

COMPUTER GRAPHICS LAB REPORT

SOULIB GHOSH and MADHURAM JAJOO



November 20, 2018

JADAVPUR UNIVERSITY

NAME – MADHURAM JAJOO

ROLL – 0016105010 54

COURSE – BCSE III

GROUP – A2

NAME – SOULIB GHOSH

ROLL – 0016105010 47

COURSE – BCSE III

GROUP – A2

The following codes contain five files – my\_label.h, my\_label.cpp, main.cpp, mainwindow.h and mainwindow.cpp.

The following files are given. The output of the following programs are also shown.

**my\_label.h**

1. #ifndef MY\_LABEL\_H
2. #define MY\_LABEL\_H
4. #include <QObject>
5. #include <QWidget>
6. #include <QLabel>
7. #include <QMouseEvent>
8. #include <QEvent>
10. **class** my\_label : **public** QLabel
11. {
12. Q\_OBJECT
14. **public**:
15. my\_label(QWidget \*parent = 0);
16. **int** x,y;
18. **protected**:
19. **void** mouseMoveEvent(QMouseEvent \*ev);
20. **void** mousePressEvent(QMouseEvent \*ev);
21. signals:
22. **void** sendMousePosition(QPoint&);
23. **void** Mouse\_Pos();
24. };
26. #endif // MY\_LABEL\_H

**my\_label.cpp**

1. #include "my\_label.h"
3. my\_label::my\_label(QWidget \*parent):QLabel(parent)
4. {
5. **this**->setMouseTracking(**true**);
6. }
8. **void** my\_label::mouseMoveEvent(QMouseEvent \*ev)
9. {
10. QPoint mouse\_pos=ev->pos();
11. **if**(mouse\_pos.x()<=**this**->size().width() && mouse\_pos.y()<=**this**->size().height()){
12. **if**(mouse\_pos.x()>0 && mouse\_pos.y()>0){
13. emit sendMousePosition(mouse\_pos);
14. }
15. }
16. }
18. **void** my\_label::mousePressEvent(QMouseEvent \*ev)
19. {
20. **if**(ev->button()==Qt::LeftButton){
21. **this**->x=ev->x();
22. **this**->y=ev->y();
23. emit Mouse\_Pos();
24. }
25. }

**mainwindow.h**

1. #ifndef MAINWINDOW\_H
2. #define MAINWINDOW\_H
4. #include <QMainWindow>
5. #include <QtGui>
6. #include <QtCore>
7. #include <QColorDialog>
8. #include<QVector>
10. **namespace** Ui {
11. **class** MainWindow;
12. }
14. **class** MainWindow : **public** QMainWindow
15. {
16. Q\_OBJECT
17. **public** slots:
18. **void** Mouse\_Pressed();
19. **void** showMousePosition(QPoint& pos);
20. **public**:
21. **explicit** MainWindow(QWidget \*parent = 0);
22. ~MainWindow();
24. **private** slots:
25. **void** on\_show\_axes\_clicked();
26. **void** on\_Draw\_clicked();
27. **void** on\_set\_point1\_clicked();
28. **void** on\_set\_point2\_clicked();
29. **void** on\_pushButton\_clicked();
30. **int** changeX(**int** x);
31. **int** changeY(**int** y);
32. **void** on\_show\_grid\_clicked();
33. **void** drawLine(QPoint, QPoint, **int**);
34. **void** drawCircle(QPoint, **int**, **int**);
35. **void** drawEllipse(QPoint,**int**,**int**);
36. **void** on\_set\_point\_3\_clicked();
37. **void** on\_Fill\_clicked();
38. **void** translate\_object(**int** x,**int** y);
39. **void** on\_draw\_ellipse\_2\_clicked();
40. **void** on\_Draw\_2\_clicked();
41. **void** on\_Draw\_3\_clicked();
42. **void** on\_Draw\_4\_clicked();
43. **void** on\_Draw\_5\_clicked();
44. **void** on\_Draw\_6\_clicked();
45. **void** on\_Draw\_7\_clicked();
46. **void** on\_Draw\_8\_clicked();
47. **void** on\_draw\_ellipse\_3\_clicked();
48. **void** on\_Draw\_9\_clicked();
49. **void** on\_comboBox\_3\_activated(**const** QString &arg1);
50. **void** on\_Draw\_10\_clicked();
51. **void** on\_Draw\_11\_clicked();
52. **void** on\_Fill\_2\_clicked();
53. **void** on\_Draw\_13\_clicked();
54. **void** on\_Draw\_12\_clicked();
56. **private**:
57. **bool** flag = **false**;
58. QVector<QPoint> vertices;
59. Ui::MainWindow \*ui;
60. QPoint p1,p2;
61. **void** point(**int**,**int**);
62. **void** point(**int** ,**int** , **int** , **int**, **int**);
63. **void** drawDDA(QPoint, QPoint);
64. **void** drawBresenham(QPoint, QPoint);
65. **void** BresGentle(**int**,**int**,**int**,**int**);
66. **void** BresSteep(**int**,**int**,**int**,**int**);
67. **void** drawCirBres(QPoint, **int**);
68. **void** drawCirMidPt(QPoint, **int**);
69. **void** floodFill(QPoint);
70. **void** floodFillRec(**int**, **int**, **int**, QRgb, **int**, **int**, **int**);
71. **void** boundaryFill(QPoint);
72. **void** boundaryFillRec(**int**, **int**, **int**, **int**, **int**, **int**, **int**, **int**, **int**);
73. **void** scanLineFill(QPoint);
74. **int** computeCode(**double**,**double**);
75. **int** x\_intersect(**int**,**int**,**int**,**int**,**int**,**int**,**int**,**int**);
76. **int** y\_intersect(**int**,**int**,**int**,**int**,**int**,**int**,**int**,**int**);
77. **void** suthHodgClip();
78. **void** clip(**int**,**int**,**int**,**int**);
80. };
81. #endif // MAINWINDOW\_H

