2D Topology for Computer Graphics

Jarek Rossignac

The data structures for representing shapes and the algorithms that operate on them are usually designed with a specific class of shapes in mind. A particular class of interest should not be, and in fact cannot be, defined by specifying a data structure. Instead, it should be defined using precise topological concepts, which we review in this chapter. We start with point sets and with Boolean and topological operators for point sets. Then we discuss how simple geometric primitives (vertices, edges, faces) may be arranged into a Selective Geometric Complex that represents a desired point set. Finally, we use these concepts to define polygons, to propose a representation for them in terms of polyloops, and to discuss algorithms that test points against polygons or compute regularized Boolean combinations or contact regions between polygons.

1 - Point sets

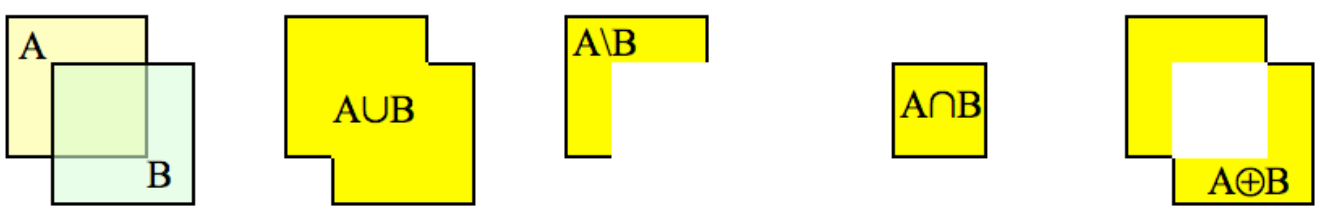
1.1 Boolean operations on point sets

The term **set** will refer to a point set that is a subset of the plane Ω. Let A, B, and S, be such sets. Let P and Q be isolated **points** of 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 **in** S. The notation p∉S indicates otherwise. The **empty** set is denoted by ∅.

Two sets, A and B, are **equal** (written A=B) when (we use the term "when" to mean "if and only if") the following predicate is true: (p∈A ⇔ p∈B). Set A is included in set B (written A⊂B) when (p∈A ⇒ p∈B). Note that this is not a strict inclusion, hence A=B ⇒ A⊂B.

The **complement** !S of S is the set {P∉S} of points not in S. Note that !!S=S. The complement operator ! has highest priority. The operators ∈ and ∉ have the second highest priority. Implications (⇒ and ⇔) have the lowest priority.

To distinguish them from Boolean operators on point sets, we use the following notation for **logical operators**: ∧ for AND, ∨ for OR, and ¬ for NOT. We will need the following set-theoretic Boolean operators: **union** A∪B={P∈A ∨ P∈B}, **intersection** A∩B={P∈A ∧ P∈B}, **difference** A\B={P∈A ∧ P∉B}, and **XOR** A⊕B=(A\B)∪(B\A). We say that A and B are **disjoint** when A∩B=∅ and that they **interfere** when A∩B≠∅.



A **collection** of two or more sets, A, B, C… is **exclusive** (pair-wise disjoint) when for all pairs (A,B) of sets in the collection, A≠B ⇒ A∩B=∅. 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 S∪!S=Ω and S∩!S=∅.

1.2 Boolean expressions

Consider a set-valued **Boolean expression** S(A,B,C…) that combines the **argument** sets (which will be called the **primitives**) A, B, C… using ∪, ∩, ⊕, \, and ! operators.

A Boolean expression may be converted into **positive form** by converting each difference into an intersection (A\B=A∩!B) and then by distributing the complement over union and intersection according to **De Morgan laws** (!(A∪B)=!A∩!B and !(A∩B)=!A∪!B). The positive form has only ∪ and ∩ operators and has as arguments the

original primitives or their complements. For example, S(A,B,C)=(A\B)\(C\D) can be written in positive form as (A∩!B)∩(!C∪D).

A Boolean expression is in **disjunctive form** when it is written as a **union of products**, each product being the intersection of primitives or of their complements. For example, (A∩!B)∪(B∩C) is in disjunctive form. One can easily convert a positive form into a disjunctive form by distributing ∩ over ∪. Indeed A∩(B∪C)= (A∩B)∪(A∩C). Similarly, A∪(B∩C)= (A∪B)∩(A∪C). Unfortunately, the number of products in the disjunctive form of n primitives may grow exponentially with n.

1.3 Properties of point sets

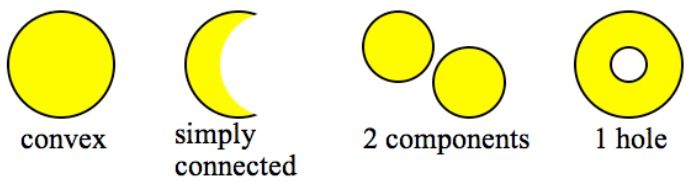
A set S is **connected** if from every point P in S one can walk to every other point Q in S along a curve C that lies in S. More formally, S is connected when ∀P∈S ∀Q∈S, ∃C⊂S: C=curve(P,Q), where curve(P,Q) is a continuous curve connecting P to Q.

