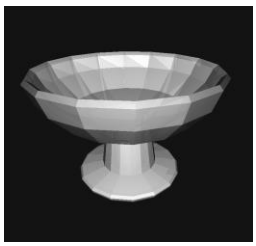
Eric Principato
May 10, 2016
COS 426 Final Project
Prof. Finkelstein



CAD.js

Introduction

Goal

The goal of this project was to create a basic CAD or 3D modeling environment. The core features I wanted to include were 2D polygonal sketches, extrusions, and revolutions, with a focus on being able to perform constructive solid geometry (CSG) operations including union, difference, and intersection. Another early-stage goal of the project was to explore forms of navigation in the CAD environment. As a mechanical engineer, I recognize the importance of CAD in design and fabrication. The CAD programs that have enough features to be truly useful to an engineer or machinist are all expensive, so this project explores how useful a simpler CAD program that uses open source tools can be.

Previous Work

CAD programs commonly used in industry include tools such as PTC Creo, Autodesk Inventor, 3DS Solidworks, and Siemen's NX suite. These tools all offer a wide range of features and reasonably user-friendly interfaces, but are very expensive. There are some open source CAD tools, but most operate in 2D. FreeCAD is perhaps the most developed 3D program, but is not at all user-friendly. OpenSCAD is a more popular tool that allows 3D shapes to be created in its own programming language. OpenSCAD uses the CGAL and OpenCSG libraries to perform CSG operations and store its data structure. It offers a wide variety of common CAD tools, but its downside is that it cannot be interacted with directly (only through its programming language).

Approach

After exploring existing programs and libraries, I chose to develop a simple non-parametric CAD program in Javascript, taking advantage of the CSG.js library for CSG operations [1] and the THREE.js library for rendering and vector math [2]. I implemented a very basic feature set including extrusions, revolutions, polygon generation (including triangularization of concave polygons), and CSG operations. I also added features that make the software easier to use including colored shading of active and selected features based on the operation in progress and tool in use. I chose to make the software non-parametric, meaning features cannot be dimensioned, choosing instead to focus on other features including user-friendliness.

Methodology

Geometry Data Structures

The requirements for geometry created by this program are that it must be able to be converted to THREE geometry objects, allow for extrusion and revolution operations on it, and be able to be converted to CSG objects. In order to meet these requirements and maintain simplicity (this is a prototype after all) I chose to maintain a global list of polygons/faces and a global list of vertices.

A polygon is created from a plane. All points in the polygon lie on this plane and the plane is be used to determine intersections and aid in other geometric operations. The only requirements for a polygon are that it can have vertices added, be triangulated, and be converted to csg geometry. Each time a vertex is added, the polygon is re-triangulated and the triangles are stored. This means that each time the triangle mesh of the entire geometry needs to be made, the underlying triangles of the polygon are just added to the list of faces. Triangles are stored independently of faces because it is important in a CAD environment for the user to be able to select and manipulate the polygon they created, and not a triangulated version of that polygon. This interface for polygons/faces allows them to work well with extrusions and revolutions. Polygon triangulation works on polygons with concave vertices, and uses a version of the ear clipping algorithm [3].

Extrusion

The requirements for an extrusion are for it to create a new set of polygons that form its outline and for it to be able to apply CSG operations on the current geometry. The extrusion also needs to be able to be dynamically updated based on user input. In order to accomplish these goals, the Extrusion class's constructor takes in a base face and immediately creates the new polygons for the top face and side faces. However, it waits until the user is done modifying it to actually apply the selected CSG operation. To apply the CSG operation, the extrusion is converted to a CSG-compatible object and passed to a CSG.js function. The CSG library creates a binary space partition (BSP) of a triangular mesh to perform its operations, so I took much care to make sure the program creates all of its triangles in the correct direction. If some triangles are in the incorrect direction, the CSG algorithms fail and create odd or invalid meshes.

Revolution

The revolution operation has similar requirements as the extrusion operation, and works in a similar manner. The key differences are that a revolution is restricted to a full 360 degree revolution and that an axis of revolution must also be chosen. Once these are both supplied, a revolution is created by rotating the original points in the polygon multiple times around the axis and connecting sets of adjacent vertices to form the sides. Because I chose to only work with simple polygonal meshes, this is the best approximation of a true surface of revolution. Points are rotated around the selected axis using a composition of matrices [4]. Once the revolution is finalized, its geometry is converted to a CSG object in a similar manner to the extrusion and applied to the existing mesh.

Rendering

The program uses the THREE WebGLRenderer to perform rendering. The scene consists of a perspective camera, a point light located at the camera, a wireframe mesh of all polygons, sets of active and selected polygons, and the resultant mesh of all CSG operations. When geometry is modified, selected, or activated, each scene object is recreated, has a THREE material applied to it, and is re-added to the scene.

