RC 16387 (#77458) 12/13/90 REVISED 1/8/92
Computer Science 22 pages

Research Report

Solid-Interpolating Deformations: Construction and Animation of PIPS

Anil Kaul

Department of Mechanical Engineering
Columbia University
New York, N.Y. 10027

Jarek Rossignac

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

Received the

Gunter Erlande's Best Paper award (First prize) at Eurographics'91

Published as

"Solid-Interpolating Deformations: Construction and Animation of PIPs", A. Kaul and J. Rossignac, Computers&Graphics, Vol. 16, No. 1, pp. 107-115, 1992.

This replaces RC 16387 (#72685) December 1990

Solid-Interpolating Deformations: Construction and Animation of PIPs

Anil Kaul [†] [‡] and Jarek Rossignac [§]

IBM Research Division,
Thomas J. Watson Research Center,
Yorktown Heights, NY 10598.

Abstract

Computer programs that simulate the deformations of geometric shapes have played a key role in the increasing popularity of software tools for artistic animation.

Previously published techniques for specifying and animating deformations are either limited in their domain or ill suited for interactive editing and visualization because the effects of alterations performed by the animator to the model's parameters may not always be anticipated, and because realtime animation may only be produced by visualizing pre-computed sequences of 3D frames, which are obtained by a slow process and require vast amounts of storage.

To support an interactive environment for animation design, we have developed a new, simple, and efficient animation primitive: a Parameterized Interpolating Polyhedron, or PIP for short. PIPs are easily specified and edited by providing their initial and final shapes, which may be any polyhedra, and need not have corresponding boundary elements.

PIPs may be efficiently animated on standard graphic hardware because a PIP is a smoothly varying family of polyhedra bounded by faces that evolve with time, but have constant orientations and have vertices that each move on a straight line between a vertex of the initial shape and a vertex of the final one. The cost of recalculating the time dependant information of a PIP is small in comparison to the display cost.

We provide simple and efficient algorithms, based on Minkowski sum operations, for computing PIPs. When both the initial and final shapes are convex, the resulting faces are the true boundary of the deforming object, otherwise subsets of the resulting faces may lie inside the object. In both cases, correct images are automatically generated using standard depth-buffer hardware.

The tools we have developed are convenient for interactively designing animation sequences that show the metamorphosis of 3D shapes. They may also be used to simulate the geometric effect of a variety of manufacturing operations and for interactively selecting the optimal compromise between two or more shapes. They are being integrated in the LAMBADA design and inspection environment for animated assemblies, where deformations and rigid-body motions may be easily combined and synchronized using a hierarchical representation.

[†]This research was supported by a grant from IBM.

[‡]Dept. of Mechanical Engineering, Columbia University, New York. NY 10027. E.mail kaul@ibm.com

[§]Interactive Geometric Modeling, J2-C03, Thomas J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598. E.mail: jarek@ibm.com

1 Introduction

1.1 Overview for animation techniques for deforming objects

Although an infinite variety of solid models may be produced by writing and executing computer programs [8, 5], most CAD developers favor higher-level interactive design tools typically confined to a few operations, for creating and combining simple shapes [3].

Similarly, commercial animation packages, marketed to artistically inclined users, who are not always vested with the skills and tenacity expected of programmers, must provide simple metaphors for specifying the behavior of 3D shapes. A variety of ergonomic tools for designing motions of rigid bodies have been proposed [19, 13], but are inappropriate for producing animations of deforming objects.

Most previously disseminated techniques for specifying and visualizing deformations of 3D shapes fall into one of the following four categories: (1) time-dependent mappings of the underlying space onto itself, (2) trivariate vector-valued functions which define one-parameter families of surfaces, (3) incremental simulation of the dynamic behavior of a discretization of the objects, and (4) parametric models, whose parameters are specified as a function of time.

The first three techniques require expensive computations and are ill-suited to interactive design and inspection of animations, because, despite recent advances [18], the effects of alterations performed by the animator on the model's parameters (such as the displacement of a control point or a change of gravity) may not always be anticipated, and because real time animation may only be produced by visualizing pre-computed sequences of 3D frames, which, for large models, are obtained by a slow process and require vast amounts of storage. We examine the limitations of these techniques in some detail below.

- A time-dependent mapping is a smoothly-varying function, $\mathcal{F}$, of time that maps the underlying three-dimensional Euclidean space onto itself. As the function changes with time, objects transformed by the mapping appear to deform. Useful mappings that can easily be parameterized as a function of time have been proposed for bending [6] and for twisting [1]. More complex mappings, based on trivariate tensor products, treat the x, y, and z coordinates as the u, v, and t variables of a polynomial form that defines the three coordinates of the image point, P(u,v,t), in the original Euclidean space. Three-dimensional control points of Bernstein polynomials [24] and natural vibration modes [17] have been used for specifying the mappings. Four dimensional point displacements have been used in [2] to interactively edit the function $\mathcal{F}$. Deformed objects are obtained by mapping the vertices or control points, which implicitly define the faces. Since there is no restriction on the displacement of these points, the faces they define may become unsuited for graphics. Furthermore, normals to the resulting faces may not in general be easily computed as images of the original normals. Alternate techniques are proposed in [2, 1] for dealing with normals.

- The trivariate vector-valued functions proposed in [24] can also be used for defining a family of parametric surfaces, S(u,v), whose control points are specified by Bspline curves parameterized by time. Thus, the surfaces deform with time. Although, by restricting

the mapping to affect only a portion of the underlying Euclidean space, a local control of the deformation may be obtained [4], the global nature of these mappings restricts their ability to stretch space locally without undesirable side effects.

- When objects are represented by a finite-element mesh or by a grid of point masses linked by springs, numerical methods may incrementally simulate the evolution of these masses, while taking into account the forces (spring, friction, gravity, wind) acting upon them [26, 16]. The result of these simulations is a sequence of 3D models that can be displayed from any angle. Animation is produced by rapidly displaying the sequence. Alterations to the animation require running the simulation again.

A deforming object may be viewed as a parameterized shape, where the parameter is time. Since construction operations used in interactive solid modellers are parameterized, a non-evaluated parameterized model may be saved [3, 23] and used to produce animations by varying the parameters. The large number of parameters in typical models requires automatic ways for specifying their evolution as a function of time [27, 28]. Unfortunately, currently available parameterized operations for creating or transforming objects may be insufficient or impractical for designing arbitrary shape deformations.

1.2 The PIP primitive for animation

To support an interactive environment for animation design, we propose in this paper a new primitive, simple to specify and efficient to animate, from which a wide class of animations can be composed. Because our animation primitive is completely defined by specifying its original and final shape, and because our implementation is restricted to polyhedral objects, we call it a Parameterized Interpolating Polyhedron, or PIP for short.

The initial and final shapes that define a PIP may be arbitrary polyhedra and need not have corresponding boundary elements. Thus, a PIP may be conveniently edited by changing the shape, position, or orientation of the two objects it interpolates. The user is not required to maintain any correspondence between the two objects, which may have different numbers of faces, edges, and vertices.