**main.cpp**

1. #include "mainwindow.h"
2. #include <QApplication>
4. **int** main(**int** argc, **char** \*argv[])
5. {
6. QApplication a(argc, argv);
7. MainWindow w;
8. w.show();
10. **return** a.exec();
11. }

**mainwindow.cpp**

1. #include "mainwindow.h"
2. #include "ui\_mainwindow.h"
3. #include <QPixmap>
4. #include <QImage>
5. #include <iostream>
6. #include <QMouseEvent>
7. #include <QPainter>
8. #include <QPaintDevice>
9. #include <QPoint>
10. QImage img=QImage(700,700,QImage::Format\_RGB888);
11. MainWindow::MainWindow(QWidget \*parent) :
12. QMainWindow(parent),
13. ui(**new** Ui::MainWindow)
14. {
15. ui->setupUi(**this**);
16. ui->x\_axis->hide();
17. ui->y\_axis->hide();
18. connect(ui->frame,SIGNAL(Mouse\_Pos()),**this**,SLOT(Mouse\_Pressed()));
19. connect(ui->frame,SIGNAL(sendMousePosition(QPoint&)),**this**,SLOT(showMousePosition(QPoint&)));
20. ui->comboBox->addItem("DDA Algorithm");
21. ui->comboBox->addItem("Bresenham Algorithm");
22. }
24. MainWindow::~MainWindow()
25. {
26. **delete** ui;
27. }
29. **int** MainWindow::changeX(**int** x){
30. **int** k=ui->grid\_size->value();
31. x-=img.width()/2;
32. x/=k;
33. **return** x;
34. }
36. **int** MainWindow::changeY(**int** y){
37. **int** k=ui->grid\_size->value();
38. y=img.width()/2-y;
39. y/=k;
40. **return** y;
41. }
43. **void** MainWindow::point(**int** x,**int** y)
44. {
45. **int** k=ui->grid\_size->value();
46. **if**(k==1)
47. img.setPixel(x,y,qRgb(255,255,0));
48. **else**{
49. **int** i,j;
50. **int** stx=(x/k)\*k;
51. **int** sty=(y/k)\*k;
52. **for**(i=stx+1;i<stx+k;i++){
53. **for**(j=sty+1;j<sty+k;j++)
54. img.setPixel(i,j,qRgb(255,255,0));
55. }
56. }
57. ui->frame->setPixmap(QPixmap::fromImage(img));
58. }
60. **void** MainWindow::point(**int** x,**int** y, **int** r, **int** g, **int** b)
61. {
62. **int** k=ui->grid\_size->value();
63. **if**(k==1)
64. img.setPixel(x,y,qRgb(r,g,b));
65. **else**{
66. **int** i,j;
67. **int** stx=(x/k)\*k;
68. **int** sty=(y/k)\*k;
69. **for**(i=stx+1;i<stx+k;i++){
70. **for**(j=sty+1;j<sty+k;j++)
71. img.setPixel(i,j,qRgb(r,g,b));
72. }
73. }
74. //ui->frame->setPixmap(QPixmap::fromImage(img));
75. }
77. **void** MainWindow::showMousePosition(QPoint &pos)
78. {
