Topology

Jarek Rossignac

For sake of simplicity, the data structures for representing shapes and the algorithms that construct or process such representations are usually limited to a specific class. For example, some systems or Application Programming Interfaces (APIs) assume that a polygonal face may not have a hole or that the boundary of a solid must be manifold and some compression algorithms assume that a triangle mesh has genus zero.

Topological and differential concepts are used to define precisely the particular class of interest and for enunciating useful properties satisfied by all shapes in the class. For example, many graphic and animation systems support the class of shapes bounded by triangle meshes. The number of handles in a connected manifold triangle mesh may be easily derived if one knows the number of triangles and vertices in the mesh.

In this chapter, we focus on topology and introduce a variety of topological concepts, terminologies, and notations. We start with sets, which are arbitrary collections of curves, faces, and volumes. We define Boolean and topological operators on them. These operators may be used to analyze shapes, to create new ones, or to study the interaction (contact, interference) between shapes. These topological concepts are independent of the analytic nature of the supporting geometries. For instance they do not distinguish between smooth and triangulated surfaces. Furthermore, they do not define or assume any specific data-structure or representation scheme.

Because most digital representation schemes for shapes are based on representations of simple primitives, which we call cells. The most common cells are the vertex and the edge (line segment) between two vertices. We explain how a Selective Geometric Complex (SGC) may be used to represent and process an extremely rich class of sets defined in terms of such simple cells. Then, by imposing restrictions on SGCs, we define several subclasses commonly used in modeling: faces, polygons, simplicial complexes, and triangle meshes.

We discuss representations of polygons and algorithms that test points against polygons or compute regularized Boolean combinations or contacts between polygons. Finally, we discuss classes of triangle meshes and their combinatorial and topological properties.

Many of the concepts discussed here, such as Boolean operations, are independent of dimension. When this is the case, for sake of simplicity and clarity, we will use 2D (two-dimensional) examples to illustrate them. Other concepts, such as the genus, are intrinsically three-dimensional.

1 - Sets

1.1 Boolean operations on sets

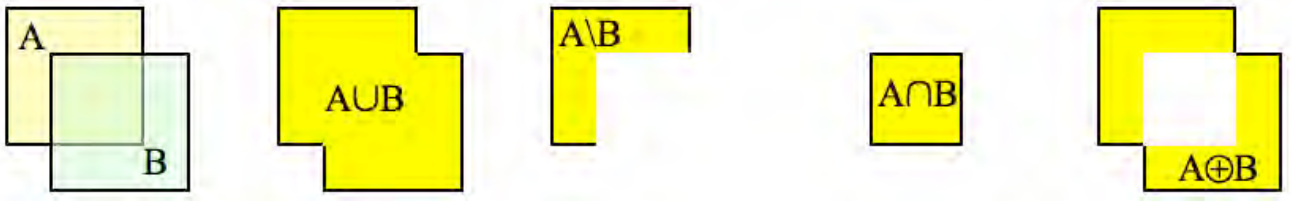
To formulate some of the definitions or properties discussed here, we will use logical expressions. In logical expressions, the ***logical operators*** have lower priority than other operators. The following logical operators are listed by decreasing priority: NOT (¬), AND (∧), OR (∨), implication (⇒), and equivalence (⇔). We use the term "when" to mean "if and only if".

The term ***set*** refers to a subset of the continuous underlying space Ω, which is either the two-dimensional (2D) plane or the three-dimensional (3D) space in which the shapes of interest are defined or animated. We will use bold uppercase letters, such as **A**, **B**, **S**, and **T** to denote such sets and bold lowercase letters, such as **p** and **q**, to denote points in Ω. We often define a set **S** by a recipe in the form **S**={**p**: f(**p**)}, which indicates that **S** is the set of points **p** for which the predicate f(**p**) is true. The notation **p**∈**S** indicates that **p** is part of **S**, i.e., is ***in*** **S**. The notation **p**∉**S** indicates the contrary. Set **A** is ***included*** in set **B** (written **A**⊂**B**) when **p**∈**A** ⇒ **p**∈**B**. **A** and **B** are ***equal*** (written **A**=**B**) when **A**⊂**B** ∧ **B**⊂**A**. The ***complement*** (written !**S**) of **S** is the set {**p**: **p** ∉**S**} of points not in **S**. Note that !!**S**=**S**. The complement operator (!) has highest priority in Boolean expressions.

To combine sets, we use the following Boolean operators: ***union*** $\mathbf{A}\cup\mathbf{B}=\{\mathbf{p}\in\mathbf{A} \vee \mathbf{p}\in\mathbf{B}\}$, ***intersection*** $\mathbf{A}\cap\mathbf{B}=\{\mathbf{p}\in\mathbf{A} \wedge \mathbf{p}\in\mathbf{B}\}$, ***difference*** $\mathbf{A}\backslash\mathbf{B}=\{\mathbf{p}\in\mathbf{A} \wedge \mathbf{p}\notin\mathbf{B}\}$, and ***symmetric-difference*** (also called ***XOR***) $\mathbf{A}\otimes\mathbf{B}=(\mathbf{A}\backslash\mathbf{B})\cup(\mathbf{B}\backslash\mathbf{A})$.

The complement operator (!) has the highest priority. ∩ has higher priority than ∪ and \. For simplicity, ∩ may be omitted, ∪ may be written as +, and \ may be written as –.

The ***empty set*** is denoted by ∅. We say that **A** and **B** are ***disjoint*** when $\mathbf{A}\cap\mathbf{B}=\emptyset$ and that they ***interfere*** when $\mathbf{A}\cap\mathbf{B}\neq\emptyset$. The notion of interference is important when discussing the validity of the model of a shape (it is typical to assume that primitive cells do not interfere), when analyzing the design of a mechanical assembly (two parts may touch but their interiors should not interfere), and when simulating physically plausible animations (object change their motions upon collision). A ***collection*** of two or more sets is ***exclusive*** (i.e., pair-wise disjoint) when for all pairs (**A**,**B**) of sets in the collection, $\mathbf{A}\neq\mathbf{B} \Rightarrow \mathbf{A}\cap\mathbf{B}=\emptyset$. Similarly, a collection is ***exhaustive*** if the union of all the sets equals Ω. For example, the collection {**S**,!**S**} is exclusive and exhaustive, since $\mathbf{S}\cup!\mathbf{S}=\Omega$ and $\mathbf{S}\cap!\mathbf{S}=\emptyset$.



1.2 Properties of sets

Some properties are only true if a shape is connected. Furthermore, several algorithms process one connected component of the shape at a time. Let us define these concepts more precisely.

A set **S** is ***connected*** if from every point **p** in **S** one can walk to every other point **q** in **S** along a curve **C** that lies entirely in **S**. More formally, **S** is connected when $\forall\mathbf{p}\in\mathbf{S}\ \forall\mathbf{q}\in\mathbf{S}, \exists\mathbf{C}\subset\mathbf{S}$: **C**=curve(**p**,**q**), where curve(**p**,**q**) is a continuous curve connecting **p** to **q**.