PIPs may be efficiently animated on standard graphic hardware, because a PIP is a smoothly varying family of polyhedra bounded by faces that evolve with time. The faces have constant orientations and vertices that each move on a straight line between a vertex of the initial shapes and a vertex of the final one. Thus, a PIP may be represented by a standard boundary model in which vertices are specified by two points (V_0 and V_1) and simply computed for each value of time t as: $V(t) = (1-t)V_0 + tV_1$. Once the current vertices are known, the standard procedure for rendering faces may be invoked. The normals for shading remain constant for each face during the entire animation. Figure 1 shows several instances of the deforming PIP at different time steps for the interpolation between a lamp and a clock.

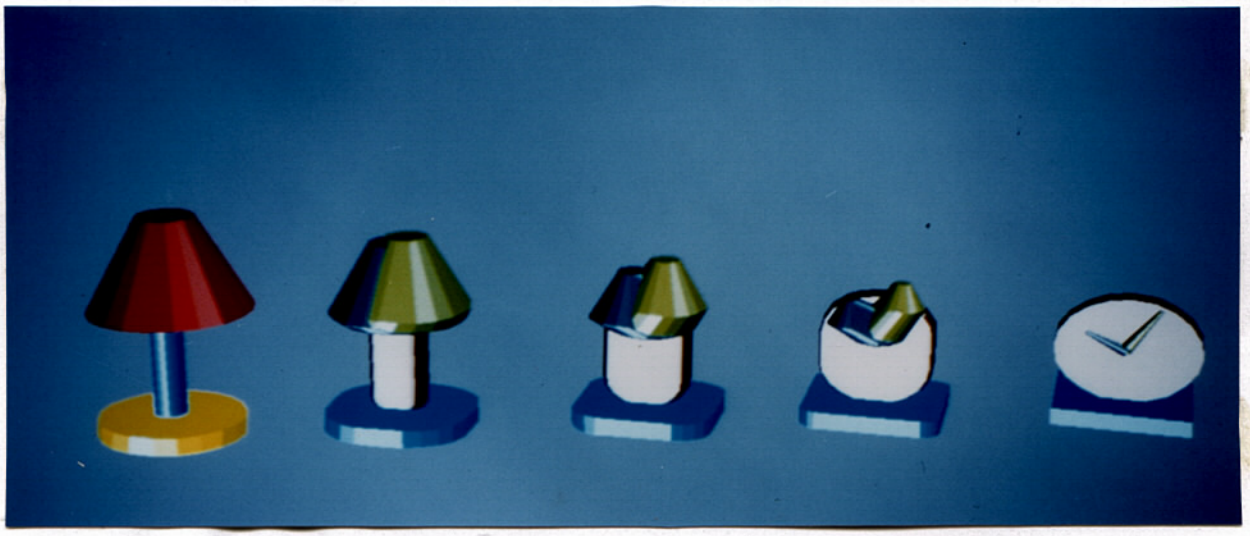

Figure 1: Figure shows the interpolation between a lamp and a clock

1.3 Linear interpolation based on Minkowski sums

A PIP, $C(t)$, is a linear interpolation between two shapes, A and B, and thus the expression:

$$C(t) = (1-t) * A + t * B \tag{1}$$

is a suitable candidate for defining PIPs. This expression combines scalars and solids. The "*" operator denotes scaling of a solid. Thus we have $C(0) = A$ and $C(1) = B$, if the "+" operator treats the empty set as the null element (i.e., if $S + o = o + S = S$ for any set S). Both the set-theoretic union and the Minkowski sum [25] satisfy this property, but yield drastically different results when used in our interpolation formula, as illustrated in two dimensions in Figure 2.

Clearly, the Minkowski sum operator is much more appropriate for our purpose than the union, because it combines the geometric characteristics of its two arguments. The definitions, properties and applications of Minkowski sums will be reviewed in Section 2. We have implemented simple and efficient algorithms for computing the Minkowski sum of polyhedra and we use them for automatically constructing the representations of PIPs. The algorithms produce a set of faces, defined in terms of vertices. The vertices are in fact references to pairs of vertices, one of the initial polyhedron, A, and one of the final polyhedron, B. The algorithms

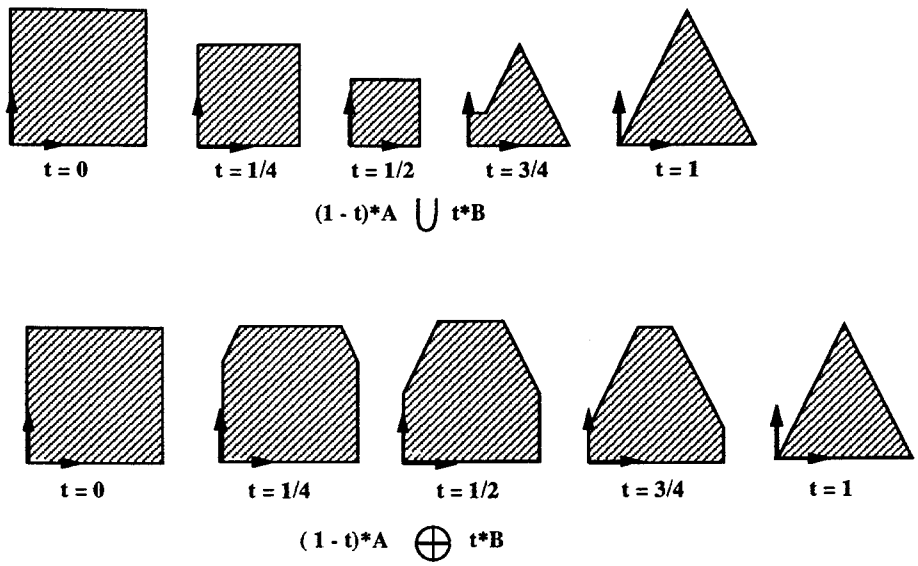

Figure 2: Union versus Minkowski sum: The top and the bottom row shows the states of a PIP that interpolates a square (left) with a triangle (right) at time values 0, 1/4, 1/2, 3/4, and 1. The top row shows a linear interpolation when "+" is interpreted as the union operator. The bottom row shows the effect of using a Minkowski sum operator for that purpose.

automatically compute these pairs and their arrangements for face boundaries. For simplicity, we assume that faces are simply connected, i.e. faces with holes have been split by bridge-edges and are thus bounded by a single loop of edges. The number of faces bounding the PIP typically exceeds the total number of faces bounding A and B.

When both the initial and final shapes are convex, the resulting weighted Minkowski combination is convex and our algorithms compute PIP faces that produce the exact boundary of the deforming polyhedra for each value of time between 0 and 1. When A or B, is not convex,the boundary of their weighted Minkowski combination is always a subset of the faces stored in the PIP's representation. These faces stored in the PIP may intersect each other; however, portions of these faces that do not lie ON the boundary of the Minkowski combination, will lie inside the combination. The correct image of the deforming object may be produced by using the standard depth-buffer or any other approach to hidden surface elimination. The algorithms for both convex and non-convex cases are presented in Section 4.

When complex objects to be interpolated by a PIP are constructed as unions of simpler con-

vex polyhedra, the interpolation may be performed between selected pairs of convex polyhedra, one from the initial object and the other one from the final object. The deforming object is the union of the PIPs each corresponding to a different pair of initial and final convex polyhedra. The optimal choice of the pairing is left to the designer and permits special effects. There is no accepted criterion for selecting a particular pairing, and thus no effort to automate the pairing was made.

