CS560-460 Spring2018

Assignment 2

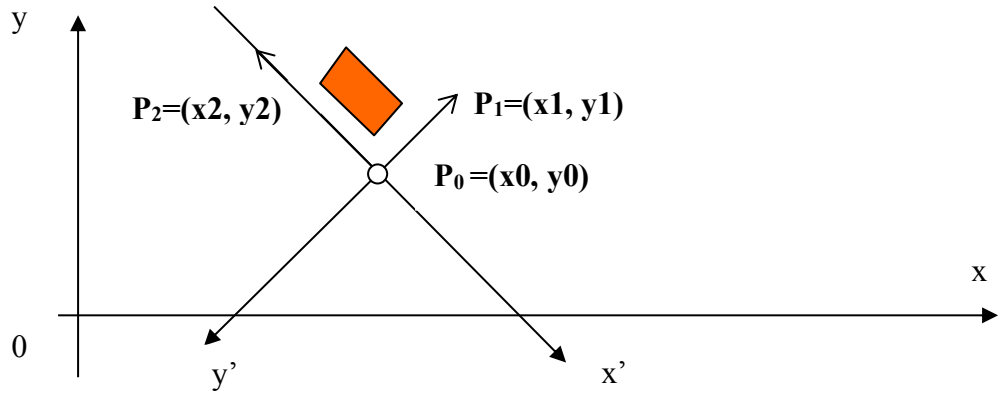
(Due on Feb 28, 2018 by 11:59pm)

Topics: 2D transformation, clipping, and filling

Part A: Theory part; Part B: Programming part.

(Part A) Theory Part (48%)

(1) (7%) Given two perpendicular vectors P_0P_1 and P_0P_2, derive a transformation matrix which can transform object description from xy coordinate system to x'y' coordinate system.



Answer:

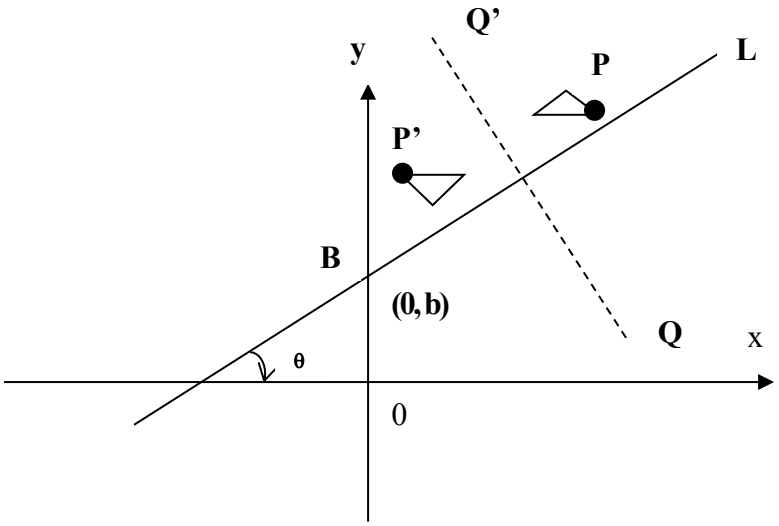
Translate: $\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Rotate: $\begin{bmatrix} \frac{x_1-x_0}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & \frac{y_1-y_0}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & 0 \\ \frac{y_0-y_1}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & \frac{x_1-x_0}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Reflection: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

So, matrix = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{x_1-x_0}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & \frac{y_1-y_0}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & 0 \\ \frac{y_0-y_1}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & \frac{x_1-x_0}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}$

(2) (7%) Let line L have a y intercept (0, b) and an angle of inclination θ (with respect to the x axis). Describe the transformation M which reflects an arbitrary point P in the triangle around the line QQ' (e.g., reflect P to P'). Note that the point Q is known, and Q' is the reflection of Q around line L.