A non-empty set **S** has one or more maximally connected components, which for simplicity we call the ***components*** of **S**. Even though the notion of component is intuitive, its precise definition is not trivial: **T** is a component of **S** if $\mathbf{T}\subset\mathbf{S}$, if **T** is connected, and if $\forall\mathbf{U}\subset\mathbf{S}$, $(\mathbf{T}\backslash\mathbf{U}\neq\emptyset)\Rightarrow$(**T** not connected). In other words, no connected subset of **S** that is different from **T** contains a component **T**.

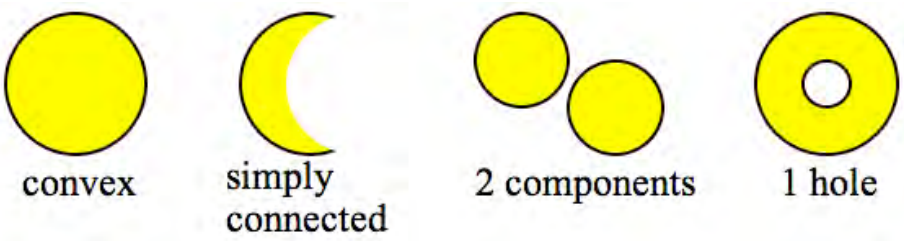
Because common representations of coordinates (integers, floats) assume finite values, many modeling systems assume that specific points and cells defined in terms of these points are within some finite distance from the origin. We say that they are bounded. On the other hand, some cells, such as a ray or a plane, are unbounded. Specifically, we say that a set **S** is ***bounded*** when it is contained in a disk D of finite radius. It is ***unbounded*** otherwise. We define a disk **D** with center **c** and radius r as the set $\{\mathbf{p}: \mathbf{cp}^2<r^2\}$ of points **p** that are closer than *r* from **c**.

A set **S** may have one or more holes. Assume that **S** is bounded. We define the ***holes*** of **S** as the components of !**S**. This precise mathematical definition of holes leads to algorithms for computing a representation of each hole.

The term hole refers to a cavity in the set. It should not be confused with a ***handle*** (or through-hole) such as the hole through in a doughnut or in the handle of a coffee mug. In fact, we do not have a definition of a handle and hence cannot decide which portion of **S** or of !**S** are in a particular handle. However, we can compute the genus of a set, which is the number of handles.

We say that **S** is ***simply connected*** when it has one component and does not have any holes or handles. **S** is ***convex*** when the line segment joining any two points of **S** is in **S**. More precisely: $\forall\mathbf{p}\in\mathbf{S}\ \forall\mathbf{q}\in\mathbf{S}$, Edge(**p**,**q**)$\subset$**S**, where Edge(**p**,**q**) is the line segment joining **p** and **q**. For example, a disk or a filled square are convex, while a circle or a crescent are not. Many calculations or operations on sets may be computed using simpler or more efficient algorithms when the sets are convex or simply connected.

The **convex hull** of a set is the smallest convex set containing it. Convex hulls are often used to accelerate collision detection and occlusion tests. Note that the convex hull of a planar set S is not necessarily the union of all line segments with endpoints in S. Can you find a counterexample?



1.3 Topological operators for sets

We want to distinguish between the interior, the exterior, and the boundary of a set. For example, we may say that two sets *interfere* if their interiors do and that they *touch* if their interiors are disjoints but their boundaries interfere. Furthermore, solid modeling systems produce *regularized* models by identifying the *membrane*, which is the portion of the boundary that actually separates the set from its complement, and discarding the *dangle*, which is the rest of the boundary. We provide here more precise definitions for these terms.

We define the boundary of a set by using the concept of neighborhood. The ***neighborhood*** N(**p**) of a point **p** is an ***open disk*** Disk(**p**,r)={**q**: $\mathbf{qp}^2<r^2$} of center **p** and radius *r*, where *r* is positive, but infinitely small. One could use an *r* that is not infinitely small, provided that the number of components in N(**p**)∩**S** and in N(**p**)∩!**S** is not affected by picking any smaller radius. A point **p** ***touches*** **S** (written **p**|**S**), if N(**p**)∩**S**≠∅.

S is **open** if it contains no point that touches !**S**. For example, the disk Disk(**p**,r) is open.

S is ***closed*** if it contains all points that touch it. The intersection of two closed sets is closed. For example, the set ClosedDisk(**p**,*r*)={**q**: $\mathbf{qp}^2\le r^2$} and Circle(**p**,*r*)={**q**: $\mathbf{qp}^2=r^2$} are both closed.

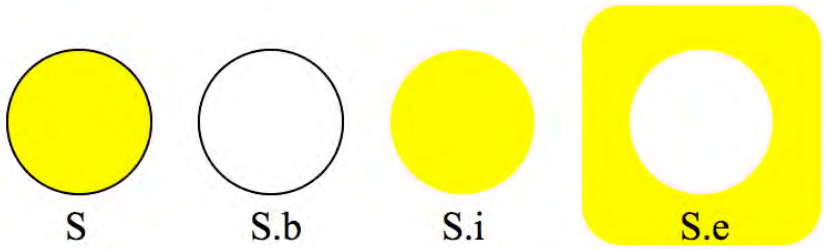
The ***boundary*** **S**.b of **S** is the set {**p** : **p**|**S** ∧ **p**|!**S**} of points that touch both **S** and !**S**. Note that a component of **S**.b may be a single isolated point. For example the difference Disk(**p**,*r*)**p**, which is an open disk with the center removed has as boundary Circle(**p**,*r*)∪**p**.

The ***interior*** **S**.i is the portion **S****S**.b of **S** that is not in **S**.b.

Similarly, the ***exterior*** **S**.e is !**S****S**.b.

In summary, a set S provides a decomposition of Ω into an exclusive and exhaustive collection: {**S**.i, **S**.b, **S**.e}.

We use (below) black curves to show boundaries that belong to a set and yellow areas to denote their interior.

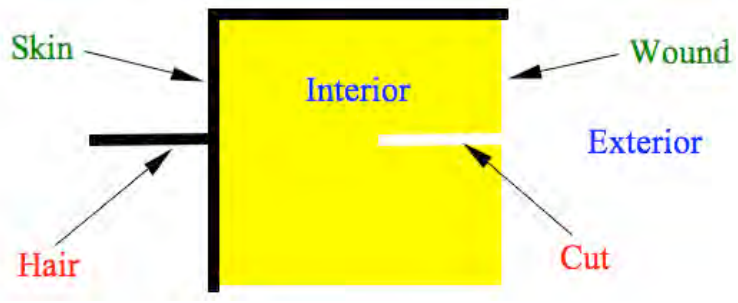


We can further decompose **S**.b using two criteria: (1) Some portions are separating **S**.i from **S**.e and some are not. (2) Some portions are in **S** and some are not.

The ***membrane*** **S**.m is the set **S**.i.b∩**S**.e.b of points that touch both the interior S.i and the exterior S.e. Note that we are using the cascading notation **S**.i.b for (**S**.i).b. The ***dangle*** S.d is the remaining set S.b\S.m of the boundary.

