The Visual Computer

Joining polyhedral objects using implicitly defined surfaces

Karan Singh[1],
Richard Parent[2]

[1] Paraform, 3052 Bunker Hill Lane, Santa Clara, CA 95054, USA E-mail: Karan@paraform.com
[2] Ohio State University, Department of Computer and Information Science, 395 Dreese Lab, 2015 Neil Ave., Columbus, Ohio 43210, USA

Published online: 2 October 2001

Complex polyhedral objects are often constructed from simpler polyhedral objects using constructive solid geometry, booleans, skinning and shrinkwrap techniques. This paper presents a new technique for incrementally building complex polyhedral objects from simpler polyhedral parts. We provide a procedural implicit function definition for a region of a polyhedral object that is star-shaped with respect to a skeletal point, called a blend center. We extend this definition to provide a single implicit function definition for an arbitrary polyhedral object, where every region is star-shaped with respect to a proximal blend center, chosen from an arbitrary set of blend centers. This allows the application of implicit function-based modeling techniques in constructing transition surfaces between arbitrary polyhedral object parts. At the same time the original detail and character of object parts are preserved in regions where they do not blend or interact with other object parts. A complete implementation of the concepts presented shows polyhedral implicit primitives to be an efficient and general technique for building complex polyhedral objects from a modular set of polyhedral object parts.

Key words: Polyhedral objects – Constructive solid geometry – Implicit surfaces – Blend surfaces – Tesselation

Correspondence to: K. Singh

1 Introduction

Forming an object from parts is a common approach to managing complexity in the data generation process. Boolean/blend operations [15, 24] can be used to combine object parts[1] in a rigid fashion. In constructive solid geometry (CSG), the unioning and differencing of overlapping primitive objects are represented in a tree structure in which the primitive objects are leaf nodes and the Boolean operations are internal nodes [9, 15, 34]. Blend operations [20, 24, 25, 33] define surfaces between intersecting object parts. These operations are useful for modeling fillets produced as a side-effect of the manufacturing process but are restrictive in the type and relative positions of blended object parts. Smoothing approaches based on implicit techniques have also been investigated [3, 11]. Skinning and shrinkwrap procedures allow a new continuous surface to envelop, possibly disjoint, geometric elements. Skinning [14, 16, 32] usually refers to interpolating a new surface through a set of disjoint curves; shrinkwrap, similar to convex hull computation [12] refers to forming a new surface around object parts while imposing some minimum curvature constraint. However, local control of the resulting polygonal complexity is usually lost because the resolution of the new mesh is globally defined. The original object part definitions are lost; shrinkwrap tends to smooth out the entire surface of the object parts as they create smooth transitions between elements. This is undesirable in situations where the object parts have been carefully modeled to include surface detail. Other approaches to merging meshes such as zippering [28, 29] show good results from registering and blending together multiple meshes that are typically obtained from multiple range scans of an object.

An implicit function formulation for star-shaped polyhedra based on ray-linear functions is presented by [1, 2]. The analytic formulation is defined by set operations [23, 24] and can become complex for large polyhedral models. The formulation also makes it hard to provide intuitive local control over the function definition. Converting between the polyhedral and implicit function formulations is not straightforward. Another approach to combining polyhedral and implicit surface methodologies uses simple analytic implicit functions to approximate and deform an underlying polyhedral mesh,

[1] In this paper, *object* refers to the object being constructed and *object part* refers to an element which is used in the construction process.

The Visual Computer (2001) 17:415–428
Digital Object Identifier (DOI) 10.1007/s003710100115

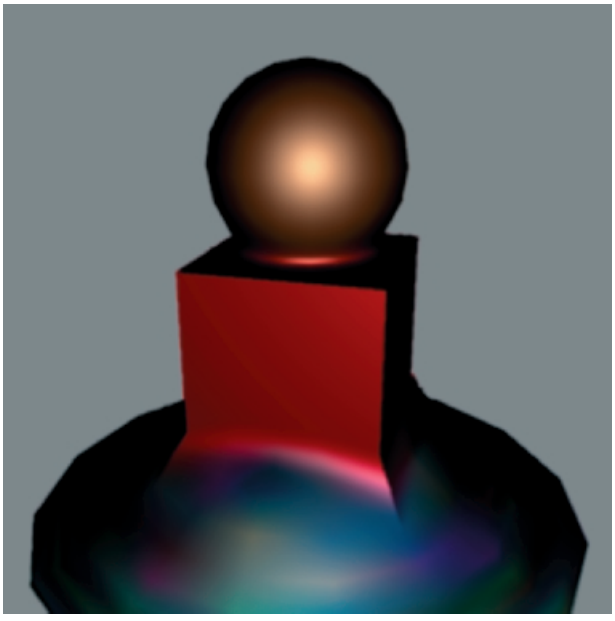


Fig. 1. Blended polyhedral object parts

without changing its topology [27]. Generation of an implicit function using distance fields for building offset surfaces can also be found in volume graphics research [10].

This paper presents an alternative approach to defining transition surfaces between possibly disjoint object parts. This approach is useful in situations such as Fig. 1, where:

1. A smooth transition surface is desired between object parts.
2. The object parts may be disjoint and at arbitrary distances from each other.
3. It is desirable to retain the original polygon definitions of the object parts whenever possible.

In order to create the smooth transition surface between separate object parts, we use an isosurface defined by a summation of implicit functions. Each implicit function is constructed procedurally, using a skeletal blend center and the surface boundary of an object part. The isosurface is defined in such a way that the original object part polygon mesh represents the isosurface wherever there is no interaction with implicit functions from other object parts. Thus, the original polygon definitions can be retained in areas of no interaction. The transition surface is generated in the area of overlap of two or more of the implicit functions and is defined so as to be tangentially continuous with the retained polygons.

While we refer to the representation of our object parts throughout the paper as being polyhedral (comprised of planar polygons), it is important to note that the implicit function formulations of Sects. 3 and 5 are directly applicable to any object part for which a ray–surface intersection can be determined. Thus, the implicit modeling techniques presented are applicable to a much wider class of objects. The surface construction approach of Sect. 4 and some of the implementation details of Sect. 6, however, are specifically intended for polyhedral objects.

