

1979

Geometric modeling using the Euler operators

Charles M. Eastman
Carnegie Mellon University

Kevin J. Weiler

Follow this and additional works at: <http://repository.cmu.edu/compsci>

This Technical Report is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Computer Science Department by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:

The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

GEOMETRIC MODELING USING THE EULER OPERATORS

by

Charles Eastman and Kevin Weiler

DRC-15-2-79

May 1979

Geometric Modeling Using the Euler Operators

Charles Eastman and Kevin Waiter
Institute of Physical Planning
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

A recent advance in the modeling of three-dimensional shapes is the joint development of *bounded shape models*, capable of representing complete and well-formed arbitrary polyhedra, and operators for manipulating them. Two approaches have been developed thus far in forming bounded shape models: to combine a given fixed set of primitive shapes into other possibly more complex ones using the spatial set operators, and/or to apply lower level operators that define and combine faces, edges, loops and vertices to directly construct a shape. The name that has come to be applied to these latter operators is the *Euler operators*.

This paper offers a description of the Euler operators, in a form expected to be useful for prospective implementers and others wishing to better understand their function and behavior. It includes considerations regarding their specification in terms of being able to completely describe different classes of shapes, how to properly specify them and the extent of their well-formedness, especially in terms of their interaction with geometric operations. Example specifications are provided as well as some useful applications.

The Euler operators provide different capabilities from the spatial set operators. An extensible CAD/CAM facility needs them both.

1. Introduction

A recent development in computer-aided design for manufacturing has been the development of *bounded shape models* of manufactured parts. The distinction between this work and earlier surface oriented 3-D models is that, in bounded shape models, individual surfaces are consistently structured together to define the complete surface bounding a shape. Well-formedness conditions are imbedded in the operations to guarantee the correct automatic forming of the shape model. The effect, in contrast to surface modeling, is faster part development and fewer user errors (Baer, Eastman, Henrion, 1978).

The structure of a bounded shape model is comprised of partial surfaces that are bounded by one or more loops of edges. The partial surfaces are called *faces*. Each *loop* is the concatenation of line segments, called *edges*, into a closed ring. Edge segments are bounded by *vertices* at their intersection. The entire assembly of faces forms one complete bounding surface, and is called a *shell*.

Two approaches have been followed by developers of bounded shape models. One is to rely on a fixed set of primitive shape models, each well formed, and to use the spatial set operators (Braid, 1975; Requicha and Voelker 1977) to derive complex shapes that are well-formed combinations of the primitives. The second approach is to rely on a lower level set of operators that combine faces, edges, loops and vertices so as to define new primitive models or to develop other high level shape operators, in addition to the spatial set operators.

This second approach gained impetus with the work of Baumgart (1972) who showed that such operators could incorporate well-formedness conditions in the composition of bounded shape models. These operators were called the Euler operators by Baumgart, because they were derived from a proof of Euler's Law. Several research groups have adopted the Euler operators (Eastman, Uvidini, Stoker, 1975; Braid, 1978). However, there has not been available an adequate description of them, in terms of:

¹This research has been supported in part by the National Science Foundation under contract MCS75-19072

- completeness, so as to allow the generation of any legal polyhedron
- *proper* specification, so that their behavior is completely known, without hidden side effects
- the extent of their well-formedness, so that their interaction with other operations are precisely understood.

In this paper, we present a detailed description of the Euler operators that include these aspects, so that they can be more readily implemented by groups wishing to use them. We also describe their implementation in GLIDE, a database language developed for CAD at Carnegie-Mellon University (Eastman and Henrion, 1977,1978) as an example. Last, we present some sample applications, to show how the functionality of the Euler operators is fundamentally different from the use of shape primitives.

2. Functionality of the Euler Operators

2.1. Topology and Geometry

The feature distinguishing bounded shape models from earlier surface models is the structure maintained between the component surfaces and their lines and points of intersection. It is this structure that incorporates most (but not all) of the well-formedness conditions in bounded shape models. Thus it is useful to distinguish the topological structure of the component surfaces from, the geometric properties of the surfaces. We call them the *Topology* and *Geometry* of a shape respectively. Thus for a cube, the geometry defines the placement of its six surfaces and possibly composite data regarding the location of their intersections, whereas the Topology identifies that each of its six Faces is bounded by one Loop made up of four Edges and Vertices. In more abstract terms, the Topology of a cube defines a particular partitioning of a surface having the *standard form* of a sphere. Geometry defines a particular distortion of the sphere, eg. cuboid, rectilinear solid, trapazoidal solid.

The distinction between a shape's Geometry and its Topology is useful on several grounds. Though both are required to characterize a particular shape, the same Topology may be appropriate for many shapes, eg. all wide-flange steel members. By using the same topology for the class, a significant reduction in data storage is allowed (see Eastman, 1976). Also many operators are strictly geometric in nature, eg. the symmetry operations of scaling, dilation, rotation and translation. At the same time it is possible to conceive many meaningful ways to define 3-dimensional shapes, eg. by deducing them from multiple orthographic views drawn with a digitizer tablet, by sculpting using the shape operators, by

rotating a line about an axis or by distorting the geometry of an existing shape, for example. Each of these but the last requires the construction of a possible unique Topology. The Euler operators provide a set of medium level tools for developing such application programs, without resorting to programming in the base system-level language. They greatly reduce the bookkeeping requirements needed to guarantee that the resulting shapes are well-formed

2.2. EuJer's Law

For topologies having the standard form of a sphere, the mathematician Euler showed that the number of faces f minus the number of edges e plus the number of vertices v always equals two, or:

$$T = f - e + v = 2$$

Known as *EuJer's Law*, this equation is easily demonstrated constructively by starting from a minimal topology partitioning a sphere. It is made of one face bounded by a single vertex $f = 1, v = 2$. With this as a start, if we add an edge, it must either close on itself, making a new face or be bounded by a new vertex. These two cases are $f = 1, v = 0$ or $f = 0, v = 1$. These two operators thus maintain the equality set up by the minimal topology. With them we can create any combination of values for the three parameters, or what is equivalent, any partitioning of a sphere by a connected graph. The first operator might be called MEF (for make edge face) and the second MEV (for make edge vertex), and are examples of the Euler operators. Figure 1 shows how these operators may be used to define the topology of a cube (technically called a hexahedron). The reader can experiment to see how these two operators may be used to derive a number of shape topologies.