The membrane may be further decomposed into two parts (i.e, is the union of two disjoint parts): the part in **S** is the ***skin***, **S**.s, of **S** and the part in !**S** is the ***wound***, **S**.w.

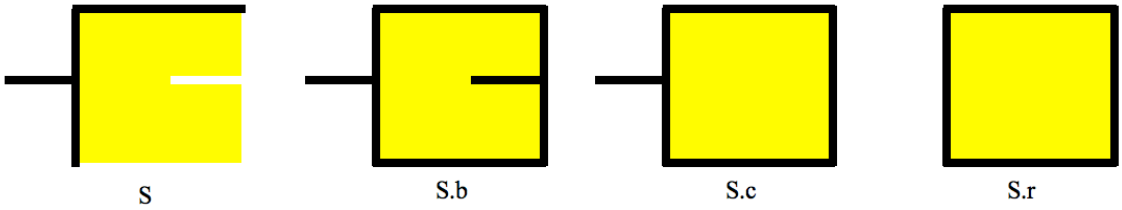
Similarly, the dangle may also be decomposed into two parts: the part in **S** is the ***hair***, **S**.h, of **S** and the part in !**S** is the ***cut***, **S**.c. Note that the hair is only touching **S**.e and that the cut is only touching **S**.i. Note that the skin, hair, wound, cut, interior and exterior of a set are exclusive (i.e. pairwise disjoint).



The collection {**S**.i, **S**.c, **S**.s, **S**.w, **S**.h, **S**.e} is exclusive and exhaustive. The membrane **S**.m= **S**.w∪**S**.s separates **S**.i from **S**.e. Often, the dangle **S**.d=**S**.c∪**S**.h is removed through a regularization process discussed below.

The ***closure***: **S**.k=**S**∪**S**.b is the union of **S** with its boundary. Note that **S**.k is in fact the union of **S** with its wound and cut, since the other parts (hair and skin) of **S**.b are already in **S**. Note that !(**S**.k)=(!**S**).i. Also note that the boundary is the difference **S**.b=**S**.k**S**.i between the closure and the interior.

Many applications are restricted to work on regularized sets. We define two kind of regularizations. The ***closed-regularization*** **S**.r of **S** is the closure **S**.i.k. of its interior **S**.i. To obtain **S**.r, we trim (subtract) the hair and heal (add) the wound and cut. We say that **S** is ***closed-regularized*** when **S**=**S**.r. Because Boolean operations on closed-regularized sets may produce sets that are not closed-regularized, solid modeling systems perform a post-processing closed-regularization step after each Boolean operation. Note that the same result may also be obtained by performing the regularization after several non-regularized Boolean operations.



We also define the ***open-regularization*** **S**.o=**S**.k.i of a set **S**. **S** is ***open-regularized*** when **S**=**S**.o. Note that an open-regularized set does not have any dangle or skin. It is an open set with no cut. It also does not contain its boundary. Note that **S**.o=**S**.r.i, **S**.r=**S**.o.k, and **S**.o.b=**S**.r.b. As a consequence, when sets are represented by their boundary, both **S**.o and **S**.r will have the same representation, except for a Boolean flag or implicit assumption indicating whether we assume closed or open regularization. Consequently, from an implementation perspective, the algorithms that perform open- or closed-regularization are identical. The theoretical distinction between closed- and open-regularization is nevertheless important when specifying properties of mechanical assemblies and of geometric complexes described below.

2 - Complexes

So far, we have discussed properties of sets and operators on sets. But we have not discussed how these sets can be represented in a computer. Furthermore, we may be interested in representing not only a set, but also its decomposition (segmentation) into meaningful parts for a given application. To address these challenges, we will use geometric complexes.

2.1 Cells and geometric complex

A ***geometric complex*** is a decomposition of Ω into an exclusive and exhaustive collection K of ***k-cells***, satisfying a specific condition, discussed below. The symbol k stands for 0, 1, 2, or 3. For example, the 3D space may be decomposed into vertices, edges, faces, and volumes. Given such a decomposition, subsets of these cells may be grouped into features of interest.

The ***0-cells***, also called ***vertices*****,** are isolated points.

The ***1-cells***, also called ***edges***, are connected curves or curve segments that **do not include their endpoints**. A line, a circle, a ray (semi-infinite line segment), a circle with one point removed, a line segment without its end-points are examples of edges.

In 2D, (i.e., when the complex is planar) the ***2-cells***, also called ***faces***, are the components of the difference between the plane Ω and the union of all the 0-cells and 1-cells. In 3D, the faces are connected components of some surface E. Note that ***faces are open and connected, but need not be simply connected or bounded, and may have cuts.***

In 3D, the ***3-cells***, also called ***volumes***, are the components of the difference between Ω and the union of all the 0-cells, 1-cells and 2-cells. Volumes are open and connected but not necessarily open-regularized.

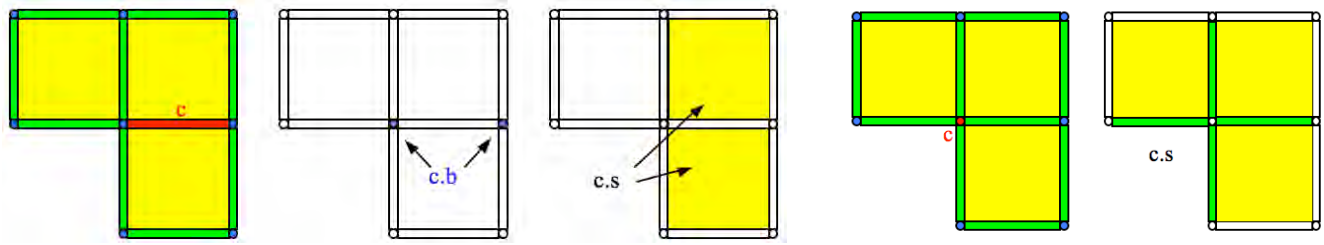
The specific ***condition*** imposed upon the cells of the complex requires that **the boundary of any cell c of K be the union of cells of K**. For example, each endpoint of an edge of K must be a 0-cell and not a member of another edge (which would create what is often called a T-junction).

A cell c of a complex K is an abstract member of a collection of cells. We define several operators on cells that are implicitly defined with respect to a given complex K. The ***set*** of cell c is denoted c.p. Its ***dimension*** (i.e., 0, 1, 2, or 3) is denoted c.d. The ***boundary*** of c is denoted c.b. Note that c.b is not c.p.b, because c.b is a collection of cells, not a point set, while c.p is a point set and hence c.p.b, its boundary, is also a point set. Instead c.b is a collection of cells b of K, such that b.p⊂c.b.p. The ***star*** c.s of c is the collection of cells f such that c∈f.b. In other words, the star of a cell c is the collection of cells bounded by c. One may think of star and boundary as being opposite operators, but of course one is not, in general the inverse of the other. Can you provide an example of such a situation? We say that each f in c.s is ***incident upon*** c. For example, a face is incident upon its bounding edges and vertices. Two cells c and d are **adjacent** when c.b and d.b interfere. For example, two faces are adjacent if they share a common bounding edge or vertex. To distinguish these two cases, we can say that two faces are edge-adjacent if they share a common bounding edge or only vertex adjacent if they do not share a common bounding edge but do share a common bounding vertex.