The rest of the paper is organized as follows. In Sect. 2, we review the basics of surfaces defined by implicit functions. In Sect. 3 we introduce a polyhedron-based implicit function. These implicit functions can be combined and blended using standard implicit modeling and animation techniques to form transition surfaces between various polyhedral object parts. Section 4 discusses how transition surfaces are constructed from the blended polyhedral implicit functions by stitching tesselated portions of a blended implicit surface seamlessly to retained pieces of the component polyhedral object parts. Section 5 discusses local spatial control over the definition of a polyhedral implicit function, allowing the same polyhedral object part to combine differently with other object parts when building complex objects. Section 6 presents implementation details and results. Conclusions are given in Sect. 7.

2 Implicit surface primitives

Implicit surfaces [6, 8, 22, 35] are defined by all points which satisfy some implicit function $F(P)=0$. A useful set of implicit surfaces can be generated as a combination of algebraic functions each of which is defined over a finite volume. For summed algebraic functions, $F(P) = \sum F_i(P) = T$, where i runs over a set of primitive algebraic functions F_i and T is a threshold value in [0, 1]. A popular class of such functions use the distance from a central skeletal element to define an *offset surface* [4, 5, 35]. An offset surface is defined by a skeletal element, S, and a radius of influence, R. We will refer to the space within a radius R of the skeletal element as the area of influence. The implicit function, F, is typically based on a scalar function $f : \mathbb{R}^+ \rightarrow [0, 1]$ referred to as a density function. Normally, f is at least C^1 and monotonically decreasing, with $f(0) = 1$, $f(x) = 0$

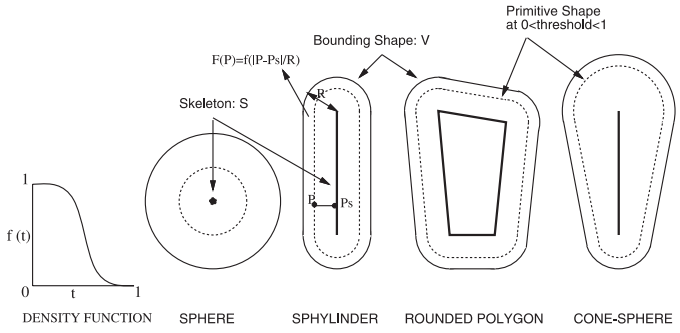


Fig. 2. Implicit primitive shapes

for $x \geq 1$ and $f'(0) = 0$, $f'(1) = 0$.[2] Either the density function f or the threshold value T can be modified to control the amount of blending between overlapping implicitly defined surfaces. The slope of the density function at the threshold value dictates the smoothness of the blend. For the example density function of Fig. 2, threshold values closer to 0.5 define a smoother blending surface than do threshold values closer to 0.0 or 1.0.

For a point P in space, a metric r with respect to the skeleton S is computed. If r is smaller than the cutoff value R, then $F(P) = f(r/R)$. For points P with metrics greater than R (outside the realm of influence of the primitive), $F(P) = 0$.

In the case of offset surfaces or distance surfaces, the metric r is merely the minimum Euclidean distance from P to S. Using P_S to refer to a point on S which minimizes Euclidean distance to P, $g(P) = \|P - P_S\|/R$ is referred to as the *distance function* of P for the primitive

$$F(P) = f(g(P)). \tag{1}$$

When such a primitive is viewed in isolation, different values of the threshold T will give surfaces of different radii offset from S.

The simplest implicit modeling primitive is a sphere (often called a *metaball* or a *blob*), which is an offset surface around a central point S. Other distance-based implicit functions are also commonly used (see Fig. 2). Any central skeletal element can be used in this formulation for which there is a well-defined distance metric. In particular, a polyhedron may be used. For a convex polyhedron in isolation, values of the threshold will result in isosurfaces which are scaled, rounded versions of the polyhedron. Figure 2 shows other examples of offset surfaces.

3 Polyhedral implicit surface primitives

All of the implicit formulations mentioned so far are based on forming a surface which is a certain constant distance from a central element. For the purposes of forming transition surfaces between object parts which smoothly blend into the object parts surface, we need to define the implicit function so that, in isolation, the implicit surface coincides with the original object part surface. One way to do this would be to consider the polyhedron as the central element and set the threshold value to 1.0. However, this would not allow us the flexibility of adjusting the threshold value to control the blending properties of overlapping implicit functions. In addition, because we often wish to blend multiple object parts together, we would like to be able to locally control the blending properties of the density functions; this is problematic if we still want first-order continuity with the object part surface using this approach.

For these reasons, we provide a fundamentally different implicit function formulation from the polyhedral offset surface primitive described in Sect. 2. Rather than use the polyhedron as a skeletal shape from which a distance surface is defined as in (1), we introduce a user-defined skeletal point C (referred to as a blend center) within the polyhedron and use

[2] In our implementation we use a family of C^1 functions $f(x) = (x^2 - 1)^2(ax^2 + 1)$ for $x \in [0, 1]$, $f(x) = 0$ for $x > 1$, defined using a parameter $a \in \mathbb{R}$, $a \neq 0$.

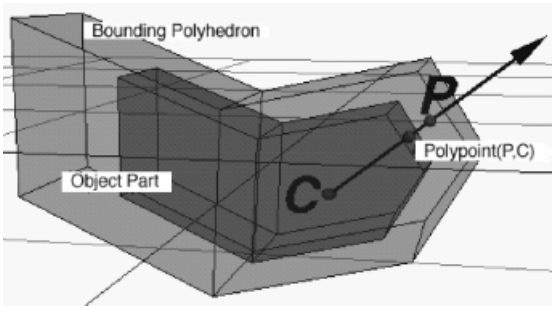


Fig. 3. Polyhedral implicit surface primitive

the surface of the polyhedron to modulate a distance function from C. We define a new *star-shaped distance function* $g(P)$ as

$$g(P) = \frac{\|P - C\|}{\|Polypoint(P, C) - C\|}. \quad (2)$$

Here $Polypoint(P, C)$ is the point of intersection of the ray $\boldsymbol{CP}$ (emanating from C and passing through P) with the surface of the polyhedron (see Fig. 3).
We now define the polyhedral implicit function $F(P)$ to be

$$F(P) = f(f^{-1}(T) * g(P)). \quad (3)$$

For a point P on the surface of the polyhedron, $P = Polypoint(P, C)$ and thus $F(P) = f(f^{-1}(T) * 1) = T$, where $f^{-1}(T)$ is essentially a constant scale factor that ensures $F(P) < T$ outside the polyhedron and $F(P) > T$ inside it. The implicit surface produced by the polyhedron-based implicit function is thus exactly the original polyhedron representing the object part for any threshold value T. For points outside the polyhedron the function value decreases monotonically in value as we move along any ray radiating from C until a point B where

$$\|B - C\| = \|Polypoint(B, C) - C\| / f^{-1}(T)$$

beyond which the function F evaluates to 0. As Fig. 3 shows, the set of these bounding points B forms a bounding polyhedron that is congruent to the original polyhedron, uniformly scaled by $1/f^{-1}(T)$ about C. The function for points inside the original polyhedron evaluates to values greater than T, converging to a value of one at C. The actual values of T and the shape of f are parameters that provide a user with control over the blending of object parts.

