

Core Extraction Software Subroutine

Tyler Huntley
Department of Mathematical and Computer Science
Bethel College
300 East 27th Street
North Newton, KS 67117. USA

Rapid Prototyping Center
Milwaukee School of Engineering
1025 N Broadway
Milwaukee, WI 53202. USA

Faculty Advisor: Vito Gervasi

Abstract

This research is aimed at preparing a portion of a larger software package, which employs the “Shell-Slice” technique, reducing complex three-dimensional objects into manufacturable elements. Currently, there are a number of software packages available designed to slice three-dimensional objects in preparation for solid freeform fabrication (SFF). There are various methods that are utilized, most exercising a layer-based raster procedure. Using SFF, each of the cross-sectional layers is transformed into the larger object by curing, sintering, or bonding. Current, layer-based methods lead to stair stepping defects. Thinner layers reduce stair stepping but lead to undesirable lengthy build times. This research is needed in order to create complex lattice structures more efficiently and with reduced error. The main focus of this core extraction software is to identify and extract the internal geometry from a CAD object through the use of C++. CAD files are in an .stl format, and the C++ program manipulates the CAD in order to extract the core(s) of CAD objects.

Keywords: Solid Freeform Fabrication (SFF), Computer-Aided Design (CAD), C++, STL format, Shell-Slice Technique

1. Introduction

Software is needed that will create solid forms of the hidden voids, or undercuts, of complex three-dimensional objects. Using these solid forms, called cores, it will be possible to reduce these three-dimensional objects into manufacturable elements. This could range from just a few to thousands of elements, depending on the complexity of the object. These parts will be created using different types of molding processes. Machining will take care of problems that are present with layer based methods. They will create models in a shorter amount of time. Also, they'll use materials that cost much less than what's used in layer-based machines. In addition to being much more time and cost efficient, the error level (stair-stepping) will no longer be present.

2. Solid Freeform Fabrication

Solid Freeform Fabrication (SFF) has been around since the late 1980s. It is a very important technology that is used by numerous companies worldwide to create prototypes for new products. What typical SFF slice software does is take a three-dimensional Computer-Aided Design (CAD) object, and split it up into many two-dimensional layers. Each of these layers is then built upon each other, one at a time, in the machine of choice. Four Examples of technologies used for SFF include the following:

- SLA (Stereolithography)
- LOM (Laminated Object Manufacturing)
- FDM (Fused Deposition Modeling)
- SLS (Selective Laser Sintering).

Each of these uses different methods and materials for creating all of the layers and gluing or fusing (with a laser) them all together.

3. Manufacturable Elements

As stated earlier, one aim of the research was to be able to take an existing object's CAD file, and manipulate it in such a way that it is broken down into manufacturable elements.

