Ryan Reynolds

2693018

CIS 457

Homework 3

**Question 1:** What are the main four rules used by Sutherland-Hogman polygon clipping algorithm?

Vector Edge SP has both start point, S, and end point, P, inside the clipping square. Output vector P

Vector edge SP has start point S inside and end point, P, outside the clipping square. Output S’ where S is the start point and the end point of is the intersection of SP and the clipping square.

Vector edge SP has both start point, S, and end point, P, outside of the clipping square. Output nothing (ie vector is clipped)

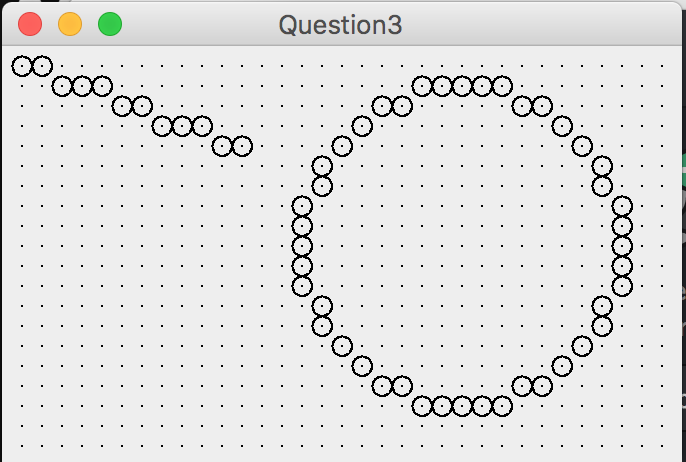
Vector Edge SP has start point, S, outside and end point, P, inside the clipping square. Output vector S’P where S’ is the start point representing the intersection of SP with the the edge of the clipping square.

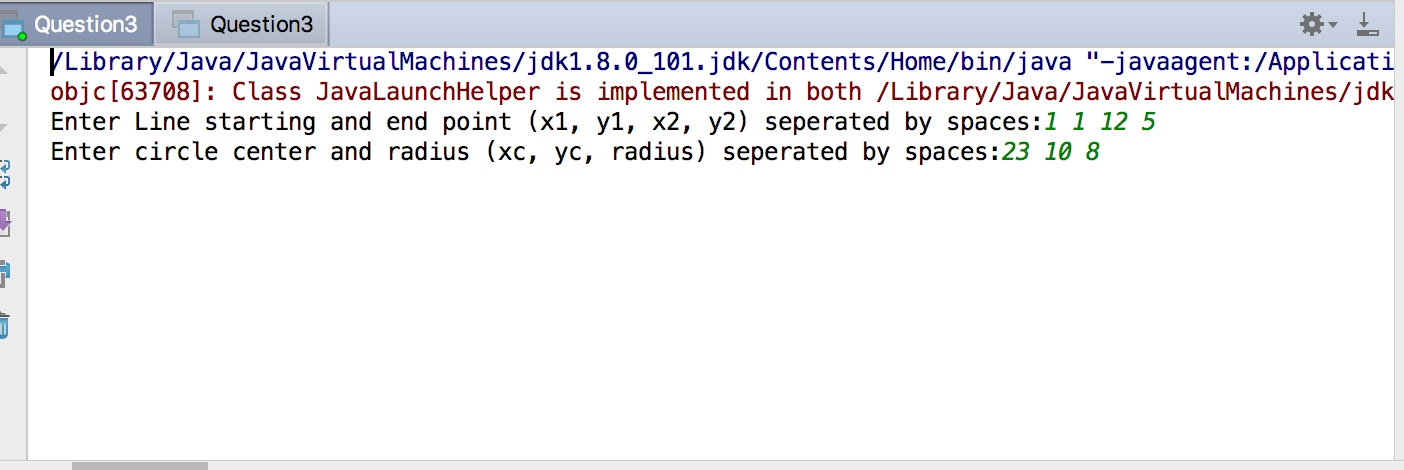
**Question 2:** What are the main difference between Beizer and B-Spline curves?

