What is Constructive Solid Geometry?

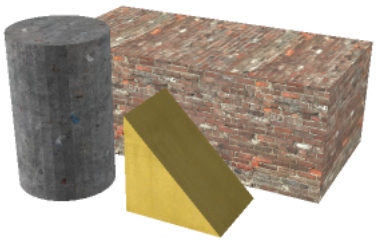
Contents

1.0 Introduction: Intelligent Geometry

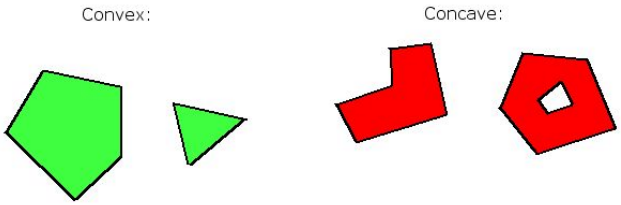
Constructive solid geometry (CSG) is a type of *intelligent geometry* that gives the programmer extra information. CSG uses simple objects called "solids", constructed according to geometric rules. The special properties of CSG solids allow mathematical operations that are not possible with an arbitrary polygon mesh. It's no wonder that games like *Half-Life 2*, *Doom 3*, and *Unreal Tournament 2004* rely on CSG as the basis of their world geometry.

2.0 Constructive Solid Geometry Defined

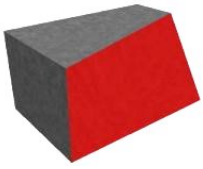
CSG solids are simple convex objects, made out of intersecting coplanar faces:



Convex means that all faces point out away from each other:

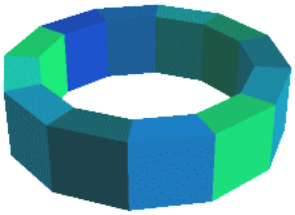


Coplanar means that all the vertices of a face lie along the same plane:



The top right vertex has been moved inwards to make the selected face non-coplanar.

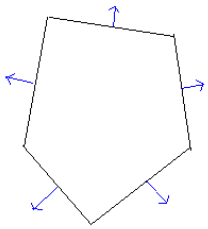
Concave objects can be created using multiple convex solids:



This concave solid still follows the rules of CSG; each individual solid is convex.

3.0 Constructive Solid Geometry In Action

The special properties of CSG solids allow mathematical operations that are not possible with an arbitrary polygon mesh. CSG solids are made out of intersecting coplanar faces. Each face has an ascertainable plane equation, defined as a 3D vector, plus a distance from the origin:



Each face has a normal indicating the direction it points, and a distance from the origin. The intersection of the planes defines the solid.

3.1 Basic CSG Operations

Two equations form the basis of more complex operations.

Equation for distance between a point and a plane

The sign of the result indicates whether the point is in front of or behind the plane.

*Distance between point A and plane B = (Bd–Bnx*Ax–Bny*Ay–Bnz*Az)*

Equation for ray-plane intersection

Determines the intersection point of a line in 3D space with a plane.

Intersection of ray AB with plane C:

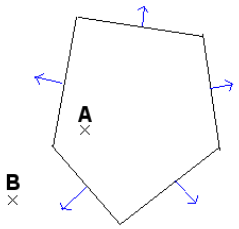
$$u = (-Cnx*Ax–Cny*Ay–Cnz*Az+d)/(-Cnx*(Ax–Bx)–Cny*(Ay–By)–Cnz*(Az–Bz))$$
$$x = Ax+(Bx-Ax)*u$$
$$y = Ay+(By-Ay)*u$$
$$z = Az+(Bz-Az)*u$$

3.2 Intermediate CSG Operations

Intermediate operations based on basic CSG operations can be useful during runtime, and form the basis of advanced CSG operations.

Point-solid intersection

If the point is behind every face of the solid, it lies within the solid. A radius value can be added to the "d" component of each plane to test a sphere instead of a point.



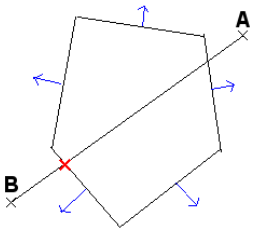
Point A lies behind each face, and is within the brush. Point B lies behind 3 faces, but in front of 2, so it is not within the brush.

Point-solid intersection can be used for:

- **Make a player step into water or an invisible trigger zone.**

Ray-solid intersection

If the intersection point of the ray and any of the planes in the solid lies behind all the other planes, the ray must intersect the solid.