1.4 Paper organization

This paper is organized as follows: Section 2 summarizes the relevant properties of Minkowski sums. Section 3 describes the data structure for PIPs and techniques for animating them. Section 4 discusses the algorithm for computing the Minkowski sums of non-convex polyhedra and then specializes it to the summation of convex polyhedra, for increased efficiency. Section 5 discusses the applications and the extensions to the present work. Section 6 concludes the paper.

2 Minkowski sums

The Minkowski sum of two sets, A and B, denoted $A \oplus B$, is defined as: $A \oplus B = \{a + b \mid a \in A, b \in B\}$. It has been used to study material properties [25], to compute collision-free trajectories [14], to define growing, shrinking, rounding, and filleting operations on solids [22], and for character design [7]. A Minkowski sum combines the shape characteristics of its arguments, as demonstrated in Figure 3.

The algorithms developed in this paper are based on the following propositions which have been derived elsewhere ([25],[15]).

Proposition 1 *The shape of the Minkowski Sum of two convex polyhedra is independent of the choice of the coordinate system and is not affected by any translation applied to either of the argument polyhedra.*

However, the final shape is in general dependant on the relative orientations of the two initial polyhedra.

Proposition 2 *The boundary, $\partial(A \oplus B)$, of the Minkowski sum of two sets A and B is a subset of the boundary of the Minkowski sum of their boundaries. $\partial(A \oplus B) \subset \partial((\partial A) \oplus (\partial B))$.*

In this paper we restrict our attention to polyhedra, which have traditionally been used in CAD and in computer graphics to approximate more general shapes. The Minkowski sum of two polyhedra is a polyhedron. Also the Minkowski sum of two convex sets is convex. Consequently the Minkowski sum of two convex polyhedra is a convex polyhedron. Throughout this paper, the symbols V_A, E_A, and F_A will refer to a vertex, an edge, or a face of A. A.vertices, A.edges, A.faces will refer respectively to the sets of all the vertices, edges and faces of A. Corresponding notation will be used for B and for C.

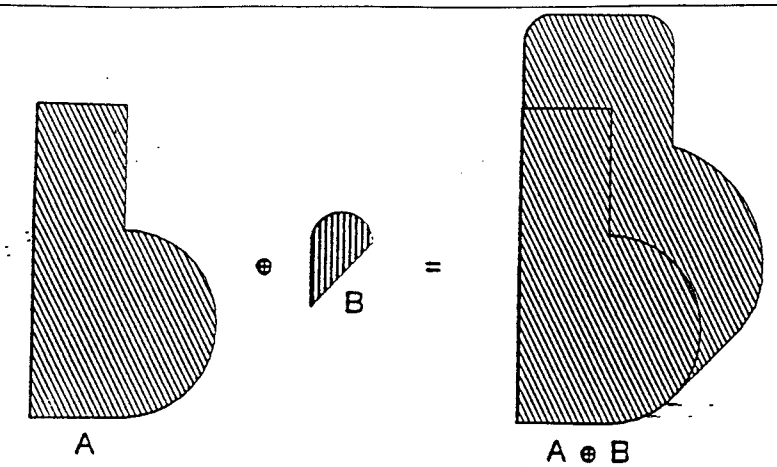

Figure 3: Minkowski sum combines the shape characteristics of its arguments A and B.

Proposition 3 *Given any two arbitrary polyhedra A and B, the faces of $A \oplus B$ are subsets of the union of faces that can be written as $V_A \oplus F_B$, $F_A \oplus V_B$ or $E_A \oplus E_B$. where V_A, E_A, F_A respectively denote a vertex, edge, face of A (similarly for B).*

Proposition 4 *Given two convex polyhedra A and B, any face of $A \oplus B$ may be written as $V_A \oplus F_B$, $F_A \oplus V_B$ or $E_A \oplus E_B$,*

Figure 4 shows an example of the different types of faces in $A \oplus B$.

These two properties provide us with the means for generating a superset of the faces of $A \oplus B$. The algorithms described in section 4 strive at improving performance by reducing the superset that needs to be considered, and by efficiently eliminating the faces that do not bound $A \oplus B$.

The following two properties guarantee that the PIP's faces have constant orientations.

Proposition 5 *Given a face F and a point V, $aF \oplus bV$ has the same normal as F for all V and all positive a and b.*

Proposition 6 *Given two non-parallel edges (line segments) E_A and E_B, the face $aE_A \oplus bE_B$ has constant normal, independent of the values of the positive scaling factors a and b.*

3 Parameterized interpolating polyhedron (PIP)

In this section we discuss the representation and animation of PIPs.

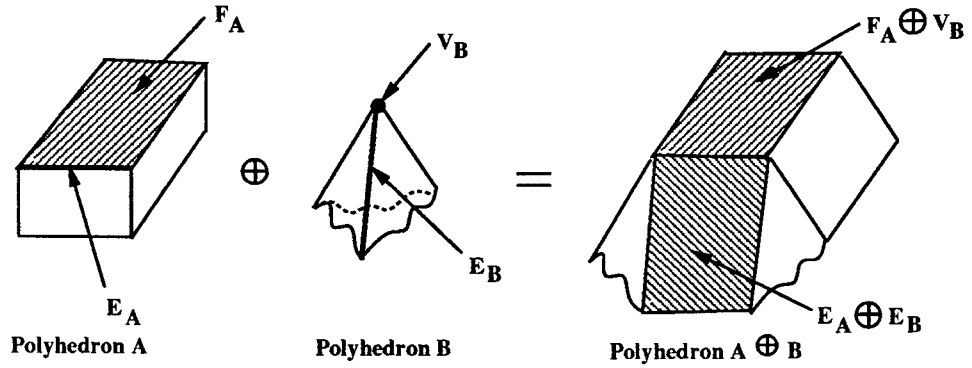

Figure 4: Different types of faces in $A \oplus B$

Definition 1 *The PIP interpolating the polyhedra A and B, denoted PIP(A,B), is a graphic-oriented representation of the family of shapes, $C(t)$, $t \in (0,1)$, such that $C(t) = (1-t)*A \oplus t*B$.*

Definition 2 *N is a locally supporting direction to the set A at point p of A if and only if there exists an open ball B around p such that its intersection with the open half space $\{x : (x-p).N > 0\}$ does not intersect A. We denote such relations by LSD(N,A,p).*

Note that interior points have no locally supporting orientation.

PIP representations are based on a boundary graph that defines faces of $C(t)$, or a suitable superset, denoted $\mathcal{L}$, of these faces called the Locally Supporting Tentative faces in terms of parameterized vertices of the original polyhedra. The faces of $\mathcal{L}$ are parameterized and thus evolve with time.

Definition 3 *The set $\mathcal{L}$ of locally supporting tentative faces of PIP(A,B) is defined as follows:*

For each time-dependant locally supporting tentative face $F(t)$ of $\mathcal{L}$, one of the following three rules holds:

- *$\exists V_A \in A.vertices$ and $F_B \in B.faces \ni F(t) = (1-t)*V_A \oplus t*F_B$ and $LSD(N_{F_B}, A, V_A)$. Such faces are of type VF.*
- *$\exists V_B \in B.vertices$ and $F_A \in A.faces \ni F(t) = (1-t)*F_A \oplus t*V_B$ and $LSD(N_{F_A}, B, V_B)$. Such faces are of type FV.*
- *$\exists E_1 \in A.edges$ and $E_2 \in B.edges \ni F(t) = (1-t)*E_1 \oplus t*E_2$, $LSD(N_E, A, a)$ and $LSD(N_E, B, b)$ for any point $a \in E_1$ and any point $b \in E_2$. T_E is the tangent along edge E and $N_E = T_{E_1} \times T_{E_2}$. Such faces are of type EE.*

N_F *represents the outward pointing normal to face* F.