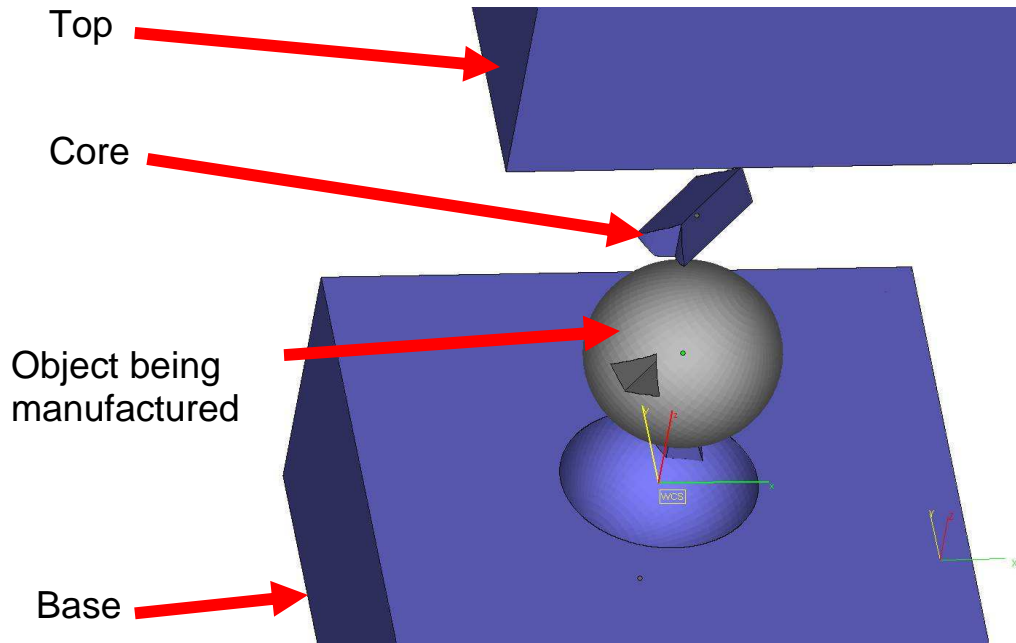


Figure 1

Figure 1 shows a simple example of an object being broken into manufacturable elements. The object being created is a simple sphere with an odd shaped hole cut through it. When looking at the object from the z axes (the top and the bottom), the hole cannot be seen. This is how the undercut identification process will work. The core is created, and so are the top and the bottom bases. Basically, a mold of sorts is created. However, the purpose of this project is to eventually be able to do this process with more complex objects, such as the one shown below. It has many cores, and, even more importantly, cores within cores.

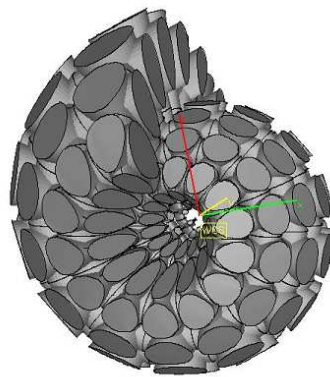
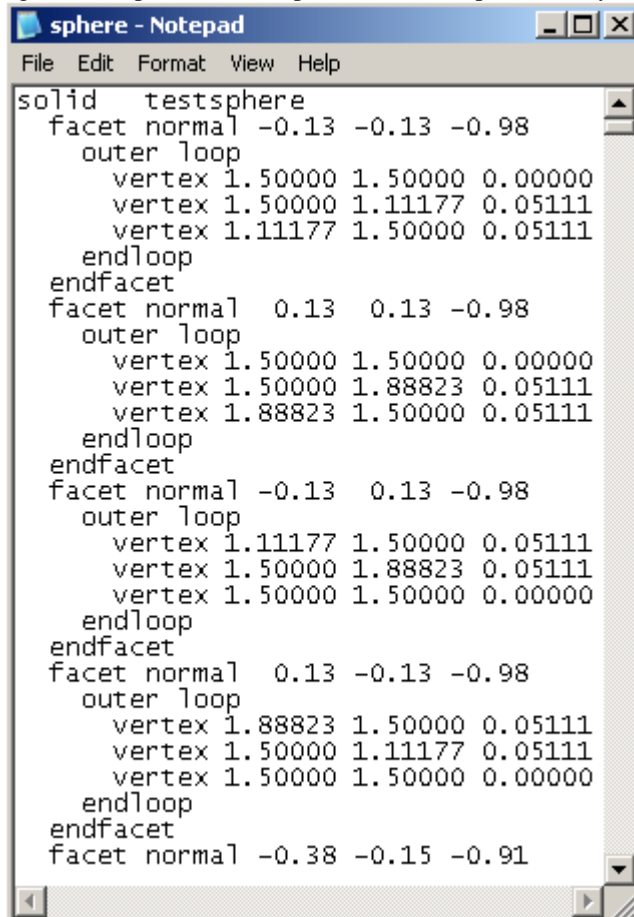


Figure 2

4. Stereolithography (STL) files

Stereolithography files are stored in a format with a .stl extension. When going from a CAD model to a rapid prototyping system, the STL file format is most commonly used. It was developed by 3D Systems, Inc., in the late 1980s. The file itself is in ASCII or binary, with the ASCII being much more readable by humans. An STL file consists of several things: the very first word is solid, followed by the name of the solid being created, and the very last word in the document is endsolid. Between these identifiers, there is a pattern that is followed throughout the file. Each part of the pattern makes up a triangle. The entire file is made up of these triangles, which, together, form the three-dimensional object. First in the pattern comes the vector that is normal to the triangle. This is identified by “facet normal” followed by the coordinates (x,y,z) of where the normal vector points to. The next line has the words outer loop. The next three lines consist of the three vertices that make up the triangle. They are in counterclockwise order, when looking from the normal vector (which denotes the “outside” of the triangle—the part that is touching air, or in a lot of cases, can be seen). At the beginning of each of these three lines is the word vertex, followed by the three coordinates (x,y,z) that make up each vertex. The next line contains the word endloop, and the next line has endfacet. This is continued on for every triangle, until the very end, with a line having endsolid, as was mentioned earlier.