The intersection point, shown in red, passes the point-brush intersection test when the intersecting face is dismissed. Therefore, the line AB intersects the solid.

Ray-solid intersection can be used for:

- **Check visibility between two players.**
- **Check visibility between a light and an object.**
- **Player collision (add a radius value to the *d* component of each plane).**

Solid-solid intersection

If any vertex of either solid lies within the other solid, the solids intersect.
If any edge of one solid intersects the other solid, the solids intersect.

Solid-solid intersection can be used for:

- **Camera frustum occlusion.**
- **Make a projector that casts an image onto objects.**

Find the silhouette of a solid from a point or vector

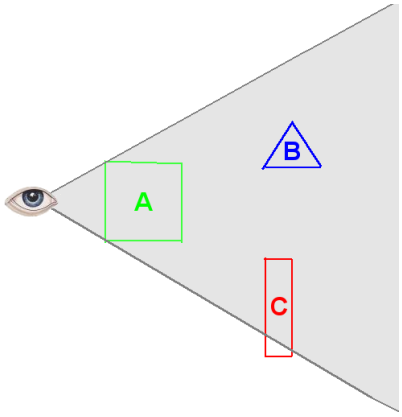
Each edge lies between two faces. If one face points towards the point, and the other points away, that edge is part of the outline. The silhouette of a convex object is always convex. This is the basis of solid extrusion.

Extrude solid from point or vector

Find the silhouette of the solid, then extrude away from the point or along the vector. This creates another solid that defines the shadow volume of the original. If a point or solid lies entirely within this volume, that object is completely occluded by the original solid. These kinds of extrusions are used frequently for lighting and portal occlusion systems.

Solid extrusion can be used for:

- **Make a solid occlude another object.**
- **Calculate the shadow volume of an object.**



Solid A occludes solid B completely, and partially occludes solid C. The silhouette of solid A has been extruded from the eye position. The resulting solid, shown in gray, encloses object B.

Solid slicing

A solid can be sliced in half with an arbitrary plane. This is used frequently in portal and BSP occlusion systems.

Solid slicing can be used for:

- **Cut a solid in half along the bounds of a rendering zone.**

3.3 Advanced CSG Operations

The previous operations described form the basis of even higher-level operations. These are advanced techniques that cannot be described with a simple equation, and will vary with implementation.

Texture mapping

Texture coordinates tell the GPU how a texture lies across the surface of an object. Solids' texture coordinates can be calculated on-the-fly, with user control over parameters like scale, position, rotation, and shearing. This is done by generating texture mapping axes per face, based on face normals. The texture coordinates for any vertex of a face can be calculated from the vertex position and face texture mapping axes:

$$u = unx*x+uny*y+unz*z+ud$$
$$v = vnx*x+vny*y+vnz*z+vd$$

Portal occlusion systems

Portal occlusion systems separate the scene into large chunks (portal zones) connected to each other through "doorways" called portals. This allows the renderer to skip geometry

that is not only off-screen, but occluded by other objects in the foreground. This is difficult, if not impossible, without CSG.

Static light calculation
Ray-tracer lightmappers create a colored texture to simulate the lighting environment. Light maps can quickly be allocated to CSG brushes, and volume intersections can quickly dismiss objects which do not cast shadows onto a surface.

Dynamic light calculation
Some rendering systems calculate lighting in real-time, usually using the stencil buffer to render shadows, and rendering the scene in multiple additive passes. These calculations are very demanding, and a good occlusion system (portal, BSP, or a combination of both) is crucial.

Binary Space Partitioning
Binary space partitioning (BSP) is a method of optimizing a static scene by splitting the geometry into a recursive hierarchy of BSP leaves and nodes, forming a BSP tree. A BSP tree can be used to optimize collision, occlusion, and other calculations, by excluding huge regions of the scene that are irrelevant. Performance gains are typically exponential when a BSP tree is utilized.

4.0 Conclusion

Simple simulations can afford to treat the world as an arbitrary polygon mesh. More advanced rendering systems need some method of culling and optimizing the scene. By implementing constructive solid geometry, designers can take advantage of simple CSG operations. These operations can be used to build more advanced systems like portal occlusion and binary space partitioning.

Without constructive solid geometry, 3D worlds are just a collection of arbitrary triangles, with no intrinsic properties. CSG adds intellegent geometry to your world.