Constructive solid geometry and Nef polyhedra

Lesson 3.2*

Contents

Until this lesson, we have only discussed data models that represent 3D geometries very *explicitly*. In other words, we have been specifying objects' shape through simple elements that have a direct geometric interpretation. For instance, a tetrahedron's shape can be known by looking only at the coordinates of its four vertices, and a polyhedron's shape by looking at the set of its bounding polygons (which have a shape that is easily obtained from a list of vertices). Even in a compact representation of a voxel, its geometry can be easily known based on a few simple parameters, such as: the absolute location and orientation of the voxel grid, the cell spacing along each axis, the order of the voxels in a linear encoding of the grid, and an index to identify the voxel on this encoding.

Since the elements in explicit representations can be interpreted easily, these kinds of explicit representations are usually the easiest for computers to process. However, they also have disadvantages: since objects are often composed of many small elements, they can be very inefficient with space and can also make it difficult for people to define objects. For instance, defining a shape that approximates a sphere using only polygons will require many small polygons to obtain a decent approximation, and manually obtaining the polygons to use (and their vertex coordinates) is not trivial.

The alternative to the explicit approach is thus to use more *implicit* representations, in which objects are represented as *sequences of operations on geometric primitives*. Thus, the exact shape of the objects being represented is only known after performing the geometric operations, which can be rather complex.
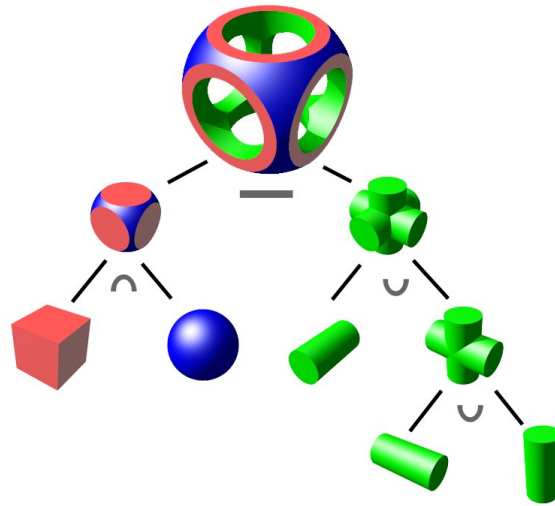

Figure 1: A CSG object represented as a tree of Boolean set operations on a sphere, a cube and three cylinders. From Wikimedia Commons.

However, the indirect approach makes it possible to use primitives that are better suited to a certain task, primitives that are easier to define, or simply fewer primitives overall.

1 What is constructive solid geometry?

Constructive solid geometry (CSG) is a general approach that combines many of techniques that are typically used with implicit representations, including primitive instancing, half-space intersections and Boolean set operations. Most other data models that use implicit representations can thus be considered as variations of CSG, usually with more restrictions on the operations that can be performed or the primitives that can be used.

CSG represents objects as *hierarchies of Boolean set operations* on other objects (Figure 1). A CSG object is thus a tree, where each non-leaf node is a Boolean set operation on its children, and where the leaves are mathematical definitions of point sets, usually describing very simple objects. In theory, arbitrary point sets can be used, although implementations usually limit them to some of the following:

Primitive instancing defines simple solids parametrically, such as a sphere based on a radius and the coordinates of its centre;

Arbitrary polyhedra defined using mesh data structures and boundary representation; and

Half-spaces (Figure 2) defined using a plane equation and a direction.

The next section of the handout is a short summary of the mathematical background for this lesson, which consists of set theory, Boolean set operations and their mathematical notation. Feel free to skip it if you are familiar with them. In the next two sections, we look at how the two main elements of CSG work in theory: (i) the definition of simple objects as point sets, and (ii) how these elements can be combined using Boolean point set operations. The final section covers Nef polyhedra, which are arguably the best known basis to implement CSG in practice.

2 Background: set theory and Boolean set operations

Set theory is the branch of mathematics that studies *sets*, which are collections of abstract objects. These objects can be anything, including other sets.
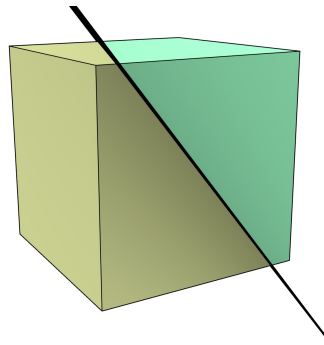
Figure 2: A plane separates 3D space into two parts on either side of it. A plane and a direction can thus be used to specify the geometry of one of these halves, which forms an unbounded space on all directions except one.

