvature flow smoothes the surface by moving along the normal $\mathbf{n}$ with a speed equal to the mean curvature $\bar{\kappa} = (\kappa_1 + \kappa_2)/2$,

$$\frac{d\mathbf{x}_i}{dt} = -\bar{\kappa}_i \mathbf{n}_i.$$

The discrete form of the curvature normal $\bar{\kappa}\mathbf{n}$ of the triangle mesh is given by

$$\bar{\kappa}\mathbf{n} = \frac{1}{4 \sum_j A_j} \sum_{j \in N_1(i)} (\cot \alpha_j + \cot \beta_j)(\mathbf{x}_j - \mathbf{x}_i),$$

where α_j and β_j are the two angles opposite to the edge $\mathbf{x}_i$, $\mathbf{x}_j$ in the two neighbouring triangles, and A_j is the area of the j^{th} triangle having $\mathbf{x}_i$ as a vertex (Figure 12).

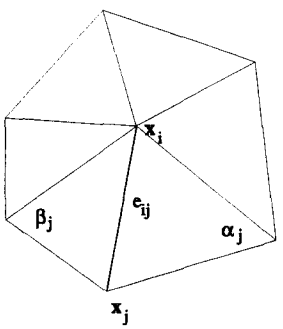

Figure 12. One edge in the curvature operator.

The solution of the differential equation by implicit integration leads again to a linear system for the mesh points

$$(I - \lambda\, dt\, K)X^{n+1} = X^n,$$

where K represents the matrix of the curvature normals. The mesh smoothing with curvature flow maintains the regions of constant curvature.

Remark that the discrete form of the curvature normal has been given also in [15] with different approach.

6. A POLYHEDRAL DATA STRUCTURE OVER THE TRIANGULAR MESH

We present an edge-oriented polyhedral data structure over the triangular mesh which is suitable for the detection of topological errors and for the computation of well-defined cross-sections with closed boundary curves. The necessity of such a data structure has been discussed in Section 2. Though several polyhedral data structures have been published in the CAD literature, they depend very much on the algorithms constructed for the solution of specific geometrical problems. In the literature of rapid prototyping mostly vertex-oriented data structures are used [7,10,12] but for computing plane sections, an edge-oriented data structure is more suitable [16,17]. Our data structure contains also the face-oriented description inherited from the STL file, and therefore, it contains redundant data.

When reading the data of a mesh from STL file, we have to check the identical vertices stored in several triangles by their coordinates. Of course, this can be done within a prescribed threshold by computing the distance of the new input point to every processed point. Therefore, the collection of the vertices takes $O(N^2)$ time, where N is the total number of vertices in the STL file. Parallel to this, the original data structure of a triangle will be transformed into a face description of the polyhedral data structure.

The first data field is the vertex list containing each mesh vertex exactly one time.

$$\mathbf{V} : \{V_i \rightarrow (x, y, z)_i\}, \qquad i = 1, \ldots, N_v,$$

where N_v denotes the number of the points and $(x, y, z)_i$ are the Cartesian coordinates of the i^{th} point. The vertex list contains numerical (also called geometrical) information about the 3D object. The vertices of a face are then determined by references (also called pointers) to the corresponding vertices denoted by V_j. They are assumed to be given in an oriented vertex cycle such that each face normal $\mathbf{n}_i$ is oriented outwards. Unfortunately, in a defected STL file not only triangular faces are generated. Consequently, the face description has to allow more than three vertices. The coordinates of the normal vector are explicitly given in the STL-file, and all the other face data can be generated while reading the file. A face data record in our representation is the following:

$$\mathbf{F} : \{F_i \rightarrow (\mathbf{n}, c, (V_1, V_2, \ldots, V_c), \text{edge}, \text{att})_i\}, \qquad i = 1, \ldots, N_f,$$

where c_i is the number of the vertices, edge_i determines a starting edge for the i^{th} face pointing to the index in the edge list, and att_i is an integer number for characterizing some face attribute (Figure 13). Since the number of the sides is equal to the number of the vertices on a face, we define so-called *oriented half edges* while identifying the vertex indices [18]. Each half edge determined by its starting and end points is sequentially accumulated in the edge list together with the pointer to the next edge in the edge cycle of the face and with the face's serial number (index) in the face list. In a correct STL representation each edge of the object will occur twice as oppositely oriented half edges. The construction of the edge list can be finished after processing the file. Namely, attaching (linking) the half edges into pairs is a searching procedure over the

