Chapter 4

MODEL DECOMPOSITION

The term **model decomposition** refers to the operation when the database describing an object scene is processed in order to produce simple geometric entities which are suitable for image synthesis. The question of which sorts of geometric entity are suitable for picture generation can be answered only if one is aware of the nature of the image synthesis algorithm to be used. Usually these algorithms cannot operate directly with the world representation. The only important exception is the ray tracing method (see chapter 9 and section 6.1 in chapter 6) which works with practically all types of representation scheme. Other types of image synthesis algorithm, however, require special types of geometric entity as their input. These geometric entities are very simple and are called **graphics primitives**. Thus model decomposition produces low level graphics primitives from a higher level representation scheme. Usually these primitives are polygons or simply triangles. Since many algorithms require triangles only as their input, and polygons can be handled similarly, this chapter will examine that case of model decomposition in which the graphics primitives are triangles. The problem is the following: a solid object given by a representation scheme, approximate its boundary by a set of triangles.

The most straightforward approach to this task is to generate a number of surface points so that they can be taken as triangle vertices. Each triangle then becomes a linear interpolation of the surface between the three vertices.

The resulting set of triangles is a valid mesh if for each triangle:

- each of its vertices is one of the generated surface points
- each of its edges is shared by exactly one other (neighboring) triangle except for those that correspond to the boundary curve of the surface
- there is no other triangle which intersects it, except for neighboring triangles sharing common edges or vertices

Some image synthesis algorithms also require their input to contain topological information (references from the triangles to their neighbors); some do not, depending on the nature of the algorithm. It is generally true, however, that a consistent and redundancy-free mesh structure that stores each geometric entity once only (triangle vertices, for example, are not stored as many times as there are triangles that contain them) is usually much less cumbersome than a stack of triangles stored individually. For the sake of simplicity, however, we will concentrate here only on generating the triangles and omit topological relationships between them.

4.1 Simple geometric objects

A geometric object is usually considered to be *simple* if it can be described by one main formula characterizing its shape and (possibly) some additional formulae characterizing its actual boundary. In other words, a simple geometric object has a *uniform shape*. A sphere with center $c \in E^3$ and of radius r is a good example, because its points p satisfy the formula:

$$|p - c| \leq r \tag{4.1}$$

where $|\cdot|$ denotes vector length.

Simple objects are also called *geometric primitives*. The task is to approximate the surface of a primitive by a triangular mesh, that is, a number of surface points must be generated and then proper triangles must be formed. In order to produce surface points, the formula describing the surface must have a special form called *explicit form*, as will soon become apparent.

4.1.1 Explicit surface patches

The formula describing a surface is in (biparametric) *explicit form* if it characterizes the coordinates (x, y, z) of the surface points in the following way:

$$\begin{aligned} x &= f_x(u,v), \\ y &= f_y(u,v), \\ z &= f_z(u,v), \qquad (u,v) \in D \end{aligned} \tag{4.2}$$

where D is the 2D parameter domain (it is usually the rectangular box defined by the inequalities $0 \le u \le u_{\max}$ and $0 \le v \le v_{\max}$ for the most commonly used 4-sided patches). The formula "generates" a surface point at each parameter value (u, v), and the continuity of the functions f_x, f_y, f_z ensures that each surface point is generated at some parameter value (the formulae used in solid modeling are analytic or more often algebraic which implies continuity; see subsection 1.6.1). This is exactly what is required in model decomposition: the surface points can be generated (sampled) to any desired resolution.

In order to generate a valid triangular mesh, the 2D parameter domain, D, must be sampled and then proper triangles must be formed from the sample points. We distinguish between the following types of faces (patches) with respect to the shape of D.

Quadrilateral surface patches

The most commonly used form of the parameter domain D is a rectangular box in the parameter plane, defined by the following inequalities:

$$0 \le u \le u_{\max}, \qquad 0 \le v \le v_{\max}. \tag{4.3}$$

The resulting patch is 4-sided in this case, and the four boundary curves correspond to the boundary edges of the parameter rectangle ($\cdot$ stands for any value of the domain): $(0, \cdot)$, $(u_{\max}, \cdot)$, $(\cdot, 0)$, $(\cdot, v_{\max})$. The curves defined by parameter ranges $(u, \cdot)$ or $(\cdot, v)$, that is, where one of the parameters is fixed, are called *isoparametric* curves. Let us consider the two sets of isoparametric curves defined by the following parameter ranges (the subdivision is not necessarily uniform):

$$\begin{array}{lll} (0,\cdot),(u_1,\cdot), & \ldots, & (u_{n-1},\cdot),(u_{\max},\cdot), \\ (\cdot,0),(\cdot,v_1), & \ldots, & (\cdot,v_{m-1}),(\cdot,v_{\max}). \end{array} \tag{4.4}$$