Set theory starts by considering the existence of a given domain of objects from which one may build sets, which is known as the *universe set* and denoted as $\mathbb{U}$. If an object a is part of a set $\mathbb{X}$, it is denoted as $a \in \mathbb{X}$, which is read as 'a is an *element* of $\mathbb{X}$'. If a is not part of a set $\mathbb{X}$, it is denoted as $a \notin \mathbb{X}$, which is read as 'a is not an element of $\mathbb{X}$'. Usually lower case is used for simple elements and upper case for sets.

There are two common ways to describe the elements in a set, both using curly braces, ie { and }. One way to do so is to list all the elements of the set one by one. For instance, the set $\{1, 2, 3\}$ is the set containing 1, 2 and 3 as elements (and no others). The other way to do so is to specify one or more rules that the elements of the set need to fulfil. For instance, the set $\{x : x \text{ is a prime number}\}$ consists of all prime numbers. It is read as 'x, such that x is a prime number'.

The order in which the elements in a set are defined does not matter. That is, $\{1, 2, 3\}$ and $\{3, 2, 1\}$ are the same set. The elements in a set are also unique, and duplicate items are ignored by convention. That is, $\{1, 2, 3\}$ and $\{1, 2, 3, 2, 1\}$ are also the same set.

A set may contain an infinite number of elements (eg as the prime number example above), or no elements at all, in which case it is a special set known as the *null set* and denoted as {} or $\varnothing$. Other commonly used sets with a special notation and name are: the natural numbers ($\mathbb{N}$), the real numbers ($\mathbb{R}$), the rational numbers ($\mathbb{Q}$) and the integers ($\mathbb{Z}$).

In order to build more complex sets, the concepts and notation from mathematical logic are used, in particular *propositional logic*. Propositional logic works with *propositions*, which are sentences that are either true or false, but not both. These propositions might be altered and combined using various symbols expressing various notions, such as: *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implies* ($\Rightarrow$), *is implied by* ($\Leftarrow$), *if and only if* ($\Leftrightarrow$), *for all* ($\forall$) and *exists* ($\exists$). These symbols correspond to their names. For instance, $a \wedge b$ is true only when both a and b are true, $a \vee b$ is true when a or b are true (or both), and $\neg a$ is true when a is false.

Using these concepts it is possible to state relationships between sets. For instance, we can define that $\mathbb{A}$ and $\mathbb{B}$ are equal ($\mathbb{A} = \mathbb{B}$) when an element is in $\mathbb{A}$ if and only if it is also in $\mathbb{B}$, which can be denoted as $\forall x : x \in \mathbb{A} \Leftrightarrow x \in \mathbb{B}$. A set $\mathbb{A}$ is called a subset of a set $\mathbb{B}$ ($\mathbb{A} \subseteq \mathbb{B}$), or $\mathbb{B}$ is a superset of $\mathbb{A}$ ($\mathbb{B} \supseteq \mathbb{A}$), when if an element is in $\mathbb{A}$ then it is also in $\mathbb{B}$, denoted as $\forall x : x \in \mathbb{A} \Rightarrow x \in \mathbb{B}$. If $\mathbb{A} \subseteq \mathbb{B}$ but $\mathbb{A} \neq \mathbb{B}$, ie there is at least one extra element in $\mathbb{B}$, then $\mathbb{A}$ is a proper subset of $\mathbb{B}$ ($\mathbb{A} \subset \mathbb{B}$), or alternatively $\mathbb{B}$ is a proper superset of $\mathbb{A}$ ($\mathbb{B} \supset \mathbb{A}$). Note that these relationships are akin to 'less than' ($<$), 'less or equal than' ($\leq$), 'equal to' ($=$), 'greater or equal than' ($\geq$), and 'greater than' ($>$) for numbers.

It is also possible to use propositional logic to create new sets by defining certain operations between sets, in particular *Boolean set operations*, consisting of intersection, union, difference and complement. The intersection of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \cap \mathbb{B}$, consists of all the elements that are both in $\mathbb{A}$ and in $\mathbb{B}$, ie $\mathbb{A} \cap \mathbb{B} = \{x : x \in \mathbb{A} \wedge x \in \mathbb{B}\}$. The union of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \cup \mathbb{B}$, consists of all the elements that are either in $\mathbb{A}$ or in $\mathbb{B}$, ie $\mathbb{A} \cup \mathbb{B} = \{x : x \in \mathbb{A} \vee x \in \mathbb{B}\}$. The difference between sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} - \mathbb{B}$, consists of all the elements that are in $\mathbb{A}$ but not in $\mathbb{B}$, ie

$\mathbb{A} - \mathbb{B} = \{x : x \in \mathbb{A} \wedge x \notin \mathbb{B}\}$. The complement of a set $\mathbb{A}$, denoted as $\neg\mathbb{A}$, consists of all the elements that are in the universe set but are not in $\mathbb{A}$, ie $\neg\mathbb{A} = \{x : x \in \mathbb{U} \wedge x \notin \mathbb{A}\}$.