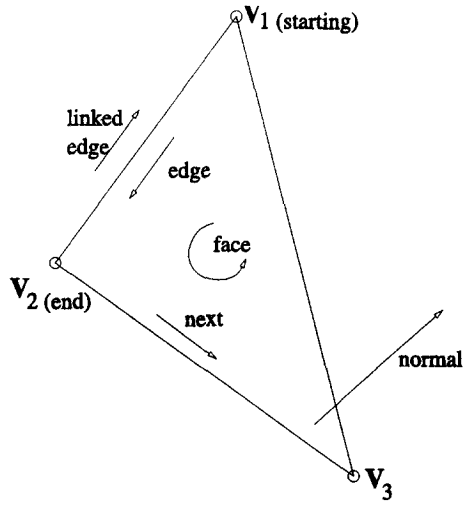

Figure 13. The polyhedral datastructure.

whole data structure for identical end points. This searching needs $O(N_e^2)$ time, where N_e is the length of the edge list. The data field of the half edges is as follows:

$$\mathbf{E} : \{E_i \to (V_{\text{start}}, V_{\text{end}}, \text{next}, \text{face}, \text{link})_i\}, \qquad i = 1, \ldots, N_e,$$

where each element of the field is a pointer to the starting point, end point, next edge, containing face, and linked half edge, respectively. Such data which define proximities are usually called topological data of the object.

Technical problems arise in the construction of this data structure when the file does not have a correct STL format and when the number of faces is too large. A numerical difficulty is the choice of an appropriate tolerance to distinguish different mesh points. If this distance threshold value between points is either too big or too small, then the original topology of the mesh is distorted.

Though the data structure is rather big, it is very effective for several algorithms.

An object described by this data structure can be *visualized* by its edges or by its faces, because of the direct access to the corresponding data.

Finding *holes* and *gaps* can be made by a straightforward collection of so-called boundary edges, i.e., elements of the edge list which are not linked to another half edge. This algorithm takes linear $O(N_e)$ time.

Collecting *feature edges* according to dihedral angles between faces sharing an edge takes linear time as well. Namely, the containing face of an edge, its linked edge, and the containing face of it can be found in constant $O(1)$ time for each element of the edge list (Figure 14).

$$E_i \to F_j \to \mathbf{n}_j \qquad \text{and} \qquad E_i \to \text{link}_i \to F_k \to \mathbf{n}_k.$$

Determining the *direct neighbourhood of a triangle*, i.e., the faces having a common edge or a common vertex with it, requires simply to step around adjacent faces with the help of *link* and *next* of the corresponding edge record. This procedure requires $O(N_e)$ time in the worst case, e.g., at the vertex of a pyramid. Let E_i be one edge of the "central" triangle and face_1, $\text{face}_2, \ldots$ its adjacent triangles.

$$\begin{array}{l} E_i \to \text{link} \to \text{face}_1 \\ \qquad \downarrow \\ \texttt{repeat}^* \ \text{next} \begin{array}{ll} \nearrow (1) & \text{link}^* \to \text{face}_i \\ \searrow (2) & \text{next} \to \text{link}^* \to \text{face}_i \end{array} \texttt{until ring closed}, \end{array}$$

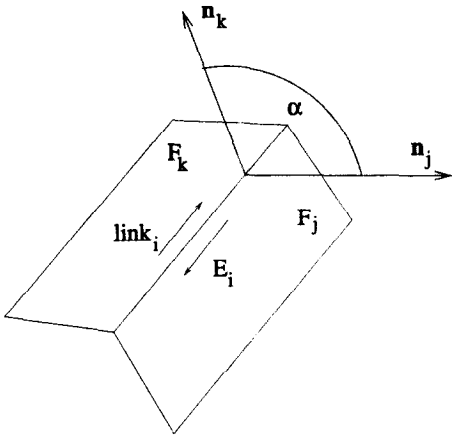

Figure 14. Characterization of feature edges.

where commands (1) are carried out if the edge determined by *next* is no edge of the central triangle. In this case, more than three edges meet at the endpoint of E_i. Otherwise commands (2) choose the next adjacent triangle at the next vertex of the central triangle.

Finding *planar regions* by collecting coplanar faces can be made in $O(N_f \log N_f)$ time by sorting the face list according their normal vectors and then checking the parallel faces for a common point.