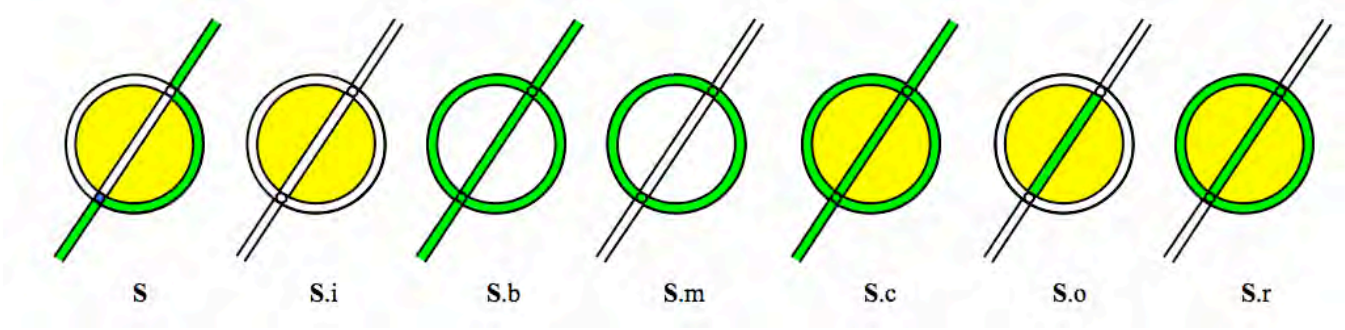
2.2 Selective geometric complex (SGC)

A ***selective geometric complex*** (abbreviated ***SGC***), G, is obtained from a geometric complex K by associating a Boolean ***selection label*** c.m with each cell c of K. For short, we say that a cell c is ***filled*** (i.e., ***in*** G) if c.m=true and is ***empty*** (i.e., ***out of*** G) otherwise. The set G.p is the union of c.p for all filled cells c. When c.m is a Boolean defining the ***in*** or ***out*** status of a cell c, the SGC can only be used to define a single point set. However, a simple extension may assign an integer (feature ID) c.m to each cell and hence partition the space into distinguished features. Finally, one may consider c.m to be an array of such feature IDs, one per layer, this defining several independent segmentation of space. New segmentations may be derived from existing ones through Boolean and Topological operators. Such a model is called a ***Selective Topological Complex*** (abbreviated STC). Each layer of an STC corresponds to a different SGC, but the SGCs of all the layers of an STC have the same underlying geometric complex.

We can now define operators on the cells of an SGC or on the whole SGC. In our illustrations, we use small darker (blue) disks to depict vertices, thin (green) rectangles to depict edges, and large lighter (yellow) areas to depict faces. Specific cells will be indicated in red. We render the cells filled when the cell is ***in*** and unfilled otherwise. On the 3 left images, below, we show first an SGC with a particular 1-cell c indicated in red. Next we show its boundary c.b (two vertices) and its star, c.s (two faces). In the two right images, we show first a 0-cell c in red and then its star (4 edges and 3 faces filled, plus the unbounded empty face).



Furthermore, we can define the SGC counterparts of the topological operators discussed above. Let G be the planar SGC shown on the left below. Note that its complex has 2-vertices, 5 edges, and 3 faces. Two faces and two edges are unbounded. The SGC has only one vertex, 3 edges, and the two bounded faces filled (shown shaded on the left). Let **S**=G.p be the corresponding set (i.e. the union of the filled cells). The subsequent figures, from left to right, show the filled cells of SGCs whose sets are in the interior, boundary, membrane, closure, open-regularization, and closed regularization of **S**.

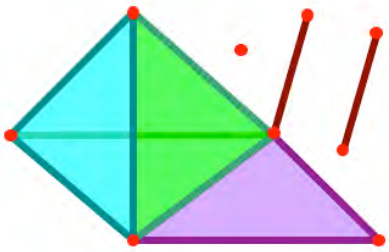


Note that the ability to represent the individual cells of a complex and their in/out status with respect to an SGC will give us the opportunity to represent a variety of regularized and non-regularized sets and to perform a variety of operations on such sets or on the underlying SGC models.

2.3 Simplicial complexes

When the corresponding set is closed, such an SGC is called a ***simplicial complex***. A simplicial complex is a restricted version of a geometric complex. Its domain is confined by two important restrictions: (1) the cells are vertices, edges, triangles, and tetrahedra; (2) all bounded cells are ***in*** the SGC.

Because the set is closed, the cells that bound a filled cell are also filled. Because the non-zero-dimensional cells are limited to be tetrahedra, triangles, and line-segments, they are open-regularized and convex. Note that a simplicial complex cannot be used to represent non-closed sets, but it may be used to represent sets with hair, but no cut or wound. Hence, a 3D simplicial complex is the union of a closed regularized set with a dangle that may comprise triangle faces, edges, and vertices that do not bound any tetrahedron (as shown below).



Two approaches are common when designing a representation scheme for models of simplicial complexes. One is explicit (each cell is represented independently of the others)—the other implicit (some cells are defined by their bounding lower-dimensional cells, and so recursively).

The ***explicit*** approach has an independent, explicit representation for each cell. The empty cells of G are not represented explicitly. For example, a triangle may be represented by its 3 bounding vertices. Hence, in an explicit representation, the coordinate of a vertex will be replicated in the explicit representation of each cell (edge, triangle, tetrahedron) in its star.

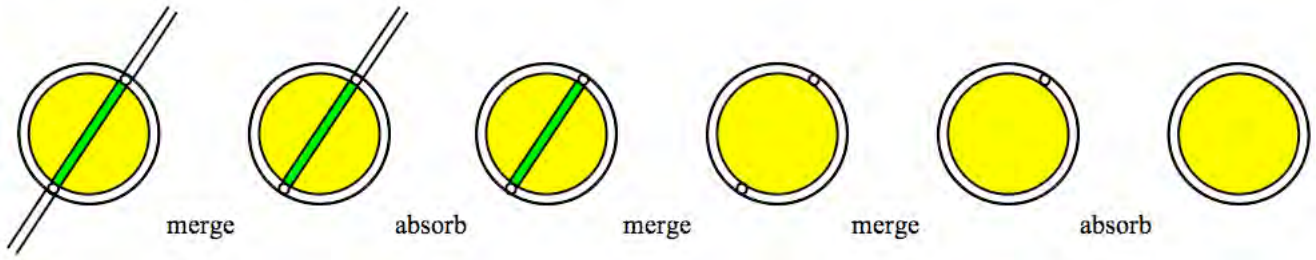
The ***implicit*** representation avoids this replications and leads to not only storage savings, but also faster processing. In an implicit representation of a simplicial complex, each k-cell is represented by a list of k+1 vertex ID (identifiers), which all reference a vertex. For example, a 2-cell triangle face will reference 3 vertices. The vertices are typically stored in a table containing their coordinates. A variety of data structures have been proposed for efficiently supporting the boundary and star operators on cells of such implicit representations and for ordering cells around others.