Apart from sets, it is also possible to consider *tuples* of elements, which are sequences of ordered elements. A tuple containing exactly two elements is known as a *pair*, one containing three elements is a *treble* and one containing n elements is an n-tuple. Tuples are denoted using parenthesis, ie (and).

A common operation that generates tuples is the Cartesian product. The Cartesian product of sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \times \mathbb{B}$, is defined as $\{(a,b) : a \in \mathbb{A} \wedge b \in \mathbb{B}\}$. In other words, it is a set of pairs, where the first element of a pair is an element of $\mathbb{A}$ and the second element of the pair is an element of $\mathbb{B}$. This can be generalised to more than two sets, such that the n-fold Cartesian product of n sets is an n-tuple. The n-fold Cartesian product of a set $\mathbb{A}$ with itself, ie $\mathbb{A} \times \mathbb{A} \times \cdots \mathbb{A}$, is denoted as $\mathbb{A}^n$.

3 Defining objects using point set geometry

Point set geometry applies the notions of set theory to define the geometry of objects as sets of points. The usual definition maps 1D space (ie the line) to the set of real numbers (ie $\mathbb{R}$), and so 2D space (ie the plane) is $\mathbb{R}^2$ and 3D space is $\mathbb{R}^3$.

Individual points in 2D and 3D space can be considered as elements of $\mathbb{R}^2$ and $\mathbb{R}^3$. For instance, we can denote a point p in 2D space as $p \in \mathbb{R}^2$ or in 3D space as $p \in \mathbb{R}^3$. Note that this notation perfectly matches the way in which points are usually defined based on their coordinates. For example, by stating $p = (x,y,z) \in \mathbb{R}^3$, we simply mean that $x,y,z \in \mathbb{R}$, ie that x, y and z are arbitrary real numbers.

Based on these definitions, we can then define sets that describe specific geometric objects. For instance, we can start by considering how any point p between two points p_1 and p_2 at different locations can be obtained as a sort of weighted average of p_1 and p_2, where the relative weight of the two points tell us that we're closer to one point than to another. If we put this into an equation, we get:

$$p = \frac{ap_1 + bp_2}{a + b}. \tag{1}$$

Note that we divide everything by $a + b$ to make sure that the weights add up to one. Also, note it is possible to use negative weights to get points that are on the line that passes through a and b but not between a and b. Expanding on this, the line L passing through p_1 and p_2 is defined by considering all possible values of a and b. That is:

$$L = \left\{ \frac{ap_1 + bp_2}{a + b} : a, b \in \mathbb{R} \right\}. \tag{2}$$

In the case of a line, we can get rid of one parameter by substituting $t = a/(a + b)$, which would yield $L = \{tp_1 + (1 - t)p_2 : t \in \mathbb{R}\}$. Note how at $t = 0$ we get p_2, at $t = 1$ we get p_1, and when $0 < t < 1$ we get the line segment between p_1 and p_2. Note also how this definition of a line works both in 2D and 3D.

Generalising from Equation 2, we can also define a similar equation for a plane P from three non-collinear points p_1, p_2 and p_3 as:

$$P = \left\{ \frac{ap_1 + bp_2 + cp_3}{a + b + c} : a, b, c \in \mathbb{R} \right\}. \tag{3}$$

