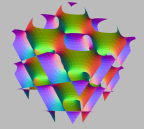
[**What is a solid?**](https://0fps.net/2012/08/29/what-is-a-solid/)

Posted on [August 29, 2012](https://0fps.net/2012/08/29/what-is-a-solid/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/)

In this post, I am going to go into a bit of a mathematical digression about the fundamentals of solid modeling.  In a nutshell, solid modeling is the study of digital representations of physical shapes.  This was a hot topic in the early days of computing and computer aided design, and led to some pretty creative stuff (see [Ivan Sutherland’s SketchPad demo](http://www.youtube.com/watch?v=USyoT_Ha_bA), for example).  Much of the early motivation came from manufacturing and engineering, where the goal was to find better ways to automate and improve design processes.  However, as time went on, the technology became more mature, manufacturing got outsourced out of the west, and interest in this area sort of died off.  But now the times are a-changing.  With the emergence of low capital, flexible manufacturing processes like 3D printing and the growing popularity of video games that make use of solid representations (like Minecraft), now is perhaps a good idea to go back and take a second look at some of these old ideas.

Today, I’ll start with something pretty simple, namely the concept of a **solid**.  We’ll start with a fuzzy definition first, and then work to turn it into something more formal.  The general idea behind a solid is that it is some shape which could conceivably exist physically as a rigid structure.  Most people have a fairly good intuition for what a solid ought to be — even if they can’t quite articulate it precisely.  If I show you some shape, you can probably tell me quickly whether or not it could ever exist in nature:

Left: A solid sphere.  Right:  Some non-solid object.

A solid has a well defined volume and is uniformly filled with the same material everywhere.  That is, it is something you could reasonably imagine building it using a processes like casting, milling, layering, etc.  Infinitesimally thin structures like points, lines or surfaces are not considered solid since they don’t have enough `thickness’ to them.  At the same time, a solid should not have any infinitesimal cracks or holes.  Formulating all of this mathematically sounds like a tall order, but fortunately we don’t have to do it from scratch.  [Requicha solved this problem](http://www-bcf.usc.edu/~requicha/PAL/papers/PAPtm28.pdf) back in the 1970s with the following definition:

A *solid* is a closed regular subset of R^n.

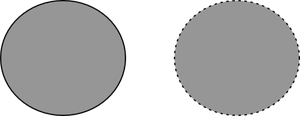
If you’ve never heard of these words before, allow me to explain:

Crash Course in Topology

To understand what Requicha’s definition means, we need to understand a bit about the standard topology of R^n (that is n-dimensional Euclidean space).  We’ll take a pretty naive approach to the subject, since developing topology properly in full abstraction would get too distracting.  In this simplified picture, we’re going to talk about two kinds of interesting sets:

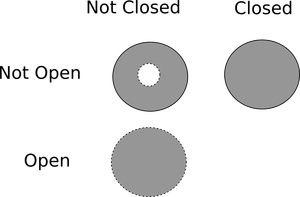
* [*Closed sets*](http://en.wikipedia.org/wiki/Closed_set): That is, subsets of R^n which contain all their limit points.
* [*Open sets*](http://en.wikipedia.org/wiki/Open_set):  That is, sets which are the complement of closed sets in R^n.

The intuition here is that a closed set has a `skin’, while an open set does not:



Left: A closed set,        Right: An open set

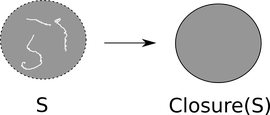
**Beware!**  It is easy to fall into the beginner’s trap of believing that a set must be either open or closed.  In fact, these two properties are completely independent from one another; a set can be open, closed, both or even none-of-the above.  For example, both R^n and the empty set are examples of [clopen sets](http://en.wikipedia.org/wiki/Clopen_set) (that is they are closed and open).  To help explain some of the differences, I made a table with some pictures:



(The set in the lower right corner is the empty set of course.)  Now here is a neat trick:  given any set, we can always turn it into an open set by peeling off any crusty limit points stuck around surface:



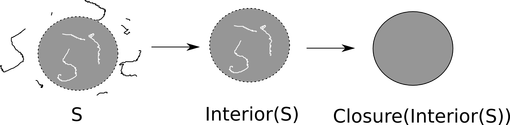
The technical name for decrustified part of S is the [*interior*](http://en.wikipedia.org/wiki/Interior_(topology))*,*written as \iota(S).  Formally, it is defined to be the largest open set contained in S.  Dual to decrustification, there is another important operation called the [*closure*](http://en.wikipedia.org/wiki/Closure_(topology))(written \kappa(S)), and the definition is basically what you think it is: \kappa(S) is the smallest closed set that contains S.  Taking the closure heals any infinitesimal cracks and regrows all the missing skin around the object:



Now that we’ve got these two concepts, we can define a *regular*set.  Briefly a set S is closed regular if:

S = \kappa(\iota(S))

This definition is a bit strange, but there is a procedural way to understand what is going on here.  Looking at the right hand side of the equation, what we are doing is combining the closure and interior operators sequentially.  But this is the same thing as removing all the extra crust around some set, and then healing any infinitesimal holes or cracks.  What you end up with at the end of this process is a nice looking set:

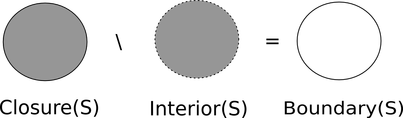


This process is called regularization, and the sets we get at the end are called solids (hence Requicha’s definition).  Note though that there are actually two kinds of regular sets: closed sets and open sets.  The other type, open regular sets, satisfy the condition that:

S = \iota(\kappa(S))

The choice of open or closed is ultimately arbitrary.  We could just as well do all of the same things using open sets, but in some situations we’d have to flip the direction of a \subset to a \supset and \cup to a \cap.  I’ve heard it suggested that you [might prefer one definition over the other](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.7200) in specific applications, but it is conventional to work with closed solids.

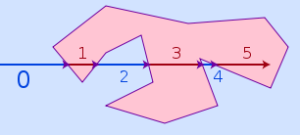
From the perspective of graphics and engineering, one really nice property of a solid object is that it is uniquely determined by its boundary.  That is, the *boundary* of a set, \delta(S), is what we get when we subtract the interior from the closure:



The idea that we can use boundaries to describe solids is the basis for most efficient representations of shapes, like polygonal models.  (In fact, even run length encoding can be thought of as a type of boundary representation.)  Converting from a boundary representation back to a solid is in general quite difficult to do robustly in all cases.  However, there is a very simple algorithm that works in quite a few cases when we have an especially nice boundary.  Here it is:

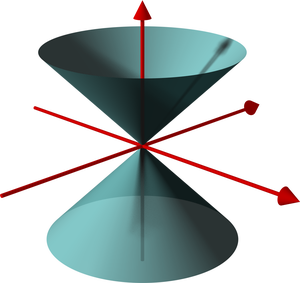
1. First you need to define the outside region, and to do this you just pick any point o \in R^n
2. Now to classify point x \in R^n, choose any path p : [0,1] \to \mathbb{R}^n.
3. Count the number of times p crosses the boundary.  If it is even, the point is outside; if it is odd the point is inside.

This method is the basis for most point membership tests of solid objects.  To illustrate how this works, here is a picture from wikipedia:

[](http://en.wikipedia.org/wiki/File:RecursiveEvenPolygon.svg)

Point membership by raycasting illustrated. Author: Melchoor (c) 2007 Wikimedia Foundation.

Of course to make this work we need to require two things.  First, the path must always intersect the boundary of the solid tangentially.  If we just glance of the surface, then we won’t correctly classify points.  Second, the path must not touch any [singularities](http://en.wikipedia.org/wiki/Singularity_theory) .  To explain this in a bit more detail, consider what happens at the vertex of a double cone:

[](http://en.wikipedia.org/wiki/File:DoubleCone.png)

3D rendering of a double cone. Author: Lars H. Rohwedder (c) 2006 Wikimedia Foundation.

In this situation, if we took a path that went directly through the point, we’d flip our parity without actually leaving the solid!  As a result, this process of tracing a path can produce ambiguous results at these points.  Note however that the situation is not all bad; the path tracing would still work at most of the other points on the solid.  As we mentioned earlier, these weird points are called [singularities](http://en.wikipedia.org/wiki/Singular_point_of_a_curve), and they are basically points on the boundary which locally do not look like a disc.  (Imagine zooming into the vertex of the cone.  No matter how close you get, it always looks like a cone; whereas if you pick any other point, eventually it just like a flat disc.)

To avoid this situation, one usually adds some additional requirements that the boundary of a solid does not contain any singularities: or more precisely, we require the boundary to be a *closed*[*orientable*](http://en.wikipedia.org/wiki/Orientability)[*manifold*](http://en.wikipedia.org/wiki/Manifold)*without boundary.*To explain this a bit, a manifold is a set where around each point it locally looks like an open disc — that is, it has no singularities.  Examples of manifolds include things like spheres and torii, while non-examples would be structures like cones or nodal curves.  We say that a manifold is orientable if around each point you can give some consistent meaning to `clockwise’ and `counterclockwise’ that matches up globally.

In general, working with boundary representations is much more efficient than solid representations.   The underlying reason for this is that they usually much more compact and so operations require fewer memory accesses.  The downside is that in order for a boundary representation to remain valid, it has to satisfy much stricter topological constraints which greatly complicates algorithms for dealing with them.  It is also quite hard to handle non-manifold shapes efficiently using a boundary representation.  (For example, winged edge and half edge data structures can not represent non-manifold surfaces, even though they may determine perfectly good solids.)