The two sets of curves form a quadrilateral mesh on the surface. The vertices of each quadrilateral correspond to parameter values of the form (u_i, v_j), (u_{i+1}, v_j), (u_{i+1}, v_{j+1}), (u_i, v_{j+1}). Each quadrilateral can easily be cut into two triangles and thus the surface patch can be approximated by $2nm$ number of triangles using the following simple algorithm (note that it realizes a uniform subdivision):

```
DecomposeQuad($\vec{f}$, $n$, $m$)                          // $\vec{f} = (f_x, f_y, f_z)$
    $S = \{\}$; $u_i = 0$;                              // $S$: resulting set of triangles
    for $i = 1$ to $n$ do
        $u_{i+1} = u_{\max} \cdot i/n$; $v_j = 0$;
        for $j = 1$ to $m$ do
            $v_{j+1} = v_{\max} \cdot j/m$;
            add the triangle $\vec{f}(u_i, v_j)$, $\vec{f}(u_{i+1}, v_j)$, $\vec{f}(u_{i+1}, v_{j+1})$ to $S$;
            add the triangle $\vec{f}(u_i, v_j)$, $\vec{f}(u_{i+1}, v_{j+1})$, $\vec{f}(u_i, v_{j+1})$ to $S$;
            $v_j = v_{j+1}$;
        endfor
        $u_i = u_{i+1}$;
    endfor
    return $S$;
end
```

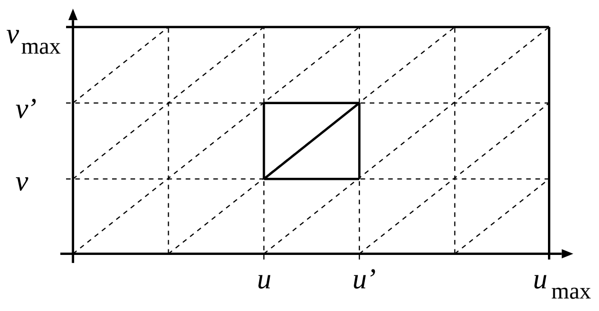

Figure 4.1: Subdivision of a rectangular parameter domain

Note that the quadrilateral (triangular) subdivision of the patch corresponds to a quadrilateral (triangular) subdivision of the parameter domain

D, as illustrated in figure 4.1. This is not surprising, since the mapping $f(u,v)$ is a continuous and one-to-one mapping, and as such, preserves topological invariances, for example neighborhood relationships.

Triangular surface patches

Triangular — and more generally; non-quadrilateral — surface patches were introduced into geometric modeling because the fixed topology of surfaces based on 4-sided patches restricted the designer's freedom in many cases (non-quadrilateral patches are typically necessary for modeling rounded corners where three or more other patches meet and must be blended). The parameter domain D is usually triangle-shaped. The Steiner patch [SA87], for example, is defined over the following parameter domain:

$$u \geq 0, \quad v \geq 0, \quad u + v \leq 1 \tag{4.5}$$

It often occurs, however, that the triangular patch is parameterized via three parameters, that is having the form $f(u,v,w)$, but then the three parameters are not mutually independent. The Bezier triangle is an example of this (see any textbook on surfaces in computer aided geometric design, such as [Yam88]). Its parameter domain is defined as:

$$u \geq 0, \quad v \geq 0, \quad w \geq 0, \quad u + v + w = 1 \tag{4.6}$$

It is also a triangle, but defined in a 3D coordinate system. In order to discuss the above two types of parameter domain in a unified way, the parameter will be handled as a vector $\vec{u}$ which is either a 2D or a 3D vector, that is a point of a 2D or 3D parameter space U. The parameter domain $D \subset U$ is then defined as a triangle spanned by the three vertices $\vec{u}_1, \vec{u}_2, \vec{u}_3 \in U$.