In 3D, if we substitute the equality (=) of the previous equation for a strict inequality (< or >), we get instead an equation to represent the half-spaces respectively below and above the plane (such as those previously shown in Figure 2). An equation of this form is typically stored in the leaves of a CSG tree, eg as the three non-collinear points in the equation above, or as the coefficients of an equation of the form $ax + by + cz + d = 0$, plus a direction to specify which half-space to use.
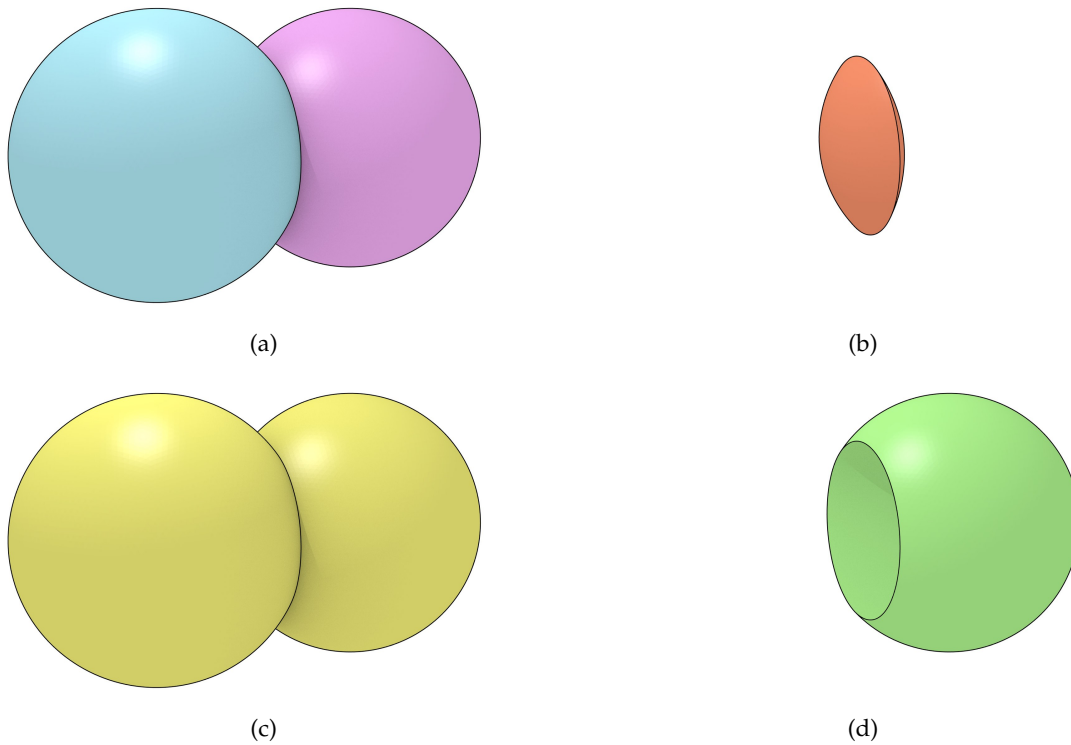
Figure 3: Based on (a) two balls $\mathbb{A}$ and $\mathbb{B}$, other objects can be defined using Boolean set operations, such as: (b) the intersection $\mathbb{A} \cap \mathbb{B}$, (c) the union $\mathbb{A} \cup \mathbb{B}$, and (d) the difference $\mathbb{A} - \mathbb{B}$.

For the sake of uniformity and ease of processing, many CSG implementations only use half-spaces. However, several simple axis-aligned 3D solids also have easy point set definitions that can be used to store them as primitives based on a few parameters, including:

balls ie the space inside a sphere, which can be defined as $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 < r^2$, where r is the radius and $c = (c_x, c_y, c_z)$ is the centre;

cuboid interiors ie box-shaped objects, which can be defined using intervals for the minimum and maximum values it has along each axis, ie $x_{\min} < x < x_{\max} \wedge y_{\min} < y < y_{\max} \wedge z_{\min} < z < z_{\max}$; and

cylinder interiors by checking whether a point lies within a radius for two axes, and within an interval for the third.

Other common objects are more general versions of these objects, such as ellipsoid interiors, parallelepipeds, cones, etc. As for non-axis aligned simple solids, they can be supported by special nodes in the CSG tree that represent geometric transformations by their parameters (rather than Boolean point set operations), such as translations, rotations and scaling, or more general ones like affine transformations or arbitrary transformation matrices by storing their elements one by one.