Having these separate scene objects achieves the objective of giving the user access to all of the "true" polygons in the scene while supplying the THREE Geometry class and the CSG main class the required triangle mesh.

In order to correctly render the selected polygon even if that polygon is part of the CSG mesh, a small z-buffer offset is applied to the material applied to the selected geometry. This makes the polygon appear slightly closer than otherwise so that it always renders in front of a coplanar non-selected face. This is important so that its red color stands out clearly. The z-buffer offset is also applied to active polygons (for example, the side polygons in an extrusion that the user is currently creating). This allows the color of the active material (green for union or yellow for difference) to also show in front of any coplanar mesh faces.

HTML Page

The HTML page (index.html) was created using the THREE.js Trackball Control example as a template [5]. In addition to the main CAD environment rendering container, it also contains text at the top explaining the program's controls.

Results

Program Flow

The program starts with an initial geometry of a square on the XZ plane. This gives the user a platform to work off, as the starting square can be selected as the basis for a new polygon, extrusion, or revolution.

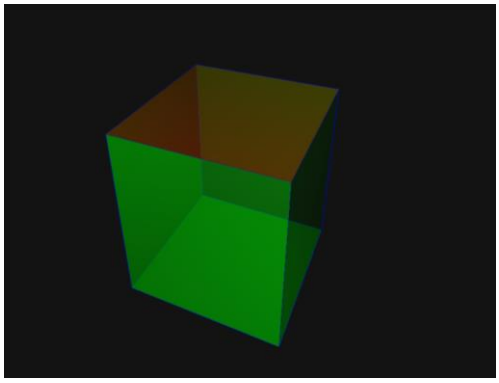
The user can create a new polygon by pressing the "3" key. The user then selects a polygon that the new polygon will share a plane with. After this, the user can start adding vertices to the new polygon by clicking anywhere. The mouse's coordinates are cast onto the polygon's plane and the vertex is added. A polygon is finalized by right clicking.

In the program, the user presses "4" to initiate an extrusion. Then the user clicks on a polygon to extrude and a preview of the operation is displayed. The length of the extrusion is based on the mouse's position relative to the base face and dynamically updates, providing an intuitive way to create extrusions. At this point the user can also select "2" to make the extrusion a union, "3" to make it a difference, or "4" to make it an intersection. Once happy with the geometry, the user clicks again and the operation is applied.

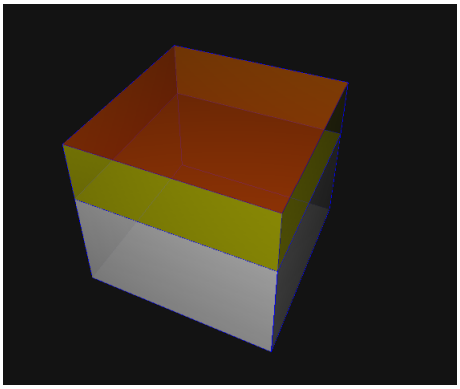
In a similar manner, a revolution is initiated by pressing “5”. The user then clicks on a polygon to revolve and clicks on the axis of revolution. At this point the revolution is displayed and the user clicks again to apply it to the current geometry.

Over the course of creation of a complex shape, many polygons may be added to the environment. Because all polygons are displayed, this can create a large amount of clutter. If the user does not want to see any lines, and just see the current CSG geometry, they can press “2” and at any time press “1” to turn lines back on.

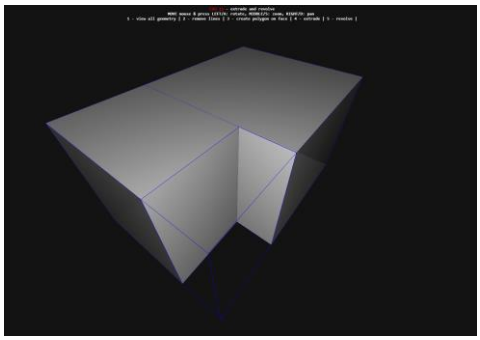
An assortment of geometries created with this program is shown below:



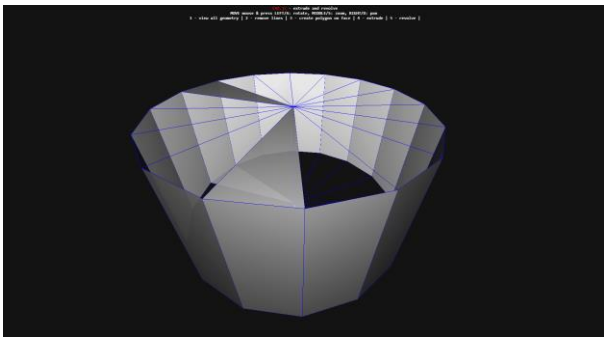
A union operation in progress: the original face is red and the new faces are green.



