Homework09: Final Project

The goal of this project is threefold. First, this project aims to successfully read a STL file and graphically display its geometry. The second goal is to read an associated SAT file and use its information to determine a unique normal vector for each known vertex. Additionally, this SAT file information will be used to categorize every vertex into a unique surface displayed with a different color. To accomplish this second goal, only the surfaces of the SAT file would need to be read. However, beyond these two required goals, this code also aims to read an SAT file's information and categorize each line into its corresponding geometry according to the B-Rep data structure in Figure 1. This data structure is similar to that found in Homework05. In addition to these classes, Cone surfaces and ellipses were added.

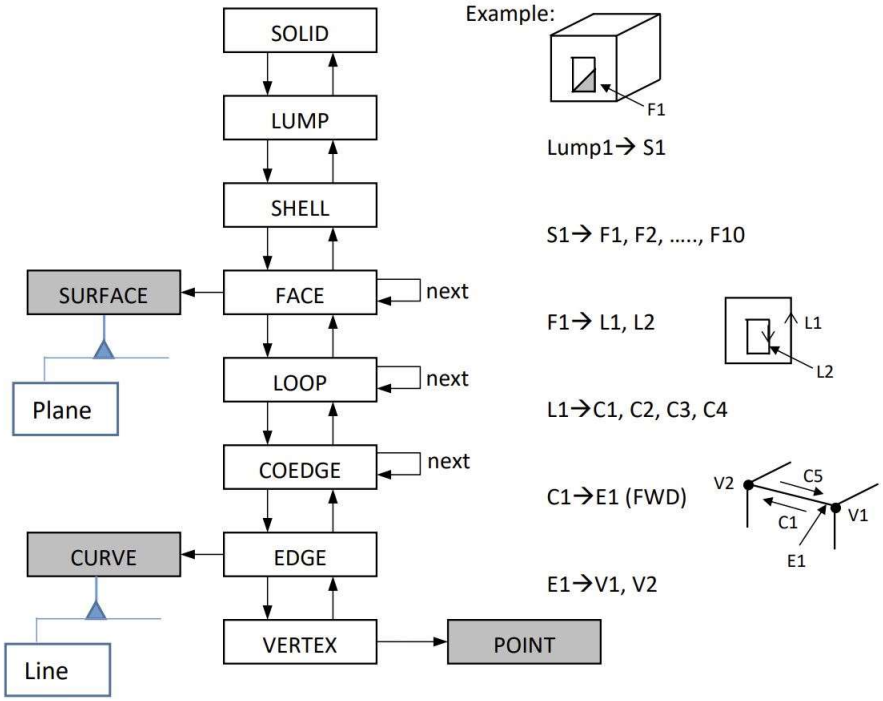

Figure 1

1 Reading and Displaying an STL File

The first arm of this project is relatively straightforward. A simple solid was built in Solidworks and its STL file was downloaded. A Buffered Reader was used to read through the STL file and record every vertex and stated normal vector. Because the normal vector is identical for every group of three vertices, it was recorded three times for every time it appeared in the data. A mesh was initialized and its Buffer updated with the vertex and normal information. Once this mesh was given a material and attached to the Root Node, the information could be displayed graphically as it is below in Figure 2.

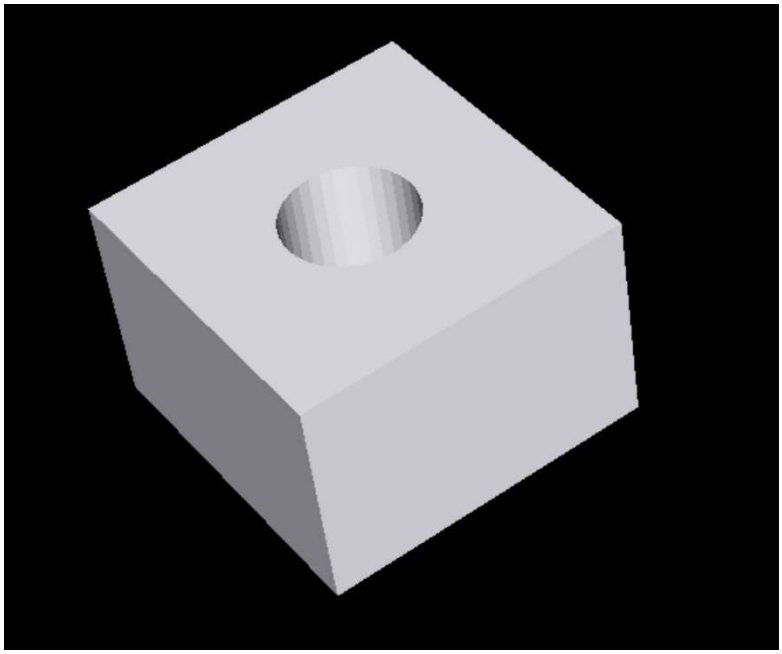

Figure 2: part1.STL

Part1 is a fairly simplistic piece. It is a rectangular prism with a cylindrical hole in its top. This hole does not pass all the way through this piece. Notice that inside the cylinder, the edges of each triangle read from the STL file can be seen. This is due to the fact that three vertices all share the same normal vector. Without the SAT file, the edges cannot be smoothed any further.