4 Boolean point set operations

In order to create objects other than those directly added to an implementation, CSG relies on Boolean point set operations (Figure 3). Arbitrary polyhedra can be represented in this way by first splitting them into convex parts (which might require Steiner vertices), and then representing the convex parts as intersections of half-spaces (one per face). These operations, which are located in the non-leaf nodes of the CSG tree, combine the geometry of the point sets described by their children (using the methods described in the previous section).
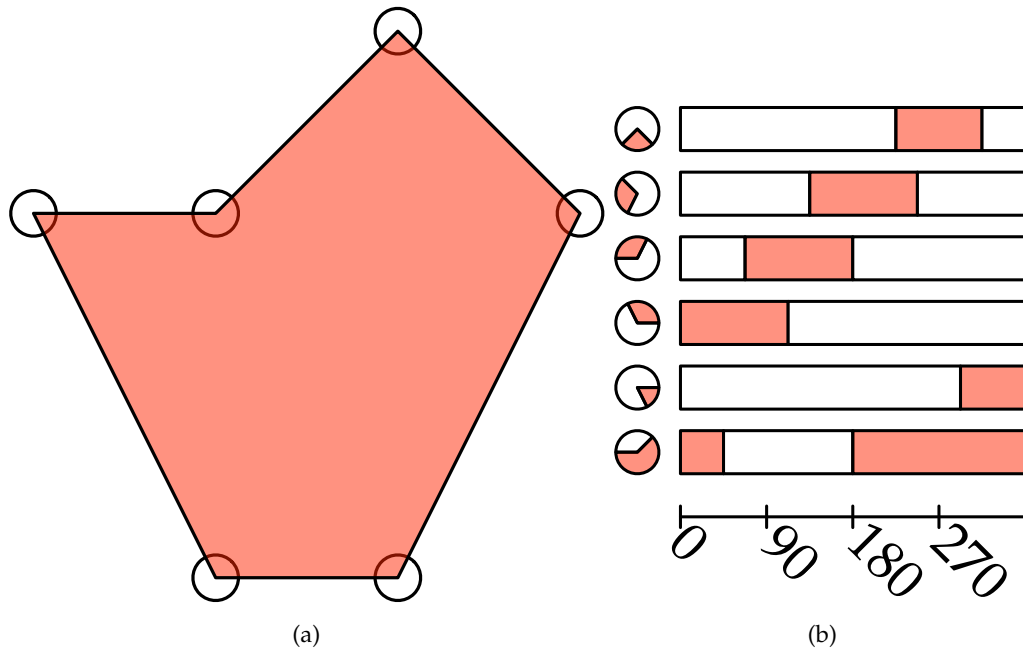
Figure 4: (a) A Nef polygon is represented indirectly as (b) a set of local pyramids (circles). At every local pyramid, the polygon (red) becomes an angular interval. Incident edges become points at the endpoints of these intervals.

Boolean point set operations are based on the Boolean operations on sets, which are mainly:

union of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \cup \mathbb{B}$, is the set containing the elements that are in $\mathbb{A}$ or $\mathbb{B}$, ie $\mathbb{A} \cup \mathbb{B} = \{x : x \in \mathbb{A} \vee x \in \mathbb{B}\}$;

intersection of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \cap \mathbb{B}$, is the set containing the elements that are in both $\mathbb{A}$ and $\mathbb{B}$, ie $\mathbb{A} \cap \mathbb{B} = \{x : x \in \mathbb{A} \wedge x \in \mathbb{B}\}$;

set difference of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} - \mathbb{B}$ or $\mathbb{A} \setminus \mathbb{B}$, is the set containing the elements that are in $\mathbb{A}$ but not in $\mathbb{B}$, ie $\mathbb{A} - \mathbb{B} = \{x : x \in \mathbb{A} \wedge x \notin \mathbb{B}\}$;

symmetric difference of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \triangle \mathbb{B}$, $\mathbb{A} \ominus \mathbb{B}$ or $\mathbb{A} \oplus \mathbb{B}$, is the set containing the elements that are either in $\mathbb{A}$ or in $\mathbb{B}$ but not in both, ie $\mathbb{A} \triangle \mathbb{B} = \{x : x \in \mathbb{A} - \mathbb{B} \cup \mathbb{B} - \mathbb{A}\}$.

The key point about these operations is that they do not need to perform any geometric computations, since it is possible to tell if an element (ie point) is in the new point set by just checking whether it is in its children's point sets. It is thus trivially easy to do Boolean point set operations on individual points.

Based on this knowledge, we could build a crude CSG implementation directly on a point cloud or voxel grid by: (i) for each leaf node, checking whether each point/voxel meets the point set definition in the node, and (ii) for each non-leaf node, applying the Boolean set operations on the point sets represented by its child nodes point by point. However, since this solution is not applicable to other data models, we will look at a better solution that works better in practice.

5   Nef polyhedra

Nef polyhedra, named after Walter Nef, are an alternative representation of polygons and polyhedra that is based on the concept of a *local pyramid*, which is a structure that stores the neighbourhood information around every vertex (Figure 4). Polygons and polyhedra can be stored as a set of local pyramids and their location (as a set of 2D/3D coordinates).

5.1 Local pyramids

The local pyramid of a vertex contains the intersection of an infinitesimally small sphere (in 3D) or circle (in 2D) with the volumes, faces and edges incident to this vertex. An incident volume thus becomes a face, an incident face becomes an edge, and an incident edge becomes a vertex on the surface of the local pyramid sphere/circle, essentially lowering the dimension of every object by one.

The key thing to understand here is the following: *a 2D/3D object represented as a set of local pyramids (and their location) can individually be stored using 1D/2D data structures*. This is a process akin to boundary representation, but it does not have problems with non-manifold objects (unlike boundary representation).