Answer:

L line: y = tanθ * x + b;

C: (Qy -Cy)/ (Qx -Cx) = -tanθ

Cy = tanθ*Cx+b;

⇨ Cx = (Qy-b+tanθ*Qx)/2tanθ Cy = (Qy+b+tanθ*Qx) /2;

Tx = - Cx; Ty = - Cy;

M = Trans(-Tx,-Ty) ·Roat(θ) ·Ref ·Roat(-θ) ·Trans(Tx,Ty)

=

$$\begin{bmatrix} 1 & 0 & \frac{q_y-b+\tan\Theta}{2\tan\Theta} \\ 0 & 1 & \frac{q_y+b+\tan\Theta}{2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} 1 & 0 & -\frac{q_y-b+\tan\Theta}{2\tan\Theta} \\ 0 & 1 & -\frac{q_y+b+\tan\Theta}{2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

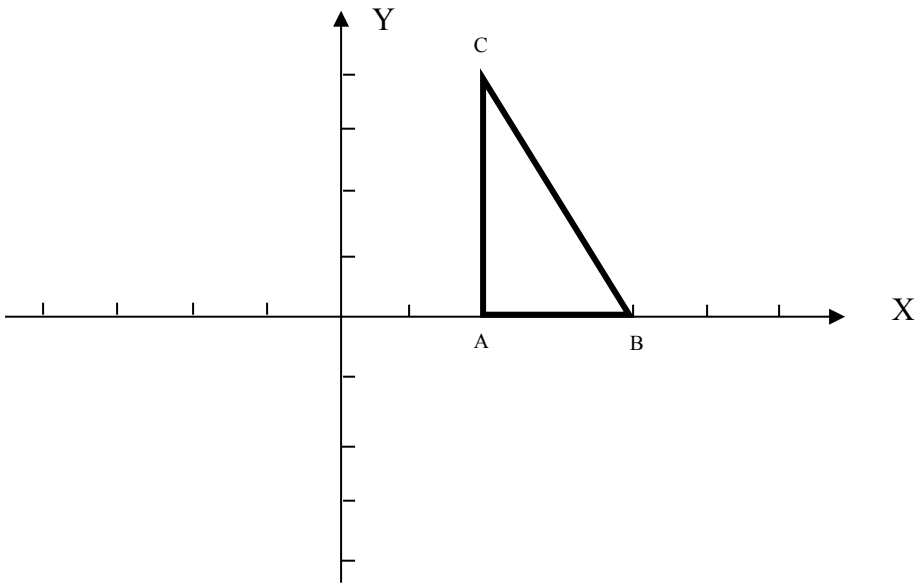
(3) (8%):

Given a triangle (ABC), where A=(2,0), B=(4,0), C=(2,4). Apply the following two transformation sequences (a) and (b) to the original triangle ABC individually, and generate two new triangles A1B1C1 and A2B2C2. Plot the two new triangles on the figure.

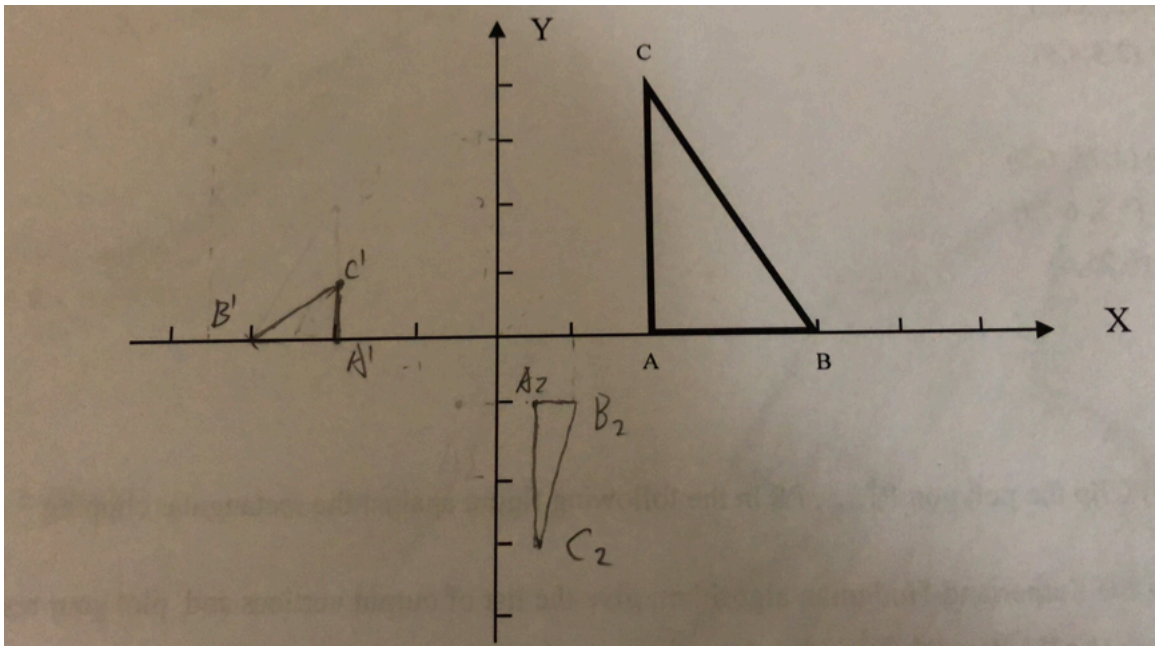
(a) T(-1, 0) S (1/2, 1/4) R (90°) Rf(y,x)

