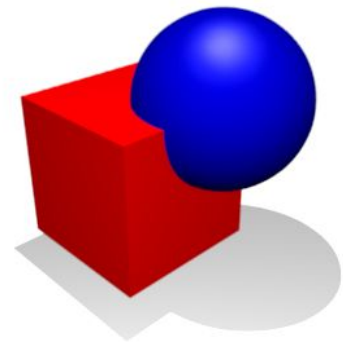
Voxel set to constructive solid geometry - OpenGL

Introduction

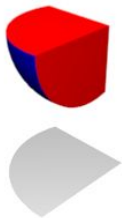
Constructive solid geometry (CSG) is a technique used in solid modeling. Constructive solid geometry allows a modeler to create a complex surface or object by using boolean operators to combine objects. Often CSG presents a model or surface that appears visually complex, but is actually little more than cleverly combined or de-combined objects.

In 3D computer graphics, CSG is often used in procedural modeling. CSG can also be performed on polygonal meshes, and may or may not be procedural and/or parametric.

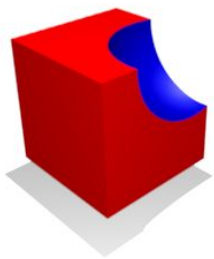
We have provided support for **union**, **intersection** and **difference** operations in our application.



Union of a cube and a sphere



Intersection of a cube and a sphere



Difference of a sphere from a cube

Types of input possible

Following types of input can be given to the software:

1. We have provided some standard 3D shapes like **cube, cylinder, sphere and cone** which can directly be used without providing any obj file as input.
2. Obj files that have voxelated objects can be provided as input using the '**Open File**' option.
3. Obj files that describe objects using their surfaces can also be given as input using the '**Open Surface File**' option.

You can apply any of the 3 operations namely - union, intersection and difference on these inputs, 2 at a time.

Other features provided

Some other features in the software are:

1. The objects can be translated in X,Y and Z directions.
2. The objects can be rotated in X,Y and Z directions (input in degrees).
3. You can save any resultant object. It gets saved by the name '**output.obj**' in the build directory.
4. The current object and the existing objects are shown in different colors to make the distinction clear.
5. You can look at the object from different positions by rotating the object inside the object plane. Arrow keys can be used to rotate in X and Y directions. A and S keys can be used to rotate object in Z direction.
6. Illumination, camera position, coloring and shading has been taken care of.

Algorithm

Input: VoxelSet1, VoxelSet2
Output: VoxelSet1 OP VoxelSet2 where OP can be union, intersection or difference

1. Find centers of the voxels in VoxelSet1 and VoxelSet2
2. Center1 = **set** of centers of the voxels in VoxelSet1
3. Center2 = **set** of centers of the voxels in VoxelSet2
4. Center = Center1 OP Center2 (just apply OP on the 2 sets considering them as normal sets)
5. VoxelSet = Construct voxels from the centers in Center.
6. return VoxelSet

Surface file to voxels

For standard shapes (cube, sphere etc.) and voxelated obj files, we already have the voxels. So this algorithm works directly. But, for obj files containing surfaces, we need to get the voxels first. The algorithm for that is mentioned below.

Input: surface file containing faces with 3 or 4 sides.
Output: voxel file containing voxels of the surface file.

1. set<Voxel> voxels = null
2. for each face in faces:
    a. if face has 3 sides:
        i. voxels.add(triangleToVoxels(face[0], face[1], face[2]))
    b. else:
        i. voxels.add(triangleToVoxels(face[0], face[1], face[2]))
        ii. voxels.add(triangleToVoxels(face[2], face[3], face[0]))
3. return voxels

triangleToVoxels() takes 3 points as input which represent a triangle in 3D. It returns the voxels lying inside that triangle.