A Bezier curve is formed using four points,2 end points and 2 control points. Points one and two represent the end points and points zero and three represent the start and end points of the curve that is formed within the space inside the polygon formed by *P0P1P2P3P4.* B-Spline on the other hand draws a curve segment between point P(n-1) and Pn for all values greater 3. Essentially Beizer curves are defined by a particular polynomial. A B-spline is defined by piece wise polynomials. Each curve of a B-spline represents a Beizer curve.

**Question 3:**

Output:



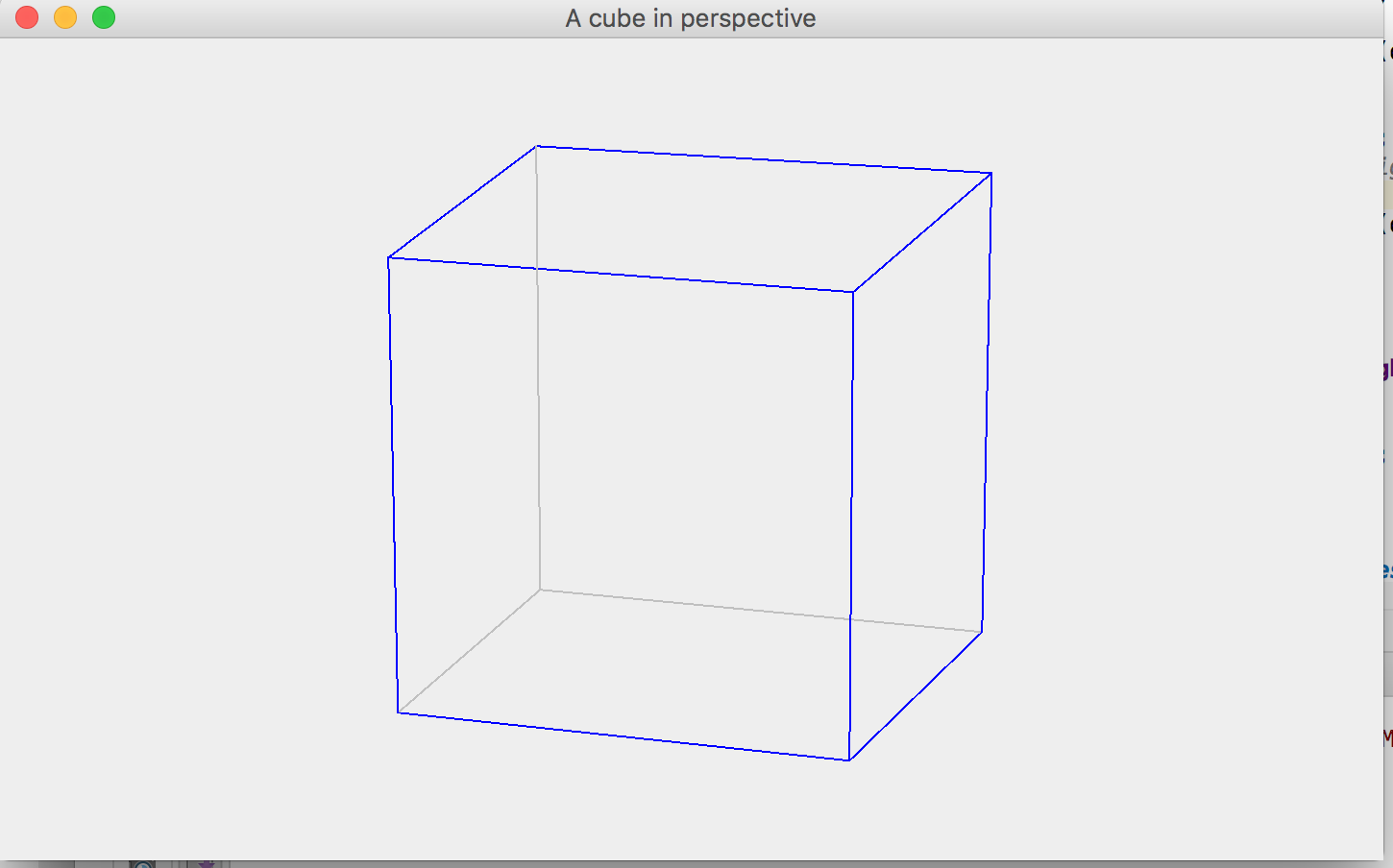


Source:

**import** javax.management.loading.MLet;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.util.\*;  
**public class** Question3 **extends** Frame {  
 **public static void** main(String[] args) {  
 **new** Question3();  
 }  
  
 Question3() {  
 **super**(**"Question3"**);  
 addWindowListener(**new** WindowAdapter() {  
 **public void** windowClosing(WindowEvent e) {  
 System.*exit*(0);  
 }  
 });  
 setSize(340, 230);  
 add(**"Center"**, **new** CvQuestion3());  
 show();  
 }  
}  
 **class** CvQuestion3 **extends** Canvas {  
 **float rWidth** = 10.F, **rHeight** = 7.5F, **pixelSize**;  
 **int centerX**, **centerY**, **dGrid** = 10, **maxX**, **maxY**;  
  
 **void** initgr() {  
 Dimension d;  
 d = getSize();  
 **maxX** = d.**width** - 1;  
 **maxY** = d.**height** - 1;  
 **pixelSize** = Math.*max*(**rWidth** / **maxX**, **rHeight** / **maxY**);  
 **centerX** = **maxX** / 2;  
 **centerY** = **maxY** / 2;  
 }  
  
 **int** iX(**float** x) {  
 **return** Math.*round*(**centerX** + x / **pixelSize**);  
 }  
  
 **int** iY(**float** y) {  
 **return** Math.*round*(**centerY** - y / **pixelSize**);  
 }  
  
 **void** putPixel(Graphics g, **int** x, **int** y) {  
 **int** x1 = x \* **dGrid**, y1 = y \* **dGrid**, h = **dGrid** / 2;  
 g.drawOval(x1 - h, y1 - h, **dGrid**, **dGrid**);  
 }  
  
 **void** drawLine(Graphics g, **int** xP, **int** yP, **int** xQ, **int** yQ) {  
 **int** x = xP, y = yP, D = 0, HX = xQ - xP, HY = yQ - yP, c, M, xInc = 1, yInc = 1;  
 **if** (HX < 0) {  
 xInc = -1;  
 HX = -HX;  
 }  
 **if** (HY < 0) {  
 yInc = -1;  
 HY = -HY;  
 }  
 **if** (HY <= HX) {  
 c = 2 \* HX;  
 M = 2 \* HY;  
 **for** (; ; ) {  
 putPixel(g, x, y);  
 **if** (x == xQ) **break**;  
 x += xInc;  
 D += M;  
 **if** (D > HX) {  
 y += yInc;  
 D -= c;  
 }  
 }  
 }  
 **else** {  
 c = 2 \* HX;  
 M = 2 \* HY;  
 **for** (; ; ) {  
 putPixel(g, x, y);  
 **if** (y == yQ) **break**;  
 y += yInc;  
 D += M;  
 **if** (D > HY) {  
 x+= xInc;  
 D -= c;  
 }  
 }  
 }  
  
 }  
  
 **void** drawCircle(Graphics g, **int** xC, **int** yC, **int** r) {  
 **int** x = 0, y = r, u = 1, v = 2 \* r - 1, E = 0;  
 **while** (x < y) {  
  
 putPixel(g, xC + x, yC + y);  
 putPixel(g, xC + y, yC - x);  
 putPixel(g, xC - x, yC - y);  
 putPixel(g, xC - y, yC + x);  
 x++;  
 E += u;  
 u += 2;  
 **if** (v < 2 \* E) {  
 y--;  
 E -= v;  
 v -= 2;  
 }  
 **if** (x > v) **break**;  
 putPixel(g, xC + y, yC + x);  
 putPixel(g, xC + x, yC - y);  
 putPixel(g, xC - y, yC - x);  
 putPixel(g, xC - x, yC + y);  
 }  
 }  
  