A non-empty set S has one or more maximally connected components, which we will call the **components** of S, for simplicity. More precisely, T is a component of S if T⊂S, if T is connected, and if ∀U⊂S, (T\U≠∅)⇒(T not connected). In other words, no connected subset of S that is different from T contains a component T.

S is **bounded** when it is contained in a disk D of finite radius. It is **unbounded** otherwise.

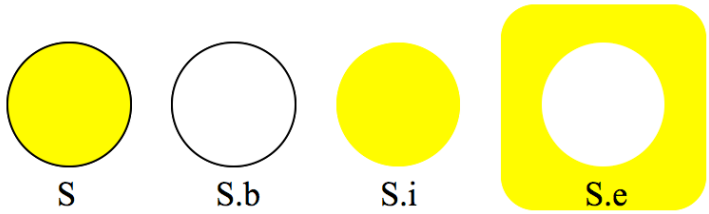
Assume that S is bounded. S is **simply connected** (i.e. has no holes) when !S is connected. When !S is not connected, the bounded components of !S are called the **holes** of S.

S is **convex** when the edge joining any two points in S is also in S. ∀P∈S ∀Q∈S, Edge(P,Q)⊂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.

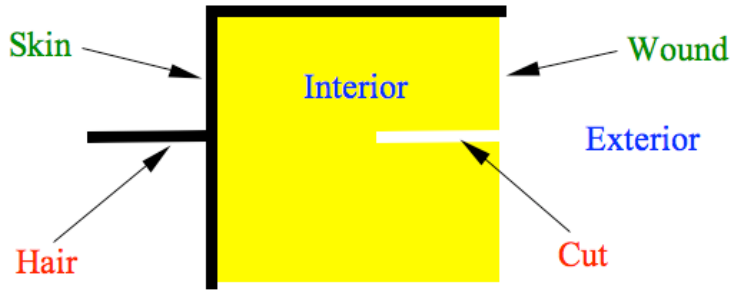


1.4 Operators for point sets

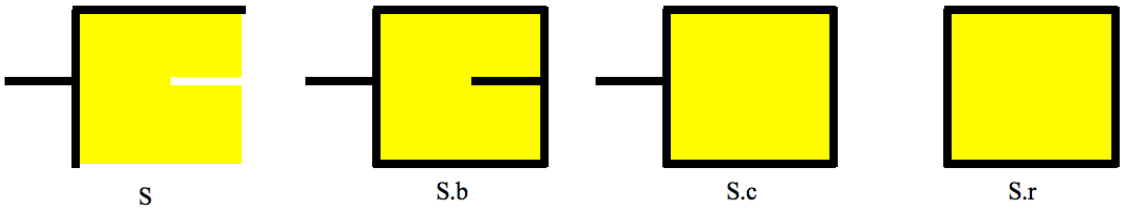
The **neighborhood** N(P) of a point P is an open disk Disk(P,r)={Q:PQ^2<r^2} of center P and radius r, where r is positive, but infinitely small. One could use an r that is not infinitely small, provided that the number of components in N(P)∩S and in N(P)∩!S is not affected by picking any smaller radius. A point P **touches** S (written P|S), if N(P)∩S≠∅. S is **closed** if it contains all points that touch it. The intersection of two closed sets is closed. S is **open** if it contains no point the touches !S. 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. We use (below) black curves to show boundaries that belong to a set and yellow areas to denote their interior.



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 decomposed into two parts (i.e, is the union of two disjoint parts): the part in S is the **skin** of S and the part on !S is the **wound**. Similarly, the dangle may be decomposed into two parts: the part in S is the **hair** of S and the part on !S is the **cut**. 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.

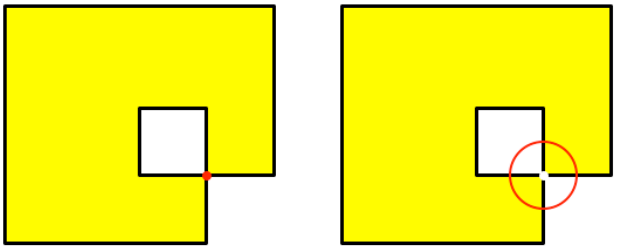


The **closure**: S.c of S is the union S∪S.b of S with its boundary. Note that S.c is in fact the union of S with its wound and cut, since the other parts (hair and skin) of S.b are already in S. The **regularization** S.r of S is the closure S.i.c. of its interior S.i. To obtain S.r, trim (subtract) the hair and heal (union) the wound and cut. A set S is **closed-regularized** when S=S.r. Many applications are restricted to deal with regularized sets.



