Presented to Eurographics UK 17th Annual Conference (Eurographics UK '99)
Fitzwilliam College, Cambridge, UK, 13-15 April 1999
ISBN: 0-9521097-8-6 Eurographics UK, PO Box 38, Abington, Oxon, OX14 1PX

Adaptive Refinement and Decimation under Free-Form Deformation

James E. Gain and Neil A. Dodgson

The Computer Laboratory
University of Cambridge

Abstract

The Free-Form Deformation (FFD) methods are a family of intuitive, efficient and versatile free-form modelling techniques, which rely on an ambient warp of the space cocooning an embedded object. However, the distortion caused by FFD significantly degrades the approximation quality of polygon-mesh objects. We present a curvature-based adaptive refinement and decimation algorithm which solves this problem in the context of repeated interactive FFD and improves on previous work [22, 13, 21] in three respects: it addresses sampling issues, incorporates decimation as well as refinement, and reduces computational complexity.

1 Introduction

The field of aesthetic free-form modelling (or interactive sculpting as it is sometimes called) addresses the task of computer-assisted shape design. It is distinguished from structured modelling techniques such as Constructive Solid Geometry, Primitive Instancing and Generalized Cylinders [10] by permitting unrestricted manipulation of an object's surface. Also, the aesthetic appearance of the final form is the only concern and no consideration is given to functional properties such as volume, tensile strength and aerodynamics.

There is considerable research into adapting physically-based methods to free-form modelling [26]. This approach allows a user to deform a solid in a physically realistic (and hence intuitive) fashion, since it directly incorporates mechanics, but it is memory and computation intensive. More recently this inefficiency has been reduced through a range of restrictions [4, 19, 23].

At the other end of the complexity spectrum are the Regular Deformations [1]. These are extremely efficient but of limited utility since they offer only specific stylized deformations - taper, twist and bend.

Between these extremes lies Free-Form Deformation (FFD) and its extensions, a family of highly intuitive, powerful and efficient modelling techniques, which rely on an ambient warp of the space containing an embedded object. An analogy [25] would be setting a deformable shape inside a block of jelly of the same consistency and then flexing this jelly, resulting in a corresponding distortion of the inset shape. FFD has other strengths apart from efficiency: it can always be applied as long as sample points can be extracted from an embedded object and is thus independent of the object's underlying representation, and it can be either local or

global in effect depending on the scope of the deformation relative to the object.

FFD in its seminal form [25] employed a Bézier hyperpatch (the three- dimensional analogue of a two-dimensional Bézier patch) in an initial parallelepiped configuration to demarcate a deformable portion of world co-ordinate space. This has since been generalized to hyperpatches on different bases [13, 8, 17] and with generalized topologies [6, 18]. FFD has also found widespread application in modelling [14, 8, 11] and animation [5, 7, 16]. In all of these, deformation is controlled by repositioning control points (much as in curve and surface design). Unfortunately, this interface is cumbersome and counter-intuitive [15] as display of the lattice (control points connected in a grid) tends to clutter the screen and obscure the object being created. Even worse, some lattice control points may be hidden within the object. It would be preferable if the user could drag object points directly and have the hyperpatch alterations necessary to induce these effects automatically calculated. This is the principle behind the direct manipulation extensions to FFD [3, 15].

Rendering objects which have undergone FFD is problematic. If Polygon Scan Conversion is employed then a continuous object should be approximated in polygon-mesh form prior to FFD. The primary difficulty is that this approximation degrades under FFD, as previously flat (and sparsely covered) regions become sharply undulating. Conversely, highly curved areas may be warped to near planarity and be wastefully oversaturated with polygons. These problems can be avoided by coupling an adaptive polygon-mesh refinement and decimation scheme to FFD. The conversion to polygon-mesh form *prior* to FFD is motivated by two concerns: a clean separation between the refinement/decimation scheme and the underlying representation (be it B-spline, implicit or CSG), and the ability to efficiently apply a possibly lengthy sequence of deformations to the object in an interactive modelling context.

There are currently three FFD rendering algorithms, but all have shortcomings with respect to repeated interactive deformation.

Greissmair and Purgathofer [13] take a triangle-element polygon-mesh and subdivide adjacent triangles at the midpoint of their common edge by examining the position of this midpoint after FFD. This approach is efficient but lacks symmetry, since only refinement is provided. This implies that under a sequence of deformations the object's complexity may grow without bound even if its overall curvature decreases.

Parry [22] subdivides triangles in a polygon-mesh based on a combination of their curvature and projected screen size. One edge in every triangle is heuristically identified as a "long side". Subdivision of an edge may only take place if it is marked as a "long side" by both adjacent faces, otherwise the surrounding triangles are recursively subdivided until this condition is satisfied. This process is more computationally costly than Greissmair's algorithm, may initiate unnecessary refinement and also lacks symmetrical decimation.

