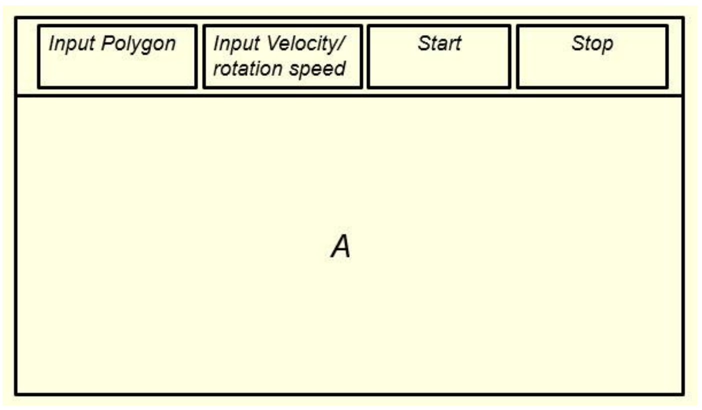
CS633 Computer Animation
Homework Assignment 1 (40 points)
Due: 1/30/2018

1. Your assignment for the first question is to download a copy of the “animated 2D bouncing sample program”, compile it, test run it, fix a problem of this sample program, and turn a copy of your revised version of the sample program in on the due date. The reason for doing these is to get yourself warmed up for the first programming assignment which requires you to develop an openGL program to perform animated 3D bouncing of a polyhedra and an embedded balloon represented by a closed, B-spline surface.

   The sample program does the following things. First, it displays a GUI on screen as follows:



   The four small rectangular areas are supposed to act as menu buttons. The first menu button is for the user to show or hide an “H” shaped polygon that contains a closed, composite Bezier curve of degree 2. By clicking on the second menu button (the one with a label "Input Velocity/Rotation Speed"), the user can input velocity and rotation speed of the polygon by clicking at two points of A to input a vector and another two points to input a degree to the program. By clicking on the third menu button (the one with a label "Start"), the user starts the animation process and by clicking on the fourth menu button (the one with a label "Stop"), the user stops the animation and resets the screen. The animation can be repeated as many times as possible.

   (Note: OpenGL is a Graphics API, not GUI, so you don't have any Buttons, Textboxes in OpenGL. But there are several external GUI libraries that will allow you to create buttons, textboxes, drop-down menus, etc. For example, http://en.wikipedia.org/wiki/GLUI)

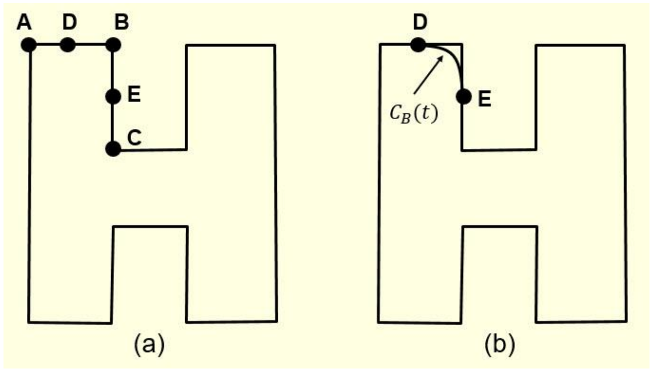
The closed, composite Bezier curve inside the “H” shaped polygon is created in two steps:

a. Compute mid-point of each edge of the polygon

b. For each vertex of the polygon, construct a Bezier curve of degree two using the vertex and mid-points of the adjacent edges as control points

For instance, in part (a) the following figure, once the midpoints of edge AB and edge BC have been computed, say D and E, respectively, one can define a Bezier curve segment of degree two for vertex B as follows (see part (b) of the following figure):

$$C_B(t) = (1-t)^2\mathbf{D} + 2t(1-t)\mathrm{B} + t^2\mathrm{E}, \qquad 0 \le t \le 1$$

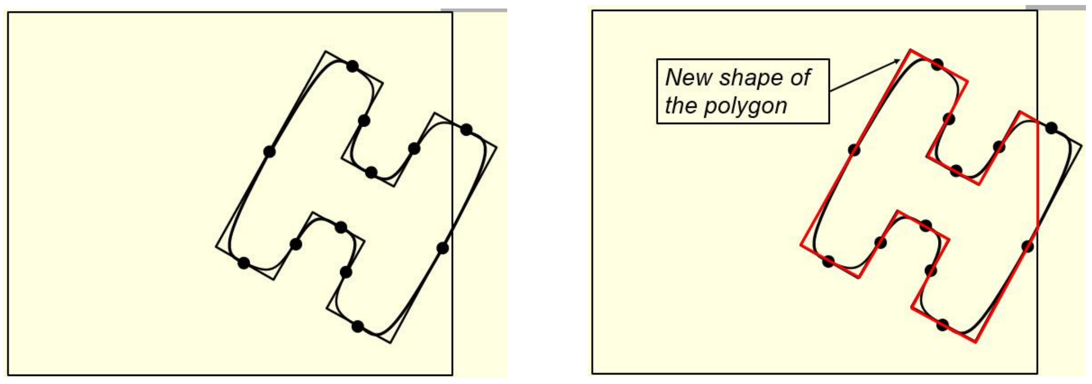


By constructing a Bezier curve segment of degree two this way for each vertex of the polygon, one gets a closed smooth composite Bezier curve.

Clicking on the third menu button starts the animation using the velocity (the vector defined by the first input point (start point) and the second input point (end point)) and rotation speed (number of horizontal pixels between the third and the fourth pixels) about the centroid of the polygon input by the user.

The program does not bounce the polygon and the Bezier curve immediately after one of the vertices touches a wall. The program starts to bounce the polygon and the Bezier curve only when the "centroid" of the polygon hits the wall.