Locally Supporting Tentative faces ($\mathcal{L}$) have the following properties:

Proposition 7 *For all values of the scaling factor t, the boundary of the final result of the Minkowski combination of A and B is a subset of the union of the faces in* $\mathcal{L}$, *i.e.*

$$\forall t,\ \partial C(t) \subset \bigcup_{F \in \mathcal{L}} F(t).$$

In short $\mathcal{L}$ contains the boundary of C.

Proposition 8 *For all values of the scaling factor t, faces in* $\mathcal{L}$ *are a subset of the closed set* $C(t)$, *i.e.*

$$\forall t,\ (\bigcup_{F \in \mathcal{L}} F(t)) \subset C(t).$$

It can be seen from the above proposition that the excess of faces of $\mathcal{L}$ that are not on ∂C lie inside C and thus will not be visible from the outside of the object if hidden surface removal techniques are used for graphics. We exploit this property and provide realtime animation of PIPs of not necessarily convex objects without having to compute the boundary of $C(t)$.

The propositions below are used to further improve animation performance.

Proposition 9 *The normal to any face in* $\mathcal{L}$ *is an invariant of time.*

Proposition 10 *Any vertex* $V(t)$ *of any face in* $\mathcal{L}$ *of PIP(A,B) may be written as* $V(t) = (1-t) * V_A + t * V_B$, *where* V_A *and* V_B *are vertices of A and B.*

3.1 Representation of a PIP

The initial and final shapes A and B are represented in the standard boundary form. We use an ***adjacency graph*** that defines faces in terms of their bounding edges and edges in terms of their end-vertices. These graphs capture face-face adjacency relations. The edge pointers are ordered around faces and around vertices to permit counter-clockwise traversal. The resulting PIP is represented by a similar graph except for its vertices, which are associated with pairs of pointers, the first pointing to a vertex in A and the second pointing to a vertex in B. In the process of constructing the PIP we keep track the "parent entities" in A and B (as will be explained later) from which each face was created.

3.2 Animation of the PIP

Whenever the parameter 't' is changed, we recompute the current value of all the vertices, as given by : $V_C(T) = (1-t) * V_A + t * V_B$, using the hardware-assisted matrix multiplication of the graphic pipeline.

Since the face normals are constant during the animation process, we need never recalculate them, and the update of the vertex coordinates is sufficiently fast to permit real time animation of PIPs of complex objects. (It is in fact faster than the display process.)

Figure 5 shows excerpts of a simple deformation of a brown rectangular block (A) into a blue cone (B). Note that faces of the evolving polyhedron move and shrink or grow, but remain parallel to themselves. For t = 0 and 1, most faces have shrunk to edges or vertices.

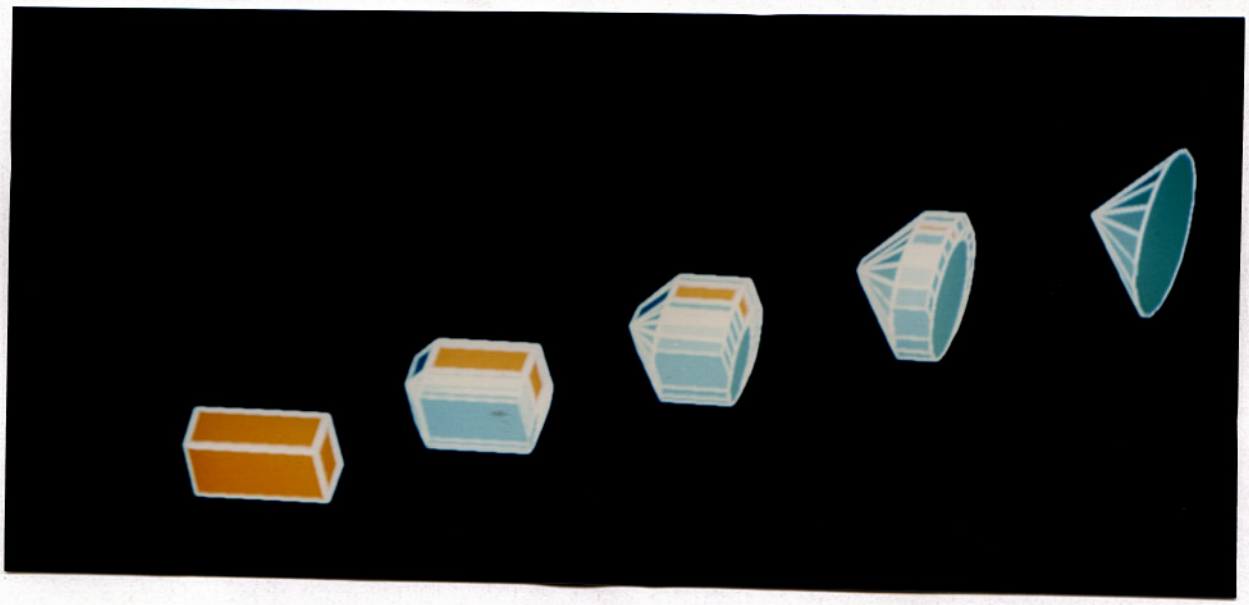

Figure 5: Deformation of a block into a cone.

The PIP represents a 3D structure that can be rotated, translated or magnified and displayed at interactive rates, using standard graphic pipelines and a z-buffer for hidden surface removal.

Some of the PIP's faces may be coplanar and may overlap. This happens in singular cases where, for example, edges of A are parallel to faces of B or vice versa. Since these faces have the same normal, we need not handle overlapping faces in any special manner. (Displaying overlapping faces one by one produces the same results as displaying their union.)

4 Algorithm for computing PIPs

In this section we describe algorithms for computing the $\mathcal{L}$ faces of PIP(A,B) that interpolates two polyhedra A and B in 3D. We first consider the case where the operands need **not** both be convex and focus on the situations where A and B are in *general position*. ie., where no edge of A is parallel to a face of B and vice versa. When both A and B are convex, we use a more

efficient algorithm defined later on in this section. For polyhedra not in general position, we refer the reader to [12].

4.1 Algorithm for non-convex polyhedra in general position

Given two polyhedra A and B, which need not be convex, the algorithm computes the $\mathcal{L}$ faces of PIP(A,B) by generating all tentative faces of type FV, EE or VF and by rejecting faces that do not satisfy definition 3.

The generation of the tentative faces requires a simple traversal of the list of the boundary graph of A and for each of its elements we traverse the corresponding list in the boundary graph of B. For example, for a vertex of A we traverse the face list of B and for each pair VF generate a tentative face. Similarly, for each edge of A, we traverse all the edges of B.

To test a tentative face, we check if one of its two normals is an LSD for both A and B at the appropriate vertices.

Let each face F of a convex polyhedron D be associated with an ***outward normal direction*** N_F, such that the point $P_F + \delta N_F$ is out of D for any point P_F in F when δ approaches 0 on the positive side.

Let the ***tangent vector*** to an edge E, denoted T_E, be the unit vector parallel to E.