 **void** showGrid(Graphics g) {  
 **for** (**int** x = **dGrid**; x <= **maxX**; x += **dGrid**) {  
 **for** (**int** y = **dGrid**; y <= **maxY**; y += **dGrid**) {  
 g.drawLine(x, y, x, y);  
 }  
 }  
 }  
  
 **public void** paint(Graphics g) {  
 initgr();  
 **int** x1, y1, x2, y2, xc, yc, r;  
 System.***out***.print(**"Enter Line starting and end point (x1, y1, x2, y2) seperated by spaces:"**);  
 Scanner input= **new** Scanner(System.***in***);  
 x1=input.nextInt();  
 y1=input.nextInt();  
 x2=input.nextInt();  
 y2=input.nextInt();  
 System.***out***.print(**"Enter circle center and radius (xc, yc, radius) seperated by spaces:"**);  
 xc=input.nextInt();  
 yc=input.nextInt();  
 r=input.nextInt();  
  
 showGrid(g);  
 drawLine(g, x1, y1, x2, y2);  
 drawCircle(g, xc, yc, r);  
 }  
 }

**Question 4:**

Output:



Source Code:  
*// CubePers.java: A cube in perspective.  
  
// Copied from Section 5.4 of  
// Ammeraal, L. and K. Zhang (2007). Computer Graphics for Java Programmers, 2nd Edition,  
// Chichester: John Wiley.  
  
// Uses: Point2D (Section 1.5), Point3D (Section 3.9).****import*** *javafx.geometry.Point3D;****import*** *java.awt.\*;****import*** *java.awt.event.\*;****public class*** *CubePers* ***extends*** *Frame {* ***public static void*** *main(String[] args) {****new*** *CubePers();}  
  
 CubePers() {* ***super****(****"A cube in perspective"****);  
 addWindowListener(****new*** *WindowAdapter() {* ***public void*** *windowClosing(WindowEvent e) {System.exit(0);}  
 });  
 setLayout(****new*** *BorderLayout());  
 add(****"Center"****,* ***new*** *CvCubePers());  
 Dimension dim = getToolkit().getScreenSize();  
 setSize(dim.****width*** */ 2, dim.****height*** */ 2);  
 setLocation(dim.****width*** */ 4, dim.****height*** */ 4);  
 setVisible(****true****);  
 }  
}****class*** *CvCubePers* ***extends*** *Canvas {* ***int centerX****,* ***centerY****;  
 Obj* ***obj*** *=* ***new*** *Obj();* ***int*** *iX(****float*** *x) {* ***return*** *Math.round(****centerX*** *+ x);  
 }* ***int*** *iY(****float*** *y) {* ***return*** *Math.round(****centerY*** *- y);  
 }* ***void*** *line(Graphics g,* ***int*** *i,* ***int*** *j) {  
 Point2D p =* ***obj****.****vScr****[i], q =* ***obj****.****vScr****[j];  
 g.setColor(Color.****BLUE****);  
 g.drawLine(iX(p.****x****), iY(p.****y****), iX(q.****x****), iY(q.****y****));  
 }* ***void*** *line2(Graphics g,* ***int*** *i,* ***int*** *j){  
 Point2D p =* ***obj****.****vScr****[i], q =* ***obj****.****vScr****[j];  
 //set the stroke of the copy, not the original  
 g.setColor(Color.****LIGHT\_GRAY****);  
 g.drawLine(iX(p.****x****), iY(p.****y****), iX(q.****x****), iY(q.****y****));  
  
 }* ***public void*** *paint(Graphics g) {  
 Dimension dim = getSize();* ***int*** *maxX = dim.****width*** *- 1, maxY = dim.****height*** *- 1,  
 minMaxXY = Math.min(maxX, maxY);* ***centerX*** *= maxX / 2;* ***centerY*** *= maxY / 2;* ***obj****.****d*** *=* ***obj****.****rho*** *\* minMaxXY /* ***obj****.****objSize****;* ***obj****.eyeAndScreen();  
 // Horizontal edges at the bottom:  
 line(g, 0, 1);  
 //****TODO, draw the line for 3 other vertices*** *line2(g,0,3 );  
 line(g, 1,2);  
 line2(g,2, 3);  
  