Figure 3 shows a picture of the very beginning of an example STL file that was opened in Notepad, and in Figure 4 is a picture of the sphere that was represented by this file.



```
solid testsphere
facet normal -0.13 -0.13 -0.98
  outer loop
    vertex 1.50000 1.50000 0.00000
    vertex 1.50000 1.11177 0.05111
    vertex 1.11177 1.50000 0.05111
  endloop
endfacet
facet normal 0.13 0.13 -0.98
  outer loop
    vertex 1.50000 1.50000 0.00000
    vertex 1.50000 1.88823 0.05111
    vertex 1.88823 1.50000 0.05111
  endloop
endfacet
facet normal -0.13 0.13 -0.98
  outer loop
    vertex 1.11177 1.50000 0.05111
    vertex 1.50000 1.88823 0.05111
    vertex 1.50000 1.50000 0.00000
  endloop
endfacet
facet normal 0.13 -0.13 -0.98
  outer loop
    vertex 1.88823 1.50000 0.05111
    vertex 1.50000 1.11177 0.05111
    vertex 1.50000 1.50000 0.00000
  endloop
endfacet
facet normal -0.38 -0.15 -0.91
```

Figure 3 An example STL file

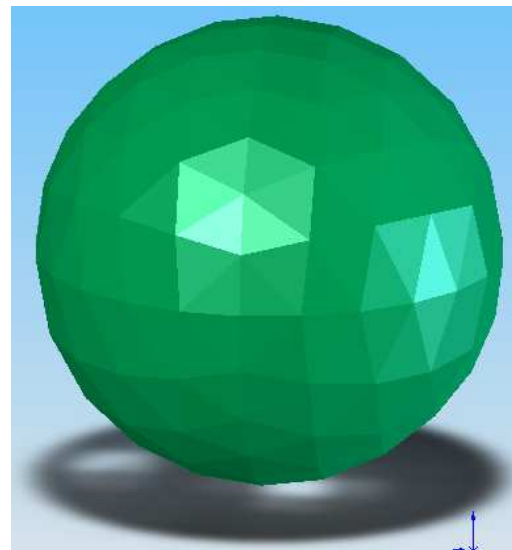


Figure 4 Sphere created from STL file in Fig 3

An important rule to keep in mind when designing STL files is the vertex to vertex rule. It states that each triangle must share exactly two vertices with each of its adjacent triangles. Figure 5 shows an image of a violation of the rule on the left, and a fix to abide by the rule on the right:

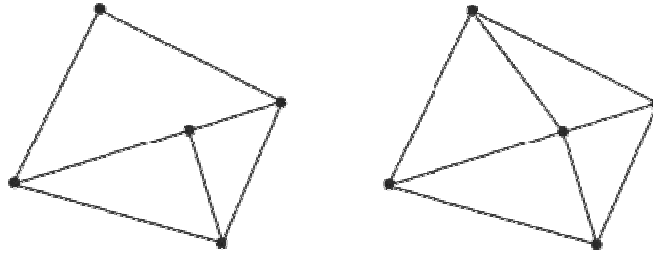


Figure 5

5. Methodology

5.1 Software Written

Small, simple programs written in C++ that read and manipulated STL files were written at first to better understand the STL file format and how it works. The first program created STL files from scratch. The program would ask the user the name of the file, shape, and how many triangles were in the object. It would then go through and ask the user the coordinates for the normal vector, and then the 3 vertices of each triangle, for as many triangles as were indicated.

The second program looked at an existing STL file. It figured out the extents of the volume of the object. Then, simply put, it stretched the object a specified number of units, elongating the object. The purpose of this was to be able to identify and manipulate specific parts of STL files.

The next step was conducting research on how to more efficiently manipulate STL files, and to be able to determine what makes up the undercuts of an object. It needs to be determined how to do this, while only given the information that the STL file contains. Research was then done on fully understanding the mathematics of vectors and intersecting lines and planes in three-space.

5.2 Identifying the Undercut

A possibility currently being looked at is that of extending each individual triangle along the z-axis, either up or down, based on what direction the normal vector for the triangle is pointing. It would be extended beyond the extents volume of the object. The way to determine whether the triangle is completely visible, partially visible/partially hidden, or completely hidden, is to compare it to all of the other triangles. If the extended volume is intersected by one or more other triangles, then it is at least partially hidden.