Given a face F and edge E, the ***inface normal*** $N_{E,F}$ to E with respect to F is defined by the unit vector with direction $N_F \times T_E$, oriented such that it points towards the interior of the face.

The tentative VF, FV, EE faces are tested as follows.

- **VF case:** Let $F = V_A \oplus F_B$. The outward normal N of F is the outward normal to F_B. To test LSD(N, A, V_A), we check that for all edges E of A, adjacent to V_A, the tangent T_E to E (defined as pointing away from V_A) has a negative dot product with N.
- **FV case:** Same as above by interchanging A and B.
- **EE case:** Let $F = E_A \oplus E_B$. Let $\mathcal{N}_E$ define the set of inface normals of all the faces bounded by the edges E_A and E_B. Also let $N = T_{E_A} \times T_{E_B}$. Then F is valid if: $\forall N_i \in \mathcal{N}_E$, $N \cdot N_i$ has constant sign. If the sign is positive, we use $-N$ as the outward normal.

The Algorithm is shown in Figure 6 and an example of computing the Minkowski sum of two tori is shown in Figure 7.

4.2 Algorithm for convex polyhedra in general position

The previous algorithm requires an exhaustive enumeration of all pairs of elements of the graphs of A and B whose dimensions sum up to 2. In 2D the boundary of the Minkowski sum of two convex polygons may be computed by traversing both polygons simultaneously. The traversal synchronization selects the edges that produce the least sharp turn [9]. Taking advantage of the convexity property in 2D reduces the quadratic performance to linear. The extension of this wrapping algorithm to 3D requires a more elaborate approach which is presented in this section.

Input : Polyhedra A and B as boundary-adjacency graphs.
Output: PIP(A,B)

1. Combine every face F_A of A with vertices V_B of B satisfying $LSD(N_{F_A}, B, V_B)$
2. Combine every face F_B of B with vertices V_A of A satisfying $LSD(N_{F_B}, A, V_A)$
3. Combine every edge E_A of A with edges E_B of B which satisfy the conditions $LSD(N, A, a)$ and $LSD(N, B, b)$ where $N = T_{E_A} \times T_{E_B}$, $a \in E_A$ and $b \in E_B$.

Figure 6: Algorithm for Minkowski Sum of Non-Convex Polyhedra

Proposition 11 *For convex polyhedra, A and B, in general position, each face of A and of B appears once and only once amongst the FV and VF pair faces of $PIP(A, B)$.*

Corollary 1 *For any face F_A of A, in general position, there always exists one and only one vertex of B such that $F_A \oplus V_B$ is a face of C.*

If there exists a pair (E_A, V_B) such that, $E_C = E_A \oplus V_B$, then E_A and V_B are said to be the *Parents* of edge E_C, and we write $P(E_C) = (E_A, V_B)$. Similarly, if $E_C = E_B \oplus V_A$ then we write $P(E_C) = (E_B, V_A)$. A Similar notation is used for faces: if $F = V_A \oplus F_B$ then $P(F) = (V_A, F_B)$ or if $F = E_A \oplus E_B$ then $P(F) = (E_A, E_B)$.

Proposition 12 *For A and B in general position, the parents $P(F_C)$ of any face F_C of C are unique.*

4.2.1 Overview of the algorithm

We construct the adjacency graph of C from the graphs of A and B by a "wrapping" approach. During this process we maintain a partial graph of C and increment it until it is complete by a procedure analogous to wrapping. This partial graph, G_C, contains faces of C. In those faces, some edges are such that there is only one face incident on the edge. Such edges are called *open*. The algorithm is based on an operation which adds a new face to an *open edge* in G_C.

We know from Proposition 11 that a face of A and B can appear only once as an FV or VF face of C. Therefore, if a face of A or B has been used once as a parent of an FV or VF face in G_C, we mark it as *used* in the graphs of A and B.

The algorithm keeps a list of faces of G_C that have open edges and keeps applying face-adding operations to the first open edge in the list until the list is empty.

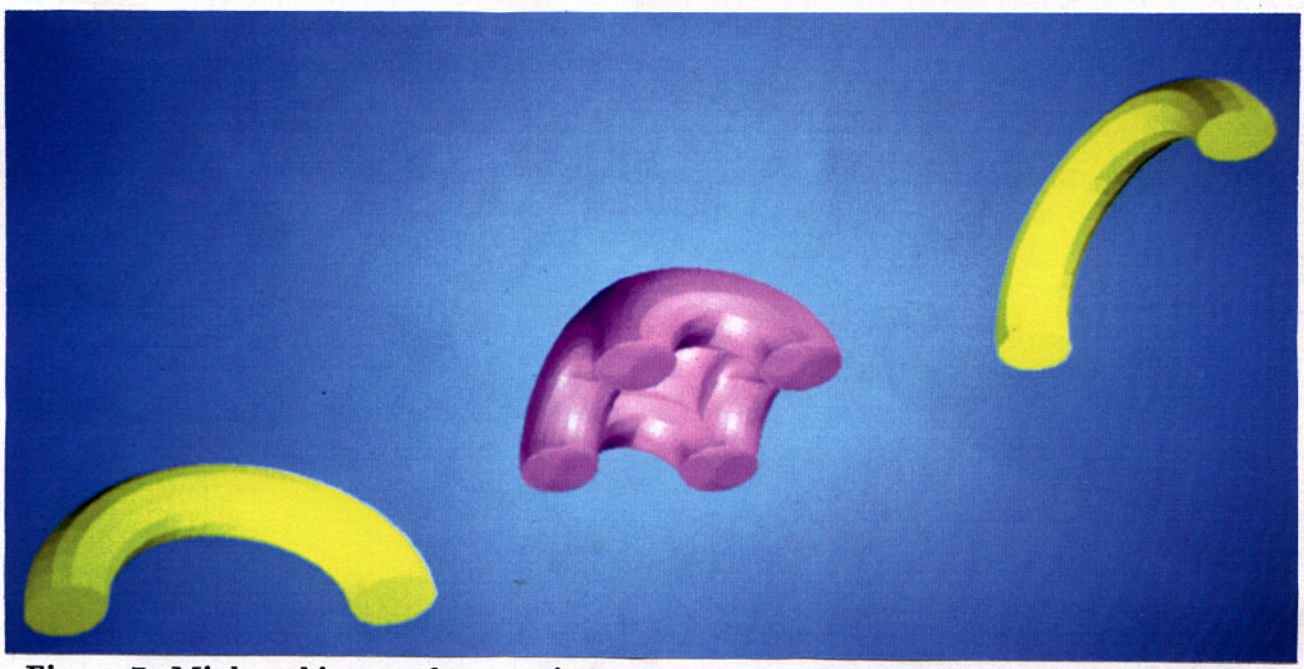

Figure 7: Minkowski sum of two tori.

The Algorithm is outlined in Figure 8. For Details the reader should consult [12].

Without loss of generality, suppose that $P(E_C) = (E_A, V_B)$, then the candidate faces are restricted to be $V_B \oplus F$, where F is the unused face of A containing E_A, or $E_A \oplus E_B$, where E_B is bounded by V_B.