The above functional description is based on the star-shaped assumption that the function *Polypoint* is well defined. For any point P the intersection of ray $\boldsymbol{CP}$ with the polyhedron exists and is unique. This imposes restrictions on the polyhedron, the skeletal point C and their placement relative to each other. We show in Sect. 5 that this restriction is not prohibitive and is averted by the introduction of multiple user-defined blend centers within the polyhedron.
It is also worth noting that it is straightforward to extend the function definition in (2) to use a more complex skeletal shape S as a blend center. The function $Polypoint(P, S)$ is then the point of intersection of the ray $\boldsymbol{P_S P}$ (emanating from P_S and passing through P) with the surface of the polyhedron. P_S is the unique point on S which minimizes Euclidean distance to P. The requirement of uniqueness of P_S restricts acceptable shapes S to a small but very useful set of elements like line segments or polygons.

4 Surface construction

We will now show how the polyhedral implicit surface primitives are used to construct a blend surface between two object parts. The polyhedral object parts must be locally star-shaped with respect to their respective blend center only in the region involved in the construction of the blend surface. The next section will then introduce the use of multiple blend centers to allow for multiple blend surfaces among an arbitrary number of object parts.
The construction of tesselated polygon surfaces from general implicit functions has been well addressed [7, 18, 26, 35]. These surface construction techniques can be directly applied to the described polyhedral implicit functions.
We briefly review the surface construction process in the case of standard implicit functions. The surface is constructed by sampling space defined by a 3D regular array of vertices, aligned with the principle axes, positioned so as to envelop the area of interest. Edges connect adjacent vertices in each of the three principle directions. Cells are defined by eight adjacent vertices and the twelve edges which connect them; adjacent cells share four vertices and four edges. The implicit function is evaluated at each vertex of the array. For each edge of the array which has one array vertex inside the isosurface (evaluating to greater than T) and one array vertex outside the isosurface (evaluating to less than T), the position of

a vertex on the isosurface is either interpolated or numerically calculated by repeated evaluations of the implicit function to be the point along the edge where the function evaluates to T. For each cell of the array, one or more polygons are formed by connecting these isosurface vertices in appropriate order [7, 26]. The resulting polyhedral structure built from these polygons and isosurface vertices represents the tesselated isosurface. A number of fitness criteria of the generated polygons help determine the resolution of sampling or level of subdivision in the case of adaptive algorithms. Additionally, tesselation algorithms can be classified as surface-tracking or convergence-based [21]. The former grows the surface around a partial tesselation and thus requires a seed point on the isosurface from which to propagate. The latter is a more robust but exhaustive approach in determining the isosurface within the grid of interest.

The polyhedron-based implicit functions described in Sect. 3 have several desirable properties that may be exploited by a tesselation mechanism.

1. Any polygon or polygon fragment of an object part, which is outside the region of influence of other implicit functions, precisely represents a section of the isosurface. It can thus be copied directly to the definition of the object being formed. The contrast can be clearly seen in the blending of two spherical polyhedral implicit primitives in Fig. 4. The entire implicit surface is tesselated on the left in Fig. 4 as opposed to a partial tesselation, stitched to clipped and retained fragments of the original polyhedra, on the right. This allows surface detail present in an object part which is not involved in the formation of transition surfaces to remain unchanged in the final object.
2. The transition surface is tangentially continuous to the surfaces of interacting object parts at the boundaries of the overlap of their implicit functions. The object parts thus provide surface tracking algorithms with a number of seeds from where to propagate the surface construction.
3. An appropriate sampling rate or cell size for the tesselation algorithm can also be determined from the length of edges of the object parts at the boundaries of overlap of their implicit functions.

The standard surface construction algorithm for implicit surfaces is modified to take advantage of the fact that segments of object parts which are outside any other implicit function are retained in the final object definition. Consider the case in which only two object parts are involved (the case in which more than two object parts are involved is a simple extension of this scenario). The procedure first finds regions where primitives interact by forming an *overlap box* which is the intersection of the bounding boxes of the bounding polyhedra of the object parts (see Fig. 5). If desired, a tighter *overlap box* can be calculated as the bounding box of the intersection object of the bounding polyhedra of the object parts. For each object part, the portion of its surface outside the *overlap box* does not interact with the other object part's implicit function.

Each object part is clipped to the overlap box and the surface outside is retained as part of the final object. The polygonal approximation to the isosurface inside the box (the transition surface) is then produced using a standard surface construction technique mentioned above. Figure 6 shows the idea using two simple object parts. Finally, the polyhedral surface generated inside the *overlap box* as in Fig. 6 needs to be stitched to the clipped and retained polyhedral fragments outside the *overlap box*.