 // Horizontal edges at the top:  
 line(g, 4, 5);  
 //****TODO, draw the line for 3 other vertices*** *line(g, 4,7);  
 line(g,5,6);  
 line(g, 6,7);  
  
  
 // Vertical edges:  
 line(g, 0, 4);  
 //****TODO, draw the line for 3 other vertices*** *line(g,6,2);  
 line2(g, 7,3);  
 line(g, 5,1);  
 }  
}****class*** *Obj { // Contains 3D object data* ***float rho****,* ***theta*** *= 0.3F,* ***phi*** *= 1.3F,* ***d****,* ***objSize****,* ***v11****,* ***v12****,* ***v13****,* ***v21****,* ***v22****,* ***v23****,* ***v32****,* ***v33****,* ***v43****;  
 // Elements of viewing matrix V  
 Point3D[]* ***w****; // World coordinates  
 Point2D[]* ***vScr****; // Screen coordinates  
  
 Obj() {* ***w*** *=* ***new*** *Point3D[8];* ***vScr*** *=* ***new*** *Point2D[8];  
 // Bottom surface:* ***w****[0] =* ***new*** *Point3D(1, -1, -1);  
  
 //****TODO*** *// Write 3D coordinates of vertices w1, w2, w3* ***w****[1]=* ***new*** *Point3D(1,1,-1);* ***w****[2]=* ***new*** *Point3D(-1,1,-1);* ***w****[3]=* ***new*** *Point3D(-1,-1,-1);  
  
 // Top surface:* ***w****[4] =* ***new*** *Point3D(1, -1, 1);  
 // Write 3D coordinates of vertices w5, w6, w7* ***w****[5]=* ***new*** *Point3D(1,1,1);* ***w****[6]=* ***new*** *Point3D(-1,1,1);* ***w****[7]=* ***new*** *Point3D(-1,-1,1);* ***objSize*** *= (****float****) Math.sqrt(12F);  
 // = sqrt(2 \* 2 + 2 \* 2 + 2 \* 2)  
 // = distance between two opposite vertices.* ***rho*** *= 5 \** ***objSize****; // For reasonable perspective effect  
 }* ***void*** *initPersp() {* ***float*** *costh = (****float****) Math.cos(****theta****),  
 sinth = (****float****) Math.sin(****theta****),  
 cosph = (****float****) Math.cos(****phi****),  
 sinph = (****float****) Math.sin(****phi****);*

*//****TODO, calculate other elements of viewing matrix***

***v11*** *= -sinth;* ***v12****=-(cosph\*costh);* ***v13****=sinph\*sinth;* ***v21****=costh;* ***v22****=-(cosph\*sinth);* ***v23****=sinph\*sinth;* ***v32****=sinph;* ***v33****=cosph;* ***v43****=-****rho****;* *}* ***void*** *eyeAndScreen() {  
 initPersp();* ***for*** *(****int*** *i = 0; i < 8; i++) {  
 Point3D p =* ***w****[i];* ***float*** *x =* ***v11*** *\* (****float****) p.getX() +* ***v21*** *\* (****float****)p.getY(),  
 y=* ***v12****\*(****float****)p.getX()+****v22****\*(****float****)p.getY()+****v32****\*(****float****)p.getZ(),  
 z=****v13****\*(****float****)p.getX()+****v23****\*(****float****)p.getY()+****v33****\*(****float****)p.getZ()+****v43****;  
  
 //****TODO, write for y and z  
  
 vScr****[i] =* ***new*** *Point2D(-****d*** *\* x / z, -****d*** *\* y / z);  
 }  
 }  
  
}  
// Point2D.java: Class for points in logical coordinates.  
  
// Copied from Section 1.5 of  
// Ammeraal, L. and K. Zhang (2007). Computer Graphics for Java Programmers, 2nd Edition,  
// Chichester: John Wiley.****class*** *Point2D {* ***float x****,* ***y****;  
 Point2D(****float*** *x,* ***float*** *y) {****this****.****x*** *= x;* ***this****.****y*** *= y;}  
}*