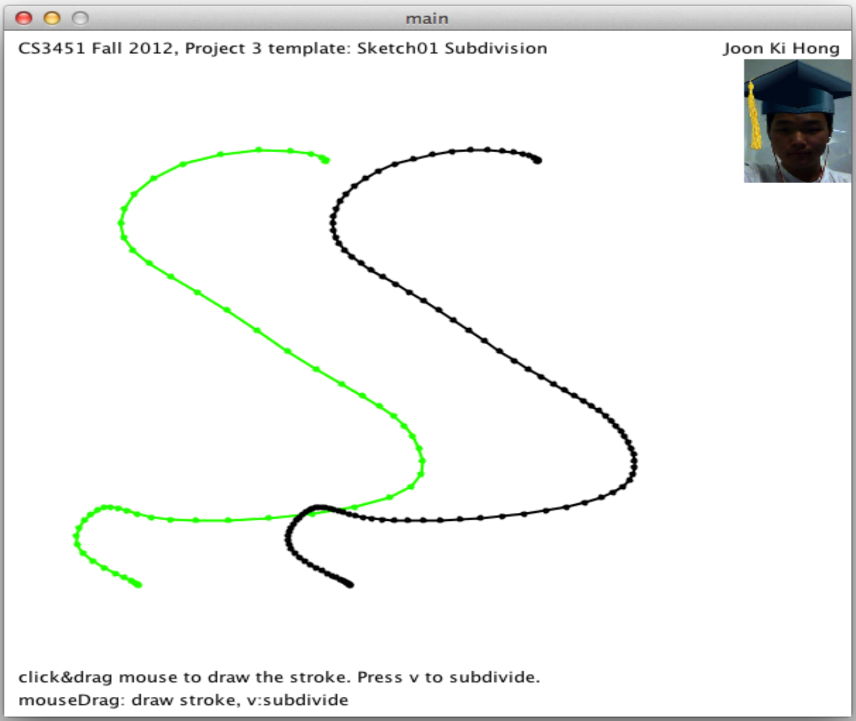
**Project 3 – P03a Stroke Subdivision**
**CS 3451 Fall 2012**
**Joon Ki Hong**
**October 4, 2012**

**Problem Statement**: Create a program that lets the user draw a curve and also allows the user to subdivide the curve and see the incremental changes

**Screen Capture**:



click&drag mouse to draw the stroke. Press v to subdivide.
mouseDrag: draw stroke, v:subdivide

**Snippet of Code:**

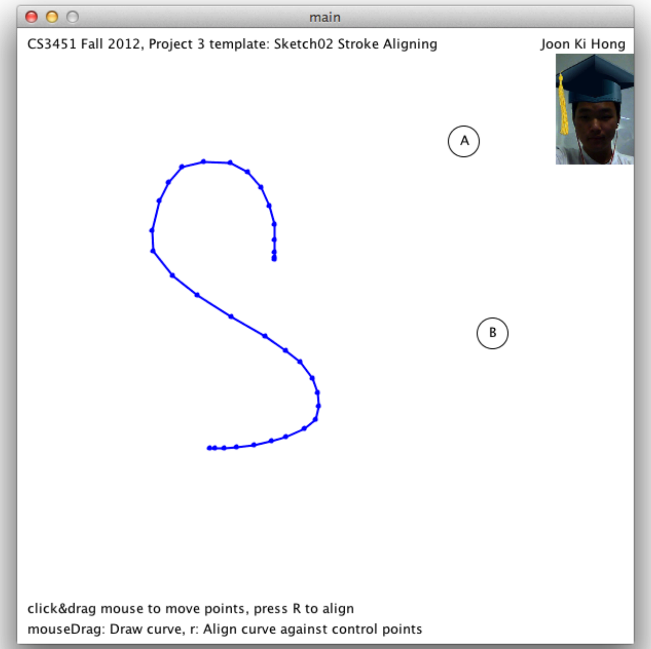
```
Subdivide: pt H1 = P( P(G[i],t,G[i+1]) , t/2 , P(G[i+1],t-1,G[i+2]) );
    pt H2 = P( P(G[i+3],t,G[i+2]) , t/2 , P(G[i+2],t-1,G[i+1]) );
    pt M = P(H1,H2);
Resample: int n = nv/num;
   pt[] R = new pt[maxnv];
   int count = 0;
   for(int i=0;i<nv;i=i+n){
     R[count] = new pt(G[i]);
     count++;
   }
```

**Approach/Critique:** I approached this program in a straight forward manner. For the subdivision, I took the average points of two parabolas ABC and DCB: H1 and H2. Then I took the midpoint of that and inserted that point into the array. There were two special cases for this however. The first and last section of the curve had to the average of a single parabola. This implementation of subdivision is a very simple way to enhance the detail of a curve if a curve didn't have many vertices and had visible straight edges along the curve. Subdivision however, as I noticed, can quickly slow the execution of the program with the increasing number of times this method is used on a curve.