2 Identifying Surfaces in the SAT File

Similar to the STL file, a Buffered Reader was used to read the SAT file. Whenever it recognized a surface, such as "plane-surface", it recorded its information into an Arraylist to be displayed later. The two major types of surfaces it can encounter are planes and cones. Although generic cones can be read by this program, every tested part has a perfect cylinder. For this reason the explanation of the code will focus on cylinders. Normal planes can be defined using a single point and a vector normal to the plane. There are a variety of ways to display this data, but this code made use of a constant called D_0, such that given any point p Equation (1) holds true.

$$p \cdot N + D_0 = 0 \qquad (1)$$

In other words, the dot product of any point on the plane against the normal vector will equal $-D_0$. Because D_0 can be determined from the given center point, every known vertex can be tested against this equation to see if it holds true. If a group of three vertices satisfy the equation, these were then categorized as belonging to said plane. Through this method, each triangle read from the STL file could be categorized into its respective surface.

Cones must be read parametrically. However, there exists a simpler method to identify if vertices lie on the surface of a cylinder. Given the center of a cylinder's ellipse, p_c, and the normal vector N_e, a point p lies on the surface of the cylinder if it satisfies Equation (2) below.

$$Radius = \frac{|N_e \times (p_c \times N_e - p \times N_e)|}{N_e \cdot N_e} \quad (2)$$

2.1 Smoothing Edges

The edges for plane surfaces, fortunately, did not need to be smoothed. The normal vector for every point on a flat plane is identical. For cylinders, the method to finding the normal vector is also quite simple. Recall Equation (2), which deals with radius. If we do not find the magnitude of the numerator, it will equal to the normal vector for each vertex N_v. Note that N is still the normal vector for the cylinder's ellipse.

$$N_v = \frac{N_e \times (p_c \times N_e - p \times N_e)}{N_e \cdot N_e} \quad (3)$$

Given any point p, this equation finds the distance vector to the nearest point on a given line N. Because all points on a cylinder are normal towards the center, this distance vector is functionally interchangeable with the point's normal vector. Figure 3 below shows the effect of determining a normal vector for each individual vertex. The rough lines in the cylinder have become smoothed over as each polygon smoothly transitions into the next.

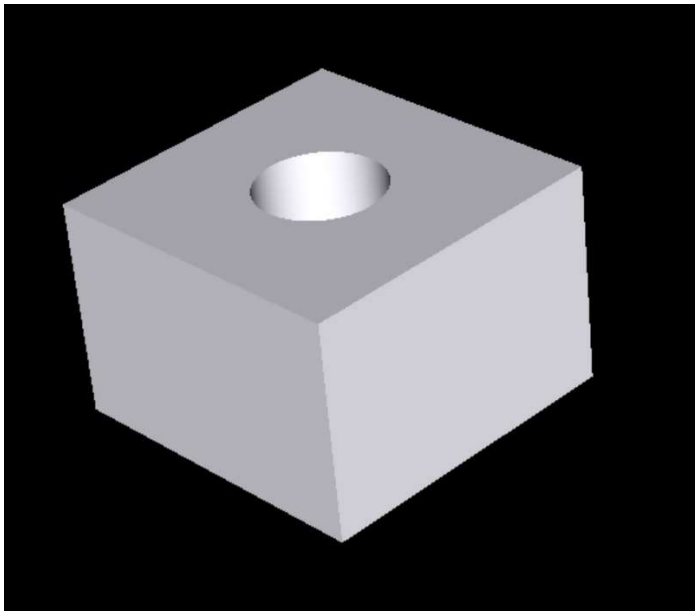

Figure 3

2.2 Coloring Unique Surfaces

Using Equations (1) and (2), every vertex can be associated with either a unique plane or cylinder. The code runs through a nested for-loop and tests every group of three vertices against every plane and cylinder. Note that unlike a normal for-loop, the vertex iterator j progresses by +3 rather than +1 every loop. The result of this is that vertices from separate triangles are not accidently mixed and matched to match a surface. Once a group of three vertices are found to match a surface, they are attached to that surface's stored vertex list. Although this method of using nested for-loops has time $O(n^2)$, the tested shapes are simple enough that the small number of surfaces being tested does not cause a noticeable delay in the program.