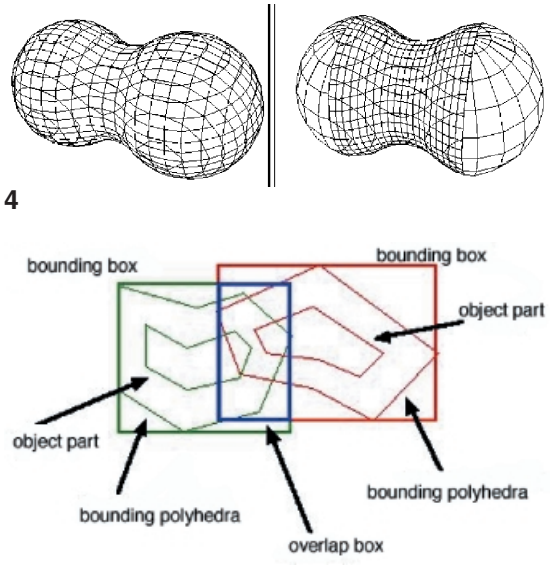
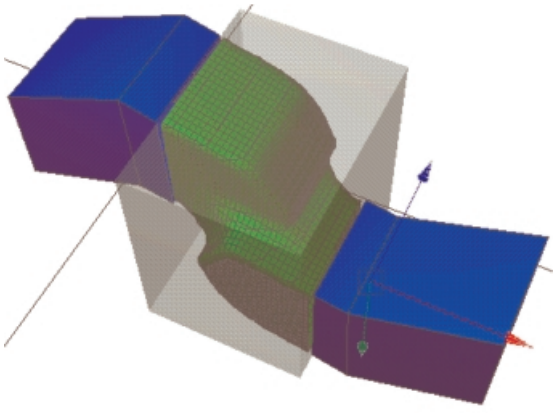
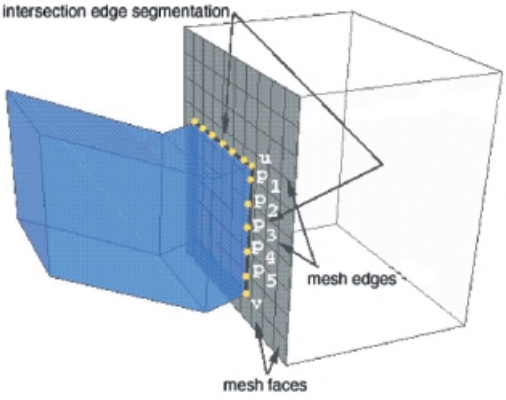


Fig. 4. Blending spherical polyhedra (without and with clipping)

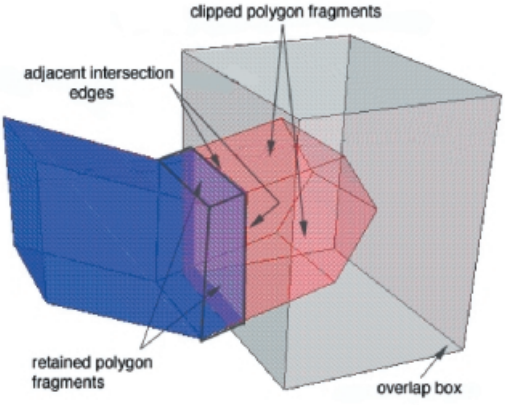
Fig. 5. Overlap box computation



6



8



7

Fig. 6. Surface construction by clipping and tesselation
Fig. 7. Polygon clipping to the overlap box
Fig. 8. Segmentation of intersection edge by mesh faces

4.1 Stitching together inside and outside surfaces

Special consideration must be given to polygons intersecting the boundary of the *overlap box* in order to connect the tesselated surface inside to the clipped and retained polygon fragments outside. Such polygons are intersected with the *overlap box* in order to determine the part of the polygon to be retained. New *intersection edges* are formed between the polygon and a face of the *overlap box* as a result of the clipping procedure (see Fig. 7). These edges are used to stitch together the tesselated surface inside and retained polygon fragments outside, as follows.

Each intersection edge lies on one of the six boundary faces of the *overlap box*. The array of cells used for tesselating the inside of the *overlap box* forms a regular 2D mesh on each of the six boundary faces of the *overlap box*, as can be seen in Fig. 8. Let $e = (u, v)$ be an intersection edge between points u, v. Each intersection edge e is segmented into a sequence of edges $u, p_1, \ldots, p_n, v$ by intersecting it with the mesh edges at points $p_1, \ldots, p_n$, in the boundary face containing the edge (see Fig. 8). Each internal edge segment (p_i, p_{i+1}) in the sequence is used as an edge in the polygon generated by the surface construction algorithm for the cell it intersects.

We now attend to the end points u, v of the intersection edge. In the fortunate case that an end point lies precisely on an edge of the mesh, we treat its corresponding segmented edge like any other internal edge segment. Typically, however, it is likely to lie within some face of the mesh. Let an end point u be adjacent to intersection edges e, e'. Suppose the edge e is segmented from end point u to be $u, p_1, p_2, \ldots$

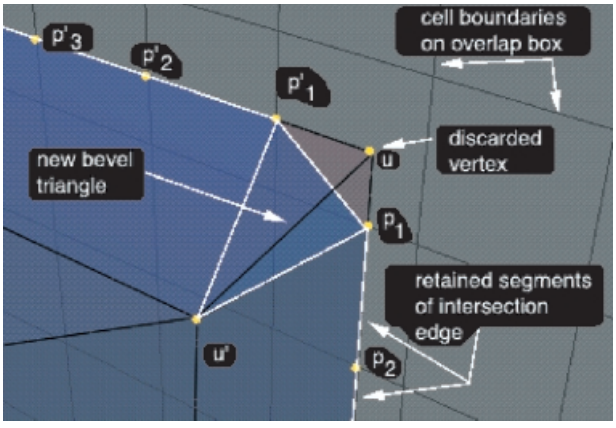


Fig. 9. Edge beveling

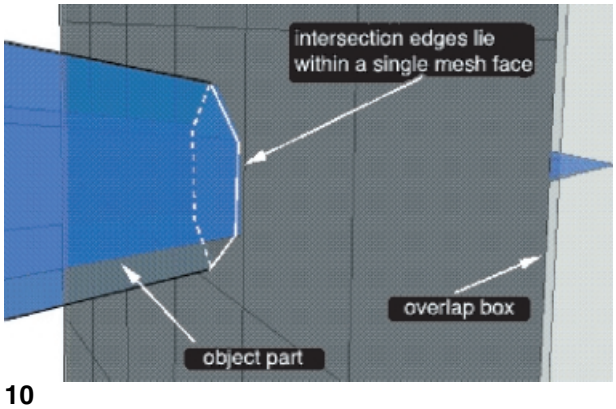
and e' is segmented $u, p'_1, p'_2, \ldots$. As Fig. 9 shows, the surface construction algorithm for this cell will cut the corner forming edge p_1, p'_1. This is likely to leave a surface discontinuity and a triangular crack p_1, u, p'_1 in the plane of the boundary face. Simply filling the crack with a triangle between points p_1, u, p'_1 would still leave a surface discontinuity. We propose three ways to handle this scenario. The first is to alter the regular spacing of the cells by moving a mesh edge so that u lies on it. This will result in the corner being represented by the surface construction algorithm. However, this method requires additional bookkeeping for each such end point and global changes in cell spacing, which can increase the resolution and complexity of the tesselation procedure. The second solution is to replace the edge p_1, p'_1 in the internally constructed surface with the edge sequence p_1, u, p'_1. A third solution is to alter the outside by throwing away the end point u altogether and generating a bevel triangle to cut the corner on the polyhedron outside the *overlap box* to match the surface constructed inside (see Figs. 9, 11a). While both the second and third solutions are acceptable, we prefer the third since it allows us to easily interface our approach with existing surface construction implementations without modifying their algorithms [21]. If u' is the vertex adjacent to u in the retained polygon fragment, the edge (u', u) is replaced by the bevel triangle u', p_1, p'_1 (Fig. 9).