**Project 3 – P03b Stroke Alignment**
**CS 3451 Fall 2012**
**Joon Ki Hong**
**October 4, 2012**

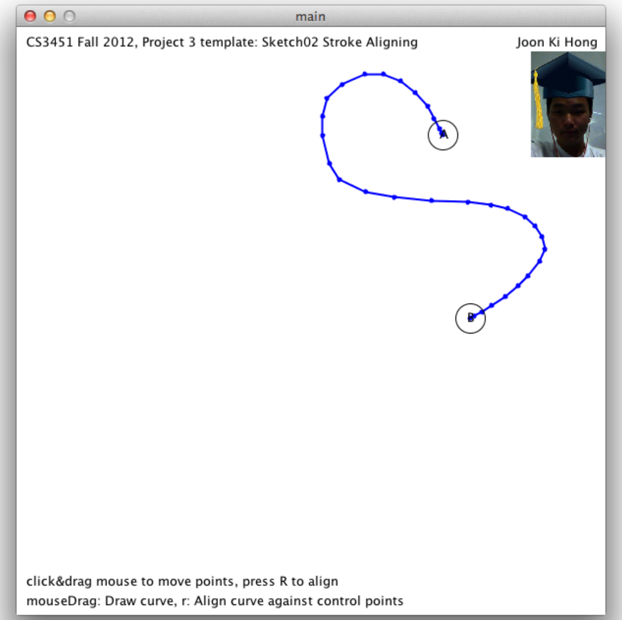
**Problem Statement**: Create a program that lets the user draw a curve on the screen and allow the user by pressing the r key, to align the curve between points A and B

**Screen Capture**:



CS3451 Fall 2012, Project 3 template: Sketch02 Stroke Aligning    Joon Ki Hong

click&drag mouse to move points, press R to align
mouseDrag: Draw curve, r: Align curve against control points



CS3451 Fall 2012, Project 3 template: Sketch02 Stroke Aligning    Joon Ki Hong

click&drag mouse to move points, press R to align
mouseDrag: Draw curve, r: Align curve against control points

**Snippet of Code:**

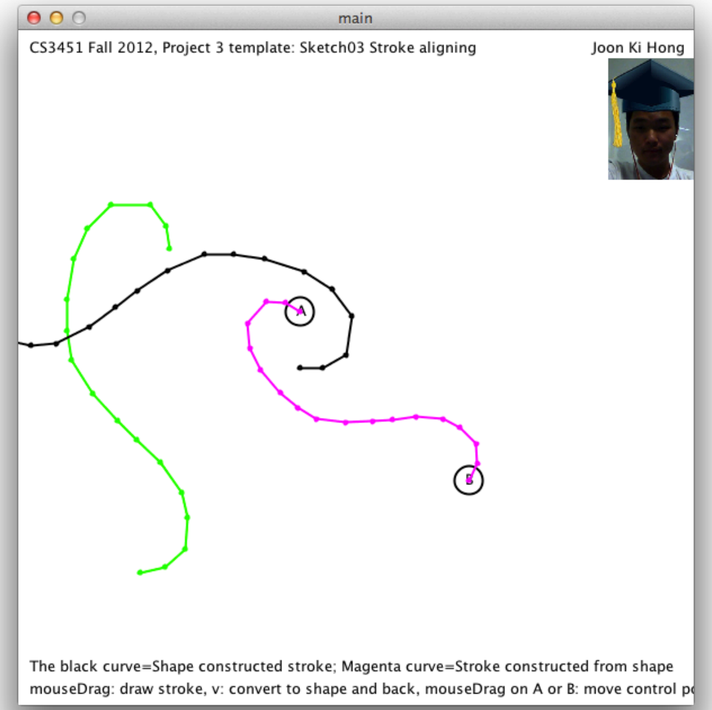
```
x = dot(CD,CV)/(nCD*nCD); //calculate relationship of x and y in original curve
y = -det(CD,CV)/(nCD*nCD);
newArray[i] = A.add(x,AB).add(y,RAB); //determine and save new x,y values
```

**Approach/Critique:** I approached this program also in a straight-forward manner. I recorded the x and y relative to the axis defined by the first point in the first curve [C] and the last point in the first curve [D]. I divided these values by the length of CD to get fractions. I then used these x and y fractions for each point to recreate the stroke but using different values for the first and last points (A,B). Doing this we get the same stroke but constrained by points A and B. This method of scaling, translating, and rotating a stroke is extremely simple and intuitive. It uses simple algebra and common sense to recreate the curve to make it seem like it rotated, translated, and scaled. The only disadvantage may be the fact that it doesn't necessarily do what it's showing (rotating, translating). There's no way of showing any intermediate steps of rotating, and translating the curve to get the end result. Computational wise, this code is very efficient.