79. ui->mouse\_movement->setText(" X : "+QString::number(changeX(pos.x()))+", Y : "+QString::number(changeY(pos.y())));
80. }
81. **void** MainWindow::Mouse\_Pressed()
82. {
83. ui->mouse\_pressed->setText(" X : "+QString::number(changeX(ui->frame->x))+", Y : "+QString::number(changeY(ui->frame->y)));
84. point(ui->frame->x,ui->frame->y);
85. ui->x\_axis->move(0,ui->frame->y);
86. ui->y\_axis->move(ui->frame->x,0);
87. //point(ui->frame->x, ui->frame->y,255,255,0);
89. **if**(flag){
90. p1.setX((ui->frame->x));
91. p1.setY((ui->frame->y));
93. **if**(vertices.size() > 0 && p1 == vertices[0]){
95. flag = **false**;
96. **return**;
98. }
100. vertices.push\_back(p1);
101. }
102. }
104. **void** MainWindow::on\_show\_axes\_clicked()
105. {
106. **if**(ui->show\_axes->isChecked())
107. {
108. **for**(**int** i=0;i<img.height();i++)
109. point(i,img.width()/2,0,255,255);
110. **for**(**int** j=0;j<img.width();j++)
111. point(img.height()/2,j,0,255,255);
112. }
113. **else**{
114. **for**(**int** i=0;i<img.height();i++)
115. point(i,img.width()/2,0,0,0);
116. **for**(**int** j=0;j<img.width();j++)
117. point(img.height()/2,j,0,0,0);
118. }
119. }
120. **void** MainWindow::on\_set\_point1\_clicked()
121. {
122. **if**(ui->draw\_line->isChecked()){
123. p1.setX(ui->frame->x);
124. p1.setY(ui->frame->y);
125. }
126. }
128. **void** MainWindow::on\_set\_point2\_clicked()
129. {
130. **if**(ui->draw\_line->isChecked()){
131. p2.setX(ui->frame->x);
132. p2.setY(ui->frame->y);
133. }
134. }
136. **void** MainWindow::on\_Draw\_clicked()
137. {
138. //int r0=ui->circle\_radius->value();
139. //QPainter painter(&img);
140. //QPen pen;
141. //pen.setWidth(1);
142. //pen.setColor(Qt::red);
143. **if**(ui->draw\_circle->isChecked()){
144. p1.setX(ui->frame->x);
145. p1.setY(ui->frame->y);
146. /\*painter.setPen(pen);
147. painter.drawEllipse(p1,r0,r0);\*/
148. drawCircle(p1,ui->circle\_radius->value(),ui->comboBox\_2->currentIndex());
149. }
150. **if**(ui->draw\_line->isChecked()){
151. //painter.setPen(Qt::red);
152. drawLine(p1,p2,ui->comboBox->currentIndex());
153. }
154. **if**(ui->draw\_ellipse->isChecked()){
155. p1.setX(ui->frame->x);
156. p1.setY(ui->frame->y);
157. drawEllipse(p1,ui->ellipse\_maj->value(),ui->ellipse\_min->value());
158. }