(b) R (90°) T (-1, 0) S (1/2, 1/4) Rf(y, -x)

Note: The transformations in the sequence are performed in the right-to-left order. Rf(y,x) refers to the reflection around a line y=x; Rf(y, -x) refers to the reflection around a line y= -x.

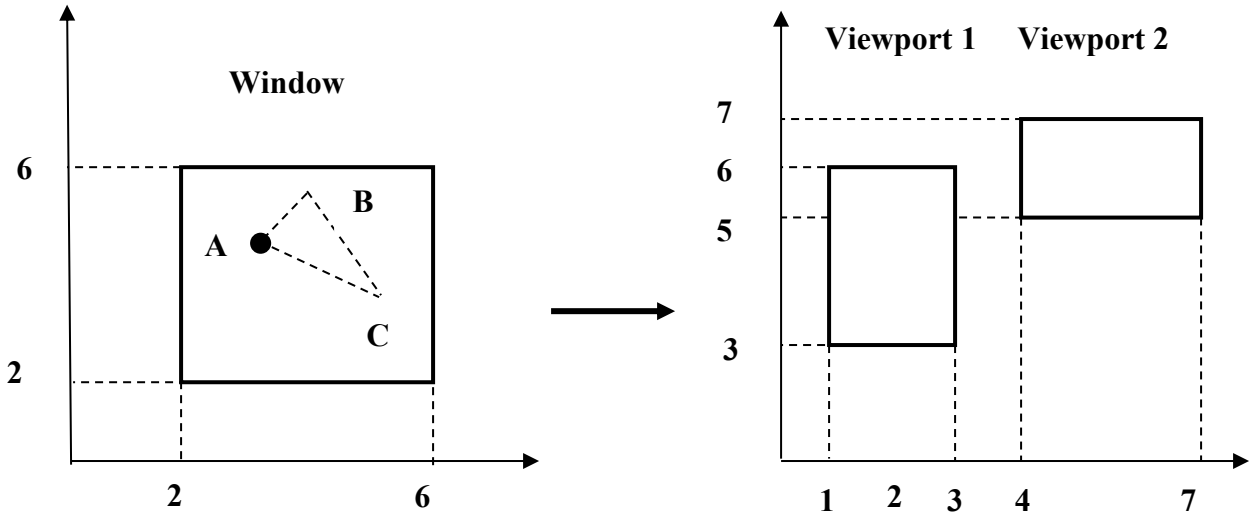


Answer:



(4) (6%) Given a triangle ABC in a window, after the window-to-viewport mapping, ABC is mapped to A'B'C' in the viewport 1, and to A"B"C" in the viewport 2. Derive the coordinates of A', B', C',A", B", C", and draw the triangle A'B'C' and A"B"C"