Several issues arise in defining and implementing the Euler operators. First, several classes of meaningful polyhedra are excluded from those allowed by the simple form of Euler's Law given above. Second, we must properly specify the behavior of each operator and the external information it requires.

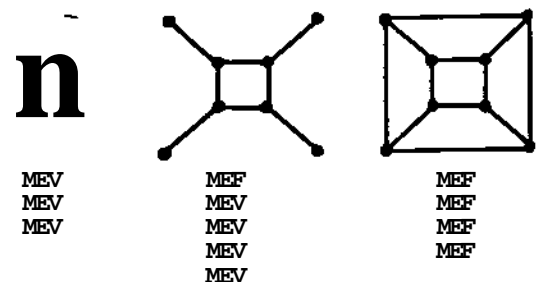


Figure 1: A sequence of seven MEV and five MEF are needed to construct a cube, once the minimal topology is created.

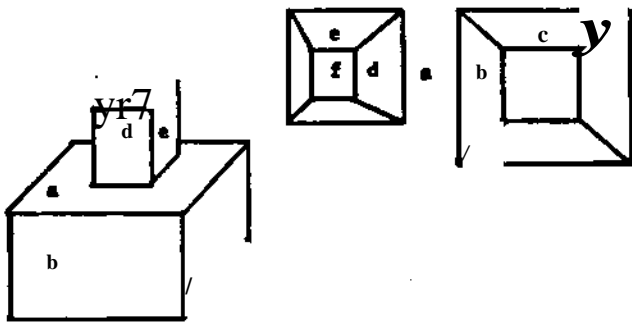


Figure 2: Meaningful shapes may be represented by disjoint graphs, when a face has an inner loop in addition to its outer one. Face A has such a loop in this example.

Three classes of shapes are excluded from the simple form of Euler's Law. First, some useful polyhedra are defined by *disjoint graphs*. See Figure 2. They consist of a graph embedded within a face of another graph. The result is that the top face of the bottom block is bounded by two disjoint sets of edges: 5 - 6 - 7 - 8 - 5 and 9 - 10 - 11 - 12 - 9. We can say that this face has an inner loop or *hole*. The corresponding extension to Euler's Law is:

$$F - E + H - 2$$

where H is the cardinal value of holes within all faces of the shape.

The second restriction on meaningful shapes imposed by the simple form of Euler's Law is that many useful shapes cannot be reduced to the normal form of a sphere, for example, piping or other hollow tube. These shapes are distinguished by having holes through them, providing a grip or *handle*. See Figure 3. Handle corresponds to *genus* of a graph in topology. The corresponding extension, and full definition of Euler's Law is

$$F - E + V - 2 \cdot \{S - G\}$$

where S is the cardinal value of shells (in these cases always one) and G is the genus or number of handles for the shape. Shells have also been traditionally called *bodies*, after Baumgart, and later sections of this paper use the words interchangeably.

The third class of restrictions responds to one other class of meaningful shapes not considered here. These are the hollow solids, for example a pressure vessel, that is composed of one or more shells inside of the outer one. Thus S, along with the other parts, may vary in number. For an insightful history of Euler's Law see Lakatos, 1975.

The new form of Euler's Law defines an abstract

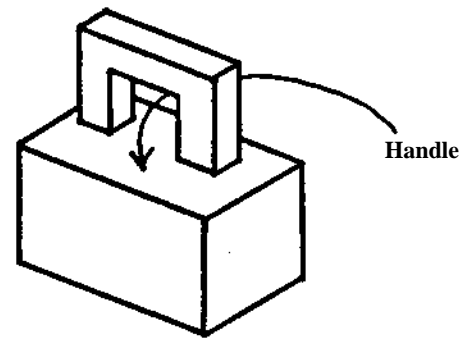


Figure 3: A handle requires a hole entirely through the shape.

combinatorial space of six dimensions that characterizes a universe of topological shapes of interest. The space is discrete, allowing only integer points. Euler's Law restricts the valid combinations to a subset of all those combinatorically possible. Desired is a set of operators that can cover the allowed set of points.

23. Well-Formedness

We have been concerned up to now with the range of shapes characterized by Euler's Law. It is equally important to specify the commonalities among all the shapes it allows. That is, what *are* the common properties of a shape that must necessarily hold for it to be accepted as an input shape for the operators. If these conditions are provided in all shapes returned by the operators, a necessary condition of *well-formedness* has been satisfied.

There are three classes of conditions that apply to all shapes of interest: that they be non-intersecting, closed and orientable (see Giblin, 1977, pp. 51-61X).

The non-intersecting condition requires that two bounded faces of a shape intersect in only two ways: they have one vertex in common or two vertices, and consequently an edge joining* them, in common. The effect of the intersection condition is that each edge is adjacent to exactly two faces (possibly the same one) and two vertices. By restricting face intersections, the topology of connections constrains the location of the faces and hence their geometry. We shall return to this issue later.

The condition that a shape be *closed* requires that every vertex on the surface can have an arc scribed about it and that the result will be exactly one closed loop or polygon. This condition overlaps the condition of being non-intersecting.

Both guarantee that no edge is left "bare", with only one face adjacent to it. However, the closed condition also guarantees that two sets of surfaces cannot join at a vertex or single edge.