We have implemented the above-mentioned algorithms in the programming language Java (1.2) on a small workstation with Linux operating system. The procedure of visualization shows boundary edges, feature edges, and coplanar triangles with different colors. Holes are described by closed chains of boundary edges.

7. CONCLUSIONS

We have given a short review about techniques in discrete surface modeling which are needed in rapid prototyping, such as management of STL files, analysis of a triangle mesh in STL format to detect and to heal topological defects, construction of appropriate data structures for shape description, and smoothing of meshes. We have selected only a few papers from the related literature which are focusing on the geometry of these problems presenting mathematical tools, and give a detailed review of the discussed topics. The most important techniques for managing triangle meshes are collected in Sections 3 and 5. Three basically different mesh representations are mentioned in Section 4, in order to give a rough build about a very large field of this actual research topic. We have not mentioned the relation of these questions to reverse engineering which is also a very actual research field using all the mathematical tools of geometric modeling for model reconstruction.

Section 6 contains our new result, the construction of an edge-oriented data structure from a triangle mesh. This data structure includes numerical data and a full topological description of the meshed model. It is suitable for the solution of the problems emerging in rapid prototyping and for shape analysis. We have implemented the algorithms in the programming language Java (1.2) on a small workstation with Linux operating system.

REFERENCES

1. X. Yan and P. Gu, A review of rapid prototyping technologies and systems, *Computer-Aided Design* **28**, 307–318, (1996).
2. B. Hamann, A data reduction scheme for triangulated surfaces, *Computer Aided Geometric Design* **11**, 197–214, (1994).
3. D. Wang and J.A. Goldak, Correcting topological defects of tesselations, In *Canadian Conference on Computational Geometry CCCG*, (Edited by F. Fiala, E. Kranakis and J.-R. Sacks), pp. 50–55, Carleton University Press, (1996).

4. G. Barequet and M. Sharir, Filling gaps in the boundary of a polyhedron, *Computer Aided Geometric Design* **12**, 207–229, (1995).
5. N. Dyn, D. Levin and S. Rippa, Data dependent triangulations for piecewise linear interpolation, *IMA Journal of Numerical Analysis* **10**, 137–154, (1990).
6. M.-E. Algorri and F. Schmitt, Mesh simplification, *Eurographics '96* **15** (3), C-77—C-86, (1996).
7. W.J. Schroeder, J.A. Zarge and W.E. Lorensen, Decimation of triangle meshes, *Computer Graphics* **26**, 65–69, (1992).
8. P. Veron and J.C. Leon, Shape preserving polyhedral simplification with bounded error, *Comput. & Graphics* **22**, 565–585, (1998).
9. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Mesh optimization, In *Proc. of SIGGRAPH '93, Annual Conference Series on Computer Graphics*, pp. 19–25, ACM SIGGRAPH, Addison-Wesley, (1993).
10. H. Hoppe, Efficient implementation of progressive meshes, *Comput. & Graphics* **22**, 27–36, (1998).
11. A. Fischer, Multi-level models for reverse engineering and rapid prototyping in remote CAD systems, *Computer-Aided Design* **32**, 27–38, (2000).
12. A. Ciampalini, P. Cignoni, C. Montani and R. Scopigno, Multiresolution decimation based on global error, *Visual Computer* **13**, 228–246, (1997).
13. L. Kobbelt, S. Campagna, J. Vorsatz and H.-P. Seidel, Interactive multi-resolution modeling on arbitrary meshes, In *Proc. of SIGGRAPH '98, Annual Conference Series on Computer Graphics*, pp. 105–114, ACM SIGGRAPH, Addison-Wesley, (1998).
14. M. Desbrun, M. Meyer, P. Schröder and A.H. Barr, Implicit fairing of irregular meshes using diffusion and curvature flow, In *Proc. of SIGGRAPH '99, Annual Conference Series on Computer Graphics*, pp. 317–323, ACM SIGGRAPH, Addison-Wesley, (1999).
15. U. Pinkall and K. Polthier, Computing discrete minimal surfaces and their conjugates, *Experimental Mathematics* **2** (1), 16–36, (1993).
16. M. Szilvási-Nagy, An algorithm for determining the intersection of two simple polyhedra, *Computer Graphic Forum* **3**, 219–225, (1984).
17. M. Szilvási-Nagy, Two algorithms for decomposing a polyhedron into convex parts, *Computer Graphics Forum* **5**, 197–202, (1986).
18. M. Mäntylä, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MD, (1988).