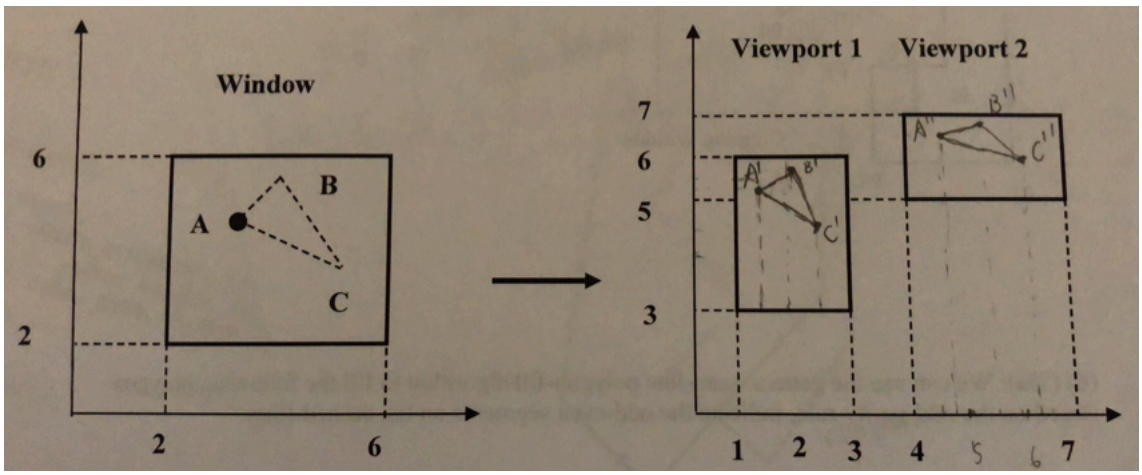
A = (3, 5); B=(4, 5.5); C=(5, 4)



Viewport1: T(1, 3) S (1/2, 3/4) T(-2, -2)

Viewport2: T(4, 5) S (3/4, 1/2) T(-2, -2)

Answer:



A’ = (1.5, 5.25)

B’ = (2, 5.62)

C’ = (2.5,4.5)

A’’ = (4.75, 6.5)

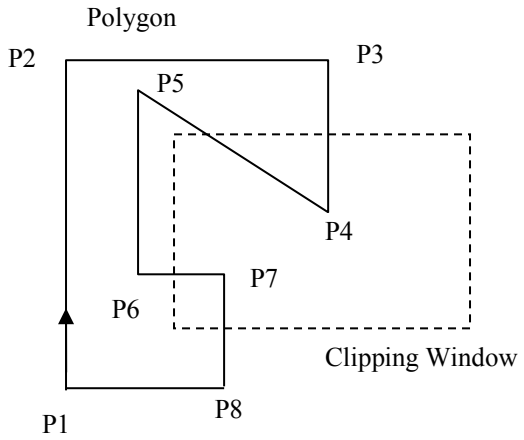
B’’ = (5.5, 6.75)

C’’ = (6.25,6)

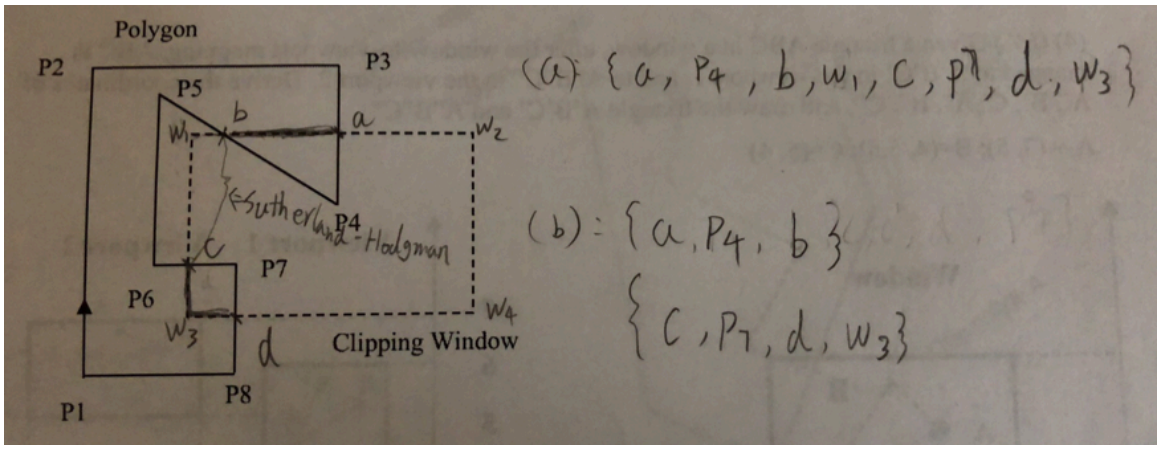
(5) (8%) Clip the polygon P1,…, P8 in the following figure against the rectangular clipping window.