In practice, computing the local pyramid at a local vertex is also a relatively simple operation. We will not go through the details here (see the notes), but in 2D, it involves computing the angle of its neighbouring vertices as you rotate around the vertex, and marking the intervals between these vertices with the polygons that you pass through while doing so. In 3D, it is a more complex operation involving the computation of an arrangement of lines in a spherical coordinate system, and the location of every neighbouring vertex is defined by two angles (rather than one).

5.2 Computing Boolean point set operations on Nef polyhedra

Boolean point set operations (and many other geometric operations) can be computed in three steps: subdivision, selection and simplification. We will discuss what each of these involves.

Subdivision involves computing an overlay of the input polyhedra, thus creating the overall structure where the result will be put (ie the vertices, edges, faces and volumes).

In 2D, this is also a computation of a line arrangement (also known in GIS as map overlay), where the output is a set of vertices and edges representing all the input lines of both polygons. Vertices (ie local pyramids) will be located at the position of all input vertices and at every new intersection between lines.

In 3D, it is a similar operation, but the new vertices (ie local pyramids) are located at line-polygon intersections as well. These can be computed by computing a plane passing through the polygon and intersecting it with the line.

Selection involves checking whether each face (in 2D) or volume (3D) should be part of the output or not, marking it as such in the relevant parts of the local pyramids. This is done by testing whether it is in the interior or exterior of the input Nef polygons/polyhedra.

Simplification involves removing unnecessary structures in a way that does not alter the point set that is represented, which is akin to the dissolving operations common in GIS. This is done by deleting local pyramids when they do not actually represent a new vertex, or when are not subdivided.

Figure 5 shows an example of how this works in practice in 2D. A 2D Boolean point set operation starts from two Nef polygons $A = (g, b, f, i)$ and $B = (a, f, k, j, e, c)$—each of which is stored as a set of local pyramids at its corresponding vertices. As shown previously in Figure 4, each of these 2D local pyramids can be stored as a list of 1D intervals, eg at vertex a, polygon $B = [225, 315]$, where the values are in degrees.

The operation first computes the intersections between the line segments (as an overlay problem), creating the new vertices d and h. The location of these vertices can be calculated using the equations of the corresponding lines. The vertices of each polygon and the intersection points between the line segments yield the local pyramids to be considered.

Then, the local pyramid intervals for both polygons at all of these locations are computed. For instance, at vertex a, $A = \varnothing$ and $B = [225, 315]$. A Boolean set operation is then computed by applying it to the local pyramids (ie to the intervals). For instance, at vertex a, $\neg A = \mathbb{U} = [0, 360]$ and $\neg B = [315, 225]$ (by inverting the range), $A \cup B = [225, 315]$ (by combining the ranges), $A \cap B = \varnothing$ (by finding common parts of the ranges), and $A - B = \varnothing$ (by removing from the ranges of A those of B).
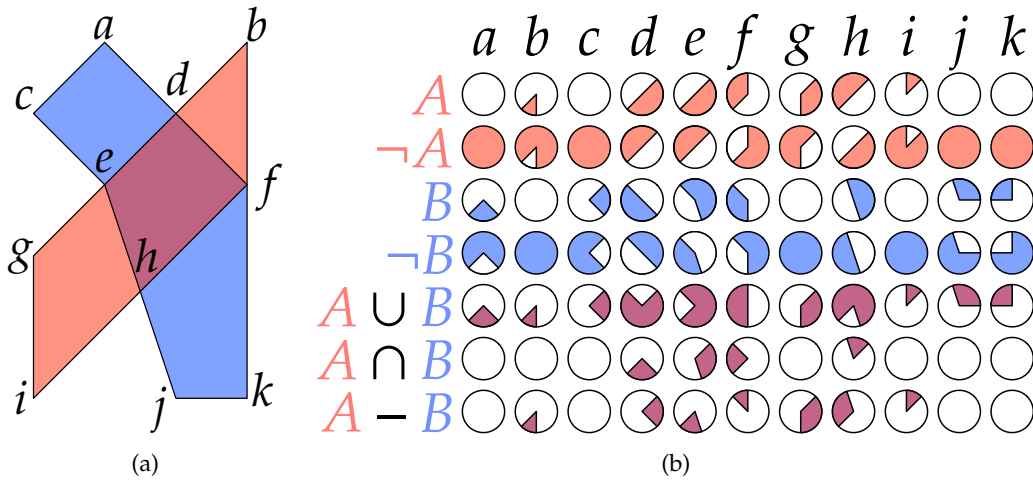
Figure 5: Various Boolean point set operations on (a) the Nef polygons A (red) and B (blue) that can be performed on (b) their local pyramids.