Nimscheck [21] adapts an advancing front finite element mesh generation algorithm to the rendering problem. The input is a completely general polygon-mesh with both convex and concave polygons and any number of holes piercing the object. The output is a triangulated approximation, which is within a user-specified tolerance of the "true" object under FFD. This power and generality comes at the expense of efficiency and this technique is not suitable for real-time applications [21].

This paper presents an adaptive refinement and decimation scheme that is based on the research of Greissmair [13] and Nimscheck [21] and tailored to repeated interactive FFD. Our contributions are as follows:

1. We incorporate a complimentary decimation process which merges faces in nearly planar areas and thereby reduces the polygon-mesh complexity (number of vertices, edges and faces).

2. We trigger refinement or decimation by testing the deviation between surface normals at edge endpoints (adapted from [21]). As will be shown, this has efficiency advantages over Greissmair's midpoint displacement condition [13].

3. We identify and develop measures to ameliorate the serious sampling problems associated with refinement and decimation under FFD.

The remainder of this paper is divided into three sections: The notation and mechanism of Free-Form Deformation are explained, with particular reference to a uniform B-spline basis; a method for evaluating the change in first derivative properties under FFD is then described and, finally, our adaptive refinement and decimation algorithm is outlined and discussed.

2 Free-Form Deformation

Free-Form Deformation is a modelling tool which warps the space surrounding an object and thereby transforms the object indirectly. This is achieved by imposing a parametric hyperpatch onto a portion of world co-ordinate space and linking distortions in the hyperpatch to object vertices.

FFD can be formulated as a mapping, $\mathcal{F}$: $\Re^3 \mapsto \Re^3 \mapsto \Re^3$, from world space, through the local parameter space of the hyperpatch, to deformed world space. Thus a vertex $X = (x, y, z)$ is mapped to parameter space as $U = (u, v, w)$ and transformed under deformation back to world space as $\bar{X} = (\bar{x}, \bar{y}, \bar{z})$. This is achieved by two functions: the embedding, $F(U) = X$, and deformation, $\bar{F}(U) = \bar{X}$, functions. The composition of $\bar{F}$ and F^{-1} constitutes FFD: $\mathcal{F}(X) = \bar{F}(F^{-1}(X)) = \bar{F}(U) = \bar{X}$.

The agency of deformation is a hyperpatch, defined as a trivariate piecewise parametric tensor product volume. This is a straightforward extension of one-dimensional curves to three dimensions. The curve control polygon which indicates the adjacency of control points generalizes to a control lattice. Likewise, just as curves may be divided into piecewise segments (on a subinterval of the univariate domain), so too a hyperpatch may be broken into cells (each defined over a parallelepiped block of the trivariate U domain). Points which lie within the hyperpatch can be formulated as a sum of control points weighted by polynomial basis functions, as follows:

$$Q = F(u, v, w) = \sum_{i=0}^{a+l-1} \sum_{j=0}^{b+m-1} \sum_{k=0}^{c+n-1} B_{i,l}(u) \cdot B_{j,m}(v) \cdot B_{k,n}(w) \cdot P_{i,j,k} \qquad (1)$$

where Q is a point within the hyperpatch and $U = (u, v, w)$ are its local parameter space co-ordinates. B are the basis functions (the first subscript denotes the index and the second the order) and P are the lattice control points.

With this background in place, FFD proceeds in three stages:

1. Object vertices which fall within the undistorted hyperpatch are assigned parametric u, v, w co-ordinates (F^{-1} is applied). In terms of our earlier metaphor, the shape being deformed is set inside the jelly.

2. A number of control points are displaced (P become $\bar{P}$), with a consequent distortion of the hyperpatch. This equates to flexing the jelly.

3. Equation 1 ($\bar{F}$ in the FFD notation) is applied repeatedly to all of the parametrised object vertices to produce a deformed version of the object. So, by the analogy, the inset shape is warped along with its cocooning jelly.