2.4 Topological simplification

Consider the 2D SGC, G, (shown on the left below). It has 11 cells (two are unbounded)**.** Yet, the point set G.p it represents is a simple open disk (shown on the right), which has a simpler SGC representation comprising only 3 cells: a filled open disk (G.p.i), its empty bounding circle (G.p.b), and the empty exterior unbounded cell (G.p.e). This simplified SGC may be obtained through the application of a series of merge and absorb operations on SGCs, as shown below.

Merge combines a k-cell with two of its incident (k+1)-cells, provided that they have the same filled/empty classification and that the k-cell is not bounding any other cell that is not also bounded by the two (k+1)-cells. In our example, the first merge combines the two empty unbounded faces with one of their empty bounding edges into a single empty unbounded face. The second merge combines the two filled faces with their common filled edge into a single face. The third merge combines the two empty half-circle edges with one of their empty bounding vertices into a single empty circular edge with one bounding vertex. Note that the third merge was impossible before the first two merges were completed, because the merged vertex was bounding other cells.

Absorb combines a k-cell with the only (k+1)-cell that it bounds, provided that they have the same filled/empty classification and that the k-cell is not bounding any other cell of dimension k or higher that is not also bounded by the (k+1)-cell. In our example, the first absorb combines an empty unbounded edge with the unbounded face of which it is the cut in the unbounded non-filled face. The second absorb combines the last vertex with the circular edge it bounds.



When the edges are restricted to **bounded line segments**, we say that the SGC is **linear**. Note that a linear SGC has a single unbounded cell that is the **outer face**, because no edge is infinite. We will often refer to linear SGCs when discussing shapes in the planes, their representations, and operators on one or two such shapes. A linear SGC may be easily represented by the location of its vertices and by a **connectivity graph** (studied in a separate chapter), which defines the boundary and star relations on cells and optionally ordering information, such as the order of the cells of c.s around c. We discuss below several examples of such linear SGCs.

3 - Triangle meshes

When a 3D simplicial complex represents a closed-regularized set, it is a ***tetrahedral mesh***. Such meshes are often used for finite element analysis.

In many applications, we are not interested in representing the internal structure (decomposition) of the interior a point set. In fact, we either are only interested in representing a two-dimensional complex (sets of surfaces) or a solid (regularized 3-cell), which can be implicitly represented by its boundary. In both cases, we may use a two-dimensional simplicial complex (no 3-cells).

3.1 Shells

When the simplicial complex has no tetrahedral 3-cells and when each edge and each vertex is bounding a face, the SGC is a ***triangle mesh***, which is the most common representation of shapes in computer graphics. Note that the boundary of a tetrahedral mesh is a triangle mesh.

A triangle mesh M is ***edge-manifold*** when each edge is bounding exactly two triangles. A vertex **p** of an edge-manifold triangle mesh M is ***manifold*** when $(M.p \cap N(\mathbf{p})) \backslash \mathbf{p}$ is connected and ***non-manifold*** otherwise. A triangle mesh is ***manifold*** when it is edge-manifold and when all its vertices are manifold.

The components of a manifold triangle mesh are called ***shells***. The ***genus***, i.e., number of handles, h of a shell with v vertices and t triangles is $h=(t-2v+4)/4$.

A shell is defined by its geometry and its connectivity. The **geometry** is typically stored in a table of vertices, each represented by the coordinates of the vertices. The **incidence** is the association of each triangle with its three bounding vertices, which is sometimes stored as three integer vertex indices per triangle.

3.2 Simple meshes

A shell M of genus zero is a ***planar triangle graph***, which means that one may find location for its vertices so that the collection of vertices, edges, and faces is a 2D complex with triangle faces, one of which is unbounded. For

simplicity, we will use the term **simple mesh** when referring to zero genus shells. A simple shell with v vertices has $t=2v-4$ triangles and $e=3t/2=3v-2$ edges.

Edge-collapse simplification steps, which are used to reduce the vertex count in a triangle mesh may produce non-manifold vertices, triangle meshes which are no longer edge-manifold, and even simplicial complexes that are no longer triangle meshes but have dangling edges and vertices that do not bound a triangle. Hence, the designers of such simplification algorithms must make the choice to either prevent such simplification steps, which reduces the potential benefits of simplification, or to support more general simplicial complexes.

3.3 Hamiltonian circuits and a naïve encoding of simple meshes

A ***Hamiltonian circuit*** H of a simple shell M is a subset of the edges of M chosen so that each vertex of M has exactly two incident edges in the circuit and so that the union of the vertices of M with the edges of the circuit is a **polyloop**, i.e. a collection of exclusive line-segment edges and vertices that form a connected set and so that each vertex has exactly two incident edges.

Note that H has v edges. We will call then the ***cut-edges***. Hence there are $2v-2$ edges of M not in H. We will call them ***hinge-edges***. The difference between M and the union of its vertices and cut-edges has two components, M_1 and M_2, each formed by $t/2=v-2$ triangles and $v-1$ hinge-edges.

Consider listing the vertices in the order in which they appear along H. The first edge E_1 of H is bounding a triangle T_1 of M_1 and a triangle T_2 of M_2. Walk from T_1 to all the other triangles of M_1 by a recursive traversal that crosses a hinge-edge to move from one triangle to an adjacent one. As you do so, for each hinge-edge you cross, append a symbol from the set {L,R,S,E} to the *lers* string. To choose the correct symbol, assume that you entered a triangle T through an edge E, you are facing two other edges, E_L and E_R of T. If only E_L is a hinge-edge, append symbol L (for ***left***) to *lers* and cross E_L to continue the traversal. If only E_R is a hinge-edge, append symbol R (for ***right***) and cross E_R to continue the traversal. If both E_L and E_R are hinge-edges, append symbol S (for ***split***), issue a recursive call to initiate a traversal starting after crossing E_R. Then, when the recursive call returns, cross E_L to continue the traversal. If both are cut-edges, append symbol E (for ***end***) and return from the (recursive) call.

The sequence of symbols is sufficient to recover the connectivity of M_1. To do so, start with T_1 and extend it by attaching triangles to hinge-edges, which are distinguished from cut-edges by the consecutive *lers* symbols. The border of the resulting mesh subset is the Hamiltonian circuit H. Note that we can build a similar list of *lers* symbols for M_2. Since we have 4 possible symbols, each symbol can be encoded using 2 bits. Therefore, the connectivity of M can be transmitted in compressed format by transmitting the symbols for the two *lers* streams and then the vertices in the order in which they appear along H. Note that we do not need to transmit the number of triangles or vertices, because the end of the first *lers* stream may be detected when the number of encountered E symbols exceed the number of encountered S symbols.

Note that with this approach, the connectivity of a simple shell may be encoded using less than $2t$ bits. Unfortunately, a Hamiltonian circuit may not exist or, if it does, may be expensive to discover.