(a)Using the Sutherland-Hodgman algorithm, give the list of output vertices and plot your result.

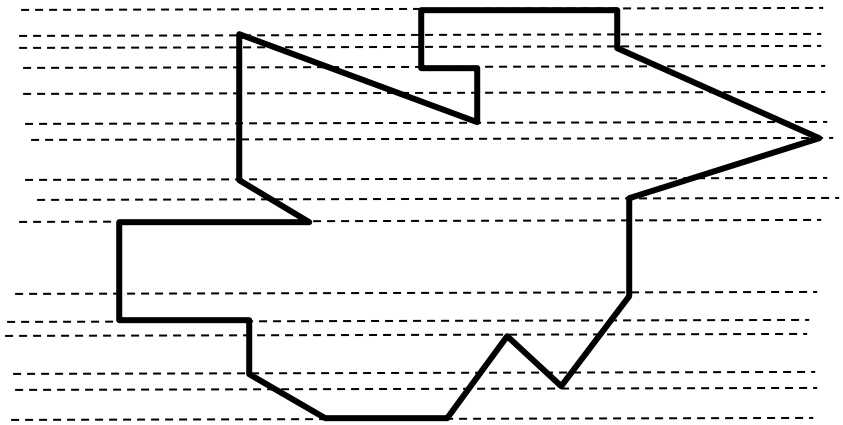
(b) If using the Weiler-Atherton algorithm instead, give the list of output vertices and plot your results.



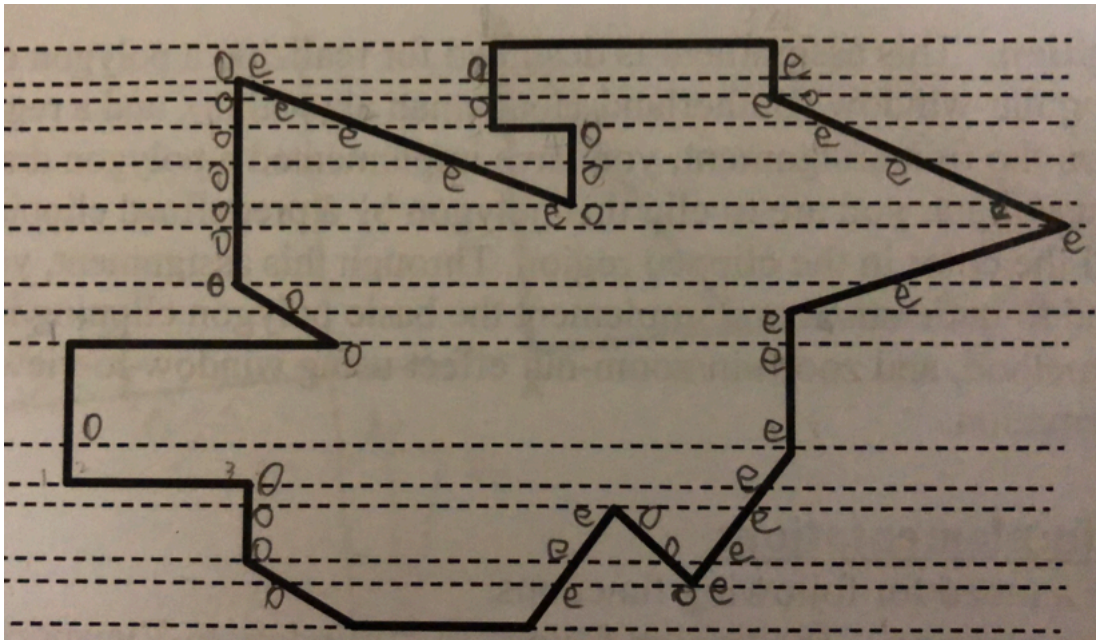
Answer:



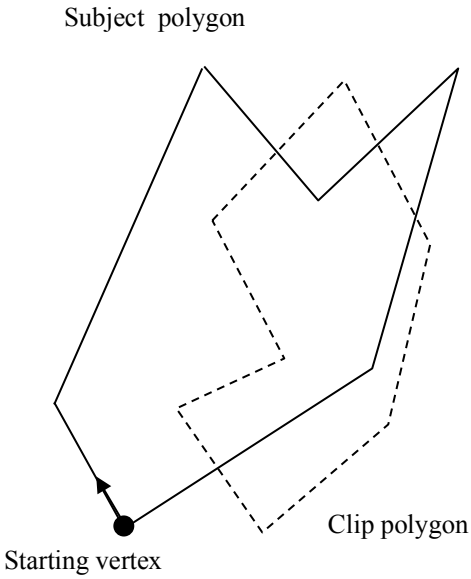
(6) (7%): We can use the general scan-line polygon-fill algorithm to fill the following polygon. Based on the odd-parity rule, indicate the odd-even segments on the dashed-lines.



Answer:

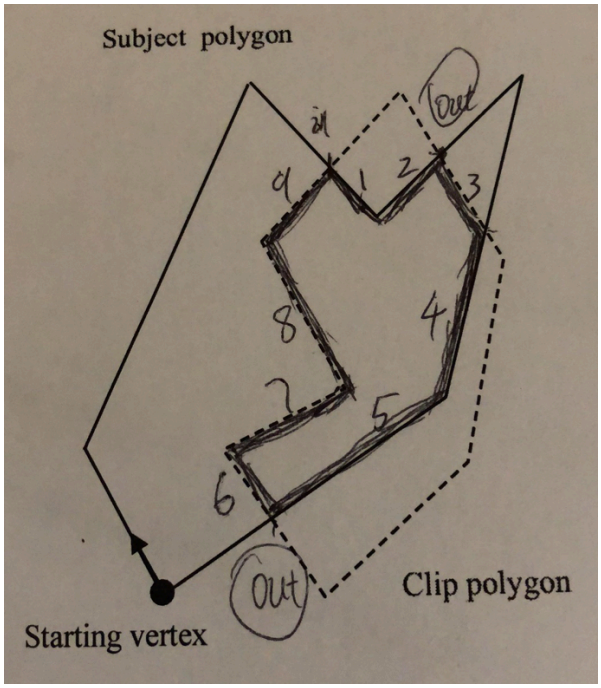


(7) (5%) Use the Weiler-Atherton algorithm to clip the polygon against another polygon as follow, plot the result. Indicate the turning points that switch the subject polygon edges to the clip polygon edges. Number the order for edges that you visited.



:

Answer:



(B) Programming Part (52%) plus extra point (10%):

1. Polygon clipping, region filling, and window-to-viewport transformation

- **Description:** This assignment is designed for realizing a polygon clipper against a rectangular window (Sutherland-Hodgeman algorithm), and a region filler. Based on the first assignment, you have implemented a polygon drawing through the mouse input, you are to clip this polygon by a predefined clipping window, then fill the color in the clipped region. Through this assignment, you are expected to understand and implement the basic polygon clipping method, region filling method, and zoom-in/zoom-out effect using window-to-viewport transformation.