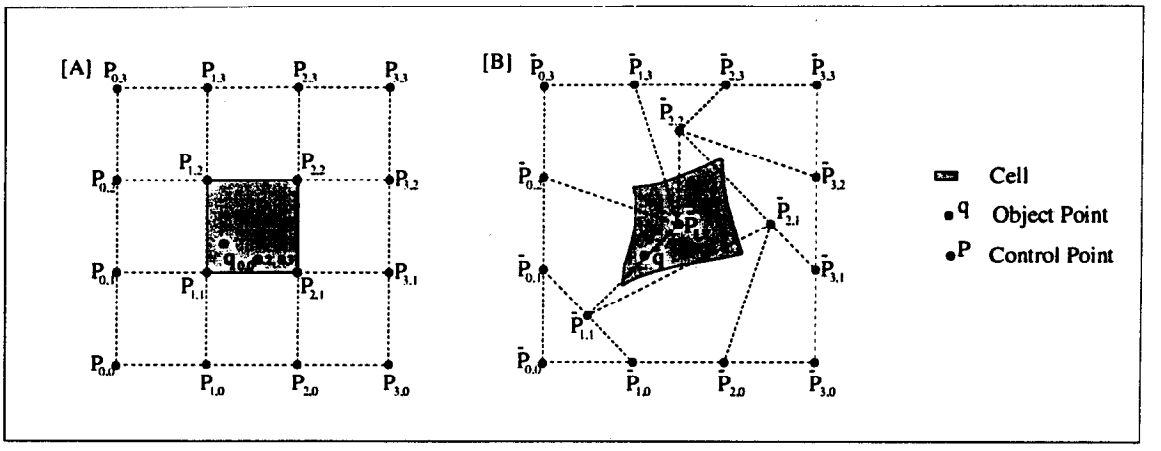

Figure 1: A two-dimensional cubic B-spline area with a single cell and an embedded vertex: [A] pre-deformation, [B] post-deformation.

A specific example will help to clarify the Free-Form Deformation process. If interactivity is an overriding concern, then the use of uniform cubic B-splines is appropriate. A uniform knot sequence ensures that all basis functions are shifted versions of each other and their evaluation is thus amenable to optimization. Also, if the initial (undistorted) lattice consists of control points evenly spaced in a parallelepiped configuration, and the parameter and world co-ordinate systems are aligned (ie. the $\vec{u}, \vec{v}, \vec{w}$ axes of the hyperpatch match the $\vec{x}, \vec{y}, \vec{z}$ axes exactly), then due to the linear precision property of B-splines [9] the embedding of vertices is immediate ($X = U$ and F^{-1} is unnecessary).

This form of FFD is illustrated in a two-dimensional analogue by figure 1. In figure 1A the lattice is shown in its initial base-state together with an associated cell and an embedded vertex. If the control point positions are altered, then the cell which they define is deformed from its initial box-shape and so too is the object vertex (see figure 1B).

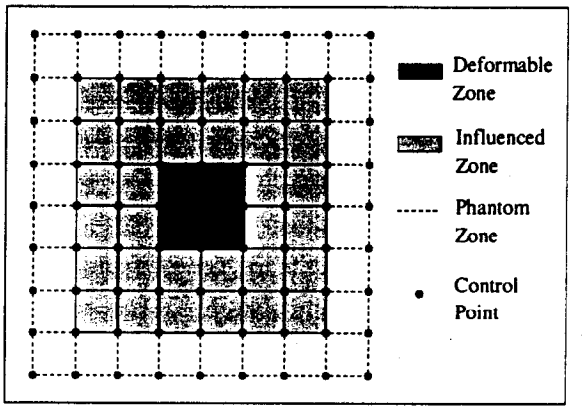

Figure 2: A FFD lattice showing deformable, influenced and phantom zones.

There are two further issues concerning this style of FFD which should be mentioned:

- There may be cells whose control points are unaltered and which do not perturb object vertices which fall within them. Instead of applying FFD only to have the original point return unchanged, a bit index, which flags cells with altered control points, can be consulted. In this way, if a control point is moved, then the cells that it influences are marked in the index.

- The fringes of the lattice may, without careful attention, produce anomalous continuity degradation. To prevent this, the lattice is partitioned into three shells (figure 2). At the centre is the deformable zone, with any number of control points and their corresponding cells. Around this lies the influenced zone, which is affected by the movement of control points in the deformable zone and is two cells thick. At the edges a single-layered phantom zone of static control points guards against boundary conditions (as outlined by Bartels et al. [2]). These phantom vertices are preferred to the tripling up of vertices previously proposed by Hsu et al. [15], which introduces additional complexity into the FFD algorithm.

3 First Derivative Properties

It is vital in certain instances to compute not only the new position of object vertices, but also their first derivative properties (normal and tangents). These provide finer detail of an object's shape and are useful for rendering algorithms (such as Phong shading [10]) and curvature-based adaptive refinement and decimation, as developed here.

The Contravariant and Covariant transformation rules [20] provide a means of mapping tangents and normals under general transformations of the form, $G : \Re^3 \mapsto \Re^3$. These can be adapted to FFD, and after some algebraic manipulation (including the discarding of scaling factors), this produces the FFD transformation rules:

$$\begin{array}{ll} \text{Covariant:} & N_{\bar{X}} = (\mathbf{J} \cdot \bar{\mathbf{J}}^{\star})^T \cdot N_X \\ \text{Contravariant:} & T_{\bar{X}} = \mathbf{J}^{\star} \cdot \bar{\mathbf{J}} \cdot T_X \end{array} \tag{2}$$

Here $\mathbf{J}$ and $\bar{\mathbf{J}}$ are the Jacobians of $\mathbf{F}$ and $\bar{\mathbf{F}}$ respectively. The superscript star (eg. $\mathbf{J}^{\star}$) denotes the adjoint of a matrix. T_X, $T_{\bar{X}}$ and N_X, $N_{\bar{X}}$ are the tangents and normals before and after deformation.

