Daniel Flanagan

First Semester Math Project

FST

17 December 2010

**Triangles**

Due to my struggles with trigonometry, I decided to create a program that would solve a triangle in order to get a better grasp for trigonometry. In the past, I have used trigonometric functions for certain functionalities in programs or games, but only because I knew what the outcome would be, but not why. I figured if I created a program to solve triangles, I would have to learn the details of trigonometry and gain an inside understanding of why this or that actually worked. So I did. I created a mostly successful program that will solve and even **draw** a triangle given a few of the triangle’s variables.

First, when developing the program, I decided it would be best to use the “*a, b, c, A, B, C*” method of defining triangles. I needed something standard, but simple to work with. Because of the 6 different available variables (3 sides and 3 angles), I had to allow the program to detect which method to use (SSS, SSA, SAS, AAS, or AAA) in order to solve the triangle. To figure out which method to use, I had the program use the first 3 variables entered into the program before deciding. To figure out which method to use, it simply adds either “S” or “A” to a string (which is some text, like this essay). In the case of SSA, which has two methods (either SAS or SSA), I simply check whether or not the angle is included, that is, if *a* and *c* are defined, it is SAS if the given angle is *B*. In the cases of SSS, SAS, SAA, and AAA, the solution was relatively simple to find. Here are the mathematical functions I used to solve a triangle for each method:

SSS: (let *a, b, c* be the given sides, and *A, B, C* be the unknown angles opposite the side of the same letter for this example)  
*A* = acos((*c2 + b2 -a2)/(2(b)(c)))  
B* = acos((*a2 + c2 -b2)/(2(c)(a)))  
C* = acos((*a2 + b2 -c2)/(2(a)(b)))*  
  
SAS: (let *a, b, C* be the given variables, and *c, A, B* be the unknown variables for this example)  
*c = √(b2 + a2-2(b)(a)cos(C))  
B = asin(b(sin(C)/c))  
A = asin(a(sin(C)/c))*SAA: (let *a, B, C* be the given variables, and *b, c, A* be the unknown variables for this example)  
*A = 180 – B- C  
b = sin(B)/(sin(A)/a)  
c = sin(C)/(sin(A)/a)*AAA: (since it is impossible to know the scale of the triangle, I assume that *a = 1* and solve with SAA)

SSA: (by far the most difficult to implement – let *a, b, B* be the known variables, and *c, A, C* be the unknown variables for this example)

*First, we must find out how many solutions this triangle has. We can that with this bit of code: (See next page)*

if (adjacentSide < oppositeSide)

{

solutions = 1;

}

else if (oppositeSide < adjacentSide)

{

if (knownAngle < 90)

{

if (oppositeSide == triangleHeight)

{

solutions = 1;

}

else if (oppositeSide > triangleHeight)

{

solutions = 2;

}

}

}

else

{

if (knownAngle < 90)

{

solutions = 1;

}

}

*This will tell us how many solutions the triangle has when given* knownSide*,* oppositeSide*, and* knownAngle*.* adjacentSide *is the same as* knownSide*, it is only renamed for legibility. Now that we know how many solutions we have, we can solve the triangle. Obviously, if no solution is found, no further calculations are necessary.*

One Solution:

a,b,B

*A = asin(a(sin(B)/b)*

*C = 180 – A – B*

*c = sin(C)/(sin(B)/b)*

The second solution is found the same way, only with this simple difference:

*A = 180 – asin(a(sin(B)/b)*

Then the triangle is solved and drawn. As you can see, solving for SSA is much more complex than any of the others combined.

Now we know how to solve triangles. After I finished the program, I thought it would be cool to draw the user’s triangle for them so it’s easier to visualize. I ran in to a couple issues which forced me to draw the longest side as the base. In fact, the math for drawing the triangles to the browser is nearly as complex as solving it. If we have a canvas with a width of 490 pixels and a height of 194 pixels and we want the triangle to be centered in that canvas in a square with a minimum margin of 7 pixels, here are the calculations: (NOTE: In an HTML5 Canvas, the point of origin is the top-left corner!)

Get the largest of *a*, *b*, and *c* and make it the baseline. Scale the other legs as such: (NOTE: We really only need to scale one other leg, which would be the next one after the baseline (e.g: if baseline is a, leg is b, if baseline is c, leg is a, etc.)

leg / baseline

This produces a decimal from 0 to 1.

If our baseline is the largest line, we may scale it on the X axis to a full 1.

Now to get the positioning within the canvas, start with this:

Canvas height – (margin x 2)

194 – (7 x 2) = 180 = square

X1 = (canvasWidth / 2) – (square / 2)

This is our starting X position for drawing on the canvas. Starting Y position is calculated similarly. First, however, we need the triangle height scaled to our 180 pixel square. We can take the negative sin of either of our scaled legs (the ones that are a decimal from 0 to 1) and multiply it by our square value of 180 to get the triangle height in pixels.

Height = (-sin(leg)\*180)

Y1 = (canvasHeight / 2) – (Height / 2)

Now that we know where we’re drawing from, we need to find where the line is being drawn **to**, this is also known as the second point of the line.

We know our baseline has a scale of 1, and therefore has a pixel width of 180, our square, so the endpoint x value is easy:

X2 = X1 + 180

Our new Y doesn’t change since this is the base of our triangle. It has no change in Y value.

Y2 = Y1

Now we simply draw from (X1, Y1) to (X2, Y2) with the color of the line it is, depending on whether it was a, b, or c.

Now we need to find where to draw this next leg. We already know where to start from. Since a triangle’s line segments are always touching, we can start from our current position, newX by newY. Where do we draw **to** though? Simple. We know the height of the triangle, so we know the Y change from the baseline, which is where we’re at know, so:

Y3 = Y2 (or Y1) + Height

Now we need to find the X difference between X2 and X3.A simple negative cosine should do the trick.

X3 = X2 – (cos(leg)\*180)

Now we simply draw from (X2, Y2) to (X3, Y3) with the color of the line it is, depending on whether it was a, b, or c.

The rest is even easier. Since we know a triangle’s lines all connect, we can draw from (X3, Y3) to (X1, Y1). Triangle drawn and everything is finished!

And so my mission was complete! I had learned how to solve triangles, learned how to draw triangles to the screen using the trigonometric functions, and best of all, I had the feeling I had mastered what I knew about trigonometry thus far. I have uploaded the entire project as a downloadable zip file for any who are interested. You can find the links to both the site and the download link on the final page of this document.

Project Source Code: <http://dl.dropbox.com/u/12199979/source.zip>

Project Site: <https://dl.dropbox.com/u/12199979/trigproj/index.html>