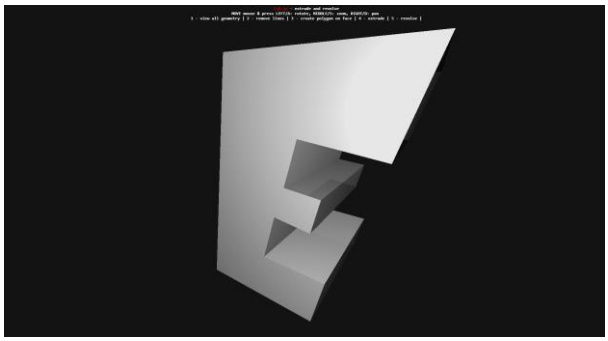
A difference operation in progress: the original face is red and the new faces are yellow.



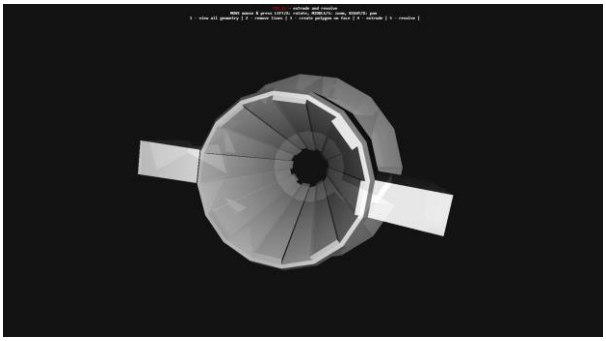
Result of a couple unions and differences: note that the polygons are outlined in blue.



Result of a revolution where the axis is an edge of the polygon: the geometry is incorrect, possibly due to a bug in the CAD.js library.



Correct triangulation of convex shape: all faces of resulting extrusion also render correctly.



Complex shape generated from a variety of extrusion and revolution unions and differences.

Issues

The created CAD program allows the user to created arbitrary geometries consisting of polygons, extrusions, and revolutions, but it does have flaws. The algorithm for triangulating concave shapes works for many concave shapes, but fails for others – primarily very complex shapes or shapes with self-intersection. Also, polygons must be created with their vertices in a counter-clockwise order or the concave triangulation routine fails. I developed code to mitigate this by detecting which order the points were generated in (calculate whether the sum of exterior angles is positive or negative 360 degrees), but commented it out because the algorithm already has other flaws.

Another issue is with revolution CSG operations. I believe this is due to a bug in the CSG library, but if a polygon is rotated about an axis that is also an edge of the polygon, the resulting CSG operation fails. There are also other odd edge-cases that cause the operation to fail.

Finally, there is an issue with tracking keyboard and mouse events. For simplicity, I made the key and mouse event handlers update key and mouse objects, and then had the main state machine poll those objects. I did this for simplicity, knowing this would cause some key and mouse presses to be missed. These could easily be fixed in the future, but for this prototype I chose to focus more on geometry and rendering.

Conclusion

CAD.js allows a user to intuitively edit polygonal meshes in a manner similar to existing CAD software such as Autodesk Inventor or 3DS Solidworks (though in a very minimalistic way). It offers a variety of features including access to CSG operations, dynamic updating of polygons and extrusions, and triangulation of concave polygons. Using THREE.js and CSG.js allowed me to focus on filling out the proposed feature set and create a tool that is fun to use, and has the potential to be useful to people such as hobbyist designers and makers who don't want to pay for a more expensive product.

Many improvements could be made to this software. From a user interface perspective, an overlay on top of the CAD environment would allow the user to more easily select tools and be aware of the

program state. From an engineering perspective, most designs are made from real dimensions, so allowing a user to edit vertex locations and line length would make the software more useable. A large variety of features exist that could be added to make this a real CAD tool including sweeps, lofts, chamfers, and tapered extrusions. Finally, an algorithm to combine coplanar faces that share an edge would allow this software to behave more realistically and more similarly to existing CAD programs.

Though relatively simple, CAD.js proves that it is possible to create sophisticated CAD software from existing open source tools. For all its simplicity, the program provides plenty of entertainment, and it shows that complex shapes can be created from a few simple operations.

Works Cited

[1] Wallace, Evan (evanw). CSG.js. https://github.com/evanw/csg.js/

[2] Cabello, Ricardo (mrdoob). THREE.js. http://threejs.org/

[3] Everly, David. Triangulation by Ear Clipping. Geometric Tools, LLC. August 16, 2015. http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf

[4] Murray, Glenn. Rotation About an Arbitrary Axis in 3 Dimensions. June 6, 2013. http://inside.mines.edu/fs_home/gmurray/ArbitraryAxisRotation/

[5] THREE.js TrackballControls Example. http://threejs.org/examples/misc_controls_trackball.html

This paper represents my own work in accordance with University regulation.