The optimized version of FFD outlined in the previous section (a uniform cubic B-spline basis combined with a parallelepiped axis-aligned initial lattice) allows these transformation rules to be further simplified:

$$\begin{array}{ll} \text{Covariant:} & N_{\bar{X}} = \bar{\mathbf{J}}^{\star T} \cdot N_X \\ \text{Contravariant:} & T_{\bar{X}} = \bar{\mathbf{J}} \cdot T_X \end{array} \tag{3}$$

where

$$\bar{\mathbf{J}} = \begin{bmatrix} \frac{\partial \bar{F}_x}{\partial u} & \frac{\partial \bar{F}_x}{\partial v} & \frac{\partial \bar{F}_x}{\partial w} \\ \frac{\partial \bar{F}_y}{\partial u} & \frac{\partial \bar{F}_y}{\partial v} & \frac{\partial \bar{F}_y}{\partial w} \\ \frac{\partial \bar{F}_z}{\partial u} & \frac{\partial \bar{F}_z}{\partial v} & \frac{\partial \bar{F}_z}{\partial w} \end{bmatrix} \tag{4}$$

The partial derivatives in equation 4 are B-spline hyperpatches influenced by deformed control points ($\bar{P}$) and take the following form:

$$\frac{\partial \bar{F}}{\partial u} = \sum_{i=0}^{a+l-2} \sum_{j=0}^{b+m-1} \sum_{k=0}^{c+n-1} B_{i,l-1}(u) \cdot B_{j,m}(v) \cdot B_{k,n}(w) \cdot [\bar{P}_{i+1,j,k} - \bar{P}_{i,j,k}] \tag{5}$$

$$\frac{\partial \bar{F}}{\partial v} = \sum_{i=0}^{a+l-1} \sum_{j=0}^{b+m-2} \sum_{k=0}^{c+n-1} B_{i,l}(u) \cdot B_{j,m-1}(v) \cdot B_{k,n}(w) \cdot [\bar{P}_{i,j+1,k} - \bar{P}_{i,j,k}] \quad (6)$$

$$\frac{\partial \bar{F}}{\partial u} = \sum_{i=0}^{a+l-1} \sum_{j=0}^{b+m-1} \sum_{k=0}^{c+n-2} B_{i,l}(u) \cdot B_{j,m}(v) \cdot B_{k,n-1}(w) \cdot [\bar{P}_{i,j,k+1} - \bar{P}_{i,j,k}] \quad (7)$$

4 Adaptive Refinement and Decimation

It is now possible, with these mathematical foundations in place, to present our curvature-based adaptive refinement and decimation algorithm. This can be visualized as stacking refinements in successive layers, which later decimations will remove in reverse order.

As a preliminary, all vertices and edges in the original (undeformed) polygon-mesh are tagged to indicate that they are at the base level of the refinement hierarchy. After each successive Free-Form Deformation the algorithm scans all edges and categorizes a subset of these as candidates for either refinement or decimation.

If either
(a) at least one of the endpoints of an edge has been deformed and the normals at these endpoints diverge after FFD by more than a certain threshold angle (θ_{max}) *or*
(b) the edge is longer than a certain maximum length (L_{max}),
then the edge is placed on the refinement heap.

This heap is ordered by decreasing edge length. This dictates an order of subdivision that tends to reduce the number of ill-formed (sliver) triangles by splitting long edges first [13]. Slivers, which are thin wedge-shaped triangles, should be avoided since they can cause rendering artefacts in Gouraud shading, Phong shading and Radiosity [21].

Conversely, *if both*
(a) the angle between deformed endpoint normals is below a decimation threshold (θ_{min}) *and*
(b) the edge is shorter than the defined maximum length (L_{max}),
then the edge's endpoints are marked as possible candidates for vertex-centered decimation.