Consider a **dual graph** where each node corresponds to a different triangle of M and where each link corresponds to a hinge edge. This graph has two components. each one is a ***binary tree***. One has as root the node corresponding to T_1, the other one has as root the node corresponding to T_2. If we add a link between T_1 and T_2, we obtain a connected graph. Picking a leaf of T_1 as root turns this graph into a binary tree, which we call a ***triangle-spanning tree*** (***TST***) of M.

The ***Topological Surgery*** compression scheme, which is the core of the MPEG-4 standard for geometry compression, builds a TST with the same approach as the one proposed in the naive scheme described above. The cut-edges defined by this TST are the edges that do not correspond to links between adjacent triangles in the TST. These cut-edges and the vertices of M form a ***vertex-spanning tree*** (***VST***), where the nodes correspond to different vertices of M and the links to different cut-edges. The VST is not a Hamiltonian cycle and needs to be encoded along with the ordering of the cut-edges around each vertex. It may be encoded using 2 bits per vertex or equivalently one bit per triangle. The connectivity may be derived through a simultaneous walk around the TST and the VST. Hence, the Topological Surgery approach guarantees to encode the connectivity with $3t$ bits ($2t$ for the TST and $1t$ for the VST).

The ***Edgebreaker*** compression scheme builds the same TST as the Topological Surgery. But instead of using the 4 {L,E,R,S} symbols, it represents it with the 5 {C,L,E,R,S} symbols, using a special C symbol for R situation

where the tip vertex is encountered for the first time. In spite of having to encode strings made of 5 possible symbols, Edgebreaker also guarantees a $2t$ bits encoding of the connectivity of a simple shell. In fact, a more elaborate binary encoding of these symbols guarantees $1.80t$ bits.

These schemes, their improvements, their extensions to more general SGCs, and a number of important variations will be discussed in detail in subsequent chapters.

4 - Polygons

4.1 Boundary representation of a cell

Tetrahedra meshes and triangle meshes were examples of enumerative schemes, because their filled cells were simplices (vertices, edges, triangles, or tetrahedra). Remember that the empty cells of the corresponding SGCs had no explicit representation. Instead they were implicitly defined as the connected components of the complement of the union of the sets of the filled cells. In particular, each finite empty cell may be represented by an SGC that defines its boundary. Note that such a cell **S** needs not be open-regularized. Also note that the boundary may be disconnected. Furthermore, it may have a cut (portion that does not separate **S** from **S**.e). We say that the SGC G of **S**.b is a **boundary representation** of **S**. Note that **S** is represented implicitly by its boundary. It is distinguished from **S**.e because it is the bounded component of the complement of S.b.

When G is a simplicial complex it may be represented by a set of vertices and by the **boundary graph** defining the triangle/vertex and edge/vertex incidence.

4.2 Boundary representation of a solid

When the cell **S** is open-regular, we say that it is a **solid** (or in fact a **polyhedron**). More precisely, most solid modeling systems use the term solid to refer to the closure S.k. But remember that the boundary representation of **S**.i and **S**.k are identical when **S** is regularized.

Because **S** is a 3-cell, it is connected. However, it needs not be manifold. Furthermore, its boundary may have several connected components.

Consider a coplanar collection of cells of the SGC G of **S**.b. We can simplify it by applying the merge and absorb operations described above, as long as they preserve the validity of the SGC. The simplification process will may combine several triangles, edge, and vertices into a single face. Note however that the face may have a cut, i.e., it is not open-regular in its supporting plane.

If we perform such a simplification for all sets of coplanar cells, we obtain a **polygonal boundary representation** of S. But how are these faces represented? Most 3D modeling systems treat them as polygons and use polyloops to represent them. We discuss below the limitations of such a choice and propose solutions.

4.3 Faces and polygons

Two related concepts are used extensively in solid modeling and computer graphics literature: face and polygon. The precise definitions of these concepts are rarely provided and the available ones are sometimes incomplete and certainly not consistent with one another.

A **face** was defined above as a connected open subset of a plane. Consequently, a face **F** equals its interior **F**.i and does not contain any of its boundary. Although in general the boundary of a planar face may contain curved edges and in fact a face needs not be planar, but could be the subset of a curved surface, we will focus on planar faces with piecewise linear boundaries.

A subset **S** of the plane is a **polygon** when it verifies the following conditions:

1) **S** is not empty.
2) **S** is connected.
3) **S** is bounded.
4) **S** is open-regularized.
5) The boundary **S**.b of **S** is a subset of a finite union of lines.

The main topological difference between faces and polygons is that polygons are bounded and open-regularized.

For example, the open-regularization of a face (2-cell) of a geometric complex is a polygon. Note that although polygons are connected, they need not be convex, simply connected, or manifold. However, they are open (hence

have no hair or skin) and regularized (hence have no cuts). Polygons are bounded. Consequently, their bounding edges are bounded line-segments.

Note that, some authors prefer to define polygons as closed-regularized. The choice is a matter of elegance in the formulation of some of the properties. Both open-regularized and closed-regularized polygons may be represented in the same manner, for example by representing their boundary.

4.4 Vertices and edges of a polygon

Every polygon **S** defines a **unique canonical decomposition** of the plane into a **linear SGC**, written SGC(S). SGC(S) has two 2-cells: f, such that f.p=S=S.i (which is connected) and g, such that g.p=S.e (which is the outer face of the complex). We also decompose the boundary S.b of S into edges and vertices. To fully specify them, we say that point P of S.b is an **edge-point** if the intersection S.b∩N(P) of the boundary of S with the neighborhood of P is an edge (line-segment without its endpoints). The **edges** (1-cells) of SGC(S) are the components of the union E of edge-points of S. The **vertices** (0-cells) are the components of S.b\E. Of course, the label of f is set to true and the labels of all edges and vertices and the label of the external face g are set to false.

A polygon may be unambiguously represented by its boundary. Indeed, given the boundary B of a polygon S, we can recover S as the bounded component of !(S.b).

We say that a polygon is **simple**, when it is manifold and simply connected. Note that the boundary of a simple polygon is connected.

4.5 Boolean and regularized Boolean operations on polygons

Consider two polygons, A and B. Remember that a polygon is open-regularized, and hence open, and that it is connected. A∩B and A∪B are open sets. However, they need not be polygons. A∩B is always open-regularized. A∪B need not be. For example, consider two square polygons that are adjacent in a checkerboard. Their union is an open set with a cut through it. Hence it is not open-regularized. When A∩B≠∅, A\B and A⊗B are not open and hence not open-regularized.

Let S be the set defined by a Boolean combination of polygon primitives. S needs not be open-regularized. However, one may be interested in ensuring that Boolean combinations of polygons produce polygons. Therefore, we define the **polygons of S** as the components of S.o. We say that S.o is the **open-regularized Boolean combination** of polygon primitives.

Solid modeling systems support closed-regularized Boolean combinations of three-dimensional closed-regularized primitives (blocks, spheres, cylinders…). Such a representation of 3D shapes is called **Constructive Solid Geometry** (abbreviated **CSG**). Using polygons in the plane instead of solid primitives, one can define CSG combinations of closures of polygons. Consider the set S defined by a Boolean combination of open polygons. Now consider the set T defined by the same Boolean combination but on the closure of these same polygons. The CSG expression T.r may be easily obtained from our open-regularized Boolean combination S.o as T.r=S.o.b.