161. //ui->frame->setPixmap(QPixmap::fromImage(img));
162. }
164. **void** MainWindow::on\_pushButton\_clicked()
165. {
166. **for**(**int** j=0;j<img.height();j++)
167. {
168. **for**(**int** i=0;i<img.width();i++)
169. {
170. img.setPixel(i,j,qRgb(255,255,255));
171. }
172. }
173. ui->frame->setPixmap(QPixmap::fromImage(img));
174. }
176. **void** MainWindow::on\_show\_grid\_clicked()
177. {
178. **int** i,j,k=ui->grid\_size->value();
179. **if**(ui->show\_grid->isChecked()){
180. **for**(i=0;i<=img.width();i+=k){
181. **for**(j=0;j<=img.height();j++)
182. img.setPixel(j,i,qRgb(255,0,0));
183. }
184. **for**(i=0;i<=img.height();i+=k){
185. **for**(j=0;j<=img.width();j++)
186. img.setPixel(i,j,qRgb(255,0,0));
187. }
188. ui->frame->setPixmap(QPixmap::fromImage(img));
189. }
190. **else**{
191. **for**(i=0;i<=img.width();i++){
192. **for**(j=0;j<=img.height();j++)
193. img.setPixel(j,i,qRgb(0,0,0));
194. }
195. **for**(i=0;i<=img.height();i++){
196. **for**(j=0;j<=img.width();j++)
197. img.setPixel(i,j,qRgb(0,0,0));
198. }
199. ui->frame->setPixmap(QPixmap::fromImage(img));
200. }
202. on\_show\_axes\_clicked();
203. }

**Line Drawing:**

**DDA Line Drawing:**

1. **void** MainWindow::drawDDA(QPoint p1, QPoint p2){
2. **int** k=ui->grid\_size->value();
3. **int** x1=(p1.x()/k);
4. **int** y1=(p1.y()/k);
5. **int** x2=(p2.x()/k);
6. **int** y2=(p2.y()/k);
7. **int** dx=x2-x1;
8. **int** dy=y2-y1;
9. **int** st=(**int**)((fabs(dx)>fabs(dy))?fabs(dx):fabs(dy));
10. **float** xi=((**float**)dx)/st;
11. **float** yi=((**float**)dy)/st;
13. **float** x=x1\*k+k/2,y=y1\*k+k/2;
14. **for**(**int** i=1;i<=st;i++){
15. x+=xi\*k;
16. y+=yi\*k;
17. point((**int**)(x+0.5),(**int**)(y+0.5));
18. }
19. }

**Bresenham Line Drawing:**

1. **void** MainWindow::BresGentle(**int** x1,**int** y1, **int** x2, **int** y2){
2. **int** k=ui->grid\_size->value();
3. **int** dx=(x2-x1)/k;
4. **int** dy=(y2-y1)/k;
5. **int** yi =k;
6. **if**(dy<0){
7. yi=-k;
8. dy=-dy;
9. }
10. **int** dif=2\*dy-dx;
11. **int** i,j;
12. j=(y1/k)\*k+k/2;
13. **int** xs=(x1/k)\*k+k/2;
14. **int** xd=(x2/k)\*k+k/2;
15. **for**(i=xs;i<=xd;i+=k){
16. point(i,j);
17. **if** (dif>0){
18. j+=yi;
19. dif-=2\*dx;
20. }
21. dif+=2\*dy;
22. }
23. }
25. **void** MainWindow::BresSteep(**int** x1,**int** y1, **int** x2, **int** y2){
26. **int** k=ui->grid\_size->value();
27. **int** dx=(x2-x1)/k;
28. **int** dy=(y2-y1)/k;
29. **int** xi =k;
30. **if**(dx<0){
31. xi=-k;
32. dx=-dx;
33. }
34. **int** dif=2\*dx-dy;
35. **int** i,j;
36. i=(x1/k)\*k+k/2;
37. **int** ys=(y1/k)\*k+k/2;
38. **int** yd=(y2/k)\*k+k/2;
39. **for**(j=ys;j<=yd;j+=k){
40. point(i,j);
41. **if** (dif>0){
42. i+=xi;
43. dif-=2\*dy;
44. }
45. dif+=2\*dx;
46. }
47. }
49. **void** MainWindow::drawBresenham(QPoint p1, QPoint p2){
51. **if**(fabs(p2.y()-p1.y())<fabs(p2.x()-p1.x())){
52. **if**(p1.x()>p2.x())
53. BresGentle(p2.x(),p2.y(),p1.x(),p1.y());
54. **else**
55. BresGentle(p1.x(),p1.y(),p2.x(),p2.y());
56. }
57. **else**{
58. **if**(p1.y()>p2.y())
59. BresSteep(p2.x(),p2.y(),p1.x(),p1.y());
60. **else**
61. BresSteep(p1.x(),p1.y(),p2.x(),p2.y());
62. }
63. }