The task is to subdivide the triangular domain D into smaller triangles. Of all the imaginable variations on this theme, the neatest is perhaps the following, which is based on recursive subdivision of the triangle into similar smaller ones using the middle points of the triangle sides. As illustrated in figure 4.2, the three middle points, $\vec{m}_1, \vec{m}_2, \vec{m}_3$, are generated first:

$$\vec{m}_1 = \frac{1}{2}(\vec{u}_2 + \vec{u}_3), \quad \vec{m}_2 = \frac{1}{2}(\vec{u}_3 + \vec{u}_1), \quad \vec{m}_3 = \frac{1}{2}(\vec{u}_1 + \vec{u}_2). \tag{4.7}$$

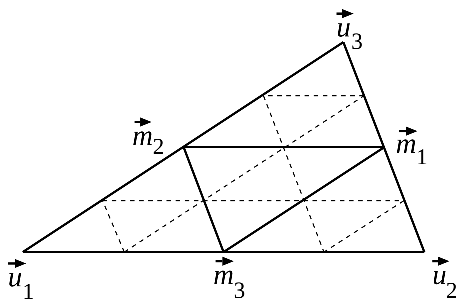

Figure 4.2: Subdivision of a triangular parameter domain

The resulting four smaller triangles are then further subdivided in a similar way. The subdivision continues until a predefined "depth of recurrence", say d, is reached. The corresponding recursive algorithm is the following:

```
DecomposeTriang($\vec{f}$, $\vec{u}_1$, $\vec{u}_2$, $\vec{u}_3$, $d$)                // $\vec{f} = (f_x, f_y, f_z)$
    if $d \leq 0$ then return the triangle of vertices $\vec{f}(\vec{u}_1)$, $\vec{f}(\vec{u}_2)$, $\vec{f}(\vec{u}_3)$;
    $S = \{\}$;
    $\vec{m}_1 = \frac{1}{2}(\vec{u}_2 + \vec{u}_3)$; $\vec{m}_2 = \frac{1}{2}(\vec{u}_3 + \vec{u}_1)$; $\vec{m}_3 = \frac{1}{2}(\vec{u}_1 + \vec{u}_2)$;
    add DecomposeTriang($\vec{f}$, $\vec{u}_1$, $\vec{m}_3$, $\vec{m}_2$, $d-1$) to $S$;
    add DecomposeTriang($\vec{f}$, $\vec{u}_2$, $\vec{m}_1$, $\vec{m}_3$, $d-1$) to $S$;
    add DecomposeTriang($\vec{f}$, $\vec{u}_3$, $\vec{m}_2$, $\vec{m}_1$, $d-1$) to $S$;
    add DecomposeTriang($\vec{f}$, $\vec{m}_1$, $\vec{m}_2$, $\vec{m}_3$, $d-1$) to $S$;
    return $S$;
end
```

General n-sided surface patches

Surface patches suitable for interpolating curve networks with general (irregular) topology are one of the most recent achievements in geometric modeling (see [Vár87] or [HRV92] for a survey). The parameter domain corresponding to an n-sided patch is usually an n-sided convex polygon (or even a regular n-sided polygon with sides of unit length as in the case of the so-called overlap patches [Vár91]). A convex polygon can easily be broken down into triangles, as will be shown in subsection 4.2.1, and then the triangles can be further divided into smaller ones.

4.1.2 Implicit surface patches

The formula describing a surface is said to be in *implicit form* if it characterizes the coordinates (x, y, z) of the surface points in the following way:

$$f(x, y, z) = 0. \tag{4.8}$$

This form is especially suitable for tests that decide whether a given point is on the surface: the coordinates of the point are simply substituted and the value of f gives the result. Model decomposition, however, yields something of a contrary problem: points which are on the surface must be generated. The implicit equation does not give any help in this, it allows us only to check whether a given point does in fact lie on the surface. As we have seen in the previous subsection, explicit forms are much more suitable for model decomposition than implicit forms. We can conclude without doubt that the implicit form in itself is not suitable for model decomposition.

Two ways of avoiding the problems arising from the implicit form seem to exist. These are the following:

1. *Avoiding model decomposition.* It has been mentioned that ray tracing is an image synthesis method that can operate directly on the world representation. The only operation that ray tracing performs on the geometric database is the calculation of the intersection point between a light ray (directed semi-line) and the surface of an object. In addition, this calculation is easier to perform if the surface formula is given in implicit form (see subsection 6.1.2 about intersection with implicit surfaces).

2. *Explicitization.* One can try to find an explicit form which is equivalent to the given implicit form, that is, which characterizes the same surface. No general method is known, however, for solving the explicitization problem. The desired formulae can be obtained heuristically. Explicit formulae for simple surfaces, such as sphere or cylinder surfaces, can easily be constructed (examples can be found in subsection

12.1.2, where the problem is examined within the context of texture mapping).

The conclusion is that implicit surfaces are generally not suited to being broken down into triangular meshes, except for simple types, but this problem can be avoided by selecting an image synthesis algorithm (ray tracing) which does not require preliminary model decomposition.