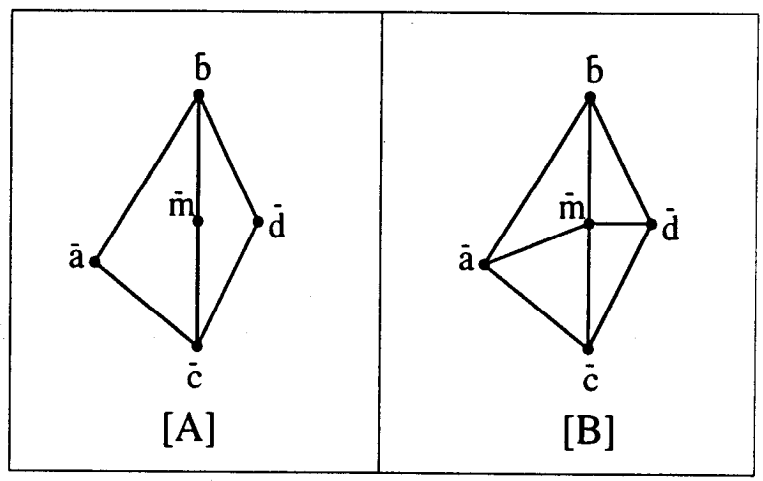

Figure 3: Refinement - subdividing an edge at its midpoint: [A] pre-subdivision, [B] post-subdivision.

The refinement stage (adapted from Greissmair and Purgathofer [13]) repeatedly extracts edges from the refinement heap until none remain. Each extracted edge, $\bar{bc}$, is subdivided according to figure 3 and the four new edges $\bar{am}$, $\bar{bm}$, $\bar{cm}$, $\bar{dm}$ are tested for refinement and if necessary inserted into the refinement heap. The only subtlety is that a refinement level, r,

is assigned to the vertex $\bar{m}$ and the two new edges, $\bar{am}$, $\bar{dm}$, and r is set at one more than the highest level amongst the vertices $\bar{a}$, $\bar{b}$, $\bar{c}$, $\bar{d}$ (ie. $r = max(\bar{a}_{level}, \bar{b}_{level}, \bar{c}_{level}, \bar{d}_{level}) + 1$). The remaining edges, $\bar{bm}$, $\bar{cm}$, are assigned the same level as their source edge $\bar{bc}$.

Decimation proceeds by grouping candidate vertices according to their refinement level. These groups are processed in descending order so that later refinements are removed first. A candidate vertex, $\bar{m}$, is decimated only if it has:
(a) exactly four incident edges which all pass the decimation test *and*
(b) all four adjacent vertices at the same or lower level than $\bar{m}$.
In this case decimation proceeds by exactly reversing a previous refinement. The vertex $\bar{m}$ and its two incident edges at the same refinement level are removed. If we label these edges $\bar{am}$ and $\bar{dm}$, then decimation is carried out according to figure 3 in reverse.

There are a number of points pertaining to this algorithm that are worth examining:

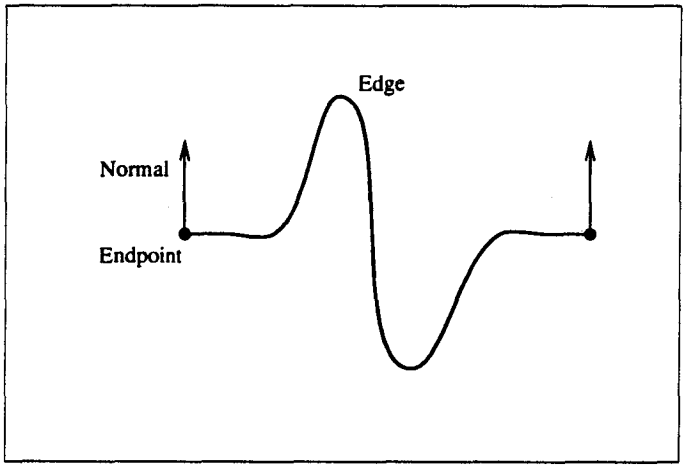

Figure 4: A condition under which the refinement classification fails.

1. Sampling issues become important in two instances. Firstly, it may occur, in testing for refinement, that the endpoint normals of an edge do not diverge significantly even though the edge undulates inbetween (figure 4). In general, sampling theory dictates that such a situation cannot be completely avoided since pathological cases can always be constructed. However, the problem can be curtailed by subdividing edges that are longer than a certain maximum and thus sampling at a higher resolution. Secondly, testing the endpoints of an edge to establish whether the edge fall within a deformed cell, may erroneously fail if only a bounded inner section of the edge is deformed. Even worse, deformation may take place within a face without affecting its edges. This suggests that a test to determine if a given face intersects any deformed cells, is required. The computational overhead for such a test is substantial and can be avoided if the longest edges in the object are shorter than the extent of a lattice cell. For these reasons a maximum edge length, L_{max}, is built into the refinement and decimation conditions. In practice, for a given FFD, L_{max} is set to the minimum cell extent.