The condition that a surface be *orientable* requires that if all edges are given a consistent ordering about a face, e.g. that they are ordered clockwise looking from outside, then each edge is traversed exactly once, in each direction. This condition, known as *Möbius' Law*, also eliminates certain pathological shapes *mnd* can identify cases of self-intersection. In any Euclidean shape not satisfying Möbius' Law, it has been proven that a self-intersection must occur (Giblin, 1977). In addition, meaningful shapes are thought in some application areas to only include those that are *bounded*. Thus the complement of a finite shape, e.g. an infinite domain with a bounded hole, is not allowed and only one orientation of a shape is acceptable.

2.4. Completeness

Given the set of shapes of interest, the criterion of completeness requires that the set of operators available can cover that set. However, several choices exist concerning the manner in which certain shape properties are represented. Several early systems did not explicitly depict holes in faces, but approximated them by connecting the inner loop to the outer one with a connecting edge (Baumgart, 1972s Eastman, Lividini and Stoker, 1975). See Figure 4a. The problem with this solution is the visibility of the connecting edge and the possibility of it creating extra vertices if cuts are later made across it. Explicit representation of holes eliminates both problems.

Shapes with handles can be approximated by having two faces of a shape geometrically coincident. See Figure 4b. This may lead, however, to misleading property evaluations of the shape, for instance surface area. Shapes made up of combinations of nested shells may not be required for some applications. Or for some uses they may be dealt with using combinations of shapes. See Figure 4c. Few of the existing bounded shape models implemented to date allow nested shells within a single shape (but see Birnbaum et al, 1978).

The above choices affect the range of Euler operators required to cover the set of shapes of interest and also influences the data structure used to model the shape. They also affect the specification of operators.

2.3. Proper Specification of Operators

A number of specification issues must be dealt with in implementing the Euler operators. Specification issues can be

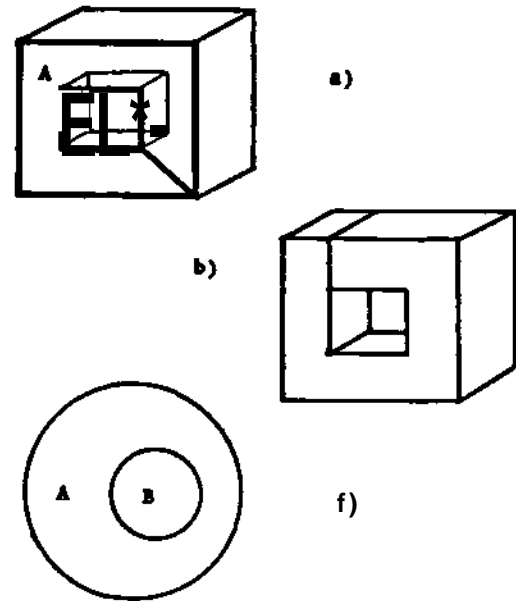


Figure 4: a)Face A has an inner loop, but may be approximated by connecting its two loops with an edge. b)This body will look almost as if it has a handle. c)Two shapes can duplicate most properties of a single hollow one; its volume here is $A-B$, its surface is $A+B$.

approached in several different ways and a mixture of these may be used in dealing with all the specification requirements of the system. While the Euler operators *&rt* primarily topological, they do impose constraints on the range of geometric specification a given shape topology can accept. Thus some specification issues are resolved by specifying the geometry with the topology, 9% each topological unit is specified. An alternative approach is to include topological specification information as arguments to the operator, to be provided when it is called. This means that the programmer must provide the *proper* specification of each operator, based on the range of geometries intended. It does, however, allow separation of the topological definition from the geometrical one. A third general approach to specification is to use local topological properties, where possible, to automatically determine the proper specification. This may result in more elaborate models and operators, but relieves the programmer of responsibility for integrity management while still separating topology from geometry. Yet another method available is to restrict the sequence in which operators may be applied to create shapes, so as to eliminate certain conditions requiring added specification.

One specification issue where several of the approaches may be applied is in specifying orientation. Orientation is required of the total shape, to identify which side of the bounding

domain is of interest. The global *property* may be evaluated automatically from a complete topological and geometrical specification. For example, one can select a point Known to be on one side of a surface and test if It Is in the bounded or unbounded domain by connecting it with a line to Infinity and testing how many times It intersects the boundary. An even number of intersections puts the point on the unbounded aide, an odd number on the bounded side. This test assumes that the shape satisfies Mobius' Law and is non-intersecting.

Alternatively, a normal vector or reference point can be entered for each face, with the condition that pairs of normal vectors adjacent to a common edge be oriented consistently. This solution relies on user provided arguments, in this case of geometrical information. As another alternative, the global test can be replaced by a number of local tests. Mobius* Law can be used directly in an implementation, if the orientation of each edge in relation to its two faces is maintained This solution relies solely on topological structure.

During construction of shapes, several combinatorial problems arise, requiring careful attention in defining the semantics of the operators. When the MEF (make edge face) operator connects two vertices within the same loop, problems can arise if either vertex is the root of a subgraph. See Figure 5. In this case, the edge loop passes through the vertex multiple times and the operation must determine wNch instance is to be connected. In other words, it must be determined whether the assignment of the subgraph is to either the new face being created or the existing old face. In the initial implementation of Baumgart's, the operators attempted to disallow construction of subgraphs with a single root by not allowing more than one edge to be constructed from any vertex interior to a face. This was thought to guarantee that any need for specification could be eliminated. However, KEF, lull edge face, was also provided, and the disallowed condition can be created by subtraction. In such cases, the behavior of the MEF operator would be undefined. One general solution is to locate the subgraphs by the geometry of their vertices, but this means that geometry must be defined with the topology. Alternatively, explicit definition of the site of connection on the loop can be given, or a listing of all edges on the connecting vertex that go on the new face.