4.2 Compound objects

Compound objects are created via special operations performed on simpler objects. The simpler objects themselves can also be compound objects, but the bottom level of this hierarchy always contains geometric primitives only. The operations by which compound objects can be created usually belong to one of the following two types:

1. *Regularized Boolean set operations.* We met these in subsection 1.6.1 on the general aspects of geometric modeling. Set operations are typically used in CSG representation schemes.

2. *Euler operators.* These are special operations that modify the boundary of a solid so that its combinatorial (topological) validity is left unchanged. The name relates to Euler's famous formula which states that the alternating sum of the number of vertices, edges and faces of a simply connected polyhedron is always two. This formula was then extended to more general polyhedra by geometric modelers. The Euler operators — which can create or remove vertices, edges and faces — are defined in such a way that performing them does not violate the formula [Män88], [FvDFH90]. Euler operators are typically used in B-rep schemes.

Although often not just one of the two most important representation schemes, CSG and B-rep, is used exclusively, that is, practical modeling systems use instead a hybrid representation, it is worth discussing the two schemes separately here.

4.2.1 Decomposing B-rep schemes

Breaking down a B-rep scheme into a triangular mesh is relatively simple. The faces of the objects are well described in B-rep, that is, not only their shapes but also their boundary edges and vertices are usually explicitly represented.

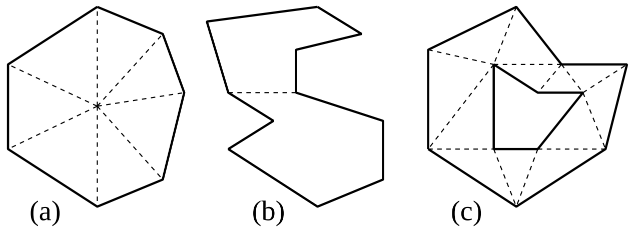

Figure 4.3: Polygon decompositions

If the object is a planar polyhedron, that is if it contains planar faces only, then each face can be individually retrieved and triangulated. Once the polygon has been broken down into triangles, generating a finer subdivision poses no real problem, since each triangle can be divided separately by the algorithm for triangular patches given in subsection 4.1.1. However the crucial question is: how to decompose a polygon — which is generally either convex or concave and may contain holes (that is multiply connected) — into triangles that perfectly cover its area and only its area. This *polygon triangulation problem*, like many others arising in computer graphics, has been studied in computational geometry. Without going into detail, let us distinguish between the following three cases (n denotes the number of vertices of the polygon):

1. *Convex polygons.* A convex polygon can easily be triangulated, as illustrated in part (a) of figure 4.3. First an inner point is calculated — for example the center of mass — and then each side of the polygon makes a triangle with this inner point. The time complexity of this operation is $O(n)$.

2. *Concave polygons without holes.* Such polygons cannot be triangulated in the previous way. A problem solving approach called **divide-and-conquer** can be utilized here in the following way. First two vertices of

the polygon must be found so that the straight line segment connecting them cuts the polygon into two parts (see part (b) of figure 4.3). This diagonal is called a *separator*. Each of the two resulting polygons is either a triangle, in which case it need not be divided further, or else has more than three vertices, so it can be divided further in a similar way. If it can be ensured that the two resulting polygons are of the same size (up to a ratio of two) with respect to the number of their vertices at each subdivision step, then this *balanced* recurrence results in a very good, $O(n \log n)$, time complexity. (Consult [Cha82] to see that the above property of the separator can always be ensured in not more than $O(n)$ time.)

3. *General polygons.* Polygons of this type may contain holes. A general method of triangulating a polygon with holes is to generate a *constrained triangulation* of its vertices, as illustrated in part (c) of figure 4.3. A triangulation of a set of points is an aggregate of triangles, where the vertices of the triangles are from the point set, no triangles overlap and they completely cover the convex hull of the point set. A triangulation is constrained if there are some predefined edges (point pairs) that must be triangle edges in the triangulation. Now the point set is the set of vertices and the constrained edges are the polygon edges. Having computed the triangulation, only those triangles which are inside the face need be retained. (Seidel [Sei88] shows, for example, how such a triangulation can be computed in $O(n \log n)$ time.)

Finally, if the object has curved faces, then their shape is usually described by (or their representation can be transformed to) explicit formulae. Since the faces of a compound object are the result of operations on primitive face elements (patches), and since usually their boundaries are curves resulting from intersections between surfaces, it cannot be assumed that the parameter domain corresponding to the face is anything as simple as a square or a triangle. It is generally a territory with a curved boundary, which can, however, be approximated by a polygon to within some desired tolerance. Having triangulated the original face the triangular faces can then be decomposed further until the approximation is sufficiently close to the original face.