There are multiple methods to color unique surfaces, however it is easiest to have every mesh a single color. Therefore, using a for-loop, a unique mesh was created for every plane and cone surface found and a color was assigned. A color wheel of 11 different colors was created, from which each cylinder mesh pulled its color. Each mesh created for a surface plane was assigned color randomly. Once the mesh and corresponding material had color, the mesh buffer was updated with each surface's position vertices and normal vectors, and a geometry created from the mesh was attached to the Root Node. As a result, each unique surface had its own unique color.

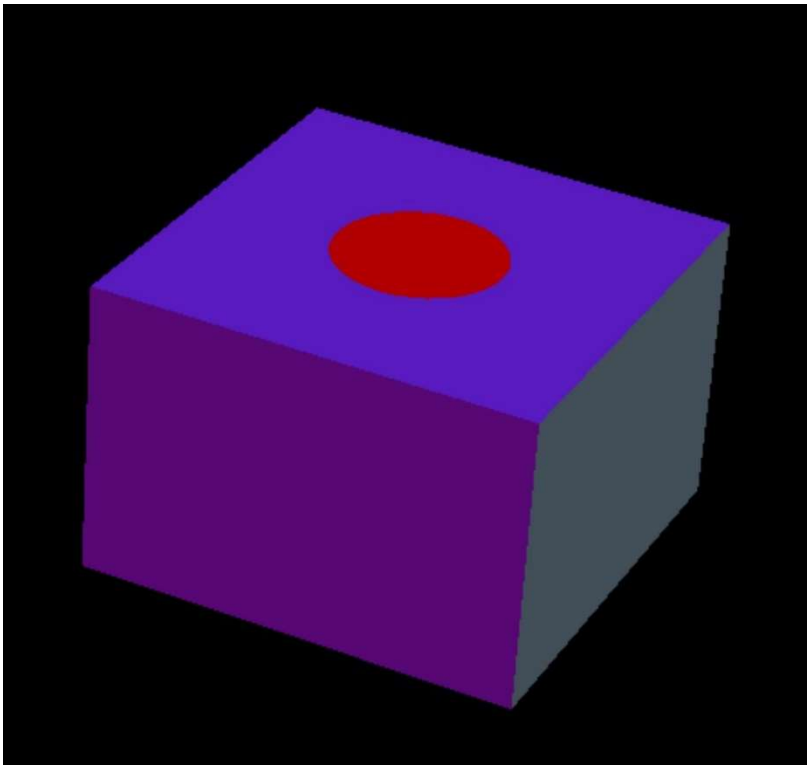

Figure 4

3 Reading the SAT File

To read the SAT file, a Geometry package created in Homework05 was imported. Additionally, the switch statement in the Buffer Reader, which previously only recognized surfaces from the SAT file, was modified to read bodies (also referred to as solids), lumps, shells, faces, loops, coedges, edges, vertices, points, ellipse curves, and straight curves, all of which are subclasses of the abstract class Geom_Added. The unusual naming for this superclass is to avoid confusion with the package name, "Geometry", or any other implemented classes. The data from the SAT file is pulled and used to create a corresponding piece of geometry which is then added to an arraylist. The position of every solid body is additionally recorded. After the STL and SAT file data is graphically displayed, the program finds and calls the *print()* method for every solid body. This *print()* method then calls the *print()* method to every referenced object, and so on. The result is a complete reprinting of the SAT file data. Note that the printed syntax follows a more reader-friendly style rather than duplicating the SAT file. As Figure 5 shows below, the solid prints, followed by the lump, and so on.

```
File Found.
1 bodies found. Printing now.

Solid 1:
It is composed of Lump 1.

Lump 1:
It is composed of Shell 1.

Shell 1:
It is composed of 9 Faces.
There are 0 linked shells.
```

Figure 5: Print() method

4 Final Modifications and Results

A Scanner class was implemented to ask the user which file should be opened, as well as whether the associated SAT file should be read. In total, four parts were developed in Solidworks and graphically displayed in the program. These parts can be viewed below in Figures 6-8. Part2 is the inverse of part1: a solid cylinder containing the cutout of a rectangular prism inside. Part3 contains more complicated geometry, including large fillets and extruded surfaces. Moreover, part3 introduces planes which are not normal to one of the three axes. Part4, meanwhile, cuts a cylinder in half, fillets two corners, and has a horizontal cylinder. In all three figures, however, the program read the STL and SAT file information, accurately organized each vertex into its corresponding surface, and colored them accordingly. Additionally, the topology of each solid was recorded and displayed with the *print()* method.