A second problem is the assignment of holes Interior to a face when a face is split using MEF. Explicit assignment of loops to one of the faces may result in a variable length list of arguments. Again if geometry Is available, then resolution is clearcut. A third alternative is the specification of bookkeeping operators capable of moving holes In a face from one to another. In each of these cases, an implementation requires selection of the most appropriate specification method for the

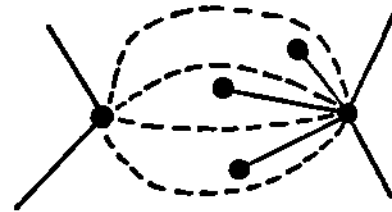


Figure 5: A vertex with multiple interior edges can be connected in many ways. Which one is desired?

range of shapes of interest.

In GLIDE, as described in detail below, holes in faces, handles in shapes *mnd* shapes made of nested shells are all represented explicitly and the operators properly manage them. In GLIDE, the Euler operators %rt strictly topological. By making them topological only, they may be used directly or, if desired, they may be combined into composite operators that include the geometric specification with them. They rely on topological properties wherever possible, eg. Mobius* Law, and use arguments to be provided by the user in cases where further specification is required. GLIDE currently provides a planar face geometry for constructing planar polyhedra, although the basic topology structure provided is also suitable for other kinds of surface definition.

3. A Specification for the Euler Operators

There are many variations on how the Euler operators can be implemented. The operators described below offer a specific example of how they can be implemented for a modeling system. Alternative sets of Euler operators may also be defined. See for instance Braid, et al 1978. They also provide an analysis of the use of each operator in covering the 6-space defined by the parameters of Euler's Law.

Five of the basic euler operators presented below, *MBFLV*, *MVE*, *ME*, *KE*, and *CLUE* are sufficient to create any topology, but others are included to add convenience and flexibility to the surface construction process. Figure 6 shows eight basic operators and their subcases as well as several others.

| Basic | Complement | Composite | Other |
|---------|------------|-----------|-------|
| MBFLV | KBFLEV | MME | LMOVE |
| MEV | KEV | ESPLIT | |
| ME: | KE: | KVE | |
| mekl | fceml | | |
| mefl | kefl | | |
| mekbfl | fcembfl | | |
| GLUE: | UNGLUE: | | |
| kflevmg | mflevkg | | |
| Mlevb | mflevb | | |

Figure 6: The Euler operators

As mentioned previously, the Euler operators create email changes in the numbers of the components in the topology. The operators can be characterized by these transitions in the 6-space defined by the parameters of the Euler Law, as shown below:

| E | E | Y | b | fi | fi | Operator |
|----|----|----|----|----|----|----------|
| 1 | 0 | 1 | 0 | 0 | 1 | MBFLV |
| 0 | 1 | 1 | 0 | 0 | 0 | MEV |
| 1 | 1 | 0 | 0 | 0 | 0 | MEFL |
| 0 | 1 | 1 | -1 | 0 | 0 | MEKL |
| -2 | -n | -n | 0 | 0 | -1 | KFLEVB |
| -2 | -n | -n | 0 | 1 | 0 | KFLEVMG |

Figure 7: Euler operator state transitions

The operators described here use parametric specification of the topological relationships for new edges that are created. The exact position of a new edge can be specified with a vertex, edge, and rotation direction (clockwise or counterclockwise), where the edge specified uses the vertex specified as an endpoint. The new edge will use the vertex specified as one of its endpoints and will lie either clockwise or counterclockwise from the specified edge about the specified vertex.

The operators are listed below in a procedure call format. First, the name of the operator is given along with all of its possible parameters. The parameters enclosed in parenthesis, "()", are output parameters; all others are input parameters. A few input parameters are sometimes optional as they are used for additional control over what the operator does or they are used to handle special situations. These are enclosed in brackets, "[]". Optional parameters in our implementation are set to zero when not used. In the paragraph below the operator call and parameter list, the function of the operator is described, along with the consequence of the operation in terms of the Euler equation and part addition and deletion. Complementary operators, if any, are also discussed with each operator.

Figure 8 describes the action of each operator graphically.

3.1. Basic Operators

MBFLV (newf,newl,newv)

"Make Body-Face-Loop-Vertex" starts a new surface in the topology, and is ultimately the first operator used in any topology construction. The single vertex created is used as a starting point from which a completely enclosing surface can be developed. The operation makes a new vertex (MHV), body, face (newf), and loop (newl). The complement operator, KBFLEV, will not only destroy the body, face, loop, and vertex as created by an MBFLV operation, but will destroy all component parts of any body owning the vertex specified.

MEV v,fe,dir,(newe,newv)

"Make Edge-Vertex" will create a new edge, newe, starting at v and running to newv. The new edge will be located such that it will be in direction dir from t about y. If this is the first edge to be constructed from a "lone" vertex (created, perhaps, by MBFLV) the e and dir w* optional and should be set to zero. If there is only one edge attached to the vertex v, the v parameter must be specified but either value for the dir parameter will have the same effect. MEV% complement operator, KEV, will take an edge and optional vertex specification and delete the edge end vertex, eliminating a strut or "squeezing" the endpoints together. K the edge is in the middle of an existing network of edges.

ME v1,fe1,dir1,v2,fe2,dir2,(newe,newf,newl)

"Make Edge" will generate an edge between two existing vertices. The new edge, newe, will be dir2 from v1 about v2 and will be dir1 from v2 about v1 with y1 and v2 as endpoints. One or both of these existing vertices may be unattached to any other edge due to previous delete operations; in this case the t and dir input fields of the corresponding vertex should be specified as zero. The complement Operator, KE, only requires an edge index as input with an optional vertex index for additional control. Three different cases can arise when connecting two vertices with a new edge; ME automatically detects which case and applies the appropriate procedure. The three possible cases are:

MEKL "Make Edge-Kill Loop" will occur when the new edge will link together two vertices of two different loops on the same face. The action will destroy the loop of the first vertex. KEs corresponding case KEML, "Kill Edge, Make Loop", applies when an edge has the same face on both sides.