The intersection edge vertices u, v in the clipped and retained polygon fragment should be replaced by the segmented vertex sequence $u, p_1, \ldots, p_n, v$, taking care to omit the end points u, v if and only if they were discarded as a result of the beveling process described above.

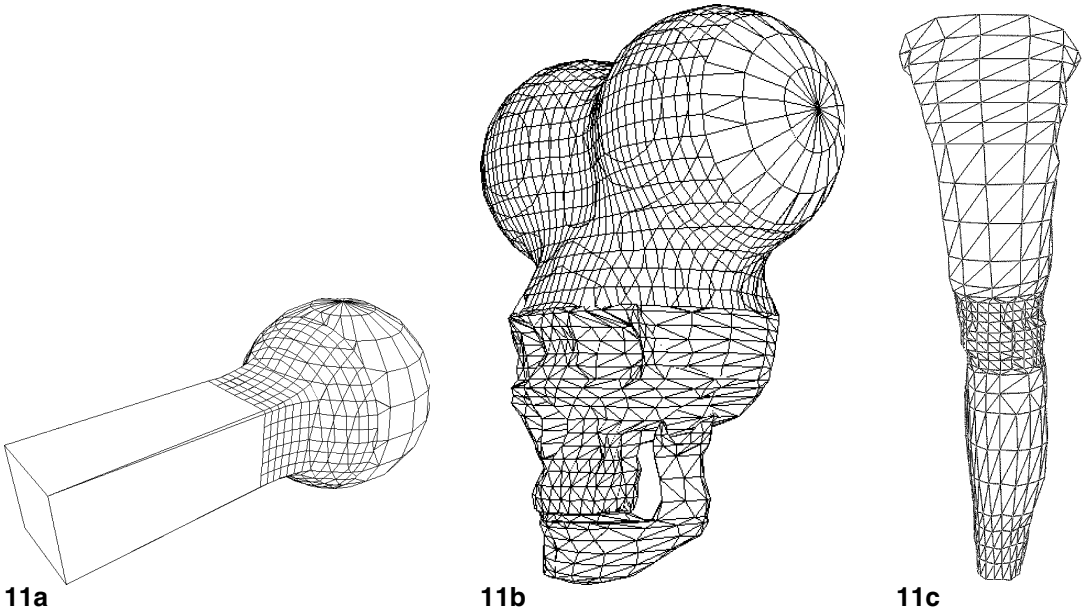
4.2 Clipping sliver polygons

There is still one issue to be resolved with respect to this approach. We have not yet considered the possibility that both end points of an intersection edge may lie entirely within a single mesh face. This can occur with sliver polygons or sharp corners intersecting an *overlap box*. In our implementation we recommend using the length of the shortest intersection edge as the cell size for the tesselation algorithm. This greatly reduces the occurrence of an intersection edge being contained entirely within a single mesh face. We do, however, address this problem for the sake of completeness and the ability to use our approach with larger cell sizes.

Once again we propose a few approaches. As in the previous subsection, we can rearrange the spacing of the cell grid so that the end points of such intersection edges lie on some mesh edge. If the number of problem intersections is large, a more brute-force solution is to use a higher-resolution grid in the surface construction process. Alternatively, as a generalization of the bevel approach, we traverse adjacent intersection edges until some intersection edge crosses the given mesh cell boundary in either direction. We thus traverse a sequence of vertices $\ldots, p, u_1, \ldots, u_n, p', \ldots$, where points p, p' intersect the mesh face boundary and $u_1, \ldots, u_n$ is a sequence of intersection edge end points all lying within a single mesh face. There will be a sequence of vertices $u'_1, \ldots, u'_m$ that are adjacent to $u_1, \ldots, u_n$ in the clipped and retained polygon fragments corresponding to the intersection edge sequence formed by $u_1, \ldots, u_n$. As was the case with the bevel approach, we discard the entire sequence of end points $u_1, \ldots, u_n$ and reconnect the sequence of vertices $u'_1, \ldots, u'_m$ to p, p' in a fashion similar to generating a loft surface between two sequences of points (the second sequence only comprising the points p, p'). As a final alternative, just as in Sect. 4.1, we can replace the edge p, p' used in the internal surface construction process with the sequence $p, u_1, \ldots, u_n, p'$. Clearly, if the sequence of intersection edges makes a closed loop within a single mesh face (see Fig. 10) or if the number of end points discarded is large, the cell grid resolution is inadequate to capture the detail being represented and should be increased.



10



11a 11b 11c

Fig. 10. Overly coarse grid resolution
Fig. 11. Polyhedral implicit surface construction

5 Multiple blend centers

We now extend the implicit function of (3) to incorporate multiple blend centers. Multiple blend centers are introduced for two main reasons.

- For the implicit function in Sect. 3 to be well defined everywhere in space, the polyhedron must be star-shaped with respect to the given blend center. This is a major restriction on the polyhedral shapes allowed. By introducing multiple blend centers the polyhedron only needs to be star-shaped with respect to a given blend center in a localized region proximal to it. An implicit function for arbitrary polyhedra by using a sufficient number of appropriately placed blend centers can thus be defined.
- When blending an object part with several other object parts, we would like local control of the blending by allowing multiple central skeletal points and corresponding density functions to be defined. This can provide intuitive local control over blending in different regions of the same object part (see Fig. 12).