2. Our second refinement/decimation trigger is Nimscheck's endpoint normal divergence test [21]. The expression $\varepsilon = 1 - \bar{n}_1 \cdot \bar{n}_2$ is an efficient method of evaluating this trigger since it approximates the angle, θ, between two unit normals, $\bar{n}_1$ and $\bar{n}_2$. As long as this angle is acute ($\theta < \frac{\pi}{2}$) the property $\varepsilon = 1 - cos(\theta)$ holds. An alternative condition is Greissmair's midpoint distance test [13]. Here the midpoint, m, of the pre-FFD candidate edge, e, is found and FFD is applied to produce $\bar{m}$. The trigger condition

		Endpoint Normal	Midpoint Distance
Algorithm[1]	Pre-Trigger	$\varepsilon = 1 - \bar{n}_1 \cdot \bar{n}_2$	$\bar{m} = F_{ffd}(midpoint(e))$ $m^\star = midpoint(\bar{e})$ find normals ($\bar{f}_1$, $\bar{f}_2$) of faces incident on $\bar{e}$ $d = \bar{m} - m^\star$ $d_1 = \lvert\bar{f}_1 \cdot d\rvert$ $d_2 = \lvert\bar{f}_2 \cdot d\rvert$ $\varepsilon = d_1 + d_2$
	Post-Trigger	$\bar{m} = \mathcal{F}(midpoint(e))$	-
Number of Operations[2]	Pre-Trigger	$3a + 3m$	$255a + 320m + 15d + 2s$
	Post-Trigger	$225a + 296m + 6d$	0
Total Cost[3]	Pre-Trigger	6.21	629.81
	Post-Trigger	553.18	0

(1) where e is the edge test prior to FFD, and $\bar{e}$ is the post FFD version with normal endpoints $\bar{n}_1$ and $\bar{n}_2$

(2) a = addition/subtraction, m = multiplication, d = division, s = square root

(3) The arithmetic operations are given the following relative cost, based on timings taken on an SGI Indy: addition/subtraction 1, multiplication 1.07, division 1.91, square root 1.88

Figure 5: An efficiency comparison of the endpoint normal and midpoint distance tests

is then the sum of the distances of $\bar{m}$ from the faces adjacent to the candidate edge. The endpoint normal divergence test is chosen over the midpoint distance test because of its efficiency advantages (as shown in table 5). In particular, the midpoint deformation is shifted from before to after the trigger and this becomes significant when there is a high volume of non-triggering edge tests.

3. In our implementation a straightforward heap with a binary tree structure is employed. A potential improvement would be to select a heap that was optimized for frequent heap operations as established by profiling the algorithm. For example, Greissmair and Purgathofer [13] proposed a Pagoda Heap.

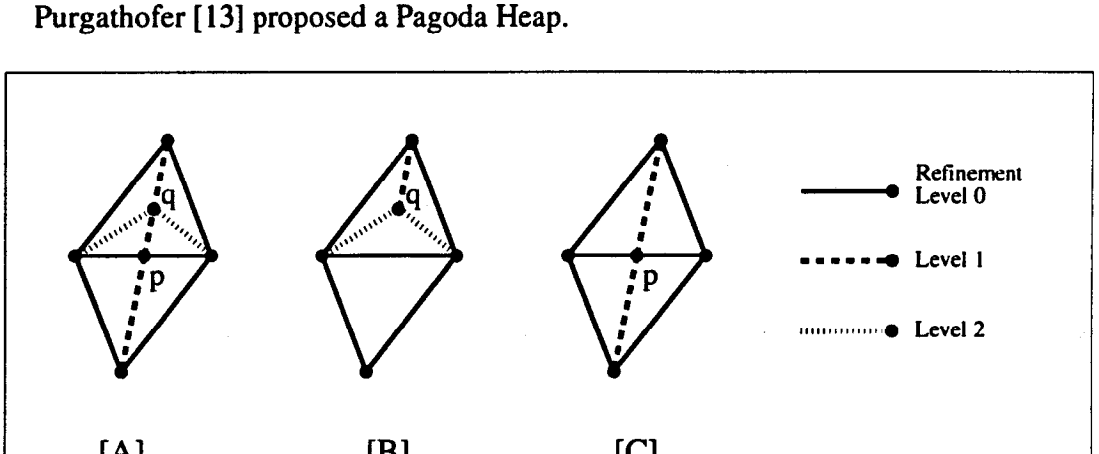

Figure 6: The importance of ordered decimation: [A] pre-decimation, [B] illegal decimation of vertex p which prevents further decimation, [C] legal decimation of vertex q which allows later decimation of p.

4. The decimation procedure is restricted to vertices with a very specific surrounding topology (which is produced in the refinement stage) and is applied in a specific order, which is controlled by the refinement level labelling of edges and vertices. These limitations are justified for several reasons:

   (a) As long as no decimation of base level vertices is allowed, the refinement/decimation algorithm is restricted to overlaying topological detail. This ensures that important aspects in the original topology (such as feature edges) are maintained.