MEFL "Make Edge-Face-Loop" occurs when two vertices of the same loop are linked together by the new edge. The new face will lie to the right hand side of the new edge running from v1 to v2 and the original face will lie to the left side of the new edge. Any other loops belonging to the original face remain on the original face. The loops may be moved to the new face, if desired, by applications of the LMOVE operator. KEs corresponding case is KEFL, "Kill Edge-Face-Loop", which eliminates an edge separating two faces; a face and a loop are also destroyed in this operation. Which face is deleted by KEFL can be optionally specified by indicating on which side of the edge the face lies.

MEKBFL "Make edge, Kill Face-Loop-Body" occurs when the two vertices newf located on separate bodies. The act of linking the two bodies together into one surface means that a body, face, and loop must be destroyed, v1 specifies the surviving face and body. KEs corresponding case, KEMBFL "Kill Edge, Make Body-Face-Loop" In some cases cannot be automatically distinguished from KEML (in the case of a "boss" or "pocket" in a surface), so an explicit indication of intent is necessary. When KEMBFL is invoked and a new body is created, a new face and loop must also

be created to "close up" the surface. *KEMBFL* can only be invoked on edges which separate the topology graph into two independent subgraphs.

GLUE f1,e1,f2,e2

"Glue Faces" will merge two faces of one body or two faces of different bodies. In the former case, a handle is created in the surface while in the latter case two surfaces are merged into one. The merge is performed such that *e1* of *f1* and *e2* of *f2* are merged into the same edge. The surviving set of edges are those of *f1*; in glues of two bodies the surviving body is that of *f1*. The two faces must have only one loop (no holes), the same number of vertices, and must have no edges in common. *UNGLUE*, the complement of *GLUE*, will take a "seam" or complete circuit of edges of a surface which has been marked using an edge marking facility and separates it, creating two new faces in its place. This may cause a handle in the surface to disappear or it may cause a surface to be split into two surfaces. The circuit marked for the *UNGLUE* operator must be well formed; it must be complete, have no struts, and must not cross itself. The two cases detected by *GLUE* are:

KFLEVMC

"Kill Faces-Loops-Edges-Vertices, Make Genus" occurs when the two faces belong to the same body. Both faces and corresponding loops are destroyed in the process, as are the edges and vertices of the second face as they are merged into the edges and vertices of the first face. The operation also results in the creation of a handle through the surface, and an increase of one in the genus of the body. *UNGLUE*'s corresponding case is *MFLEVKG*, "Make Faces-Loops-Edges-Vertices, Kill Genus", which occurs when the seam unglue operation does not result in two bodies, but removes a handle from the surface, reducing the genus by one.

KFLEVB

"Kill Faces-Loops-Edges-Vertices-Body" occurs when the two faces to be glued are from two different bodies. As above, both faces and loops are destroyed, as well as the edges and vertices of the second face as they are merged into the edges and vertices of the first face. The operation results, in addition, in the destruction of the body of the second face as both of the surfaces have been joined into one. *UNGLUE*'s corresponding case is *MFLEVB*, "Make Faces-Loops-Edges-Vertices-Body", occurs when the seam unglue operation creates two separate bodies.

3.2. Composite Operators

MME edgnum,[vstart,e,dir],[newv,ebeg,vend,eend]

"Make Multiple Edges" can create a chain of edges starting either at a given vertex (*vstart*) or starting with a new body if no vertex is specified. An appropriate number of edges and vertices are created; the number of edges created is always *edgnum*. The *startv*, *e*, *dir* triplet is used to position the first edge of the chain relative to the rest of the topology, and is only optional for starting vertices

which have no other edges attached to them (which also is the case when no vertex is specified and a new one is created). This routine is often used as a replacement for an optional use of *MBFLV* coupled with repeated applications of *MEV*.

ESPLIT e,[v],[newv,newv]

"Edge-Split" will split edge *e* into two edges, *e* and *newe*. A new vertex, *newv*, will be created between the vertices of the old edge. The optional *v* parameter is used to control which vertex of the old edge will be used on the new edge. The effect of this operator could also be accomplished with the application of a *KE* followed by *MEV* and *ME*.

KVE v

"Kill Vertex-Edge" will delete the vertex *v* specified and any edges which use this vertex. If necessary, faces and their corresponding loops are deleted. Ownership of "hole" loops of deleted faces always falls to the remaining surrounding face. The three cases handled by *KVE* are:

Vertex is only member of body

- the operation is equivalent to an application of *KBFLEV*.

Vertex is only member of a loop of a face with another loop

- the operation is equivalent to an application of *ME(KL)* followed by *KEV*.

Vertex is used by one or more edges

- if there are *n* edges using the vertex, then the operation is equivalent to *n-1* applications of *KE* followed by a single

3.3. Miscellaneous Operators

LMOVE l,f

"Loop Move" is not strictly an Euler operator, as it doesn't involve any changes in the parameters of the Euler equation. *LMOVE* moves the loop *l* from its current face to a different existing face *f*. This operation is particularly helpful when, after having invoked the *ME(FL)* operator, one wants to move some of the hole loops from the original face to the newly created face.