4.6 Contacts between polygons

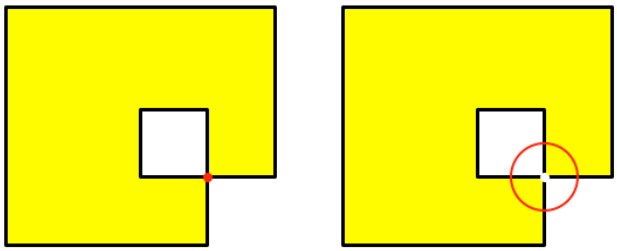
Two polygons, A and B, that do not interfere (i.e., A∩B=∅) are **in contact** when A.b∩B.b≠∅. The **contact region** between two non-interfering polygons A and B is A.b∩B.b. Note that the contact region is closed and non-regularized.

Interference tests and the computation of contact regions are essential for the analysis of assemblies and for the proper treatment of collisions and frictions in physically–based animations. Specifically, the parts in a mechanical assembly need not be exclusive. In fact, often pairs of them are in contact. However, no two of them can interfere.

Consider the **open-regularized Boolean combination** S.o of polygon primitives. Its connected components, which are the polygons of S are exclusive. However, their closures are not. Let A and B be two polygons of S. Their contacts region A.b∩B.b is either empty or contains one or more isolated vertices. Note that the contact region cannot contain an edge, because, if it did, A and B would have been merged into a single component of S.

4.7 Non-manifold singularities

Many data-structures and algorithms for 2D and 3D shape processing assume that the shapes are manifold. A point **p**∈**S**.b is **manifold** when **S**∩N(**p**) is homeomorphic to a half-ball. P is said to be a **non-manifold** point otherwise. A set S is **manifold** if all points in S.b are manifold. Some early applications assumed that the sets they manipulate are manifold, because this assumption made representation and processing easier.



5 - Polyloops

5.1 Polyloop and point-membership classification

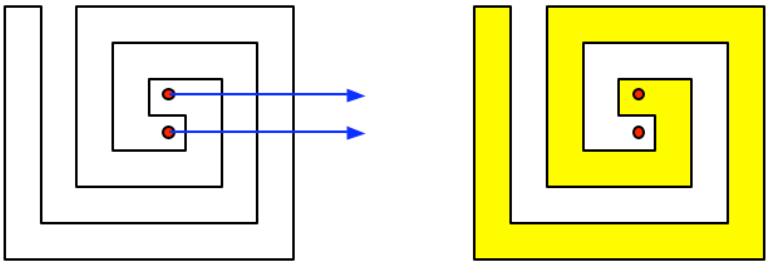
One may represent the boundary S.b of a **simple polygon** S by a **cyclically ordered collection of vertices**. For example, assume that the vertices are stored in an array V[vc] of vc vertices with coordinates (V[i].x,V[i].y). Let n(i) be a function that returns (i+1)%vc. The **edges** of S.b are not represented explicitly. They are defined as the line segments E[i]=Edge[V[i],V[n(i)]. Such a collection of edges and vertices is called a **polyloop**.

A polyloop is **clean** (i.e. **not self-intersecting**) when its edges and vertices are exclusive (i.e., pairwise disjoint). The polyloop representation of the boundary of a simple polygon is clean.

One often needs to test whether a given point P lies in a polygon S or not. To be more precise, we want to classify P against S. The **point-membership-classification**, PMC(P,S), of point P with respect to a polygon S returns IN when P∈S.i, ON when P∈S.b, and OUT P∈S.e. We can test whether P∈S.b by checking whether P coincides with a vertex of S or lines inside an edge of S. These test reduce to simple geometric tests on point and vectors, although exact answers may require the use of extended rational arithmetic.

Assume now that P∉S.b. We want to decide between P∈S.b and P∈S.e. To do so, we shoot (create) a ray R (semi-infinite line segment) from P to infinity in a direction chosen to ensure that R does not intersect any of the vertices of S. (In practice, the direction of R may be selected randomly.) If we discover that R hits a vertex of S, we restart with another random direction. One rarely needs more than one attempt.) We then count the **parity** p of the number of edges that are intersecting R. If p is odd, then P∈S.i and we return IN. Otherwise, we return OUT.

To justify this approach, consider the fact that S if bounded, hence, the outer face is not in S. Now place a point Q at infinity along R. Q is initially OUT of S. As you continuously slide Q towards P. Its status will toggle between OUT and IN each time you cross S.b. Note that the parity of the number of edges of the polyloop that intersect R may be tested by considering the edges in any order.

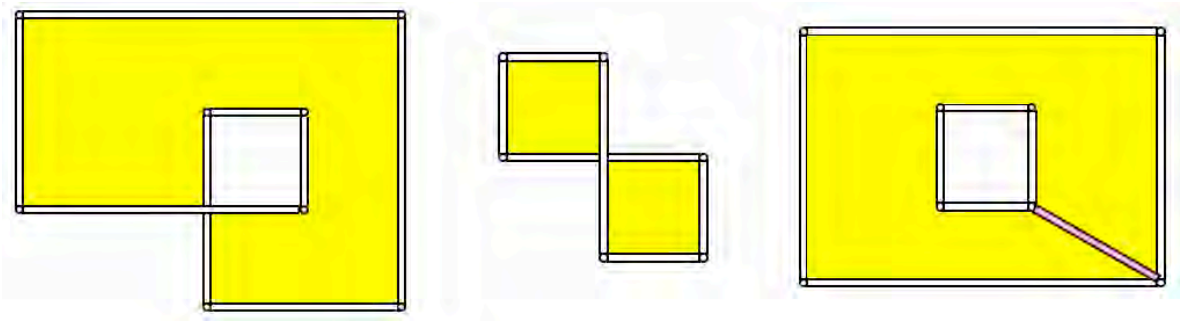


So far, we have shown that a simple polygon may be represented by a polyloop and that this polyloop may be used for PMC. Now we would like to extend these observations in two ways: (1) to polygons that are not simple and (2) to polyloops that are not clean. In fact, we will achieve both extensions simultaneously.

We define the **interior** L.i **of a polyloop** L as the set of points that are not on any edge or vertex of L and for which the PMC algorithm described above returns IN. Note that, even though we have expressed L.i in terms of an algorithm, this definition is mathematically precise and independent of its implementation. We then define the **boundary** L.b of a polyloop L as the boundary L.i.k.b of the closure L.i.k of L.i. Finally, we define the **exterior** L.e as the remaining part !(L.i∪L.b) of the plane.

Armed with this extension, we can represent a non-manifold polygon by a polyloop that is self-intersecting (below, left). The polygon here is L.i. Note that L.i may be disconnected (below, center), in which case, L.i is not a polygon, but its components are.