We also define the **open regularization** S.o=S.c.i of a set S. A set 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.

A point P∈S.b is **manifold** when the difference (S.b∩N(P))\P between the part of S.b in N(P) and P has exactly two components (two edges incident upon P). 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.



2 - Complexes

2.1 Cells and geometric complex

A **geometric complex** is a decomposition of the plane Ω into an exclusive and exhaustive **collection** K of **k-cells**, satisfying a specific condition, discussed below.

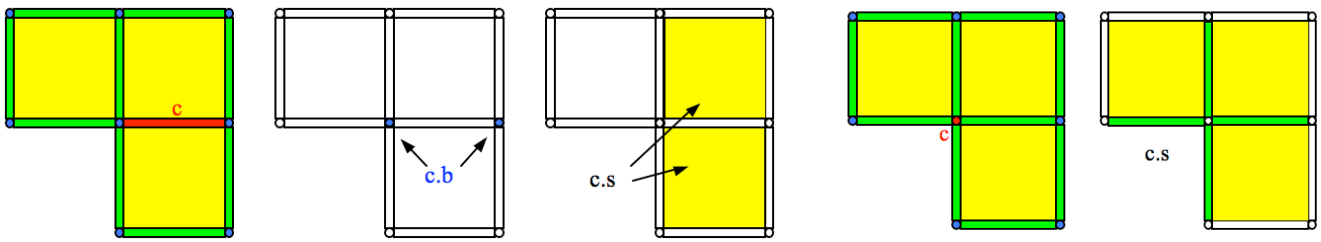
The symbol k stands for 0, 1, or 2. The **0-cells**, also called **vertices,** are isolated points. The **1-cells**, also called **edges**, are connected curves or curve segments that do not include their endpoints. A line, a circle, a ray (semi-infinite line segment), a circle with one point removed, a line segment without its end-points are examples of edges. The **2-cells**, also called **faces**, are the components of the difference between the plane Ω and the union of all the 1-cells and 2-cells. Note that faces are open and connected, but need not be simply connected or bounded, and may have cuts.

The specific **condition** imposed upon the cells of the complex requires that **the boundary of any cell c of K is the union of cells of K**. For example, the endpoint of an line-segment edge of K must be a cell and not a point in another edge.

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 **in** G if c.m=true and is **out of** G otherwise.

A cell c of K is an abstract entity of a collection. We define several operators on cells with respect to a given complex K. The **set** of cell c will be denoted c.p. Its **dimension** (i.e., 0, 1, or 2) will be denoted c.d. The **boundary** of c will be denoted c.b. Note that c.b is not c.p.b, because c.b is a collection, not a set, while c.p is a set and hence c.p.b, its boundary, is also a 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. The set G.p is the union of c.p for all c for which c.m=true. 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 selected and unfilled otherwise.



When the edges are restricted to **bounded line segments**, 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.

3 - Polygon

3.1 Definition

Two related concepts are used extensively in computer graphics: faces and polygons. A **face** was defined above as a connected open subset of the plane. Consequently, a face F equals its interior F.i and does not contain any of its boundary. Note that, in general, the boundary of a face is not necessarily restricted to be the union of edges and isolated points, but can contain curved edges. For example, Disk(P,r)\P is a face.

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 may be decomposed into a finite union of edges and vertices.

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 will be represented in the same manner, for example by representing their boundary.

3.2 Vertices and edges

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 the exterior 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.

3.3 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.

3.4 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 - Polyloop

4.1 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 be 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**.

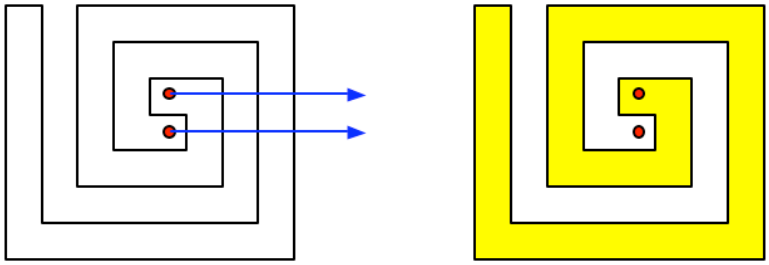
A polyloop is **clean** (i.e. **not self-intersecting**) when its the edges and vertices are exclusive (i.e., pairwise disjoint). The polyloop representation of the boundary of a simple polygon is clean.

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. When 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.