4. Interaction of Topology with Geometry

Close inspection of the construction sequence used in building shapes with the Euler operators will show that intermediate objects often have some unusual properties. While they all satisfy Euler's Law and the well-formedness conditions defined earlier, they may be:

- the topology of a shell made of one face and one loop made up of a single vertex, is what can be called the minimal topology.
- a topology made of two vertices connected by one edge. The resulting shape cannot have surface area or volume. We call this a wire topology, after Baumgart.
- a topology made of two faces, defined by one or more vertices and edges. Depending on the surfaces allowed, such a topology may or may not specify a shape enclosing volume. This is called a

lamina topology.

- a topology with an edge connected to the loop bounding a face, but with the same fact on both its sides. This we call a strut
- a topology with an interior loop made up of only one vertex or two vertices and an edge. We call this a minimal disjoint graph.

For some classes of desired shapes, the above cases could be considered ill-formed, if encountered in completed topologies. A loop made up of a vertex or edge may not be meaningful if encountered on topologies intended to be used for strictly planar surface shapes. However, these are meaningful if conical surfaces are supported. Consider the vertex at the apex of a cone, for example. Both lines and lamina may be considered ill-formed or useful shapes, depending on the application area. If some of these topologies are not to be allowed, tests can be readily constructed to detect them.

We have emphasized that the topology constrains the geometry of a shape. This was based on the assumption that topology was to be defined not later than geometry. One can conceive of methods that enter geometry first, then use it to constrain topology, eg. in the graphical entry of shapes. The more accurate statement is that the two parts of a shape specification are interdependent, in the following ways: intersections of surfaces must agree with the structure provided by the topology; intersections must combine in such a way that Mobius* Law is satisfied. In addition, a number of purely geometric conditions must be satisfied. These include consistency conditions regarding the geometry of adjacent parts, eg. the surface geometry, the line geometry and point geometry of adjacent parts must be consistent. Also, the geometry must be finite.

S. Examples Using the Euler Operators

Many high level and interactive functions can be built upon the Euler operators. The following program samples demonstrate how the Euler operators described can be used to construct surface topologies of polyhedral solids.

The examples are written in a Pascal language syntax where the Euler operators are available as procedures with the same formal parameter format described in the section on the Euler operators.

Every topology component (body, face, loop, edge, vertex) in the following examples has an associated *intUx*. This index is a *Mtne* and is used to refer to the component; the indices start at 1 for the first part of a given type and increase by 1 for every new part created. The indices of parts *m%* typed as a subrange of integers (in Pascal a typical index type declaration would be: *type vindex = 1-MAXVERTS*;) Some indices of parts

are determined by calculation in the following examples since the index assignment method is well defined. The rotation direction value is specified as either clockwise with the symbol CW or counterclockwise with the symbol CCW (this type definition in Pascal is: *type dir = (CW,CCW)*;).

6.1. Construction of a Tetrahedron

Below is a short program segment showing how a tetrahedron topology can be constructed. In line 1 of the example a connected sequence of three edges is created using the *MMB* operator (the indices of the parts are shown in figure 9). The effect of the *MME* operator in this case is the same as a call to *MBFLV* followed by three calls to *HEV*. Line 2 joins two of the vertices with a new edge along the base of the tetrahedron, making a complete circuit and a new face and loop. This is followed by two more calls to *ME* in lines 3 and 4 which also create edges and close off two additional faces. The completed object has a total of four faces, four loops, six edges, and four vertices.

```
{ example of building a tetrahedron topology }
{ numbers to left are for text references }

var vbeg,vends vindex*
    ebegin,eend*,nevei eindex!
    newfi flindex          lindex! lindex!
                                { sake series of 9 lines }
begin
1' Me(3,0,0,0,vbeg,ebeg,vend,eend);
    { close up base }
2  a>e((vbeg4l),(ebeg+1),CW,vend,eend,CCW,
    newe,newf,newl)i
    { close up side }
3  «e(vbeg,ebeg,CU,vend,newe,CCU,
    newe,newf,newl)t
    { close last side }
4  M(vbeg,neve,CMvend-l),eend,CCU,
    nowe,newf,newl)
end
```

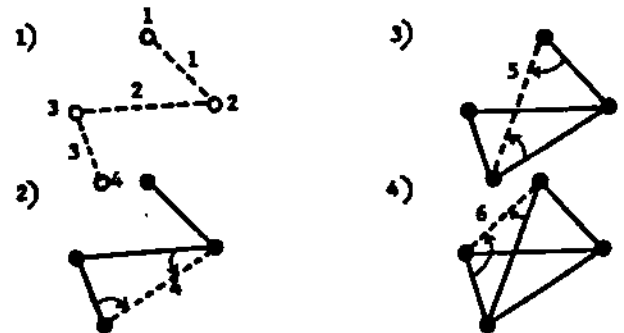


Figure 9: Constructing a tetrahedron

5.2. Extruded Topologies

A general procedure can be written which will generate a topology of an extruded or prismatic solid of any number of sides. The basic strategy is to first construct the top fact ring of edges of the solid. Then, starting from a vertex on that ring, one side edge and the edge ring of the bottom face is completed. Finally, the remaining side edges and faces are completed with a loop.

{ a general procedure to Make an extruded topology)

```

procedure extrude(m Integer)!
var vbegl,vbeg2,vlast,vindex*
    ebevl,ebeg2,elasts eindex*
    newft flindex*      newft lindex*      It Integer*
begin
    { top }
    »Me(n-l,0,0,0,vbegl,ebegl,vlast,elast)t
    { close top }
    are(vlast,elast,CCU,vbegl,ebegl,CU,
        neve,newf,newl)t
    { first side and bottoa }
    ame(n,vbegl,ebegl,CCM,vbeg2,ebeg2,
        vlast,elast,newf,nevl)i
    { close bottom }
    »e(vbeg2,ebeg2,CU,vlast,elast,CU,
        neve,newf,newl)
    { do sides }
    for l l* 1 to (n-1) do
        fte((vbeg2+l),(ebeg2+t),CCU,(vbegl+l),
            <ebegWi-l),CW,newe,newf,n«wl)
    end f { extrude procedure }

```

5.3. Pyramid Topology

This final example shows a procedure which will construct a pyramid topology with any number of sides. It follows the same strategy as the previous example except that only a single vertex is needed at the top of the topology structure. Consequently, one can start directly with the creation of a side edge and the ring of the bottom face.

