3D POINT LOCATION FOR SPECIAL FEATURE PRESERVATION DURING SURFACE SMOOTHING

M.M. Madi & D.J. Walton

Department of Computer Science
University of Manitoba, Winnipeg, Manitoba R3T 2N2
Canada
{mmadi,walton}@cs.umanitoba.ca

ABSTRACT

The development of a smooth geometric model from a crudely approximated object is often sought in CAD/CAM, computer-aided geometric design and similar applications. For example, many surface fitting techniques may have as their input data that are crudely sampled from free form objects to digitally reproduce such objects to look smooth again. However, regardless of the surface fitting technique used whose choice is usually application dependent, there are always particular features on the approximated models that should not undergo the smoothing procedures to maintain visual realism. A technique for interactively *marking* such sampled points for preservation and special treatment is described.

Keywords: 3D Point Location, Surface Fitting and Refinement.

INTRODUCTION

The display of 3D *modeled* objects on a computer screen is usually the final phase of a long process of data manipulation. A scenario that is relevant to the application discussed below involves obtaining sampled data points from a physical object by laser scanning or a coordinate measuring machine [Seile96]. A crude wire-frame triangulated model is then produced from the obtained sampled data using techniques such as that shown in [Fang95]. Next, refinement algorithms are used to fit triangular patches to the produced triangulated model to generate a refined triangular model consisting of many smaller triangles. A common method of refinement is to first fit surfaces to the crude triangular patch [Hagen96, Neamt94]. Smaller triangles are then generated from the fitted surface. These refinement algorithms are applied uniformly on all sampled points with the intention of preserving positional and tangent plane continuity across the triangular patches to obtain visual smoothness.

In the technology of layered manufacturing, which uses stereolithography apparatus (SLA) for producing physical models of products or parts in a short time (rapid prototyping), the generated refined triangles are input into the SLA to build a plastic model of the triangulated object by curing liquid polymer into a solid with an ultra violet (UV) laser.

Because features such as cusps or tangent discontinuity are sometimes inherent in the object being modeled, it is not always desirable that the smoothing process be applied uniformly to all vertices and edges composing the model. This has the effect of producing artificial results that are not intended on the produced physical model. It would be useful to have an interactive and immediate way of locating vertices and edges that define such features in order to exclude them from the smoothing process. A manual search of vertices and edges that define the special features of an object in the data file is not practical, for the model may be composed of several hundred vertices.

Once the location of special features data are determined in the data file, they are *marked* by having their associated flags set accordingly. Marked data are excluded from the smoothing process to preserve special features on the refined triangulated objects. The technique for marking vertices and edges in a crude triangulated model for special feature preservation is described here. It is devised to complement smoothing algorithms in general, but with reference to the refinement algorithm for smoothing as described in [Walto97, Walto96].

Generalization of the 3D point location algorithm to tensor-product surfaces to direct the refinement process as in [Forse88] is possible. One way of achieving this is to first triangulate the parametric surface by adding a bisecting diagonal through each patch, and then re-arrange the representation of the triangles into a special data structure that is described below.

The remainder of the paper is organized as follows: A convenient data structure for storing sampled data points to allow for interactive 3D point location is described in the next section. The actual process of locating and marking 3D points is then described. Some implementation issues are discussed in the following section. The paper is concluded with some visual examples and concluding remarks.

TRIANGULAR LOOP DATA STRUCTURE

A wire-frame triangulated model of an object is defined by a number of triangles in space whose vertices are traversed in a consistent order (e.g., counter-clockwise). The display of such a model involves reading from a file all the triangles, three vertices at a time, and then calculating their normals for shading purposes before rendering. The triangles as input, i.e., sets of three vertices, are not in a form that is suitable for processing. For example, if a vertex $\mathbf{v} = \{x, y, z\}$ is a common vertex in S surrounding triangles, then an update in the status of $\mathbf{v}$ requires an update in all the S surrounding triangles that share $\mathbf{v}$. An alternative data structure, referred to as the *triangular loop* data structure (TLDS) is described below. This data structure is convenient for interactively locating and modifying the statuses of 3D vertices in $O(1)$ time.