Finally, unnecessary local pyramids can be removed from the output: f in $A \cup B$; a, b, c, g, i, j and k in $A \cap B$; and a, c, j and k in $A - B$.

5.3 3D Nef polyhedra in practice: selective Nef complexes

3D Nef polyhedra and Boolean point set operations on them can be implemented using different data structures, but an excellent open implementation (see notes) uses a data structure called *selective Nef complexes* (Figure 6).

Selective Nef complexes (SNC) use a combination of two half-edge data structures:

- a standard half-edge data structure on 3D space, which stores each face of each polyhedron as a cycle of *edge-uses* connecting vertices;

- a half-edge data structure to represent each local pyramid (one per vertex) as a subdivision on the surface of an (infinitesimally small) sphere.

Each vertex is linked to its sphere map, where each incident volume corresponds to a face on its sphere map (sface), each incident face corresponds to an edge (sedge), and each incident edge corresponds to a vertex (svertex). These corresponding elements are linked to each other, which makes it possible to navigate both on the half-edge data structure in 3D space (eg by going from an edge-use to the next to cycle around a face) and on the half-edge data structure of a sphere map (eg by going from one sedge to the next to cycle around an sface).

6 Exercises

1. How can you define a cube using:

    a) a parametric representation

    b) a b-rep data structure

    c) an intersection of half-spaces

2. In the line $L = a p_1 + (1 - a) p_2$, what is the geometry given by $a < 0$ and $a > 1$?

3. Compute the ranges for the local pyramids of some other vertices in Figure 5.

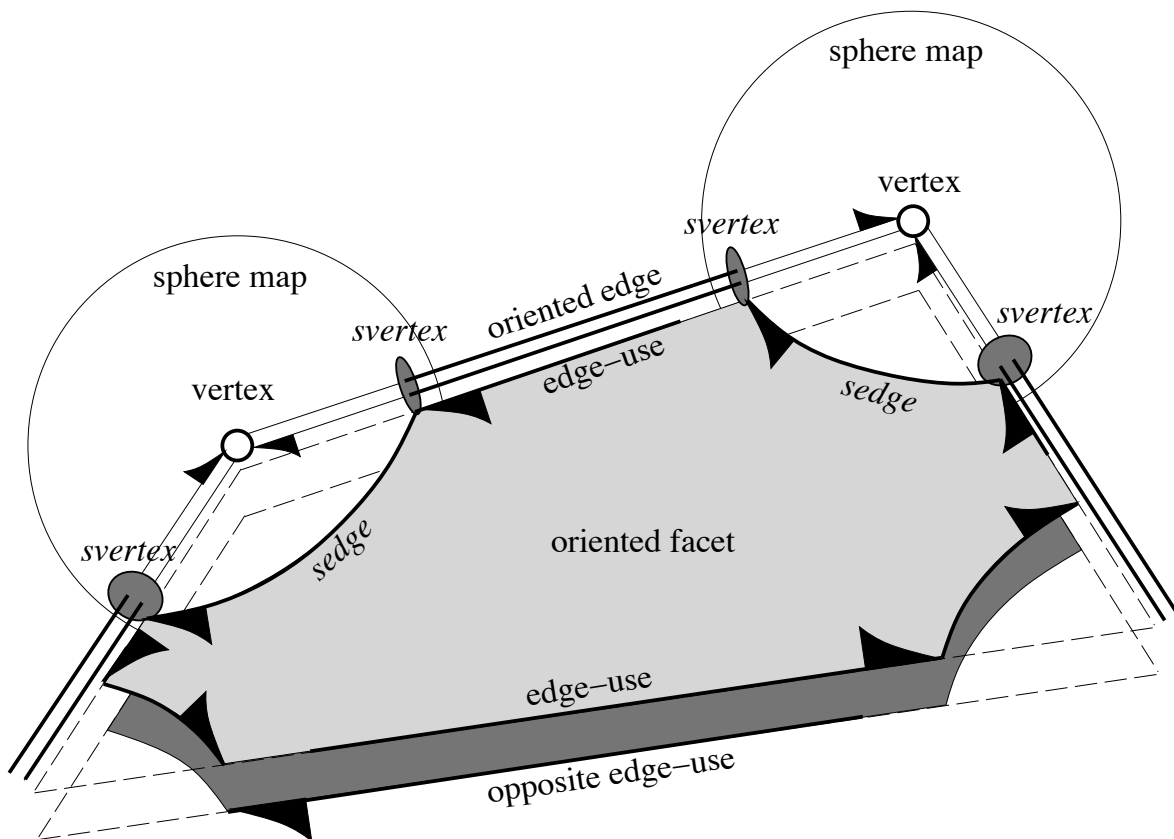4. In which cases is simplification needed in 2D?