**Project 3 – P03c Stroke Alignment**
**CS 3451 Fall 2012**
**Joon Ki Hong**
**October 4, 2012**

**Problem Statement**: Create a program that lets the user draw a curve on the screen. The user then presses the v key to show the converted stroke into a shape that has its first edge parallel to the x axis and centered, and the recreated stroke from the shape that is also constrained by points A and B.

**Screen Capture**:



main

CS3451 Fall 2012, Project 3 template: Sketch03 Stroke aligning

Joon Ki Hong

The black curve=Shape constructed stroke; Magenta curve=Stroke constructed from shape
mouseDrag: draw stroke, v: convert to shape and back, mouseDrag on A or B: move control p

**Snippet of Code:**

Convert to Stroke:

```
nextPoint = new pt(nextPoint.x+cos(relAngle)*lengths[v],nextPoint.y+sin(relAngle)*lengths[v]);
```

Convert to Shape:

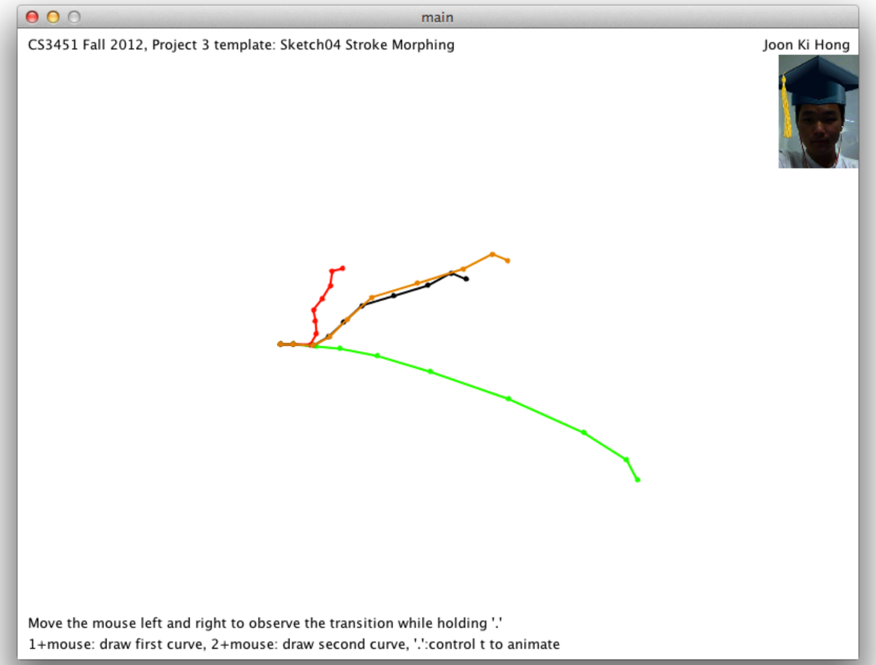
```
newShape.addLength(G[i],G[i+1]);
newShape.addAngle(G[i],G[i+1],G[i+2]);
```

**Approach/Critique:** This approach was pretty straight -forward as well. I converted the user drawn stroke by saving the lengths and angles. I created a draw method for the shape class that keeps track of the relative angle by adding each angle to it each time a length is drawn. First we needed a starting point, which was chosen to be the center of the screen with the first edge parallel to the x-axis. The equation I used was pt( x+cos(relative angle)*length, y+sin(relative angle)*length). Using this, I was able to draw the shape. To convert it back, I noticed that I was already computing each vertex as I was converting the stroke to a shape. I merely added these vertices to an array and used the alignment method to align the new stroke with A and B. This method of saving the stroke by lengths and angles is very clever. I don't see any real disadvantages to using this method, although the code can become a little complicated having to use 2 arrays to store the angles and lengths. This method is a neat way of showing 2 different representations of the same curve.

**Project 3 – P03d Stroke Morphing**
**CS 3451 Fall 2012**
**Joon Ki Hong**
**October 4, 2012**

**Problem Statement**: Create a program that lets the user draw 2 curves on the screen. The first curve represents the beginning stroke and the second represents the last. The user then holds the '.' Key and drags his/her mouse left to right to control the time variable. Doing this, will allow the user to see the morphing take place between two implementations. The black morphing curve represents the exponential interpolation of lengths where as the orange curve represents the linear interpolation of lengths.