In TLDS, object data are pre-processed such that the number of vertices $\mathbb{V}$, edges $\mathbb{E}$, and triangles $\mathbb{T}$ are given. In addition, the surrounding triangles for each vertex, the end points for each edge, and the constructing edges for each triangle are identified.

Each vertex ($V_i, i = 1, 2, \ldots, \mathbb{V}$) is represented only once by the following data:

- An identification number vID_i, which indicates its order in the data file.
- Space coordinates $\mathbf{v}_i = \{x_i, y_i, z_i\}$.
- A binary flag (vFlag_i) preset to zero to indicate that all vertices are initially non-special.
- The number of the surrounding triangles S_i, together with a list of S_i identification numbers, each referring to one of the surrounding triangles obtained during the *single-time* data pre-processing.

Each edge ($E_j, j = 1, 2, \ldots, \mathbb{E}$) is represented by:

- An identification number eID_j.
- Two indices E_{j_1} and E_{j_2}, as an ordered pair. Each index refers to a unique vertex through its vID field.
- A binary flag (eFlag_j) preset to zero to indicate that all edges are initially not marked.

Finally, each triangle ($T_k, k = 1, 2, \ldots, \mathbb{T}$) is represented by:

- An identification number tID_k.
- A sequence of three indices T_{k_1}, T_{k_2}, and T_{k_3}, as an ordered triple. Each index refers to an edge ID number. For rendering purposes, each index is preceded by a "+" or a "−" sign to indicate the input order of vertices in every edge, to guarantee that all triangles are read in a consistent order (e.g., counter-clockwise).

A TLDS thus has the following format:

$$\begin{array}{llll} \mathbb{V} & \mathbb{E} & \mathbb{T}, & \\ V_1 & V_2 & \cdots & V_{\mathbb{V}}, \\ E_1 & E_2 & \cdots & E_{\mathbb{E}}, \\ T_1 & T_2 & \cdots & T_{\mathbb{T}}. \end{array}$$

Therefore, linear arrays V, E, and T are defined by:

$$\begin{aligned} V_i &= \{\mathsf{vID}_i, \mathbf{v}_i, \mathsf{vFlag}_i, S_i, \\ &\quad \{\mathsf{tID}_{i_1}, \mathsf{tID}_{i_2}, \ldots, \mathsf{tID}_{i_{S_i}}\}\}, \\ E_j &= \{\mathsf{eID}_j, (\mathsf{vID}_{E_{j_1}}, \mathsf{vID}_{E_{j_2}}), \mathsf{eFlag}_j\}, \\ T_k &= \{\mathsf{tID}_k, (\pm\mathsf{eID}_{T_{k_1}}, \pm\mathsf{eID}_{T_{k_2}}, \pm\mathsf{eID}_{T_{k_3}})\}. \end{aligned}$$

Note from the above definitions how triangles point to edges, edges point to vertices, and vertices point to triangles, from which the data structure get its name "triangular loop". This structure allows for immediate data access, and for more general operations to be performed on surfaces, as discussed in the conclusion of this paper.

Fig. 1 illustrates an example of a simple triangulated object. The arrows indicate the direction of each edge according to the order of indices in its definition. Under TLDS, a triangle such as T_3 in Fig. 1 is defined by $\{3, E_4, E_5, -E_2\}$.

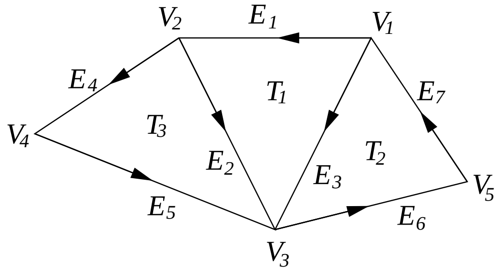

Fig. 1: A simple crude triangulated object.

Although the TLDS is reminiscent of the *winged-edge* data structure (WEDS) [Foley90], the WEDS is not as suitable for the kinds of applications described here. WEDS is based on edges having pointers to their defining vertices and adjacent faces. It is therefore not convenient for operations such as, for example, determining a normal at a vertex using data of all faces that share that vertex. This is because the surrounding polygons are not directly available from vertices; in the case of TLDS they are.