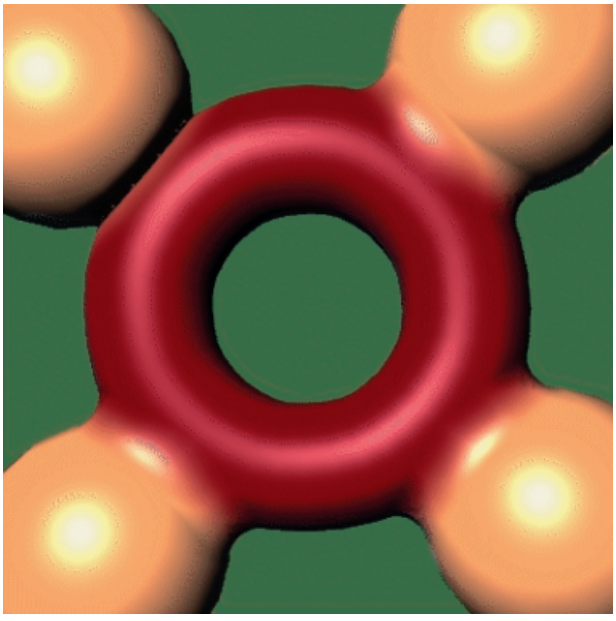


Fig. 12. Multiple blend centers with locally controlled blending

Let $C_1, \ldots, C_n$ be a set of skeletal centers with corresponding density functions $f_1, \ldots, f_n$. Given a query point P, the closest skeletal center in the set for which the *Polypoint* function is well defined is used to define the implicit function at P. We can thus provide an implicit function definition anywhere in an arbitrary polyhedron, by introducing additional skeletal centers proximal to regions where the *Polypoint* function is ill defined.

Figure 12 shows a torus blended with four different objects, each using a different central point and density function. Notice the various amounts of hard and soft blending taking place on the same object. Local control is also illustrated by a softer blend at the forefinger than at the thumb in Fig. 15.

Defining the implicit function value at a point, based on an arbitrarily picked Euclidean closest blend center, introduces discontinuities in the implicit function around Voronoi boundaries of the set of blend centers. The function defined by any blend center evaluates to T for all points on the polyhedron. At other points in space, however, the function value as defined by different blend centers is likely to vary, causing discontinuities in the function value of points around Voronoi boundaries.

Function evaluation across a Voronoi boundary is made continuous by weight-averaging the values of functions as defined by a subset of the blend centers. Blend centers whose distance to a point is within some tolerance *tol* of the closest blend center distance to the point contribute to the weight-averaging of the implicit function value. The approach is detailed as follows:

1. Let C_i be a blend center with the shortest Euclidean distance to P.
2. Let

   $$\forall k \in 1, \ldots, n \; (d_k = \|C_k - P\| - \|C_i - P\|).$$

3. Define a smoothing tolerance *tol*, which will control the smooth interpolation of function values across Voronoi boundaries.
4. Let

   $$\forall k \in 1, \ldots, n \; (w_k = f(d_k/tol)),$$

   where f is a sigmoid density function as defined in Sect. 2.
5. $F(P)$ is then obtained as a weighted average

   $$F(P) = \sum_{k=1}^{n} w_k * F_k(P) / \sum_{k=1}^{n} w_k,$$

   where $F_k(P)$ is the function definition with respect to the blend center C_k. We assume here that $F_k(P)$ is well defined for all non-zero w_k, implying that regions of the polyhedron in the vicinity of a Voronoi boundary must be star-shaped with respect to all the blend centers that define the boundary. Violation of this assumption can be fixed in practice by introducing additional blend centers or reducing the value of *tol*.

6 Implementation

The procedure outlined above has been implemented within the *Maya* modeling and animation system. *Maya* has a dependency graph architecture that lends itself automatically to the modular combination and filtering of polyhedral and other implicit function primitives intuitively by a user. A blend center is located at the local origin of the object by default in our implementation. A user can then reposition or add new blend centers to the set of blend centers. This implementation can conceivably be extended to using topologically more complex blend centers like line segments or the medial axis of objects as part of our implicit function formulation.

Table 1. Polygonization timings (seconds) and statistics

Figure	*# primitives (n)*	*# total polygons (n)*	*# candidate polygons (n)*	*# polygons retained (n)*	*# polygons generated(n)*	*preprocessing, clipping time*	*polygonization time*
11a	2	206	98	189	536	0.11	0.24
11b	3	2484	503	2159	1841	0.25	0.36
11c	3	1062	288	774	243	0.24	0.29

The overall surface construction algorithm for the figures shown in the paper takes 0.5 s on average on an SGI O_2 machine. More detailed times can be found in Table 1.
Surface construction is implemented as described in Sect. 4.

1. *Overlap boxes* are computed as the intersection of the bounding boxes of the bounding polyhedra corresponding to the object parts being blended.
2. The polyhedral object parts are clipped against the *overlap boxes*, regions outside the box retained and intersection edges computed.
3. The cell size for the tesselation is set to the larger of a user-defined minimum and the length of the shortest intersection edge. It may also be adjusted by the user.
4. The intersection edges are segmented once the cell size is determined using a simple Bresenham-like traversal of mesh faces.
5. The end points of the intersection edges are inspected and discarded as described if necessary. The clipped polygons are then correctly generated and bevel or loft triangles created where necessary.
6. The cells corresponding to the segmented intersection edges now form a number of seed cells for a fixed resolution surface tracker as described in [7].

A seamless integration of the clipped and polygonized structure can be clearly seen in Fig. 11a, where a cuboid blends with a spherical polyhedron. The beveling of edges to prevent cracks has been accentuated to be clearly visible. Figure 11b shows the polygonization of the skull in Fig. 13. Figure 11c shows the polygonization of the arm from Fig. 16. As shown in these figures, the clipped polygons at the boundaries of *overlap boxes* may have a large number of vertices along intersection edges. These polygons may need to be triangulated to render properly on some graphics systems.