**Screen Capture**:


main
CS3451 Fall 2012, Project 3 template: Sketch04 Stroke Morphing
Joon Ki Hong
Move the mouse left and right to observe the transition while holding '.'
1+mouse: draw first curve, 2+mouse: draw second curve, '.':control t to animate

**Snippet of Code:**

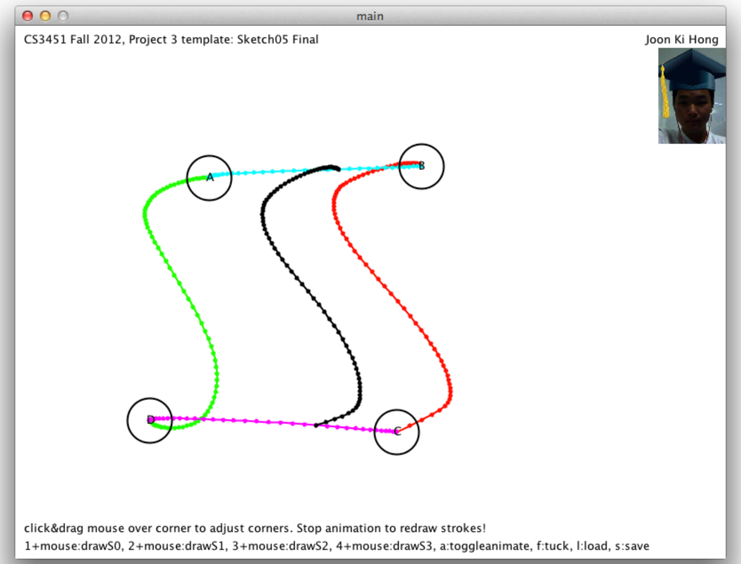
```
WAG (exponential):
St.angles[i] = Si0.angles[i]*(1-t)+t*Si1.angles[i];
St.lengths[i] = Si0.lengths[i]*pow(Si1.lengths[i]/Si0.lengths[i],t);

WAG linear:
St.angles[i] = Si0.angles[i]*(1-t)+t*Si1.angles[i];
St.lengths[i] = Si0.lengths[i]*(1-t)+t*Si1.lengths[i];
```

**Approach/Critique:** This approach was simple as well. Since we have the angles and lengths, we can interpolate these parameters however way we want to morph it into another curve. Simply, we use a weighted average to compute the new angles. The lengths however, we use an exponential equation. (M0*[M1/M0]^t). I implemented both exponential and linear to give an idea of the differences. The exponential morph seemed to have a more natural progression where the lengths would ease into the ending curve and speed up at the very end. The linear curve would have a linear behavior and have a constant rate of change in lengths, which seemed less natural.

**Project 3 – P03e Stroke Morphing Bounded by 2 Paths**
**CS 3451 Fall 2012**
**Joon Ki Hong**
**October 4, 2012**

**Problem Statement**: Create a program that lets the user draw 4 curves on the screen by holding 1,2,3,4 respectively. The first curve represents the beginning stroke and the second represents the last. The 3rd and 4th strokes represent the paths that the morph is constrained by. The whole thing is then animated to show the progression between the first and second curve

**Screen Capture**:



CS3451 Fall 2012, Project 3 template: Sketch05 Final

Joon Ki Hong

click&drag mouse over corner to adjust corners. Stop animation to redraw strokes!
1+mouse:drawS0, 2+mouse:drawS1, 3+mouse:drawS2, 4+mouse:drawS3, a:toggleanimate, f:tuck, l:load, s:save

**Snippet of Code:**
**Animate process:**
```
St = WAG(S0,S1,f).scaleRotateTrans(S2.G[i],S3.G[i]);
        if (i>=S2.nv-1){
                iDelta = -iDelta;
        }
        if(t>=4){
                t=0;
                i=0;
                iDelta=1;
       }
        i+=iDelta;
       t+=2.0/(S2.nv-1);
       f=sin(t*PI/4);
```
**Approach/Critique:** This approach was slightly complicated. I allowed the user the draw 4 curves by holding 1,2,3,4 and dragging their mouse. The curves are immediately aligned with each other, refined, subdivided, and resampled. The user may drag the corners of the curves to adjust them. I basically calculated the morph between the beginning and last using the WAG function from the previous sketch. I then aligned this with the points along the path curves and incremented the t value for WAG and the index value for the path points. My implementation was complicated in that I had to synchronize the t and index values so that when I'm aligning the morph along the path curves, I show the correct morph at

that event. I knew how many points were on the paths and that they have the same number of points, since they were resampled. I used simple algebra to synchronize the time and index values as shown above in the snippet of code.