INTERACTIVE 3D POINT LOCATION

The process of interactively locating 3D points on crude triangulated objects for marking/un-marking purposes is a two-fold process: (1) storing the points in special two-dimensional arrays, and (2) accessing those points when needed in $O(1)$ time. The work presented below is partly based on ideas presented in [Madi98].

Storing the Points

In order to interactively toggle the binary value of vFlag to mark/un-mark vertices of a displayed crude triangle model before surface refinement, each vertex is stored in a two-dimensional $R \times C$ vertex pointer (VP) table, where R and C are the numbers of rows and columns of the canvas used for displaying the object, respectively.

Initially, all cells in the VP table are initialized to a "null" value (i.e., the cells are currently not pointing to any vertices on the modeled object). While the object is being transformed (i.e., rotated, translated, scaled) into a position selected by the user, each vertex V_i that goes through the transformation process is stored in the VP table depending on two factors:

1. The (x_i, y_i) value of $\mathbf{v}_i$ in V_i. This pair specifies the current (r, c) location in the VP table ($r = 1, 2, \ldots, R$, and $c = 1, 2, \ldots, C$), where r and c are the integer parts of of x_i and y_i, respectively.

2. The z_i value of $\mathbf{v}_i$ in V_i. If the current (r, c) cell in the VP table has a null value, it is set to point to V_i. Otherwise, z_i is compared to the z value of the vertex pointed to by the current cell in the VP table. If z_i is greater, the current cell in the VP table is reset to point to V_i (i.e., on a right-handed system, V_i is closer to the viewer than the vertex already resident in the current cell in the VP table).

In this regard, the VP array is used in the same fashion as the z-buffer array in the *depth buffer* hidden surface removal algorithm [Hill90], but on a vertex level rather than on a pixel level.

Similarly, by identifying each edge with a point that lies on the edge center (i.e., the average of its two end points), the same process explained above for vertices may be applied to edges using a different two-dimensional $R \times C$ edge pointer (EP) table for storing points that refer to edges. The definition of edges E_j defined above may therefore be modified to include the middle point $\mathbf{m}$ in their definition as follows:

$$E_j = \{\mathsf{eID}_j, (\mathsf{vID}_{E_{j_1}}, \mathsf{vID}_{E_{j_2}}), \mathbf{m}_j, \mathsf{eFlag}_j\}.$$

Accessing the Points

Coplanar patches that share the same vertices and/or edges may not be distinguishable from each other when shaded, therefore, it is necessary to use special highlighting to indicate the locations of all vertices and edges on the modeled object. On the displayed model, vertices are highlighted by small squares. In order to select a highlighted vertex for marking/un-marking, the user moves a pointing device (e.g., a mouse) on that square and then selects it (e.g., by clicking a designated mouse button) to toggle the value of vFlag, interactively.

When the user selects the desired vertex for marking/un-marking, the coordinates of the location selected are recorded as an (r, c) pair of values on the two-dimensional pixel canvas. The (r, c) pair is then used to access the corresponding (r, c) cell which points to the corresponding vertex V_i in the VP table. As a result, the value of vFlag is toggled immediately.

Since objects may consist of thousands of vertices, some of which may be clustered in the area of special features (i.e., where marking is to be done), functionalities such as scaling (to zoom in) and different color highlighting that changes interactively to reflect the status of vFlag for vertices are needed. The user may select between two display modes, depending on whether vertices or edges are selected for marking/un-marking. An illustration of a crude triangulated object with visible vertices highlighted differently to reflect their marking statuses is shown in Fig. 2(a).

Similarly, when displaying in edge mode, each edge E_j is highlighted by a small circle that lies on its center defined by $\mathbf{m}_j$. The user may modify the eFlag_j value of the selected edge by selecting the identifying circle. The coordinates of the circle are recorded as an (r, c) pair that are used to access the corresponding (r, c) in the EP table. As a result, the value of eFlag_j is modified accordingly. Fig. 2(b) shows the same figure shown in Fig. 2(a), but rendered in edge-display mode.