{ » general procedure to Bake a pyramid topology)

```

procedure pyramid <n: Integer)!
var vbeg,vlastt vindex:
    e,ebeg,elastt eindex*
    newft flindex*      newli lindex*      It Integer*
begin
    { first side end bottoa }
    Me(n,O,O,O,vbeg,ebeg,vlast,elast)t
    e lB ebevt
    { rest of sides }
    for lts 2 to n do
        »e((vbeg+l),(ebeg+l-l),CCU,vbeg,e,CU,
            e,newf,newl)t
    { close bottoa }
    aie(vlast,e,CCU,<vbeg4l),ebeg,CU,newf,nevl)
    end t { tpyramid routine }

```

6. Higher Uvel Functions Using Euler Operators

Many higher level geometric modeling operators can and have been based on the use of the Euler operators, often adding geometric capabilities as well as the topological ones Inherent in the Euler operators.

There is a large variety of *mv-p* operators which can be used to sweep a line into a face, a face into a solid, etc Two of the more interesting of the sweep operators are rotational sweep operators which sweep a series of connected line segments or alternatively, a lamina around an axis (see figure 10). *RifUction* operators ere often useful when constructing Objects having symmetry. While the *spatial #et* operators (subtracting, adding, and intersecting solids with each other) have not been entirely based on the Euler operators to date, our own implementation (Birnbbaum, at al, 1978) heavily relies on their functions for a major portion of the manipulations involved.

As a specific example of the ease of using the Euler operators as a basis for higher level functions, we show below how the rotational sweep operators can be implemented entirely in terms of the operators and functions previously described in this paper.

6.1. Connected Line Segment Rotational Sweep Operator

The connected line segment rotational sweep operator takes *m* connected line segments (edges) which begin and end at an axis, and rotates them around the axis incrementally for a total of *n* sides.

It is often convenient in the case of sweep operators to develop the geometry simultaneously with the topology. The algorithm below omits such geometric calculations for sake of brevity. The algorithm also assumes the availability of the *CLUE*, *MEV*, *ME*, and *KE* operators *m>d* the extrude and pyramic procedures previously described, with the exception that for brevity the latter additionally return the indices of the base faces (as opposed to side faces) created

The basic approach of the algorithm presented below is to sweep each line segment in turn around the axis into an extruded *n* sided prism (or pyramid if it is the first or last segment); as each of the solids is created it is glued onto the developing shape with the *GLUE* operator. If one of the line segments is "vertical" or normal to the axis of rotation, it is not necessary to sweep a solid with it; a new face installed on the end of the adjacent extruded shape is sufficient.

Algorithm for Rotational Sweep of Connected Line Segments

Step 0. (Initialization)

$m \leftarrow$ number of connected edges
 $n \leftarrow$ number of sides of final rotated object
 $lastf \leftarrow 0$

Step 1. (First End of Object)

if edge₁ is not normal to axis of rotation
 then call pyramid(n , baseface) and $lastf \leftarrow$ baseface

Step 2. (Loop on Middle Segments)

for i from 2 to $m-1$ do
 if edge _{i} is not normal to axis of rotation
 then call extrude(n , baseface₁, baseface₂)
 if $lastf \neq 0$
 then if edge _{i} is normal to the axis
 then if its first vertex is above its
 last then make a hole in $lastf$
 and GLUE(new face, baseface)
 otherwise make a hole in baseface₁
 and GLUE($lastf$, new face)
 else GLUE($lastf$, baseface₂)
 $lastf \leftarrow$ baseface₂

Step 3. (End of Object)

if edge _{m} is not normal to axis of rotation
 then call pyramid(n , baseface)
 if $lastf \neq 0$
 then if edge _{m} is normal to the axis
 then if its first vertex is above its
 last then make a hole in $lastf$
 and GLUE(new face, baseface)
 otherwise make a hole in baseface₁
 and GLUE($lastf$, new face)
 else GLUE($lastf$, baseface)

The algorithm uses the MEV, ME, and KE operators to create holes (and corresponding new faces) inside of base faces of the extrusions when vertical line segments are encountered in the middle of the shape being created.

2. Lamina Rotational Sweep Operator

The lamina sweep solid of rotation operator requires a very simple algorithm; it calls the extrude procedure ft times and glues the results together with n CLUE operations.

6.3. A Rotational Sweep Operator Using Graphic Input

An implementation of a solid of rotation operator based on the Euler operators and graphical input is shown in figure 11. A connected series of edges can be drawn on the X-Y grid shown and the function will rotate it about the X axis for a specific number of sides. Output from the operator is shown in figure 12 (backward facing planes have been removed by the display system).