4.2.2 Boundary evaluation for CSG schemes

As described in section 1.6.2 CSG schemes do not explicitly contain the faces of the objects, shapes are produced by combining half-spaces or primitives defining point sets in space. The boundary of the solid is *unevaluated* in such a representation. The operation that produces the faces of a solid represented by a CSG-tree is called **boundary evaluation**.

Set membership classification: the unified approach

Tilove has pointed out [Til80] that a paradigm called **set membership classification** can be a unified approach to geometric intersection problems arising in constructive solid geometry and related fields such as computer graphics. The classification of a *candidate set* X with respect to a *reference set* S maps X into the following three disjoint sets:

$$\begin{aligned} C_{\text{in}}(X,S) &= X \cap iS, \\ C_{\text{out}}(X,S) &= X \cap cS, \\ C_{\text{on}}(X,S) &= X \cap bS, \end{aligned} \tag{4.9}$$

where iS, cS, bS are the interior, complement and boundary of S, respectively. Note that if X is the union of boundaries of the primitive objects in the CSG-tree, and S is the solid represented by the tree, then boundary evaluation is no else but the computation of $C_{\text{on}}(X,S)$. The exact computation of this set, however, will not be demonstrated here. An approximation method will be shown instead, which blindly generates all the patches that *may* fall onto the boundary of S and then tests each one whether to keep it or not. For this reason, the following binary *relations* can be defined between a candidate set X and a reference set S:

$$\begin{aligned} X \text{ in } S &\quad \text{if} \quad X \subseteq iS, \\ X \text{ out } S &\quad \text{if} \quad X \subseteq cS, \\ X \text{ on } S &\quad \text{if} \quad X \subseteq bS \end{aligned} \tag{4.10}$$

(note that either one or none of these can be true at a time for a pair X, S).

In constructive solid geometry, S is either a primitive object or is of the form $S = A \circ^* B$, where $\circ^*$ is one of the operations $\cup^*, \cap^*, \setminus^*$. A **divide-and-conquer** approach can now help us to simplify the problem.

The following relations are straightforward results:

$$\begin{array}{rll}
X \text{ in } (A \cup B) & \text{if} & X \text{ in } A \ \vee \ X \text{ in } B \\
X \text{ out } (A \cup B) & \text{if} & X \text{ out } A \ \wedge \ X \text{ out } B \\
X \text{ on } (A \cup B) & \text{if} & (X \text{ on } A \ \wedge \ \neg\, X \text{ in } B) \ \vee \ (X \text{ on } B \ \wedge \ \neg\, X \text{ in } A) \\
\\
X \text{ in } (A \cap B) & \text{if} & X \text{ in } A \ \wedge \ X \text{ in } B \\
X \text{ out } (A \cap B) & \text{if} & X \text{ out } A \ \vee \ X \text{ out } B \\
X \text{ on } (A \cap B) & \text{if} & (X \text{ on } A \ \wedge \ \neg\, X \text{ in } cB) \ \vee \ (X \text{ on } B \ \wedge \ \neg\, X \text{ in } cA) \\
\\
X \text{ in } (A \setminus B) & \text{if} & X \text{ in } A \ \wedge \ X \text{ in } cB \\
X \text{ out } (A \setminus B) & \text{if} & X \text{ out } A \ \vee \ X \text{ out } cB \\
X \text{ on } (A \setminus B) & \text{if} & (X \text{ on } A \ \wedge \ \neg\, X \text{ in } B).
\end{array} \tag{4.11}$$

That is, the classification with respect to a compound object of the form $S = A \circ B$ can be traced back to simple logical combinations of classification results with respect to the argument objects A, B.

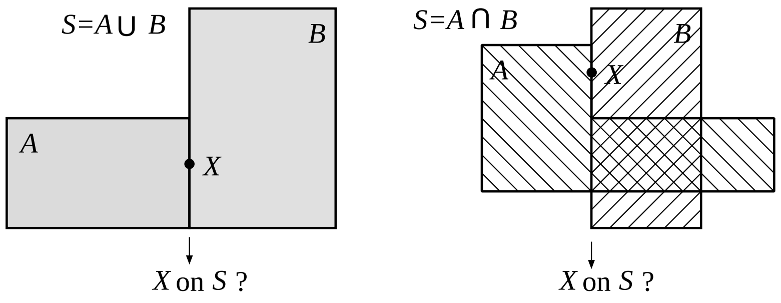

Figure 4.4: Some problematic situations in set membership classification

There are two problems, however:

1. The events on the right hand side are not equivalent with the events on the left hand side. The event X in $(A \cup B)$ can happen if X in A or X in B or (this is not contained by the expression) X_1 in A and X_2 in B where $X_1 \cup X_2 = X$. This latter event, however, is much more difficult to detect than the previous two.