- **Your implementation:**

Create a menu for following functions:
(a) "polygon clipping" (b) "region filling" (c) "Window-to-Viewport Mapping"

(1) Draw a clipping rectangular window by dashed lines (using OpenGL). The window size is determined by yourself (e.g., 150 by 150 pixels)
(2) Draw a polygon (with solid lines) around the clipping window through the mouse input (using OpenGL)
(3) When choose the menu "polygon clipping", perform the polygon clipping procedure (using Sutherland-Hodgeman algorithm);
(4) When menu "region filling" is clicked, fill the clipped polygon using a polygon filling algorithm (either boundary-filling, or flood-filling, or scan-line) to fill a red color.
(5) Define a viewport (e.g., 40*80 pixels or other size defined by yourself), and map the clipped polygon in the window to the viewport (Take the Part-A Question 4 as an example).
(6) Change the viewport size by using the mouse-click to drag the top-right corner of the viewport and display the scaling effect of the clipped polygon simultaneously.
(7) Change the window size and map the clipped polygon from the window to viewport dynamically, achieving the *zoom-in* and *zoom-out* effect.
(8) Move the window horizontally (or vertically) and map the clipped polygon from the window to viewport, achieving the *panning* effect.

Extra point: (10%)

(1) Clipping a polygon against a rectangular window using Liang-Barsky Polygon clipping algorithm (10%)

2. Hand-in

- **Code package:** Your program package and the assignment report must be compressed in a ZIP file, and submit it to the TA by email on/before the due date.
- **Write-up (report)**: In your hand-in you should document the way in which you solved the problem. This documentation should describe your solution so that the reader understands the problem that you are solving and then understands the code that you hand in. It should approximately include
  (1) problem statement
  (2) algorithm design
  (3) kernel codes of your own implementation
  (4) sample images

Your file must be named as follows:

YourLastName_CS560_HW2_PartB.zip or YourLastName_CS460_HW2_PartB.zip

3. Mark distribution (100%)

- Your assignment will be marked as follows: Part A (48%) and Part B (52%)

Demo:
Draw a polygon overlapping with a clipping window (4%)
Draw the clipped polygon (12%)
Show the clipped polygon with a color filled (10%)
Show the clipped polygon in a viewport by window-to-viewport mapping (6%)
Show the scaling effect by changing the viewport size dynamically (5%)
Show the Zoom-in/Zoom-out effect by changing the window size dynamically (5%)
Show the Panning effect by moving the window dynamically (5%)

Write-up:
Clearly explain your code design, and the solution to the problem you solved (5%)

Hint:

(1)
In order to avoid the over-stack problem in VC++ programming, you can set the stack as follows:

In the VC++ environment:

Project → Setting → Link →

Category→Output→Stack allocations→ Reserve → type “2000000”

(2)
Using OpenGL to access pixel colors:
glReadPixels(x, y, 1, 1, GL_RGBA, GL_UNSIGNED_BYTE, *pixel);

Set (write) pixel:
glVertex….(…)

(3)
Using Windows API to read and set pixels:

SetPixel(m_hdc, x, y)
GetPixel(m_hdc, x, y)

(4) Using OpenGL (alternative):

//create buffer
float buffer[width*height*3];
//read pixels
glReadPixels(0, 0, width, height, GL_RGB, GL_FLOAT, buffer);
//access buffer (assuming x and y are a position on the screen)
buffer[y*width*3+x*3] //R
buffer[y*width*3+x*3+1] //G
buffer[y*width*3+x*3+2] //B
//write back
glDrawPixels(width, height, GL_RGB, GL_FLOAT, buffer);

(5) Using OpenGL (alternative)

```
GLubyte data[height*width*3];

   Or

GLubyte parray[1060][1060][3];


void CAssignment_2View::readBuffer()
      {
            glReadBuffer(GL_BACK);
            glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
            //glReadPixels(0, 0, width, height, GL_RGB, GL_UNSIGNED_BYTE, data);
            glReadPixels(0, 0, 1060, 1060, GL_RGB, GL_UNSIGNED_BYTE, parray);
```

```
	}


void CAssignment_2View::writeBuffer()
	{
		//glDrawPixels(width, height, GL_RGB, GL_UNSIGNED_BYTE, data);
		glDrawPixels(1060, 1060, GL_RGB, GL_UNSIGNED_BYTE, parray);
		SwapBuffers(m_hDC);
	}
```

(6) Note: once you call glBegin(), glReadPixels will return all garbage values. A solution to this problem is to use glReadPixels to read all the pixel values you will need prior to calling glBegin( ).