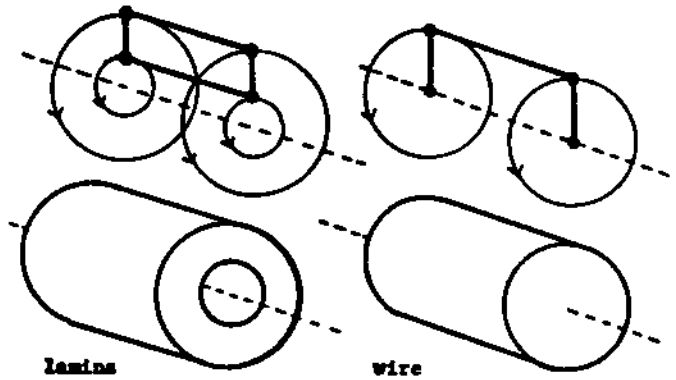


Figure 10: The rotational sweep operators

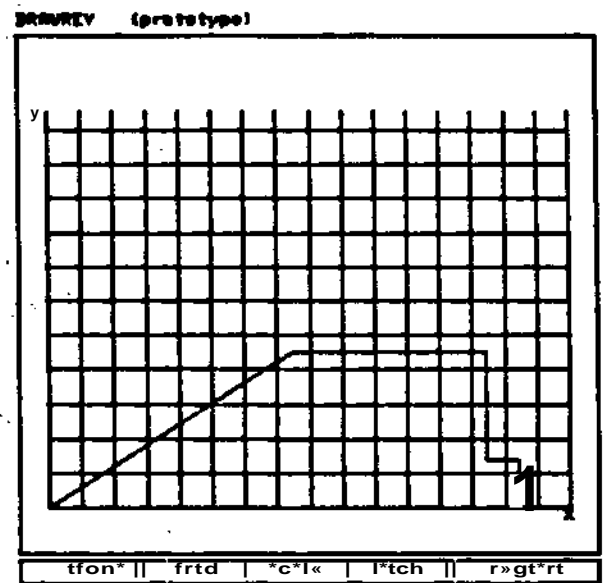


Figure 11: Graphic input screen during rotational sweep operation

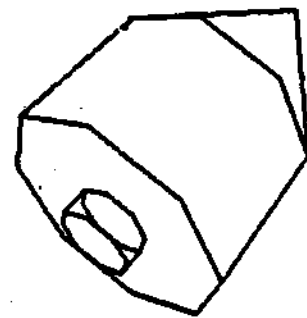


Figure 12: Output from rotational sweep operators

7. Shape Data Structure

The data structure needed to implement the Euler operators varies according to the range of shapes to be explicitly allowed and the expected applications to which the data is to be applied.

The data structure used in the GLIDE implementation is a variation of the "winged edge" structure developed by Baumgart. It is a composition of six different data structures, each instance being dynamically allocated as needed during construction of a Topology. The root of the data structure is a body record, storing header type information. Because bodies may be nested, a shape is composed of a tree of bodies. As in most tree structures, each body has a sibling and child pointer to other bodies. Each body also has pointers to rings of its constituent faces, loops, edges and vertices. Each face record holds a pointer to a ring of its loops and a pointer back to the body to which it belongs. The loop records have pointers to the ring of edges that make them and the face to which they belong. The loop records also have a pointer to a vertex which is only used when a loop consists of a single vertex. Each edge record points to the two loops to which it belongs, to its two bounding vertices, and to the clockwise and counterclockwise adjacent edges of its two loops. The vertex records have no backpointers. Each record type has associated attribute fields, for geometric, display or other uses. Notice that each face, loop, edge and vertex record has fields for the body ring pointers. In addition, the loop records have fields for the face ring. The clockwise and counterclockwise pointers serve a similar purpose at the edge level.

8. Conclusion

The Euler operators provide a powerful tool for application developers that need to provide a variety of programs for specifying shape. Their benefit is in the bookkeeping that they internally provide regarding the well-formedness conditions. They allow application developers to define new methods of shape specification without attention to data structures and with only modest attention to well-formedness issues. With proper use, the Euler operators provide a strong base for a wide spectrum of geometric modeling tools ranging from local operators to high level functions.

Note:

While this paper was being prepared, the authors received the paper by Braid, Hillyard and Stroud (1978). It describes other aspects of the Euler operators and is complementary to this one. It is highly recommended by anyone interested in the Euler operators.

9. References

- Beer, A. C. Eastman and M. Henrion, "A survey of geometric modeling", Institute of Physical Planning Report No. 66 (revised), Carnegie-Mellon University, Pittsburgh, April, 1978.
- Baumgart, B.G. "Winged edge polyhedron representation", Stanford Artificial Intelligence Report No. CS-320, October, 1972.
- Birnbaum, M et al, GLIDE REFERENCE MANUAL, Institute of Physical Planning, Carnegie-Mellon University, October, 1978.
- Braid, L R. Hillyard and L Stroud, "Stepwise construction of polyhedra in geometric modelling", O.A.D. Group Document No. 100, University of Cambridge Computer Laboratory, October, 1978.
- Braid, L, "New directions in geometric modeling", PROCEEDINGS OF THE WORKSHOP OF GEOMETRIC MODELING, CAM-1 Inc. Arlington, Texas, March, 1978.
- Braid, L, "The synthesis of solids bounded by many faces", CACM, 18:4, pp. 209-216, April 1975.
- Braid, L, DESIGNING WITH VOLUMES, Computer-aided Design Group, University of Cambridge Computer Laboratory, England, 1973.
- Eastman, C and M. Henrion, "GLIDE: Language for design information systems", PROCEEDINGS ACM SIGGRAPH CONFERENCE, 1977, San Jose, CA, ACM, New York, 1977.
- Eastman, C, "The concise structuring of geometric data for CAD", in DATA STRUCTURES, COMPUTER GRAPHICS AND PATTERN RECOGNITION, A. Klinger, K.S. Fu, and T. Kunii (eds) Academic Press, New York, 1977.
- Eastman, C J. Lividini and D. Stoker, "A database for designing large physical systems", PROC. 1975 NATIONAL COMP. CONF., AFIPS Press, New Jersey, pp. 603-611, 1975.
- Giblin, P.O, GRAPHS, SURFACES AND HOMOLOGY, Chapman and Hall, London, 1977.
- Lakatos, Imre, PROOFS AND REFUTATIONS, Cambridge University Press, Cambridge, 1976.
- Requicha, A.A.G. and K. Voelker, "Constructive solid geometry", Production Automation Project, Report TM-25, University of Rochester, NY, November, 1977.