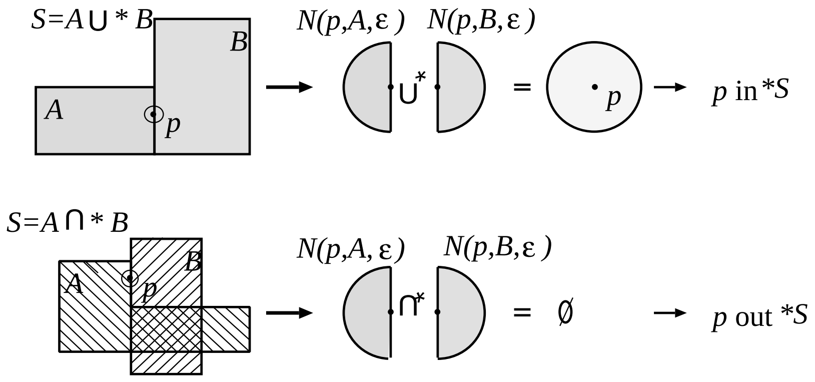

Figure 4.5: Regularizing set membership classifications

2. There are problematic cases when the above expressions are extended to regular sets and regularized set operations. Figure 4.4 illustrates two such problematic situations (the candidate set X is a single point in both cases).

Problem 1 can be overridden by a nice combination of a generate-and-test and a divide-and-conquer strategy, as will soon be shown in the subsequent sections.

The perfect *theoretical* solution to problem 2 is that each point $p \in X$ of the candidate set is examined by considering a (sufficiently small) neighborhood of p. Let us first consider the idea without worrying about implementation difficulties. Let $B(p,\varepsilon)$ denote a ball around p with a (small) radius ε, and let $N(p,S,\varepsilon)$ be defined as the ε-neighborhood of p in S (see figure 4.5):

$$N(p,S,\varepsilon) = B(p,\varepsilon) \cap^* S. \tag{4.12}$$

Then the regularized set membership relations are the following:

$$\begin{aligned} p\,\mathrm{in}^*(A \circ^* B) &\quad \text{if} \quad \exists\varepsilon > 0\colon\ N(p,A,\varepsilon) \circ^* N(p,B,\varepsilon) = B(p,\varepsilon), \\ p\,\mathrm{out}^*(A \circ^* B) &\quad \text{if} \quad \exists\varepsilon > 0\colon\ N(p,A,\varepsilon) \circ^* N(p,B,\varepsilon) = \emptyset, \\ p\,\mathrm{on}^*(A \circ^* B) &\quad \text{if} \quad \forall\varepsilon > 0\colon\ \emptyset \neq N(p,A,\varepsilon) \circ^* N(p,B,\varepsilon) \neq B(p,\varepsilon). \end{aligned} \tag{4.13}$$

Figure 4.5 shows some typical situations. One might suspect disappointing computational difficulties in actually performing the above tests:

1. It is impossible to examine each point of a point set since their number is generally infinite. Intersection between point sets can be well computed by first checking whether one contains the other and if not, then intersecting their boundaries. If the point sets are polyhedra (as in the method to be introduced in the next part), then intersecting their boundaries requires simple computations (face/face, edge/face). Ensuring regularity implies careful handling of degenerate cases.

2. If a single point is to be classified then the ball of radius ε can, however, be substituted by a simple line segment of length 2ε and then the same operations performed on that as were performed on the ball in the above formulae. One must ensure then that ε is small enough, and that the line segment has a "general" orientation, that is, if it intersects the boundary of an object then the angle between the segment and tangent plane at the intersection point must be large enough to avoid problems arising from numerical inaccuracy of floating point calculations.

The conclusion is that the practical implementation of regularization does not mean the perfect imitation of the theoretical solution, but rather that simplified solutions are used and degeneracies are handled by keeping the theory in mind.

Generate-and-test

Beacon *et al.* [BDH+89] proposed the following algorithm which *approximates* the boundary of a CSG solid, that is, generates surface patches the aggregate of which makes the boundary of the solid "almost perfectly" within a predefined tolerance. The basic idea is that the union of the boundaries of the primitive objects is a superset of the boundary of the compound solid, since each boundary point lies on some primitive. The approach based on this idea is a **generate-and-test** strategy:

1. The boundary of each primitive is roughly subdivided into patches in a preliminary phase and put onto a list L called the *candidate list.*

2. The patches on L are taken one by one and each patch P is classified with respect to the solid S, that is, the relations $P \,\text{in}^*\, S$, $P \,\text{out}^*\, S$ and $P \,\text{on}^*\, S$ are evaluated.