Finally, we can use a single polyloop to represent polygons with holes (below, right). It suffices to create pairs of **bridge-edges** (in pink below) that join the polyloops that bound the outer face and the holes into a single polyloop. The cyclic order of the vertices around the polyloop may be obtained by considering the edges as streets and the vertices are street-intersections. Consider that each street has a **sidewalk** on each side. Start on a sidewalk of a bridge-edge. Orient yourself so that the street is on your left. Walk from one sidewalk to the next, never crossing a street. At each intersection, you will have to take the right-most sidewalk. Each time you reach an intersection, append the vertex ID of the intersection to the list. You are done when you are back to your starting point. Note that some vertices appear more than once. Note that the addition of the bridge-edge pairs does not alter the PMC algorithm nor the result.

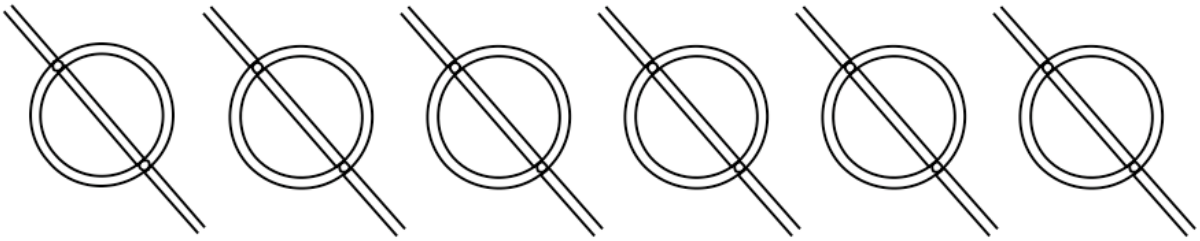


Practice

Exercises

1) Use predicates and quantifiers to express the condition for two sets, A and B to be disjoint.
2) Let S=Disk(P,r)\P. Formulate S using predicates and logical operators.
3) Use predicates and quantifiers to express A⊗B.
4) Prove that A⊗B = (A∪B)\(A∩B), A∩B ⊂ A∪B, and A∩(B∪C) = (A∩B)∪(A∩C).
5) The line L=LineImplicit(Q,N) separates the plane Ω in 3 sets that are exhaustive and exclusive. Provide implicit equalities/inequalities that define each set. Give each set s meaningful name.
6) Use Boolean operators to formulate the conditions for 3 sets, A, B, and C, to be exclusive and the conditions that no point P belongs to all three sets.
7) Illustrate the De Morgan law !(A∪B)=!A∩!B in successive drawings that show the boundaries of A and B and shade: A∪B, !(A∪B), !A, !B, and !A∩!B. Then prove that !(A∪B)=!A∩!B.
8) Explain each step of the conversion of the Boolean expression (A\B)\(C\D) into its positive form (A∩!B)∩(!C∪D). Justify each step by pointing out which equivalence relation is used.
9) Convert ((A\B)∪C)\((D\E)∩(G\H)) into positive form. Note which primitives appear complemented in the positive form. These are called the negative primitives. Provide a simple recipe for deciding whether a particular primitive is negative directly on the original expression, without first converting it to positive form.
10) Convert (A\B)\(C\D) into disjunctive form.
11) Provide the pseudo-code of an algorithm for testing whether a point P lies in a Boolean expression in disjunctive form.
12) Draw a set S, such that S is connected, but S.i is not. Draw a set that has two components, only one of which is simply connected.
13) Assume that S.b is connected. can you conclude that S is connected and if so that it is simply connected? Justify your answer.

14) Is the set S (Edge(A,B)∪Edge(B,C)∪Edge(C,A)).k convex?
15) Can a set that is not simply connected be convex? Justify your answer.
16) Can an unbounded set be convex? Justify your answer.
17) Describe and draw a set that has one unbounded and one bounded component.
18) Let S=Disk(P,r)\P. Describe N(P). Is P|S? Explain why.
19) Explain why the intersection of two closed sets is closed and why the intersection of two open sets is open.
20) Let S=Disk(P,r)⊗LineImplicit(P,N). The series of figures, below, show the SGC, G, defined by S (i.e., S=G.p). From left to right, identify (fill in) the cells for SGCs yielding the following pointsets: S, S.e, S.b, S.i, S.k, and S.r. Make sure that you correctly classify the vertices (0-cells).



21) Invent a simple set S whose hair, cut, and wound are each reduced to a single point. Draw its SGC and clearly mark the interior, exterior, skin, wound, cut, and hair.
22) Give an example of an exclusive and exhaustive collection of cells that is not a geometric complex.
23) Consider a manifold set S. Is S.i manifold? Is S.b manifold?
24) Let S={P} be the set made of a single point. Is it manifold?
25) S=Disk(P,r)\P. Is S manifold? Is S.k manifold?
26) When is a set S open-regularized? Provide a formulation in terms of set operators (closure, interior…) and also a formulation using the concepts of (hair, skin….).
27) Show an example where S.o.k≠S.k.
28) Write the 5 conditions satisfied by a polygon. For each condition, draw a simple set that violates it and explain which condition is violated and why.
29) Draw a non-manifold polygon and indicate its canonical decomposition as a linear SGC. How are the edge-points distinguished from the vertices? How are the edges defined?
30) When is a polygon simple? What can you say about its boundary?
31) Explain and justify the PMC algorithm for testing a point against a polygon. What does it return?
32) What is the distinction between a polyloop and the boundary of a polygon?
33) Provide the precise definitions of the L.b, L.i, and L.e for a polyloop L.
34) State precisely when a polyloop is clean. Draw several examples of self-intersecting polyloops illustrating different ways in which the self-intersection may occur. For each one, indicate the resulting polygons.
35) Why is the union of two polygons a closed set? Why is it regularized? Why may it not be a polygon?

Projects

1) Design, implement, and debug an algorithm for testing whether a polyloop is self-intersecting.
2) Design, implement, and debug an algorithm for computing a representation of an arbitrary polygon S as a single polyloop L.
3) Design, implement, debug an algorithm for computing the polygons of L.i for an arbitrary polyloop L.
4) Design, implement, and debug an algorithm for performing regularized Boolean operations on polygons.
5) Design, implement, and debug an algorithm for computing the contacts between polygons.

Problems

1) Provide the pseudo-code for an algorithm that will convert a Boolean expression into a positive form.

2) Consider a Boolean expression of n primitives. Explain when its disjunctive form may have an exponential number of products.
3) Show that S.r=S.o.k and that S.o=S.r.i.
4) Let c be a cell of an SGC, G. Let S=G.p. Use the cell operators, c.b, c.s, and c.m to formulate tests for the following properties: c.p⊂S.i, c.p⊂S.b, c.p⊂S.e, c.p⊂S.m, c.p⊂S.d.
5) Let A and B be manifold polygons. Assume A∩B≠∅. Draw an example where A∩B has 4 components: one is a manifold polygon, one is a non-manifold polygon, the other two would be merged in (A∩B).o.
6) Why did we restrict polygons to be bounded? Why did we restrict them to have connected interior?
7) Assume that a polyloop L has no right-turn. Can you conclude that L.i is convex? If so, provide a proof. If not provide a counter-example and suggest an additional condition on L that would lead to the desired conclusion.