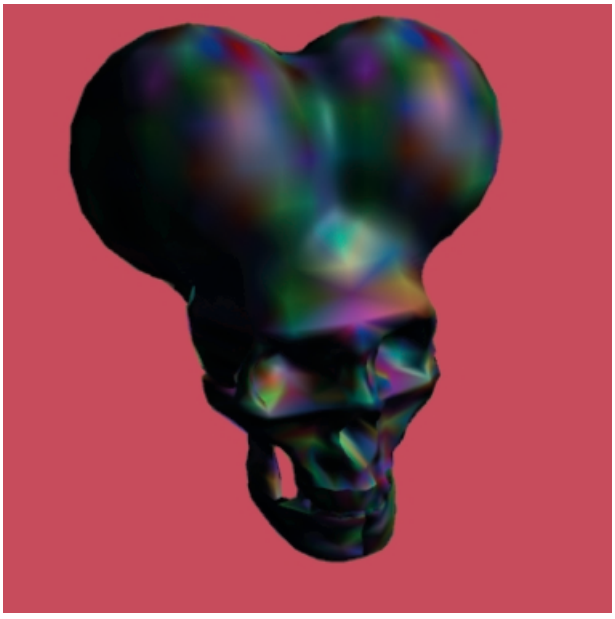


Fig. 13. Blended polyhedral implicit primitives

Function evaluation for a polyhedral implicit primitive may seem at first to be an expensive operation involving a ray–polyhedron intersection. For a point P, let the ray $\boldsymbol{CP}$ intersect a polygon $Poly_P$ of the polyhedral implicit primitive. Using the properties of similar triangles, observe that

$$\frac{\|P-C\|}{\|Polypoint(P,C)-C\|} = \frac{(P-C)\odot N_{Poly_P}}{R_{Poly_P}},$$

where $\odot$ indicates the dot product of two vectors, N_{Poly_P} is the normal vector to the plane of $Poly_P$ and R_{Poly_P} the normal distance from C to the plane of $Poly_P$. Both N_{Poly_P} and R_{Poly_P} may be easily precomputed for every polygon of the polyhedron. Equation (3) now becomes

14

15

Fig. 14. Blended polyhedral implicit primitives
Fig. 15. Mutiple blend centers at fingers of hand

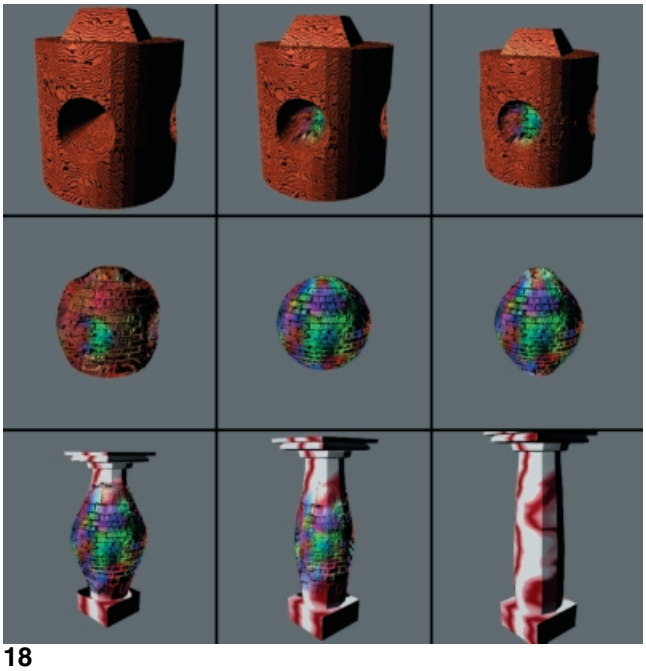
$$F(P) = f\left(\frac{f^{-1}(T)(P-C)\odot N_{Poly_P}}{R_{Poly_P}}\right),$$

making function evaluation at a point P a simple matter of determining the polygon of intersection $Poly_P$ for any given point P. To facilitate this we preprocess space around C, firing a spherical distribution of rays to determine the intersecting polygon for the ray. If the sampling resolution is fine enough, then, given any point P, we can look up the intersecting polygon efficiently in the distribution table using the spherical coordinates of $P-C$. In our implementation we bilinearly interpolate the function value at a point P, obtained using the intersecting polygons precomputed for the four ray samples that bracket $P-C$. While this can be an approximation to the formulation of Sect. 3, it is efficient and proves to work well in practice.

As can be seen in Figs. 13 and 15, polyhedral attributes such as normal vectors or texture parameters translate directly from the polyhedron to the implicit primitive. Figure 14 shows two highly detailed poly-

16

17



18

Fig. 16. Arm animated using CSG blending: 1
Fig. 17. Arm animated using CSG blending: 2
Fig. 18. Shape transformation

hedral models being blended at the heel while preserving the original detail everywhere else.

Figures 16 and 17 show the application of multiple blend centers to character animation. The arm is modeled as two limbs blended together with an analytic spherical primitive. The limbs are laser-scanned polyhedra. The arm is shown outstretched as well as bent, where collision–deformation interaction [13] between the polyhedral primitives causes the formation of a precise crease, while the primitives remain smoothly blended together due to the analytic sphere.

Figure 18 illustrates a shape transformation. The pillar and wooden block are two superposed polyhedral objects whose implicit primitive functions are

weighted and added together. The isosurface representing the combined function provides a shape transformation on interpolation of the weights. The transformation of various color and texture attributes is also illustrated.

7 Conclusion

A simple and effective technique for constructing tangentially continuous transition surfaces between disjoint object parts has been presented. A technique has been shown in which a tangentially continuous transition surface is constructed between blended object parts. The object parts may be disjoint and at arbitrary distances from each other and the original polygon definitions of the object parts are retained whenever possible.

The transition surface is defined as an implicit surface and issues related to the construction of a polygonal approximation to the surface have been addressed. These include:

- Forming a continuous surface in the area where the transition surface blends into the original polygonal surface
- Beveling the object part to provide a more esthetically pleasing transition to the transition surface
- Handling multiple blends on a single object part

As can be seen from the images, polyhedral primitives generally behave well and in a similar manner to their analytic counterparts. This gives the user an intuitive notion of the results while modeling. The tesselation efficiency obtained from the implementation is reasonable and implicit function evaluation time comparable with that of an analytically defined primitive. The resolution of transition areas is controlled independently of the object part's resolution. If the overlap boxes of multiple transition areas are disjoint then their respective resolutions can be controlled independently. The tesselation algorithm has been implemented and successfully used to build a variety of objects from various parts. In particular, the technique was used for an application in which segments of the human figure were digitized separately and then combined to form a single polygonal mesh.