3. If $P\,\mathrm{on}^* S$ then P is put onto a list B called the *definitive boundary list*. This list will contain the result. If $P\,\mathrm{in}^* S$ or $P\,\mathrm{out}^* S$ then P is discarded since it cannot contribute to the boundary of S.

4. Finally, if none of the three relations holds, then P intersects the boundary of S somewhere, although it is not contained totally by it. In this case P should not be discarded but rather it is subdivided into smaller patches, say $P_1, \ldots, P_n$, which are put back onto the candidate list L. If, however, the size of P is below the predefined tolerance, then it is not subdivided further but placed onto a third list T called the *tentative boundary list.*

5. The process is continued until the candidate list L becomes empty.

The (approximate) boundary of S can be found in list B. The other output list, T, contains some "garbage" patches which may be the subject of further geometric calculations or may simply be discarded.

The crucial point is how to classify the patches with respect to the solid. The cited authors propose a computationally not too expensive approximate solution to this problem, which they call the method of *inner sets and outer sets*; the ISOS method.

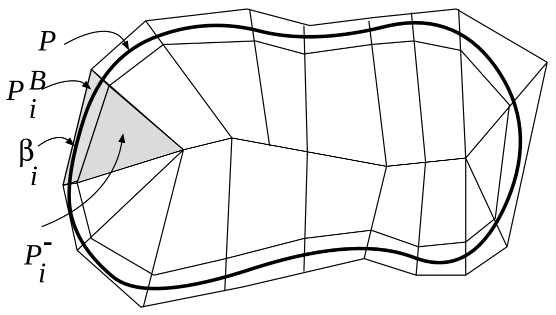

Figure 4.6: Inner and outer segments

Each primitive object P is approximated by two polyhedra: an *inner* polyhedron P^- and an *outer* polyhedron P^+:

$$P^- \subseteq P \subseteq P^+. \tag{4.14}$$

Both polyhedra are constructed from polyhedral segments. The segments of P^- and P^+, however, are not independent of each other, as illustrated in figure 4.6. The outer polyhedron P^+ consists of the *outer segments*, say $P_1^+, \ldots, P_n^+$. An outer segment P_i^+ is the union of two subsegments: the *inner segment* P_i^-, which is totally contained by the primitive, and the *boundary segment* P_i^B, which contains a boundary patch, say β_i (a part of the boundary of the primitive). The thinner the boundary segments the better the approximation of the primitive boundary by the union of the boundary segments. A coarse decomposition of each primitive is created in a preliminary phase according to point 1 of the above outlined strategy.

Set membership classification of a boundary patch β with respect to the compound solid S (point 2) is approximated by means of the inner, outer and boundary segments corresponding to the primitives. According to the **divide-and-conquer** approach, two different cases can be distinguished: one in which S is primitive and the second is in which S is compound.

Case 1: Classification of β with respect to a primitive P

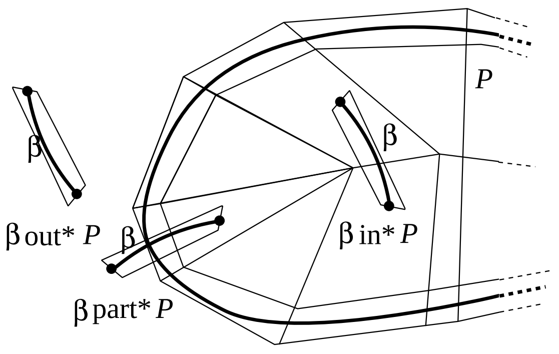

Figure 4.7: Relations between a boundary segment and a primitive

The following examinations must be made on the boundary segment P^B containing the boundary patch β with respect to P in this order (it is assumed that β is not a boundary patch of P because this case can be detected straightforwardly):

1. Test whether P^B intersects any of the outer segments $P_1^+, \ldots, P_n^+$ corresponding to P. If the answer is negative, then $P^B \,\text{out}^*\, P$ holds,

that is $\beta \text{out}^* P$ (see figure 4.7). Otherwise go to examination 2 (β is either totally or partially contained by P).

2. Test whether P^B intersects any of the boundary segments $P_1^B, \ldots, P_n^B$ corresponding to P. If the answer is negative, then $P^B \text{in}^* P$ holds, that is $\beta \text{in}^* P$ (see figure 4.7). Otherwise go to examination 3.