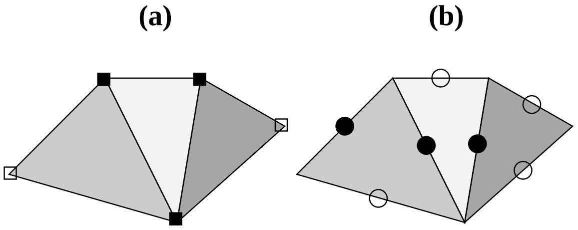

Fig. 2: The same figure rendered in different modes. Highlighted vertices/edges reflect their different marking statuses.

Observe that because of the structure of TLDS, each vertex V_i and edge E_j are stored only once, but may be pointed to several times when identifying edges and triangles, respectively. A change of the value of any vFlag in the VP array or any eFlag in the EP array is a global change in the definition of all the related edges and triangles, respectively, that is performed by the single set/reset operation. In addition, locating vertices in the VP array, or edges in the EP array is only a matter of accessing the corresponding cell, whose locations are simply determined by the pointing device used.

VISUAL EXAMPLES

For paper size considerations, the number of refining triangles per triangle is limited to 36. In the two examples shown below in Fig. 3 and Fig. 4, each shows the object in the following order:

1. A crude triangulated form of the object,
2. The object after the application of uniform surface refinement as described in [Walto97], and
3. The object after the application of surface refinement with preservation of special features, by the aid of marking techniques developed thus far.

An elaboration on the preservation of special features follow.

Example 1

This example illustrates a typical design problem, for example, creating a geometric model of an automobile's wheeltrim (or hubcap). In Fig. 3(a), the object is approximated with $\mathbb{V} = 405$, $\mathbb{E} = 1125$, and $\mathbb{T} = 720$. After the application of the smoothing algorithm, each triangle in Fig. 3(a)is refined into 36 triangles, resulting in $25,920$ triangles for each refined figure.

The uniform smoothing process results in the shape shown in Fig. 3(b). The peaks of three (each alternate one) of the humps on the outer rim were marked; the edges at the other humps on the outer rim were also marked. The vertices and edges at all humps on the inner rim were marked. Finally, the edges along the contour above the circumference were marked. The resulting model is shown in Fig. 3(c).

Example 2

This example uses data from an actual object. A physical model of an ornamental donkey was laser scanned. The data were then used to produce a crudely triangulated model. The crude triangulated model is shown in Fig. 4(a) with $\mathbb{V} = 439$, $\mathbb{E} = 1311$, and $\mathbb{T} = 874$ [Walto97]. The refined models have $31,468$ triangles each.

The application of uniform surface refinement caused special features such as cusps on the ankles or tail-end of the donkey to be smoothed, resulting in an artificial look of those parts as shown in Fig. 4(b). By rendering the donkey in vertex mode and "zooming in" on those parts when necessary, some vertices were marked so that they are not included in the smoothing process. In addition, edges on the bottom of the donkey's hooves and neck are also marked for exclusion during the refinement process. Results are shown in Fig. 4(c). The difference in the right hind legs are particularly noticeable.

IMPLEMENTATION ISSUES

In transparent objects, some hidden polygons are visible, implying that more edges and vertices are also visible. However, visible, non-hidden vertices should be highlighted, so that the surface image is not densely cluttered with highlighted vertices and edges. This conveniently limits the user to selecting points that are directly visible to them.

When marking/un-marking vertices, a group of related edges may be automatically marked/un-marked instead of individual processing. For example, in Fig. 3(c), by marking the top vertex of the closest hump on the outer rim, all the edges sharing that vertex may be automatically marked. This is possible since each vertex points to its set of surrounding triangles, and triangles point to their constructing edges, from which a subset shares the common vertex that causes them to be marked. Other scenarios for automating the process of marking/un-marking are possible and are application dependent.