Although this specialized algorithm for convex polyhedra is on average considerably more efficient than the general one, they have the same worst case complexity. If one of the polyhedra has 'n' faces and the other has 'm' faces then it can be shown [11] that the output can have upto $\mathbf{O}(mn)$ faces. Both our previous algorithms have a worst case complexity of $\mathbf{O}(mn)$. A more efficient approach is presented in [10], where they employ a topological sweep in the dual of the boundary graph, where faces map to graph nodes, edges to links between these nodes, and vertices to the connected regions bounded by dual edges. The topological sweep maintains a complex vertical ordering and can thus ensure that a single loop of open edges is maintained during the algorithm. Because the complexity of the polyhedra we plan to manipulate is small, we have chosen not to implement this more complicated approach.

Input : Polyhedra A and B as boundary-adjacency graphs.
Output: Polyhedron $C = A \oplus B$

1. Determine a starting face in C
   (a) For a face of A, find the corresponding "supporting" point in B. The Minkowski combination of this pair gives the starting face F_C.
   (b) Use F_C to initialize G_C.
2. **While** G_C contains an open edge E_C **do**
   (a) Determine candidate faces that connect to E_C.
   (b) Select the appropriate face from the list of candidate faces.
   (c) Use the selected face to update G_C

   End do

Figure 8: Algorithm for convex polyhedra

5 Combining motions and deformations

The solids A and B that define a PIP may be specified using any of the available techniques for constructing solid representations (invocation of solid primitives, sweeps and extrusions, boundary evaluation from CSG, and even Minkowski sums).

Their relative position, which may be used to alter the PIP, may also be specified in different ways. For example, the LAMBADA design environment developed by the Interactive Geometric Modeling group at IBM Research offers interactive facilities for positioning the components of an assembly through incremental rotations and translations expressed in any coordinate system [21]. Integrated in LAMBADA, the MEDITOR module [13] may be used to specify and edit rigid body motions of subassembly components relative to their parent nodes in the assembly graph. By supporting PIP objects in LAMBADA, we can combine and synchronize motions and deformations. For example, one can specify the motion of the local coordinate system of a PIP and synchronize that motion with the PIP's deformation.

The motion of an object is designed in MEDITOR by interactively specifying the positions that the object should occupy at the desired time values. The simplest motion is specified by selecting the starting and the ending positions for the object. MEDITOR generates a smooth motion so that the object's trajectory and orientation interpolate the specified positions at

specified time values.

Given two objects A and B, we can design a time-dependent transformation which combines a PIP deformation with a rigid body motion and which produces an object that smoothly evolves from A to B. The PIP's face normals may change during such a transformation, but these changes are related to the rigid body motion transformation and are automatically performed by the appropriate matrix multiplications in a graphic pipeline architecture.

The designer may control the contribution of the motion to this combined transformation. For example, restricting the motion to be identity will produce the standard PIP. On the other hand, if B is a rotated version of A, a smooth motion may be used for the interpolation without any deformation.

Let the final positions of A and B be defined by two rigid body transformations, M_A and M_B. The user may specify the rigid body motion of the PIP by using M_A as the starting position of the motion and by specifying a transformation, M'_B that brings B into the desired position relative to A so as to minimize the shape difference between $M_A * A$ and $M'_B * M_B * B$. ($M_A * A$ defines the solid A transformed by M_A.) The PIP is now defined at any intermediate time by PIP($M_A * A, M'_B * M_B * B$). The rigid body motion interpolates the local coordinate systems $M'_B * M_B$ and M_B, and compensates the adjustment performed by the designer to optimize the deformation.

Criteria for deformation optimization may be artistic. To automate the adjustment specified above, we could define the optimal position as one which minimizes the Hausdorf distance between the boundaries of both objects, i.e. minimizes the maximum distance between a point on one object and the boundary of the other object. A more practical approach would be to align, during the adjustment, the centers of mass and the principal axes of inertia of both objects, but this approach may fail to produce acceptable results for non-convex objects. Finer techniques should be investigated. Figure 9 shows the interpolation of a long, thin block with a long, thin cone. Note that in the intermediate stages, the PIP is much "fatter" than the original objects. Figure 10 shows the same interpolation in which the cone is given a motion. The cone was aligned with the block, the Minkowski sum computed, and then the cone was transformed to its final position. The results from this sequence of operations are remarkably different from those obtained in Figure 9.

5.1 Bilinear combination of solids

For PIPs of convex polyhedra, the faces and edges of the PIP have constant orientation through time. Therefore, the boundary graph of a PIP (with vertex coordinates at any valid time value) may be viewed as a standard polyhedron and thus may be used as an argument for constructing extended PIPs.

For example, $PIP(PIP(A,B), PIP(B,C))$ is a parabolic (Bezier) curve in the space of all convex polyhedra having as its control points the convex polyhedra A, B, and C. The extended PIP interpolates A and C, but its intermediate positions are influenced by the shape B. Cubic and higher degree Bezier curves may be produced in the same manner. By controlling the shapes, sizes, and positions of the control "solids", the designer controls the evolution of complex animations.

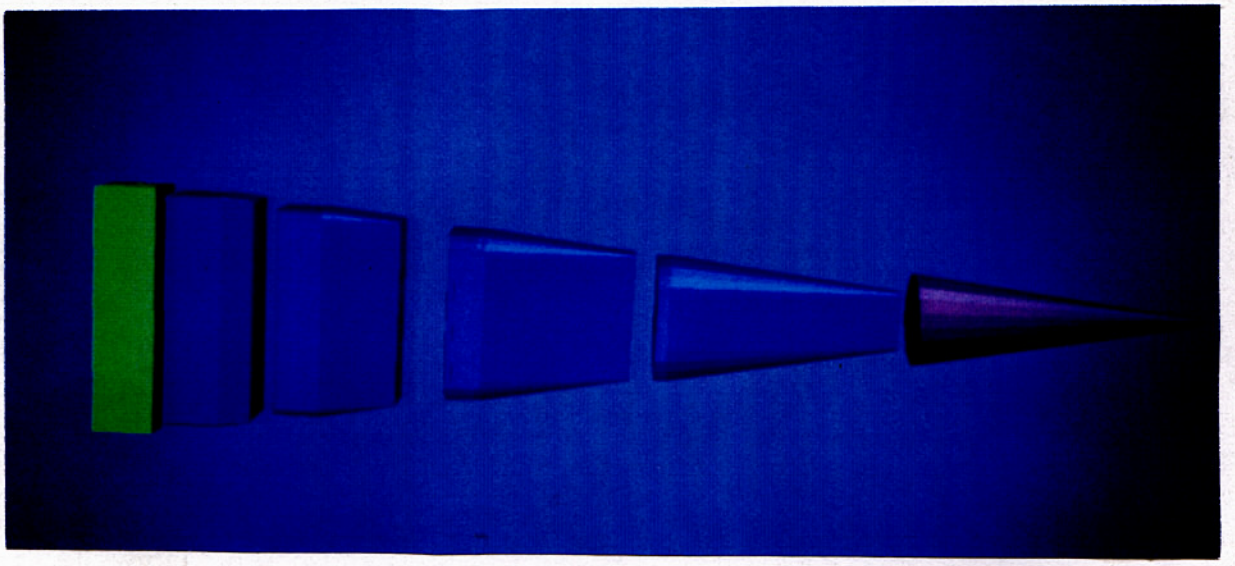

Figure 9: Interpolation of a long block with a long cone with no motion.

Extended PIPs may be represented hierarchically, in which case vertices of the final extended PIP faces point to vertices of its two constituents, which in turn point to vertices of objects or PIPs used in its own definition, and so on.