3. In this case, due to the polyhedral approximation, nothing more can be stated about β, that is, either one or none of the relations $\beta \text{in}^* P$, $\beta \text{out}^* P$ and (accidentally) $\beta \text{on}^* P$ could hold (figure 4.7 shows a situation where none of them holds). β is classified as *partial* in this case. This is expressed by the notation $\beta \text{part}^* P$ (according to point 4 of the generate-and-test strategy outlined previously, β will then be subdivided).

Classification results with respect to two primitives connected by a set operation can then be *combined.*

Case 2: Classification of β with respect to $S = A \circ^* B$

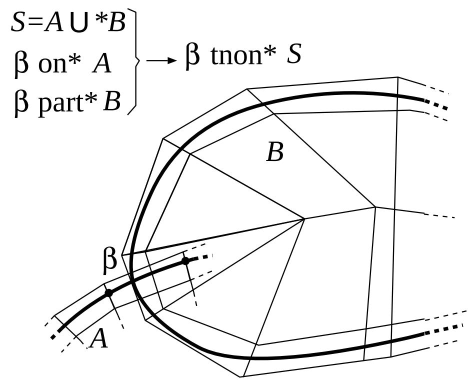

Figure 4.8: A boundary segment classified as a tentative boundary

After computing the classification of β with respect to A and B, the two results can be combined according to the following tables (new notations are defined after):

$S = A \cup^* B$	$\beta\,\mathrm{in}^* B$	$\beta\,\mathrm{out}^* B$	$\beta\,\mathrm{on}^* B$	$\beta\,\mathrm{part}^* B$	$\beta\,\mathrm{tnon}^* B$
$\beta\,\mathrm{in}^* A$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{in}^* S$
$\beta\,\mathrm{out}^* A$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{on}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$
$\beta\,\mathrm{on}^* A$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{on}^* S$	$*$	$\beta\,\mathrm{tnon}^* S$	$*$
$\beta\,\mathrm{part}^* A$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$
$\beta\,\mathrm{tnon}^* A$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{tnon}^* S$	$*$	$\beta\,\mathrm{tnon}^* S$	$*$

$S = A \cap^* B$	$\beta\,\mathrm{in}^* B$	$\beta\,\mathrm{out}^* B$	$\beta\,\mathrm{on}^* B$	$\beta\,\mathrm{part}^* B$	$\beta\,\mathrm{tnon}^* B$
$\beta\,\mathrm{in}^* A$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{on}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$
$\beta\,\mathrm{out}^* A$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$
$\beta\,\mathrm{on}^* A$	$\beta\,\mathrm{on}^* S$	$\beta\,\mathrm{out}^* S$	$*$	$\beta\,\mathrm{tnon}^* S$	$*$
$\beta\,\mathrm{part}^* A$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{tnon}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$
$\beta\,\mathrm{tnon}^* A$	$\beta\,\mathrm{tnon}^* S$	$\beta\,\mathrm{out}^* S$	$*$	$\beta\,\mathrm{tnon}^* S$	$*$

$S = A \setminus^* B$	$\beta\,\mathrm{in}^* B$	$\beta\,\mathrm{out}^* B$	$\beta\,\mathrm{on}^* B$	$\beta\,\mathrm{part}^* B$	$\beta\,\mathrm{tnon}^* B$
$\beta\,\mathrm{in}^* A$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{in}^* S$	$\beta\,\mathrm{on}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$
$\beta\,\mathrm{out}^* A$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{out}^* S$
$\beta\,\mathrm{on}^* A$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{on}^* S$	$*$	$\beta\,\mathrm{tnon}^* S$	$*$
$\beta\,\mathrm{part}^* A$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$	$\beta\,\mathrm{part}^* S$	$\beta\,\mathrm{tnon}^* S$
$\beta\,\mathrm{tnon}^* A$	$\beta\,\mathrm{out}^* S$	$\beta\,\mathrm{tnon}^* S$	$*$	$\beta\,\mathrm{tnon}^* S$	$*$

Two new notations are used here in addition to those already introduced. The notation $\beta\,\mathrm{tnon}^* S$ is used to express that β is a *tentative boundary* patch (see figure 4.8). The use of this result in the classification scheme always happens at a stage where one of the classification results to be combined is "on*" and the other is "part*", in which case the relation of β with respect to S cannot be ascertained. The patch can then be the subject of subdivision and some of the subpatches *may* come out as boundary patches. The other notation, the asterisk ($*$), denotes that the situation can occur in case of degeneracies, and that special care should then be taken in order to resolve degeneracies so that regularity of the set operations is not violated (this requires further geometric calculations).