   (b) The topology restrictions have considerable efficiency benefits. The alternative is a scheme which allows any number of edges incident on the decimation vertex (rather than just four). Schroeder et al. [24] achieve this by removing the vertex and all incident edges and then triangulating the resulting gap. The problem is thus reduced to tessellating an arbitrary polygon, which represents a significant additional overhead.

   (c) The ordering constraint ensures that refinement is fully reversible. Ideally, if a region is distorted from planar to convoluted and back to planar over a sequence of Free-Form Deformations, it should return to its original topology. Given the topology restrictions imposed on decimation, this is only possible in general (as illustrated by figure 6) if refinements are placed in a stack from which only the top element can be removed by decimation.

5 Conclusion

In this paper we have extended and improved Greissmair's adaptive refinement algorithm [13] by:

- incorporating an efficient, fully symmetrical decimation scheme that maintains the underlying initial topology,
- addressing sampling problems caused by a disparity between the scope of a Free-Form Deformation and the size of individual faces in the polygon-mesh object,
- reducing the computation cost of the refinement/decimation trigger condition.

Figure 7 shows the application of our algorithm to a sequence of two sample deformations. This is contrasted against the same deformations without subsequent refinement or decimation. Figure 7A displays an undistorted low-resolution object. In figure 7B the object has undergone an extruding Free-Form Deformation and adaptive meshing. This involves 200 refinements at parameter value $\theta_{max} = \frac{\pi}{7}$ and executes in 0.0168 seconds on an R10000 $\times$ 195Mhz SGI Octane processor. Figure 7C shows the same deformation without any refinement. A second flattening deformation is then imposed on the object in figure 7B to produce figure 7E. Notice that the centre of the nearly flat crater is now wastefully oversaturated with polygons. Figure 7D illustrates the result of an adaptive meshing step ($\theta_{max} = \frac{\pi}{7}$ and $\theta_{min} = \frac{\pi}{9}$) comprising 32 refinements and 92 decimations which is completed in 0.0095 seconds. These screen shots demonstrate both the necessity of adaptive refinement and decimation, and the efficacy of our technique.

References

[1] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics (SIGGRAPH '84)*, 18(3):21–30, July 1984.

[2] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modelling*. Morgan Kaufmann, 1987.

[3] P. Borrel and D. Bechmann. Deformation of n-dimensional objects. *International Journal of Computational Geometry and Applications*, 1(4):427–453, 1991.

[4] G. Celniker and D. Gossard. Deformable curve and surface finite elements for free-form shape design. *Computer Graphics (SIGGRAPH '91)*, 25:257–266, July 1991.

[5] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. *Computer Graphics (SIGGRAPH '89)*, 23(3):243–252, July 1989.

[6] S. Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. *Computer Graphics (SIGGRAPH '90)*, 24(4):187–196, August 1990.

[7] S. Coquillart and P. Jancéne. Animated free-form deformation: An interactive animation technique. *Computer Graphics (SIGGRAPH '91)*, 25(4):23–26, July 1991.

[8] O. R. Davis and R. P. Burton. Free-form deformation as an interactive modeling tool. *Journal of Imaging Technology*, 17(4):181–187, August 1991.

[9] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 4th edition, 1997.

[10] J. Foley, A. Van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, New York, 2nd edition, 1991.

[11] J. Gain. Virtual sculpting: An investigation of directly manipulated free-form deformation in a virtual environment. Master's thesis, Rhodes University, South Africa, February 1996.

[12] J. Greissmair and W. Purgathofer. Deformation of solids with trivariate B-Splines. *Computer Graphics Forum*, pages 137–148, 1989.

[13] U. Gudukbay and B. Ozguc. Free-form solid modeling using deformation. *Computer Graphics*, 14(3/4):491–500, 1990.

[14] W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics (SIGGRAPH '92)*, 26(2):177–184, July 1992.

[15] P. Kalra, A. Mangili, N. Magnenat Thalmann, and D. Thalmann. Simulation of facial muscle actions based on rational free form deformations. *Computer Graphics Forum (Eurographics '92)*, 11(3):59–69, 1992.

[16] H. J. Lamousin and W. N. Waggenspack Jr. Nurbs-based free-form deformations. *IEEE Computer Graphics and Applications*, 14(6):59–65, November 1994.

[17] R. MacCracken and K. I. Joy. Free-Form deformations with lattices of arbitrary topology. *Computer Graphics (SIGGRAPH '96)*, pages 181–188, August 1996.

[18] D. Metaxas and D. Terzopoulos. Dynamic deformation of solid primitives with constraints. *Computer Graphics (SIGGRAPH '92)*, 26:309–312, July 1992.

[19] R. S. Millman and G. D. Parker. *Elements of Differential Geometry*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[20] U. M. Nimscheck. Rendering for free-form deformations. Technical Report 381, University of Cambridge, Computer Laboratory, October 1995.