This means the sample program will need to clip the polygon against the wall periodically and update the corresponding Bezier curve accordingly (see the following figure). At some point, the polygon and the associated Bezier curve might be broken into several smaller, disjoint polygons and Bezier curves. Clipping is done using the Weiler-Atherton's algorithm.



The sample program uses the following function in the implementation of this work:

```
glutIdleFunc(animation)
```

This function registers a callback function 'idle' for timeouts and when the program is "idling". The function 'idle' will be called whenever there is nothing else to do (no other events are pending). If each time 'idle' is called the program renders a new scene, the window is continuously animated. Event processing happens between calls to the function 'idle', so be careful not to spend too much time in your idle function or you risk compromising your program's interactivity. Only one idle function can be registered at a time. If you call 'glutIdleFunc()' with a NULL, the idle function is disabled. Idle callbacks are very much like X Toolkit work procedure. Here is an example idle callback for animating a scene:

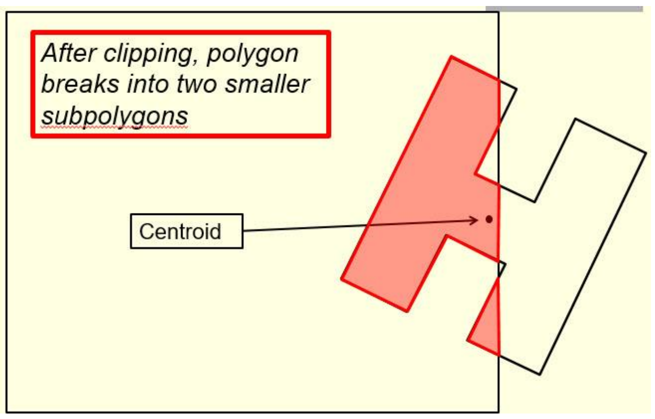
```
void animation( void ) {
      advanceSceneStateOneFrame();
      glutPostRedisplay();
}
```

Notice that the 'idle' routine does not actually render using OpenGL. 'idle' calls the 'advanceSceneStateOneFrame()' routine to update the program's state variables determining how the scene should be rendered. The 'glutPostRedisplay()' tells GLUT that the window's display callback should be triggered; that is, a redisplay is necessary. This ensures that the window is re-rendered. The display callback will redisplay the scene based on the state variables updated by 'advanceSceneStateOneFrame()'.

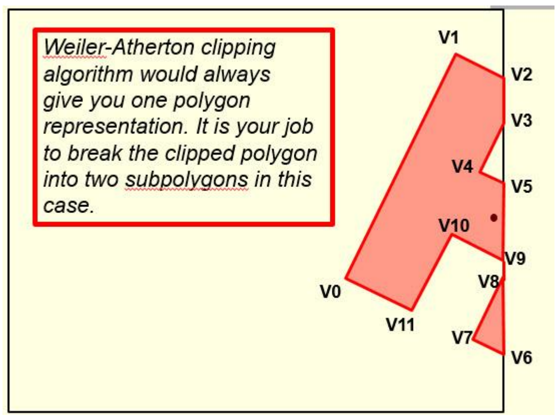
Once you get the sample program to work, try different velocities and rotation speeds to identify a problem of the sample program and have that problem fixed. Turn a copy of your revised program with the solution of Question 2 in on the due date.

Hint:

The polygon clipping process can be performed one window edge at a time, you don't need to use the general algorithm. The following is an example that the polygon after clipping breaks into two smaller sub-polygons.



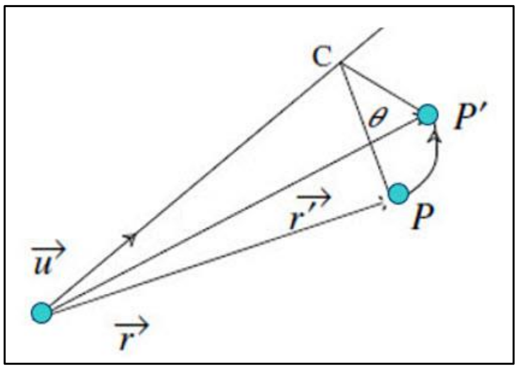
Note that the Weiler-Atherton algorithm would always give you a single polygon representation (see below). It is your job to break the clipped polygon into two sub-polygons in this case.


Weiler-Atherton clipping algorithm would always give you one polygon representation. It is your job to break the clipped polygon into two subpolygons in this case.
V1
V2
V3
V4
V5
V10
V9
V8
V0
V11
V7
V6

2. Fixed angle representation $(\theta_x,\ \theta_y, \theta_z)$ has problems when representing an arbitrary orientation and performing orientation interpolation. Using quaternions to represent points and rotations can avoid those problems, but it seems that rotation implemented

using quaternions cannot be integrated with transformations implemented using homogeneous coordinates. Is this so? In the following figure, let

$\vec{u} = (u_x, u_y, u_z)$	- rotation axis, a unit vector
$\vec{r} = (r_x,\ r_y,\ r_z)$	- vector to be rotated
$\overrightarrow{r'} = (r_x{}',\ r_y{}',\ r_z{}')$	- vector after rotation
θ	- rotation angle



Show that if $[0,\ \overrightarrow{r'}] = [cos\left(\frac{\theta}{2}\right),\ sin\left(\frac{\theta}{2}\right)\vec{u}] \cdot [0,\ \vec{r}] \cdot [cos\left(\frac{\theta}{2}\right), -\ sin\left(\frac{\theta}{2}\right)\vec{u}]$ then we have

$$\begin{bmatrix} r_x{}' \\ r_y{}' \\ r_z{}' \\ 1 \end{bmatrix} = M \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix}$$

for a 4x4 matrix M. What does this mean?