CONCLUSION

A 3D point location algorithm was developed to aid in the preservation of special features before a crude triangulated model undergoes procedures of surface refinement. It is shown that the process of locating 3D points requires no searching to be done. It takes a single $O(1)$ operation to determine the location of the required memory cell containing the selected points for marking/un-marking for special features preservation. Although the work shown in this paper applies to crude triangulated models, the data structure may be generalized to work on geometric polyhedral models composed of more general n-sided polygonal patches. In fact, by adding an extra field representing the number of edges for each triangle in the T_k definition shown above, the TLDS is equipped to represent surfaces with different n-sided polygonal patches, which is important in other computer graphics applications [Macle93].

Future work focuses on interactive shape modification and surface segmentation using the algorithms and data structure introduced here.

ACKNOWLEDGEMENT

This work was supported in part by the Natural Science and Engineering Research Council of Canada.

Data for the donkey was obtained from the Integrated Manufacturing Technologies Institute, NRC, London, Ontario (courtesy of M. Yeung).

The authors appreciate the careful work of the anonymous referees; their comments helped to improve the presentation of this paper.

References

[Fang95] Fang, T., Piegl, L., "Delaunay triangulation in three dimensions", *IEEE CG&A*, Vol. 15, No. 5, pp.62–69, 1995.

[Foley90] Foley, J., van Dam, A., Feiner, S., Hughes, J., *Computer Graphics: Principles and Practices*, Second Ed., Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[Forse88] Forsey, D., Bartels, R., "Hierarchical B-splines refinement", *Computer Graphics*, Vol. 22, No. 4, 1988, pp.205–212.

[Hagen96] Hagen, H., Nielson, G., Nakajima, Y., "Surface design using triangular patches", *Computer-Aided Geometric Design*, No. 13, 1996, pp.895–904.

[Hill90] Hill, F., *Computer Graphics*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[Macle93] MacLeod, R., Johnson, C., Matheson, M., "Visualizing bioelectric fields", *IEEE CG&A*, Vol. 13, No. 4, July 1993, pp.10–12

[Madi98] Madi, M., Walton, D., "Modeling and visualization of layered objects", submitted to *C&G*.

[Neamt94] Neamtu, M., Pfluger, P., "Degenerate polynomial patches of degree 4 and 5 used for geometrically smooth interpolation in $\mathbb{R}^3$", *Computer-Aided Geometric Design*, No. 11, 1994, pp.451–474.

[Seile96] Seiler, A., Balendran, V., Sivayoganathan, K., Sackfield, A., "Reverse engineering from unidirectional CMM scan data", *Int. J. Adv. Manuf. Technol.*, Vol. 11, pp.276–284, 1996.

[Walto96] Walton, D., Meek, S., "A triangular G^1 patch from boundary curves", *Computer-Aided Design*, Vol. 28, No. 2, pp.113–123.

[Walto97] Walton, D., Yeung, M., "G^1 refinement of data for rapid prototyping", *Int. Conf. on Manuf. Automation Proc.*, ICMA 97, Hong Kong, Vol. 1, pp.214–219, 1997.