The applicability of our implicit function formulation in Sects. 3 and 5 to general object representations for which ray–surface intersections may be determined makes for a much tighter coupling between boundary representation and implicit function-based modeling and animation techniques. This should open up new avenues for research on hybrid techniques that utilize the complementary advantages of the two representations. Future directions for work on polyhedral object parts include the precise evaluation of the function efficiently and extending the surface construction algorithm to handle adaptive tesselation techniques.

Acknowledgements. We would like to thank Steve May (ACCAD), Eugene Fiume and the R+D team of Alias|wavefront for their help and support in the implementation of the concepts presented in the paper. We would also like to thank Hans Pedersen and the anonymous referees for their comments, which have contributed greatly to improving the presentation and completeness of this paper.

References

1. Akleman E (1996) Interactive construction of smoothly blended star solids. Proceedings of Graphical Interface '96, May
2. Akleman E (1998) Interactive Construction of Ray-Quadrics. Proceedings of Implicit Surfaces '98, pp 105–112, June
3. Bajaj C, Ihm I (1992) Smoothing polyhedra using implict algebraic splines. Comput Graph 26(2):79–88
4. Barthe L, Gaildrat V, Caubet R (1998) Combining implicit surfaces with soft blending in a CSG tree. CSG '98, Set Theoretic Solid Modeling: Techniques and Applications, pp 17–31. Information Geometers
5. Blanc C, Schlick C (1995) Extended field functions for soft objects. Implicit Surfaces '95, Grenoble France, pp 21–32
6. Blinn J (1982) A generalization of algebraic surface drawing. ACM Trans Graph 1(3):235–256
7. Bloomenthal J (1988) Polygonization of implicit surfaces. Comput Aided Geom Des 5:341–355
8. Bloomenthal J, Shoemake K (1991) Convolution surfaces. Comput Graph 25(4):251–256
9. Bloomenthal J, Wyvill B (1990) Interactive techniques for implicit modeling. Comput Graph 24(4):109–116
10. Breen D, Mauch S, Whitaker R (1998) 3D scan conversion of CSG models into distance volumes. Proceedings of IEEE Symposium on Volume Visualization, pp 7–14
11. Dahmen W, Micchelli CA (1985) Line average algorithm: a method for the computer generation of smooth surfaces. Comput Aided Geom Des 2:77–85
12. Day A (1990) The implementation of an algorithm to find the convex hull of a set of three-dimensional points. ACM Trans Graph 9(1):105–132
13. Gascuel M (1993) An implicit formulation for precise contact modeling between flexible solids. Proc. of SIGGRAPH, pp 313–320
14. Guo B (1993) Representation of arbitrary shapes using implicit quadrics. Visual Comput 9(5):267–277
15. Hoffman C (1989) Solid and geometric modeling: an introduction. Morgan Kaufmann, San Francisco
16. Kaklis P, Ginnis A (1996) Sectional-curvature preserving skinning surfaces. Comput Aided Geom Des 13:7, 601–619

17. Kent J, Carlson W, Parent R (1992) Shape transformation for polyhedral objects. Proc. of SIGGRAPH, pp 47–54
18. Lorensen W, Cline H (1987) Marching cubes: A high resolution 3D surface construction algorithm. Comput Graph 21(4):163–169
19. Max N (1990) Cone-spheres. Computr Graph 24(4):59–62
20. Middleditch A, Sears K (1985) Blend surfaces for set-theoretic volume modeling systems. Comput Grap 19(3): 161–170
21. Ning P, Bloomenthal J (1993) An evaluation of implicit surface tilers. IEEE Comput Graph Appl 13(6):33–41
22. Nishimura H, Hirai M, Kawai T, Kawata T, Shirakara I, Omura K. (1985) Object modeling by distribution functions. Electronic Communications 68(4):718–725
23. Pasko A, Adzhiev V, Sourin A, Savchenko V (1995) Function representation in geometric modeling: concepts, implementations and applications. Visual Comput 11:429–446
24. Ricci A (1973) A constructive geometry for computer graphics. Comput J 16(2):157–169
25. Rockwood A (1989) The displacement method for implicit blending surfaces in solid models. ACM Trans Graph 8(4):279–297
26. Schmidt M (1993) Cutting cubes – vizualizing implicit surfaces by adaptive polygonization. Visual Comput 10:101–115
27. Singh K, Parent R (1995) Implicit function based deformations of polyhedral objects. Eurographics workshop on Implicit Surfaces, Grenoble, France, pp 113–128
28. Turk G (1992) Re-tiling polygonal surfaces. Comput Graph 26(2):55–64
29. Turk G, Levoy M (1994) Zippered polygon meshes from range images. Proc. of SIGGRAPH, pp 311–318
30. Wilhelms J, Van Gelder A (1992) Octrees for faster isosurface generation. ACM Trans Graph 11(3):201–227
31. Witkin A, Heckbert P (1994) Using particles to sample and control implicit surfaces. Proc. of SIGGRAPH, pp 269–278
32. Woodward C (1988) Skinning techniques for interactive B-spline surface interpolation. Comput Aided Des 20(8):441–451
33. Woodwark J (1987) Blends in geometric modeling. The mathematics of surfaces II. Institute of Mathematics and its Applications, Oxford, pp 255–297
34. Wyvill B, Guy A, Galin E (1998) Extending the CSG tree: warping, blending and boolean operations in an implicit surface modeling system. Implicit Surfaces 3:113–121
35. Wyvill G, McPheeters C, Wyvill B (1986) Data structures for soft objects. Visual Comput 2:227–234

KARAN SINGH is currently a computer graphics person at Paraform Inc. He received a BTech in Computer Science from the Indian Institute of Technology, Madras, in 1991 and a PhD in Computer Science from Ohio State University in 1995. His research interests include implicit functions, character modeling and animation and, more recently, modeling techniques for conceptual automotive design.

RICK PARENT is an Associate Professor in the Department of Computer and Information Science at Ohio State University. He received his PhD in Computer Science from Ohio State in 1977. He co-founded the Computer Animation Company in 1980 and then joined the faculty at Ohio State in 1985. His interests include all aspects of computer animation and particularly the modeling and animation of the human form.