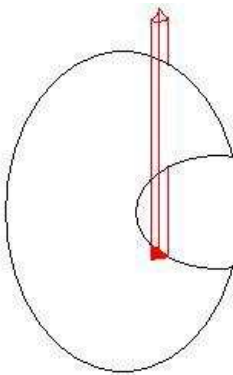


Figure 6 Triangle Extrusion

This is a simple example of a triangle extension. The normal vector of the triangle highlighted points upward, and so the triangle is extended upward. As you can see, the triangle passes through other parts of the object, which would also be triangles. In this case, the triangle highlighted is to some extent hidden.

5.3 Triangle Cases

As mentioned, the three categories for the triangle cases when looking at the extended triangles compared to other individual triangles are the following:

- Completely hidden
- Completely visible
- Partially hidden

There turned out to be numerous cases. For completely hidden, there were 9. For completely visible, there were 7. For partially hidden, the largest category by far, there were 22 cases. They are shown below.

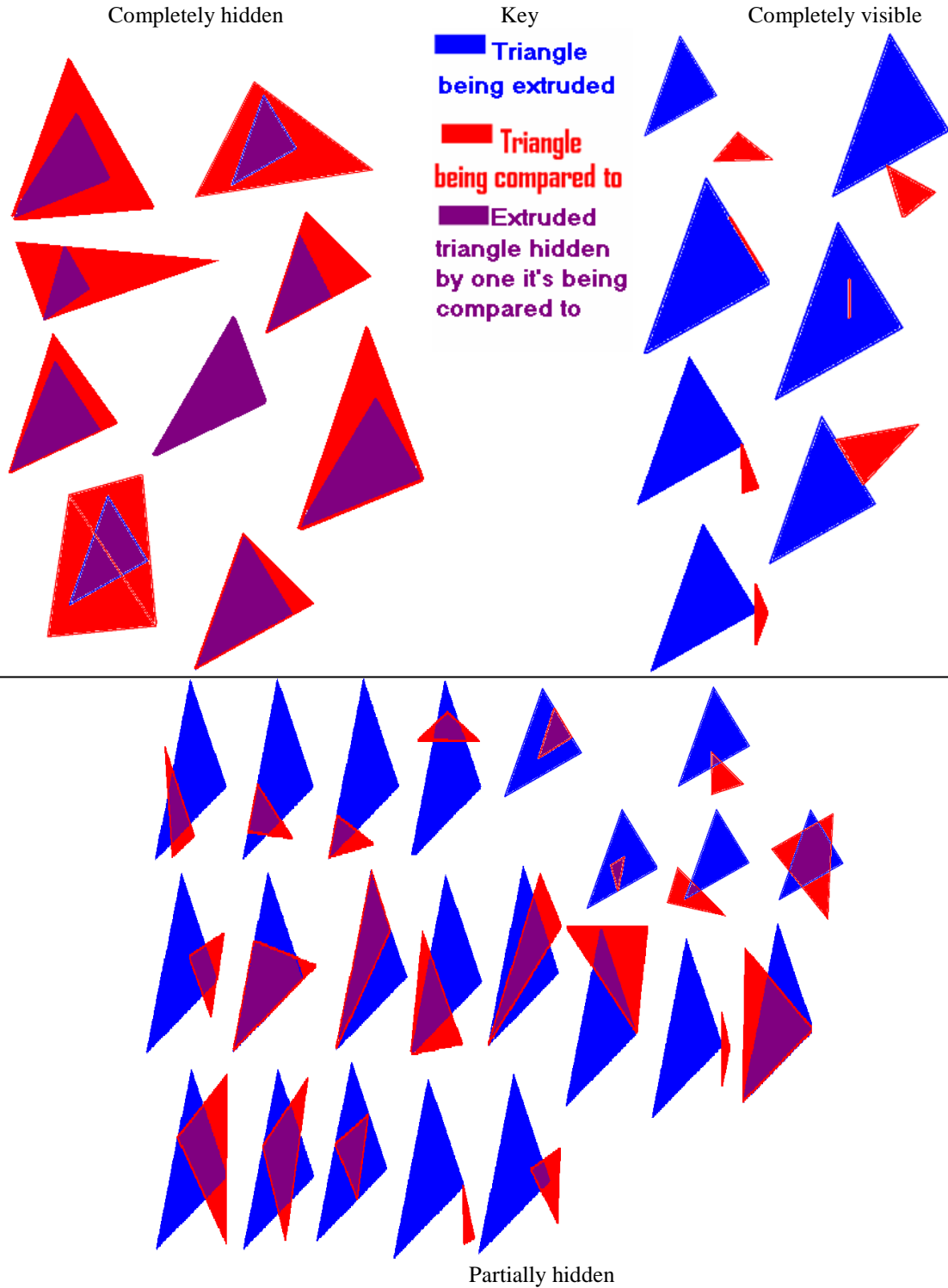


Figure 7 All of the triangle cases

5.4 Shell-Slice Technique Overview

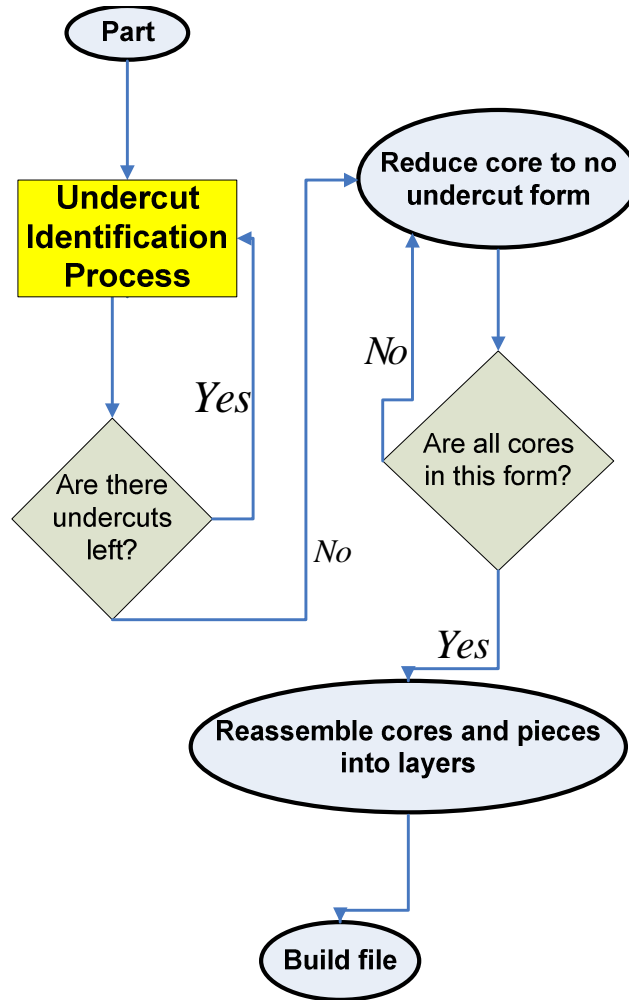


Figure 8

Here is an overview of the Shell-Slice Technique. First, there is the part. It then enters the undercut identification process, which is the main focus of this research. If there are still undercuts, the part goes through the process again, and continues to do so until there are no triangles shaded. The part then goes into a phase of reducing the core to where there are no undercuts. After all of the cores of the object are in this form, they, along with the rest of the pieces, are reassembled into layers. From there, they are sent to build the file.

5.5 Undercut Identification Process

Below, in figure 9, is a flowchart of the undercut identification process.

1. To start out, each triangle is evaluated to determine whether the normal vector points up or down. There are the select few which are put into another category, those that are parallel to the z-axis. These are looked at separately.
2. After splitting the triangles into groups A (facing up) and B (facing down), a subroutine is to be entered in which each triangle is extruded to the extents volume of the object, into a prism shape. This shape is divided into 8 separate triangles.
3. Then, all of the line segments of other triangles that are to be compared to this prism are figured out. From here, there's detailed math that is used to determine whether or not a line segment intersects a specified

triangle. This is shown in detail in figure 10. The output of this math is a value of 0, 1, or a value between 0 and 1. These determine the relationship of the triangle to the prism as completely hiding the prism, partially hiding the prism, or not hiding the prism at all. These are the 3 triangle cases discussed in depth above and shown in Figure 7.

4. After this, all of the partially hidden triangles are trimmed. They are transformed into new triangles—completely hidden and completely visible. Then, all of the hidden triangles are combined with these new hidden ones and are stitched together to reconstruct the missing geometry surface. The core of the object is now made.

