Answer Keys to Written Questions of Assignment 4

Q.2

a.

```
const int Xmin = 0;
const int Xmax = 1200;
const int Ymin = 0;
const int Ymax = 800;
const unsigned char b0 = 8;        // in binary 00001000
const unsigned char b1 = 4;        // in binary 00000100
const unsigned char b2 = 2;        // in binary 00000010
const unsigned char b3 = 1;        // in binary 00000001

unsigned chat outcode(vec2 p)
{
        unsigned char tcode = 0;  // definition of outcode b0b1b2b3

        if (p[1] > Ymax) tcode = tchode | b0;        // set b0
        if (p[1] < Ymin) tcode = tchode | b1;        // set b1
        if (p[0] > Xmax) tcode = tchode | b2;        // set b2
        if (p[0] < Xmin) tcode = tchode | b3;        // set b3
        return tcode;
}
```

b.

```
int Cohen-Sutherland-Clipping(vec2 p1, vec2 p2)
{
        unsigned outcoud1, outcode2;
        outcode1 = outcode(p1);
        outcode2 = outcode(p2);
        do {
                if (outcode1 == 0 && outcode2 == 0) return 1;        // case 1
                if ((outcode1 & outcode2) != 0) return 0;            // case 3
                if (outcode1 != 0) {
                        p1 = intersection_point(p1, p2, outcode1);
                        outcode1 = outcode(p1);
                } else if (outcode2 != 0) {
                        P2 = intersection_point(p1, p2, outcode1);
                        Outcode2 = outcode(p2);
                }
        } while (1)        // “infinite” loop
}
```

```
vec2 intersect_point(vec2 p1, vec2 p2, unsigned char outcode)
{	vec2 tp;

	// check which bit is "1" in the outcode
	If ((outcode & b0) != 0) {
		//compute intersect with Ymax here
		return tp;
	} else if ((outcode & b1) != 0) {
		// intersect with Ymin
	}	// two other cases
	……
}
```

c.

Example: p1 = (-50, 600), p2 = (600, 900)

outcode1 = 0001 //in binary

outcode2 = 1000 // in binary

loop 1:

case 1 is not true

case 3 is not true

outcode1 != 0: so compute the intersection with the Xmin boundary

To help the calculation, we have the line equation first:

$$y = (300/650)*(x + 50) + 600$$

Substitute x = Xmin = 0 into the above equation, we obtain y = 600+30/13

Replace p1 by (0, 600+30/13), and outcode1 = 0000

Loop 2:

Case 1 is not true

Case 3 is not true

Outcode2 != 0: so, compute the intersection with the Ymax boundary

Substitute the y = Ymax = 800 into the above equation, we obtain x = (2/3)*650-50

Replace p2 by ((2/3)*650-50, 800), and outcode2 = 0000

Loop 3:

Case 1 is true. Return 1.

Q3.

I provide you the code I wrote for Brensenham (MidPoint) line algorithm in the old CS 405 class. Its handling of situations should be similar to that for the question in your assignment. This implementation uses extensively symmetry properties to have a very compact code. However, it may not be efficient at run-time because of many “if” conditions.

Alternatively, you can have four cases handled by separate codes being optimized for each case. This code will be long, but very efficient at run-time.

Either answer will be ok in the final exam if there would be a question similar to this in the final.

```
/*=================================================================
    •  Function name:        MidPointLine
Purpose:      Implement the basic midpoint algorithm
              and extend it to arbitrary situations.
Parameters:  x1, y1, x2, y2     Integer    End points coordinates
              color              Integer    Gray level ( 0 -- 255 )
    By Xue Dong Yang
-----------------------------------------------------------------*/
void  MidPointLine( int x1, int y1, int x2, int y2, int color)
{
    int dx, dy, incrE, incrNE, d, x, y;
    int y_step = 1;     /* the motion in y direction. */
    int mirror = 0;     /* a flag indicating if the line is flipped
                           regarding to the 45' axis. */

    dx = x2 - x1;
    dy = y2 - y1;

    /* If the the absolute value of the slope is greater than 1,
      mirror the line to be drawn around the 45' axis first. */
    if (abs(dx) < abs(dy)) {
            my_swap(&x1, &y1);
            my_swap(&x2, &y2);
            my_swap(&dx, &dy);
            mirror = 1;                  /* set the flag to be true. */
    }

    /* Make sure x1 is less than x2 because we draw the line
      from left to right. */
    if ( x2 < x1 ) {
            my_swap(&x1, &x2);
            my_swap(&y1, &y2);
            dx = -dx;
            dy = -dy;
    }
```

```
	/* Check if the slope is negative. */
	if (dy < 0) {
		dy = -dy; /* Flip this line to positive slope. */
		y_step = -1;		/* But, remember it goes downward. */
	}

	/* Now all the situation have been reduced to the basic
	   standard sitution. We can draw it now. */
	d = dy * 2 - dx;
	incrE = dy * 2;
	incrNE = ( dy - dx ) * 2;
	x = x1;
	y = y1;
	if (mirror != 1)
		writepixel( x, y, color );
	else	/* Mirrow it back. */
		writepixel( y, x, color);

	while ( x < x2 ) {
		if ( d <= 0 ) {			/* Choose E */
			d += incrE;
			x++;
		} else {		/* Choose NE (SE) */
			d += incrNE;
			x++;
			y += y_step;	/* Whether it moves up or
					   down depending on the
					   value of y_step. */
		}
		if (mirror != 1)
			writepixel( x, y, color );
		else	/* Mirror it back. */
			writepixel( y, x, color );
	}
	return;
}


void my_swap(int *x, int *y)
{
	int t;

	t = *x;
	*x = *y;
	*y = t;
}
```

Q4.

This question may be simpler than what some of you might think. A linked list is maintained for all edges that intersect with the current scanline y under rasterization. The algorithm you are to write is to compute the x values in that linked list.

The standard polygon scan conversion algorithm can be found in many textbooks and web sites.

Here is the answer:

```
dx = (x2 – x1)/(y2 – y1);     // increment of x when y takes 1 unit step
x = x1;
for (y = y1; y <= y2; y++)
{
        x = x + dx;
        // store x into the data structure for polygon scan conversion.
}
```