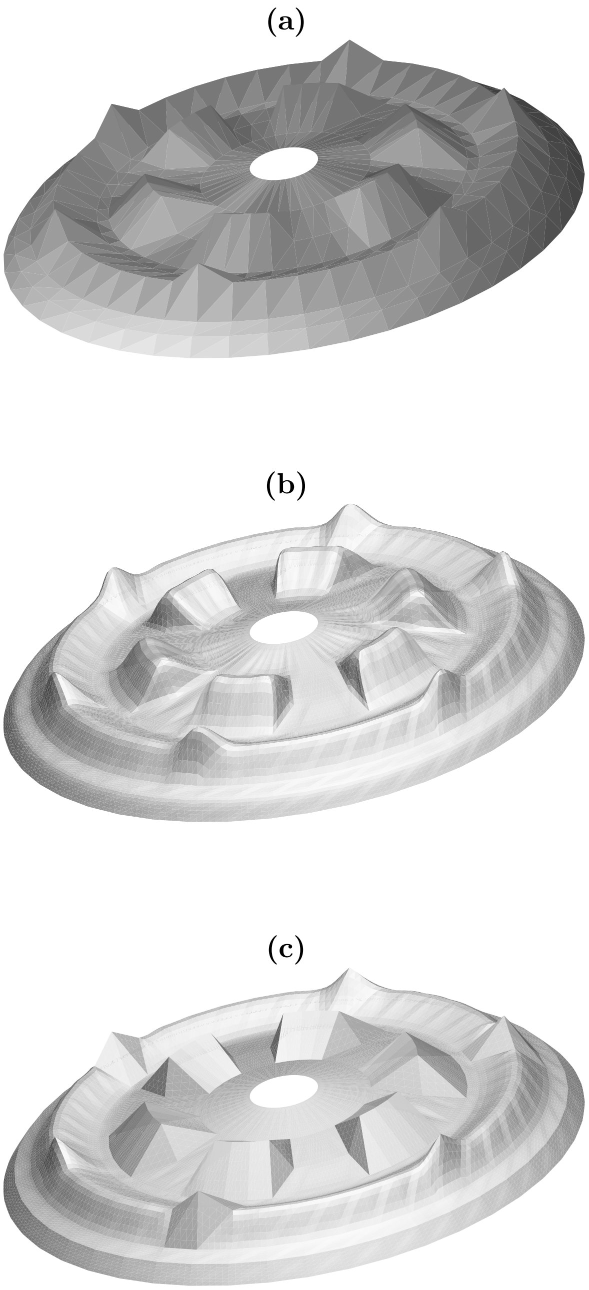

Fig. 3: Preserving special features on an artificial object.

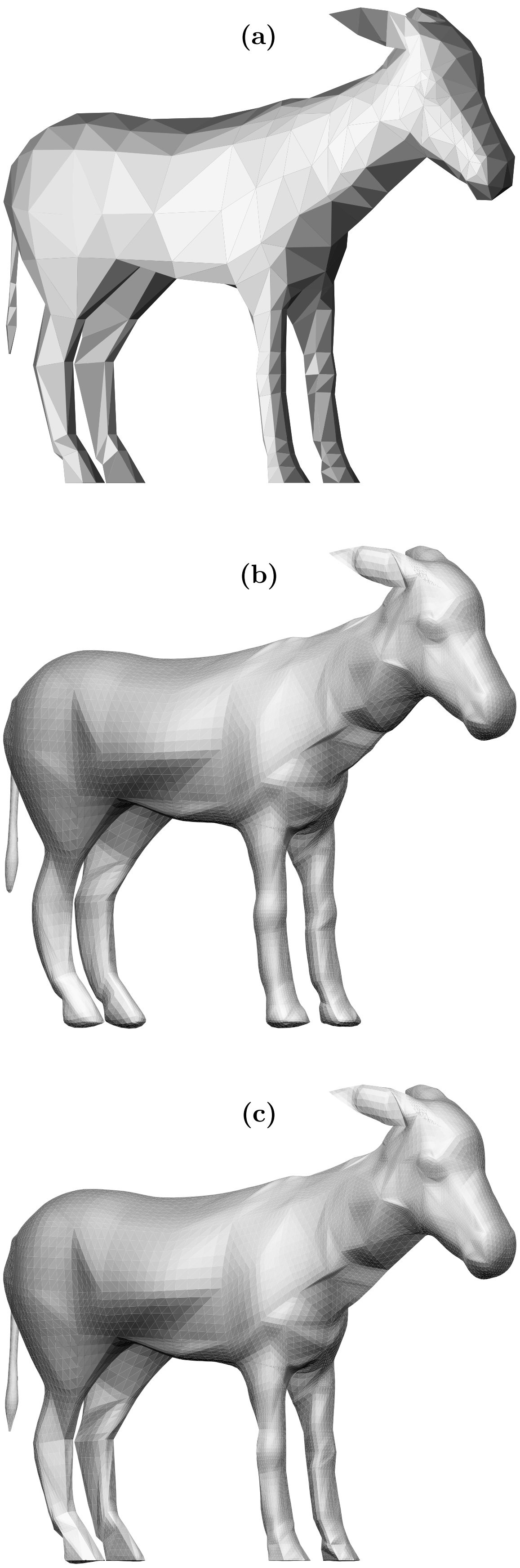

Fig. 4: Preserving special features on an actual object.