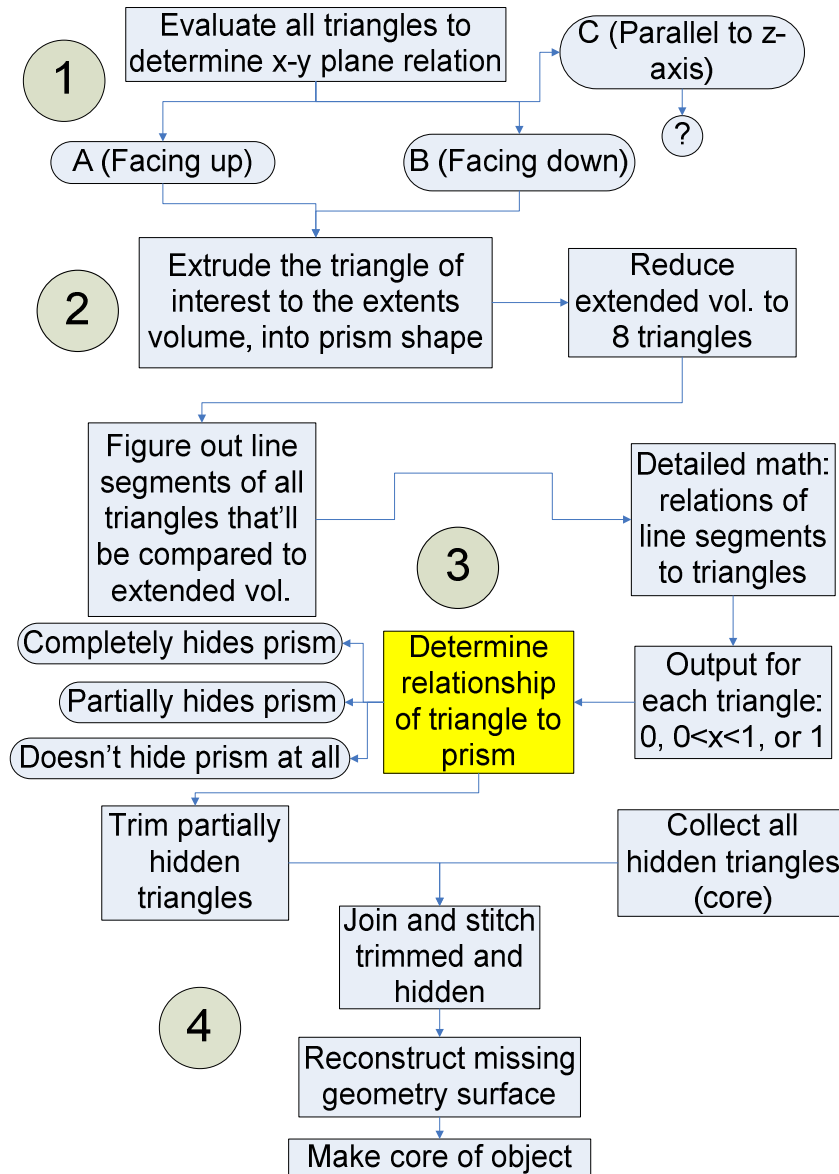
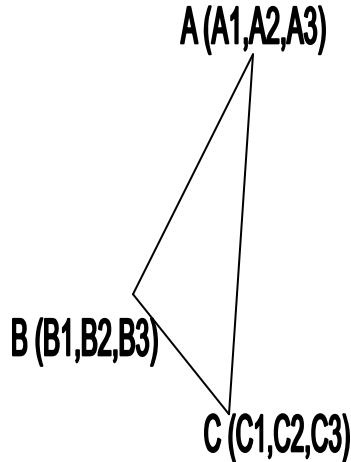


Figure 9

Figure 10a

How to Determine Whether or Not a Line Segment Intersects a Specified Triangle

- I. Figure out the equation of the plane that the triangle lies on
- a. Figure out the vector that is perpendicular to one of the triangle's vertices
- i. \rightarrow \rightarrow
- $BA = (A_1 - B_1, A_2 - B_2, A_3 - B_3)$ $CA = (A_1 - C_1, A_2 - C_2, A_3 - C_3)$
- $D = (X_1, Y_1, Z_1)$ $E = (X_2, Y_2, Z_2)$



- ii. Cross product of vectors D & E

$$D * E = \begin{vmatrix} i & j & k \\ X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \end{vmatrix}$$

\rightarrow

$$N = i(Y_1 * Z_2 - Z_1 * Y_2) - j(X_1 * Z_2 - Z_1 * X_2) + k(X_1 * Y_2 - Y_1 * X_2)$$

$$= F * i + G * j + H * k \quad \text{where F, G, and H are the coefficients figured out by the above equation}$$

Pick any point of the triangle to plug into this to find out the equation of the plane.