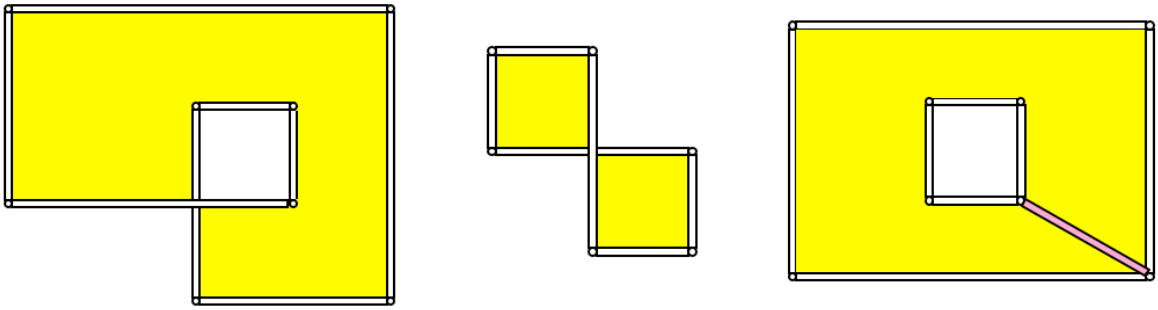


So far, we have shown that a simple polygon may be represented by a polyloop and that this polyloop may be used for PMC. Now we would like to extend these observations in two ways: (1) to polygons that are not simple and (2) to polyloops that are not clean. In fact, we will achieve both extensions simultaneously.

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.c.b of the closure L.i.c 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.

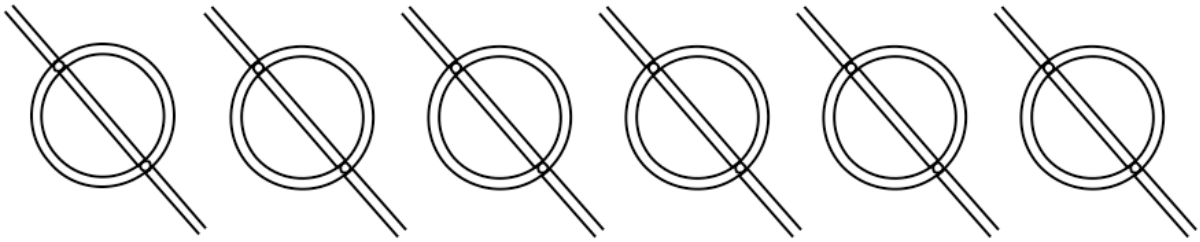
Finally, we can use a single polyloop to represent polygons with holes (below, right). It suffices to create pairs of **bridge-edges** (in pink below) that join the polyloops that bound the outer face and the holes into a single polyloop. The cyclic order of the vertices around the polyloop may be obtained by considering the edges as streets and the vertices are 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 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.



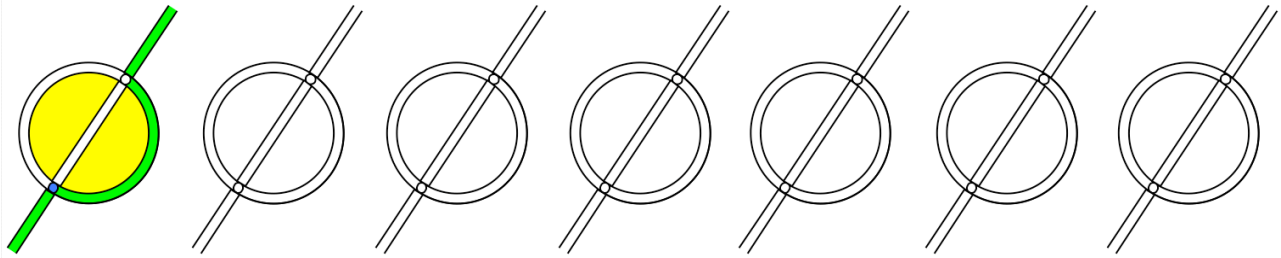
Practice

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)).c 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.c, and S.r. Make sure that you correctly classify the vertices (0-cells).



21) Consider the SGC G, shown below left. Let S=G.p. In the subsequent figures, from left to right, mark (i.e. fill in) the cells of SGCs whose point sets are in the skin, wound, membrane, hair, cut, and dangle of S.



22) 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.
23) Give an example of an exclusive and exhaustive collection of cells that is not a geometric complex.
24) Consider a manifold set S. Is S.i manifold? Is S.b manifold?
25) Let S={P} be the set made of a single point. Is it manifold?
26) S=Disk(P,r)\P. Is S manifold? Is S.c manifold?
27) 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….).
28) Show an example where S.o.c≠S.c.
29) 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.
30) 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?
31) When is a polygon simple? What can you say about its boundary?
32) Explain and justify the PMC algorithm for testing a point against a polygon. What does it return?
33) What is the distinction between a polyloop and the boundary of a polygon?
34) Provide the precise definitions of the L.b, L.i, and L.e for a polyloop L.
35) 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.
36) 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.c 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.