A PIP blends the shape characteristics of the objects it interpolates. As time varies, the weights of the shape contributions of each object vary. Designers may use this technique to interactively select optimal shapes. For example, if A is a block and B a (finely tesselated) ball, $PIP(A, B)$ will produce a block with rounded corners, where the amount of rounding may be interactively specified by changing time. Note that this technique is not equivalent to constant radius rounding, as defined in [22]. Indeed, constant radius rounding is defined as a sequence of two Minkowski operations: first B is subtracted from A (by a Minkowski subtraction which has the same effect as modifying the complement of A by a Minkowski sum with B), then B is added to the result through a Minkowski sum. The PIP, on the other hand, is composed of a scaling of A followed by a Minkowski sum with B. Note that when A is convex, the two approaches produce congruent results (i.e. shapes that only differ by a translation).

For convex polyhedra A, B, C, and D, by using the same parameter v as time for both $PIP(A, B)$ and $PIP(C, D)$; and by using a second parameter u for the extended PIP combining

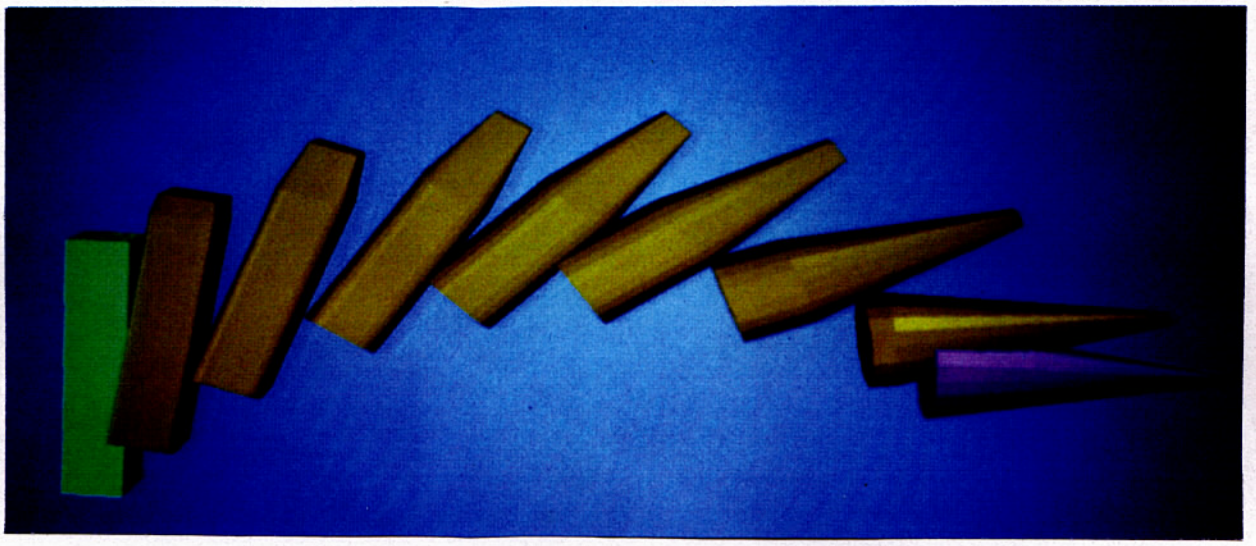

Figure 10: Interpolation of a long block with a long cone in which the cone is given a motion.

them, we obtain a bi-parametric patch in the space of polyhedra:

$$S(u,v) = PIP(PIP(A,B), PIP(C,D))$$

that interpolates A, B, C, and D. Specifically, $S(0,0) = A$, $S(0,1) = B$, $S(1,0) = C$, and $S(1,1) = D$. By changing the values for u and v, designers may interactively explore this family of shapes. By specifying the evolution of u and v as a function of time (for example through a cubic curve in the $u - v$ space and parameterized by time), one can define more complex animations. Figure 11 illustrates the use of such a bi-parametric patch, where the PIP patch (S) interpolates four shapes: a tetrahedron, a block, a pentagonal prism and a cylinder. The interpolated solids in the top and bottom rows correspond to $S(0.25,0)$, $S(0.5,0)$ and $S(0.75,0)$ and are interpolations of the two corner solids in their respective rows. The solids along the diagonal correspond to $S(0.25,0.25)$, $S(0.5,0.5)$ and $S(0.75,0.75)$ and define solids which combine all four shapes and are obtained by interpolating the top and the bottom solids in the respective rows.

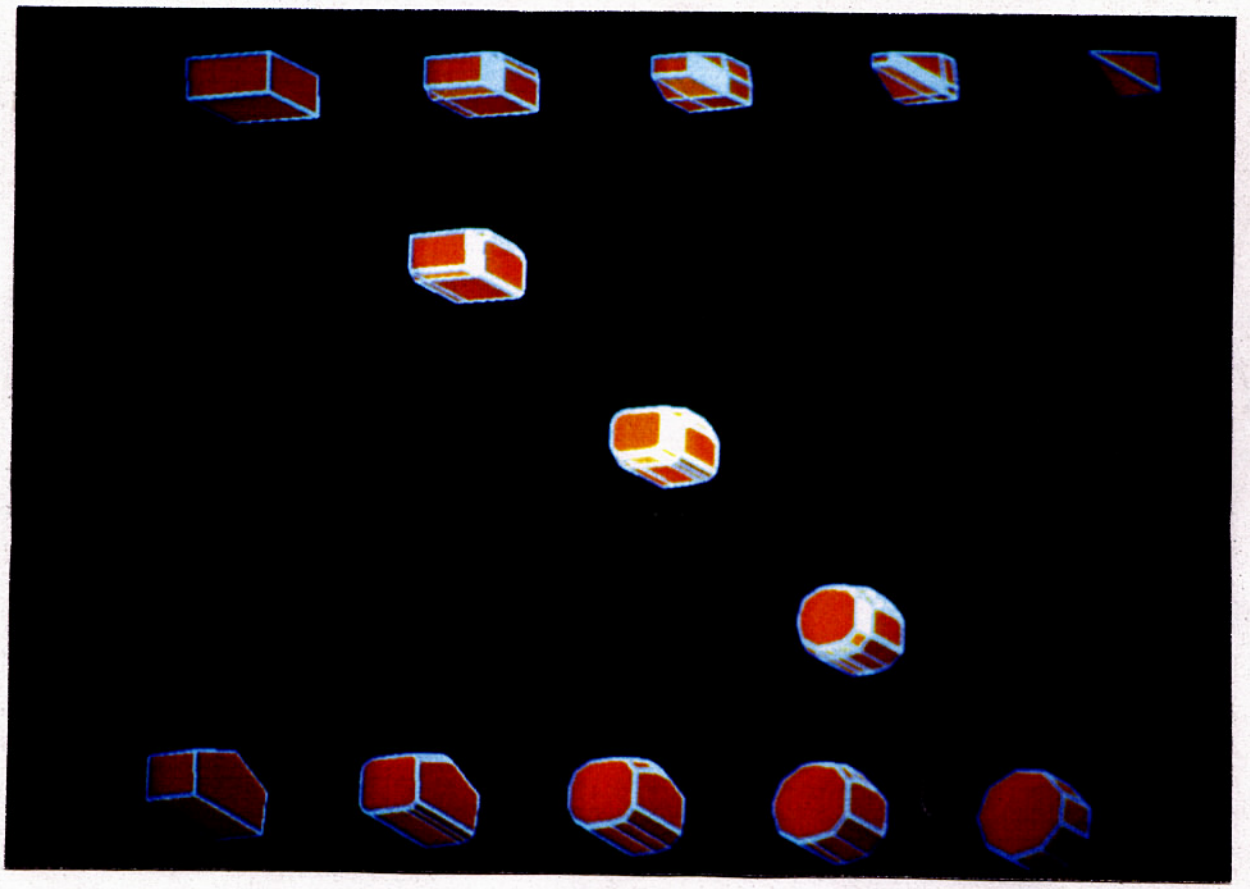

Figure 11: Example of bi-parametric patch interpolating 4 shapes.