[21] S. R. Parry. *Free-Form Deformations in a Constructive Solid Geometry Modeling System*. PhD thesis, Department of Civil Engineering, Brigham Young University, April 1986.

[22] H. Qin and D. Terzopoulos. Dynamic manipulation of triangular B-splines. In *SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 351–360. ACM, May 1995.

[23] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65–70, July 1992.

[24] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *Computer Graphics (SIGGRAPH '86)*, 20(4):151–160, August 1986.

[25] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics (SIGGRAPH '88)*, 22:269–278, August 1988.

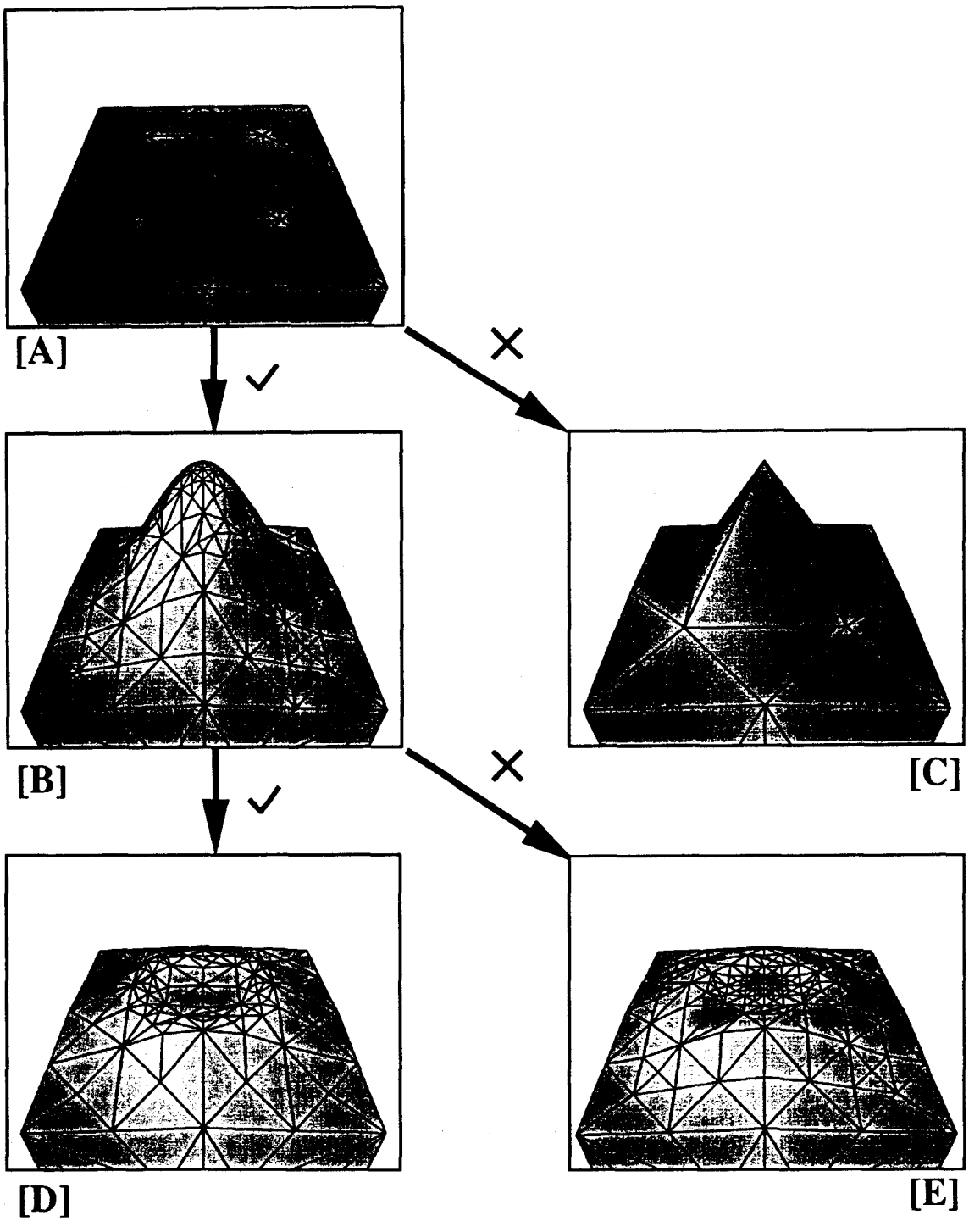

Figure 7: Refinement and Decimation: [A] an initial object, [B] an extruding deformation with adaptive refinement, [C] the same extruding deformation without refinement, [D] a flattening deformation with refinement/decimation, [E] the same flattening deformation without any refinement/decimation.