Let's pick A. The equation to use for this is $F(x - x_0) + G(y - y_0) + H(z - z_0) = 0$

So: **Equation Q:** $F * x + G * y + H * z + P = 0$ where P = the new coefficient generated by plugging in the point into the equation

- II. Line segment being checked whether it intersects the triangle

- a. Create a normal vector of the line

Endpoints of line: P1: (x1, y1, z1)

P2: (x2, y2, z2)

$$x_3 = x_2 - x_1 \quad y_3 = y_2 - y_1 \quad z_3 = z_2 - z_1$$

N: (x2-x1, y2-y1, z2-z1)

Use the equation: $(x - x_1)/x_3 = (y - y_1)/y_3 = (z - z_1)/z_3$

Plug in the x value for the y and z in the equation in Equation Q, solve for x

Plug in the value received for x into the y = something*x

Plug in the value received for x into the z = something*x

This gives you the point of intersection: (X, Y, Z)

Figure 10b

Figure 10c

- III. Determine whether or not the point of intersection of the plane falls within the triangle.
- a. Back to the triangle's vertices:
 - i. a (A1, A2, A3)
 - ii. b (B1, B2, B3)
 - iii. c (C1, C2, C3)
 - b. $\underline{X} = a*A1 + b*B1 + c*C1$
 $\underline{Y} = a*A2 + b*B2 + c*C2$
 $\underline{Z} = a*A3 + b*B3 + c*C3$

The lines below are a direct quote from Dr. Peter Kuhfittig:

If the numbers are positive, less than 1, and add up to 1, the point, and hence the intersection of line and plane, lies inside the triangle. (We already know that the point lies in the same plane as the triangle.) Now, the numbers must not be negative, but any one of them could be 0 or 1.

If a=1, b=0 and c=0, then the intersection is the vertex (0,0,3).

Suppose c=0, a=1/2, and b=1/2; then the point lies on the side of the triangle joining the top vertex to the vertex on the right. More precisely, the point is the midpoint of the side joining (0,0,3) and (0,2,0).

6. Conclusion

The research done for this project will provide a stepping stone to the next part of putting this into works by translating this process into code for the software.

The algorithm written in figure 10 should provide quite useful in coding the mathematical intensive portions of the software. It lays out the entire process of being able to determine whether or not a line segment intersects a given triangle, which is exactly what this software will need to do when figuring out which triangles are hidden, to determine the core of the object.

There are 38 cases of possible extruded triangle to triangle relations. The mathematical approach to understand the relationship between triangles and line segments will be a critical part of the core extraction software.

The entire project will help with ideas for the shell-slice technique software package that is to be developed in the near future.

7. Acknowledgements

The author would like to express his appreciation to the National Science Foundation and The Milwaukee School of Engineering for sponsoring this project. Special thanks go to Ann Bloor, Betty Albrecht, and Vito Gervasi for supporting this project. Thank you to Dr. Peter Kuhfittig for helping with the math side of the research. Also, thanks goes to all of the Research Experience for Undergraduates (REU) advisors this summer, the Rapid Prototyping Center, and the other REU participants for offering input, advice, and assistance.

This material is based upon work supported by the National Science Foundation under Grant No. EEC-0139142. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. References

Books

1. Ron Larson, Robert P. Hostetler, and Bruce H. Edwards, *Calculus with Analytical Geometry*, 7th ed. (Boston: Houghton Mifflin Company, 718:759).

Web Resources

2. MSOE, "What is Solid Freeform Fabrication?" 2004, <http://msoe.edu/reu/ssf.shtml>.
3. SDSC, "STL Format Description," Tele-Manufacturing Facility Project, <http://www.sdsc.edu/tmf/Stl-specs/stl.html>.
4. Ennex, "The STL Format: Standard Data Format for Fabbers," 1999, <http://www.ennex.com/~fabbers/StL.asp>.

Interviews

5. Kuhfittig, Peter, personal interviews, June 28, 2005 and July 20, 2005