Figure 6: A selective Nef complex. The standard half-edge structure on 3D space uses faces (as an oriented facet for each incident polyhedron), edges (as an edge-use for each incident oriented facet) and vertices. The half-edge structure on the surfaces of the spheres representing local vertices, known as sphere maps, uses *sfaces* (per incident polyhedron but not shown here), *sedges* (per incident face) and *svertices* (per incident edge). From Hachenberger (2006).

7 Notes and comments

If you need more help with the mathematical background, check the Wikipedia pages on sets[1], set algebra[2].

The earliest description of CSG is likely Requicha and Voelcker (1977, §12.3) and its properties is Requicha and Tilove (1978). However, it is more of a culmination of efforts of many people. For instance, Shamos and Hoey (1976) and Preparata and Muller (1979) show that it is possible to represent any convex object (of any dimension) as the intersection of a finite number of half-spaces. In order to see how such a decomposition can be done, see Chazelle and Dobkin (1979) or Bajaj and Dey (1990). A nice example of how half-spaces can be stored in practice is Naylor (1990).

Nef polyhedra were originally described in Nef (1978), although a much better description including the way they work with Boolean set operations is available in Bieri and Nef (1988).

For more background on the line arrangement problem, see the relevant Wikipedia page[3]. For a clear description of how to compute one, see de Berg et al. (2008, §2) or the user manual of the `Arrangements_2` package of CGAL[4].

Nef polyhedra as a CSG engine in practice is only possible thanks to Seel (2001) in 2D and Hachenberger (2006) in 3D. They discuss how to compute local pyramids in 2D and 3D, as well as Boolean point set operations on polygons and polyhedra. They are implemented in the CGAL packages 2D Boolean Operations on Nef Polygons[5] and 3D Boolean Operations on Nef Polyhedra[6].

The general scheme to perform geometric operations in three steps (subdivision, selection and simplification) is discussed by Rossignac and O'Connor (1989).

References & further reading

Bajaj C and Dey TK (1990). Convex decomposition of polyhedra and robustness. Technical report, Purdue University.

Bieri H and Nef W (1988). Elementary set operations with *d*-dimensional polyhedra. In Noltemeier H, editor, *Computational Geometry and its Applications*, volume 333 of *Lecture Notes in Computer Science*, pages 97–112. Springer Berlin Heidelberg.

Chazelle B and Dobkin D (1979). Decomposing a polygon into its convex parts. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 38–48.

de Berg M, van Kreveld M, Overmars M, and Schwarzkopf O (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition.

Hachenberger P (2006). *Boolean Operations on 3D Selective Nef Complexes Data Structure, Algorithms, Optimized Implementation, Experiments and Applications*. Ph.D. thesis, Saarland University.

Naylor B (1990). Binary space partitioning trees as an alternative representation of polytopes. *Computer-Aided Design*, 22(4).

Nef W (1978). *Beiträge zur Theorie der Polyeder: mit Anwendungen in der Computergraphik*. Herbert Lang, Bern.

Preparata F and Muller D (1979). Finding the intersection of *n* half-spaces in time $o(n \log n)$. *Theoretical Computer Science*, 8(1):45–55.

---

[1] `https://en.wikipedia.org/wiki/Set_(mathematics)`
[2] `https://en.wikipedia.org/wiki/Algebra_of_sets`
[3] `https://en.wikipedia.org/wiki/Arrangement_of_lines`
[4] `https://doc.cgal.org/latest/Arrangement_on_surface_2/index.html`
[5] `https://doc.cgal.org/latest/Nef_2/index.html`
[6] `https://doc.cgal.org/latest/Nef_3/index.html`

Requicha AAG and Tilove RB (1978). Mathematical foundations of constructive solid geometry: General topology of closed regular sets. Production Automation Project Technical Memorandum 27, University of Rochester.

Requicha AAG and Voelcker HB (1977). Constructive solid geometry. Technical Memorandum 25, College of Engineering & Applied Science, The University of Rochester.

Rossignac J and O'Connor M (1989). SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In Wosny M, Turner J, and Preiss K, editors, *Proceedings of the IFIP Workshop on CAD/CAM*, pages 145–180.

Seel M (2001). *Planar Nef Polyhedra and Generic Higher-dimensional Geometry*. Ph.D. thesis, Saarland University.

Shamos MI and Hoey D (1976). Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on the Foundations of Computer Science*, pages 208–215.