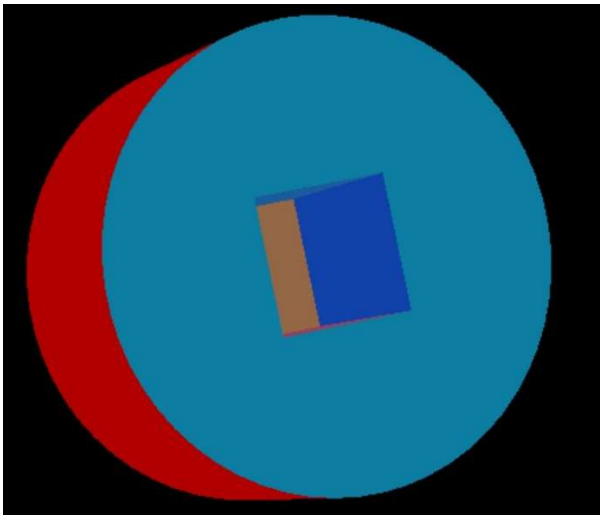

Figure 6: part2.STL

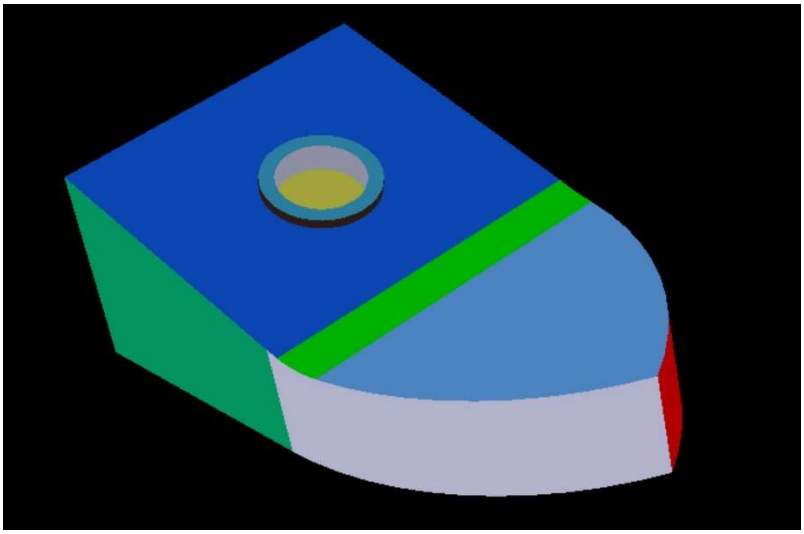

Figure 7: part3.STL

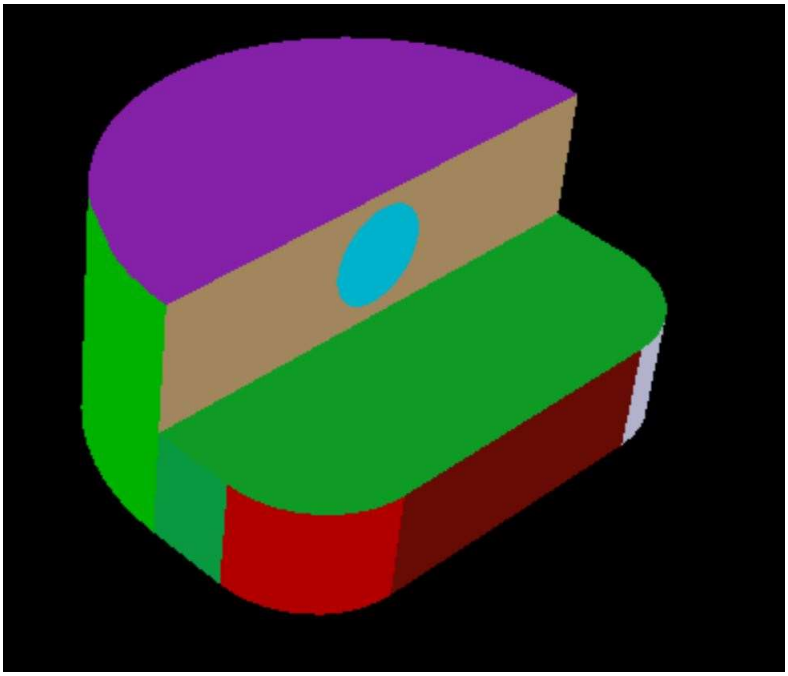

Figure 8: part4.STL