6 Conclusion

We have presented a new, simple primitive for interactively constructing and visualizing animation sequences. The primitive, called PIP for Parameterized Interpolating Polyhedron, is a parameterized boundary representation graph which defines a list of faces that evolve through time. Their evolution is composed of a translation and a scaling operation that are linear functions of time, and thus the normals are constant throughout the evolution.

PIPs are animated by computing, for each new time value, the current coordinates of the vertices, and by simply rendering the faces using a depth-buffer architecture. The computation

of the vertices only requires additions and multiplications, which can be efficiently performed by using hardware-supported scaling operations available in standard graphic pipelines.

PIPs are completely defined by the initial and final shapes, A and B, which can be any arbitrary polyhedra. It is important to note that the two polyhedra need not have corresponding boundary elements, nor any parameterization of their bounding surfaces. The PIP behaves as the weighted Minkowski average of A and B, which can be written: $(1-t)*A+t*B$, where t is time. When A or B are not convex, the faces of the PIP are a convenient superset of the boundary of the corresponding solid, and yield correct images for all values of time in the specified interval.

We have described two algorithms for computing the boundary graphs of PIPs from the boundary graphs of A and B. The first is a simple algorithm for general polyhedra. The second only applies to convex arguments, but is more efficient. The correct treatment of singular cases, where faces or edges of A are parallel to faces or edges of B, is presented elsewhere [12].

PIPs can be combined with rigid body motions and also used as arguments to construct extended PIPs. Both techniques yield a variety of tools for the interactive selection of optimal shapes and for the interactive design of animations.

7 Acknowledgements

We are indebted to V. Srinivasan for supporting this work at IBM and to J. Mastrogiulio for integrating our programs within LAMBADA.

References

[1] Barr A. H., "Global and local deformation of solid primitives", PROC. ACM SIGGRAPH'84, Minneapolis, MN, pp. 21-30, July 23-27, 1984.

[2] Borrel P. and Bechmann D., "Deformation of n-dimensional objects", IBM Research Report, RC16205, IBM Research, Yorktown Heights, New York, 10598.

[3] Brown C. M., "PADL-2: A technical summary", IEEE Computer Graphics and Applications:, Vol. 2, No. 2, pp. 69-84, March 1982.

[4] Coquillart S., "Extended Free-Form Deformation: a Sculpturing tool for 3D Geometric Modeling" ACM Computer Graphics, Vol. 24, No. 4, 1990.

[5] Dietrich W. C. Jr, Nackman L. R., Sundaresan C. J., and Gracer F., "TGMS: An Object-Oriented System for Programming Geometry", IBM Research Report RC 13444, January 1988.

[6] Fournier A. and Wesley M. A., "Bending polyhedral objects", Computer Aided Design, Vol. 15, pp. 79-87, 1982.

[7] Ghosh P. K, "A Computational Theoretic Framework for Shape Representation and Analysis using the Minkowski Addition and Decomposition Operators", PhD Dissertation, Tata Institute of Fundamental Research, Bombay. 1986.

[8] Grossman D. D., "Procedural representation of three-dimensional objects", IBM Journal of Research and Development, Vol. 20, pp. 582-589, November 1976.

[9] Guibas L, Ramshaw L, Stolfi J, "A Kinetic Framework for Computational Geometry", IEEE 24th annual symposium on Foundations of Computer Science, 1983.
[10] Guibas L, Seidel R, "Computing Convolutions by Reciprocal Search", Discrete and Computational Geometry Vol. 2, pp. 175-193, 1987.
[11] Kaul A, Srinivasan V, "Output Size of Minkowski Sums of Polygons and Polyhedra", in preperation.
[12] Kaul A, Rossignac J, "Solid Interpolating Deformations: Construction and animation of PIPs", IBM Research Report RC 16387 December 1990.
[13] Kim J. and Rossignac J., "A screw motion primitive for the interactive design of rigid body movements", IFIP WG 5.2 Workshop on Geometric Modeling for Product Engineering June 17-21, 1990 Rensselaerville, NY.
[14] Lozano-Parez T., Wesley M. A., "An algorithm for planning collision free paths among polyhedral obstacles" Comm. ACM Vol. 22, pp 560-570, 1979.
[15] Matheron G, "Random Sets and Integral Geometry." Wiley, New York.
[16] Norton A, Turk G, Bacon R, Gerth J, and Sweeney P, "Animation of fracture by physical modelling", to appear in The Visual Computer.
[17] Pentland A., Essa I., Friedman M., Horowitz B., and Sclaroff S., "The ThingWorld Modeling System: Virtual Sculpting By Modal Forces", ACM Computer Graphics, June 1990.
[18] Platt J. and Barr A., "Constraints Methods for flexible models", ACM Computer Graphics, Vol. 22, No. 4, August 1988.
[19] Pletinckx D., "Quaternion calculus as a basic tool in computer graphics" The visual computer Vol 5 pp 2-13, 1989.
[20] Requicha A. A. G., "Representations for rigid solids: Theory, methods, and systems", ACM Computing Surveys, Vol. 12, No. 4, pp. 437-464, December 1980.
[21] Rossignac J., Borrel P., Kim J., and Mastrogiulio J., " BIERPAC: Basic Interactive Editor for the Relative Position of Assembly Components," IBM Research Report.
[22] Rossignac J., and Requicha A. A. G., "Offsetting operations in solid modelling", Computer Aided Geometric Design, Vol 3 pp 129-148, 1986.
[23] Rossignac J. R., Borrel P., and Nackman N. R., "Interactive design with sequences of parameterized transformations", Proceedings of the Second Eurographics Workshop on Intelligent CAD Systems: Implementation Issues, April 11-15, 1988, Veldhoven, The Netherlands, pp. 97-127. IBM Research Report RC 13740, IBM Research Division, Yorktown Heights, NY, May 1988.
[24] Sedeberg T. W. and Parry S. R., "Free-Form Deformation of Solid Geometric Models", SIGGRAPH 1896, Vol. 20, No. 4, 1986.
[25] Serra J, "Image Analysis and Mathematical Morphology", Academic Press, 1982.
[26] Terzopoulos D., Platt J., Barr A., and Fleischer K., "Elastically Deformable Models" ACM Computer Graphics, Vol. 21, No. 4, July 1987.
[27] Witkin A., Fleischer K., and Barr A., "Energy Constraints On Parameterized Models", ACM Computer Graphics, Vol. 21, No. 4, July 1987.
[28] Witkin A. and Kass M., "Spacetime Constraints", ACM Computer Graphics, Vol. 22, No. 4, August 1988.