**Circle Drawing:**

**Bresenham Circle Drawing:**

1. **void** MainWindow::drawCirBres(QPoint p, **int** r){
2. **int** xc=p.x(),yc=p.y();
3. **int** i,k=ui->grid\_size->value(),x=0,y=r\*k,d=(3-2\*r)\*k;
4. **int** dirx[]={+1,+1,-1,-1};
5. **int** diry[]={+1,-1,+1,-1};
6. xc=(xc/k)\*k+k/2;
7. yc=(yc/k)\*k+k/2;
8. **while**(y>=x){
9. **for**(i=0;i<4;i++)
10. point(xc+dirx[i]\*x,yc+diry[i]\*y);
11. **for**(i=0;i<4;i++)
12. point(xc+dirx[i]\*y,yc+diry[i]\*x);
13. x++;
14. **if**(d>0){
15. y--;
16. d+=4\*(x-y)+10;
17. }
18. **else**
19. d+=4\*x+6;
20. }
21. }

**Mid-Point Circle Drawing:**

1. **void** MainWindow::drawCirMidPt(QPoint p1, **int** r){
2. **int** k=ui->grid\_size->value();
3. **int** xc=(p1.x()/k)\*k+k/2,yc=(p1.y()/k)\*k+k/2;
4. **int** x=r\*k,y=0;
5. point(xc+x,yc+y);
6. **if**(r>0){
7. point(xc+x,yc-y);
8. point(xc-x,yc+y);
9. point(xc-x,yc-y);
10. }
11. **int** p=(1-r)\*k;
12. **while**(x>y){
13. y++;
14. **if**(p<=0)
15. p+=2\*y+1;
16. **else**{
17. x--;
18. p+=2\*y-2\*x+1;
19. }
20. **if**(x<y)
21. **break**;
22. point(xc+x,yc+y);
23. point(xc+x,yc-y);
24. point(xc-x,yc+y);
25. point(xc-x,yc-y);
26. **if**(x!=y){
27. point(xc+y,yc+x);
28. point(xc+y,yc-x);
29. point(xc-y,yc+x);
30. point(xc-y,yc-x);
31. }
32. }
34. ui->frame->setPixmap(QPixmap::fromImage(img));
35. }
37. **void** MainWindow::drawLine(QPoint p1, QPoint p2, **int** i){
38. **if**(i)
39. drawBresenham(p1,p2);
40. **else**
41. drawDDA(p1,p2);
42. }
44. **void** MainWindow::drawCircle(QPoint p, **int** r, **int** i){
45. **if**(i){
46. drawCirBres(p,r);
47. }
48. **else**
49. drawCirMidPt(p,r);
50. }

**Ellipse Drawing:**

1. **void** MainWindow::drawEllipse(QPoint p1,**int** a,**int** b){
2. **int** k=ui->grid\_size->value();
3. **int** xc=(p1.x()/k)\*k+k/2,yc=(p1.y()/k)\*k+k/2;
4. **int** x=0,y=b,xk,yk;
5. **int** a2=a\*a,b2=b\*b,ta2=2\*a2,tb2=2\*b2;
7. **int** px=0,py=ta2\*y;
9. **double** p=b2-a2\*b+a2/4;
10. **while**(px<py){
11. xk=x\*k;
12. yk=y\*k;
13. point(xc+xk,yc+yk);
14. point(xc+xk,yc-yk);
15. point(xc-xk,yc+yk);
16. point(xc-xk,yc-yk);
17. x++;
18. px+=tb2;
19. **if**(p<0){
20. p+=b2+px;
21. }
22. **else**{
23. y--;
24. py-=ta2;
25. p+=b2+px-py;
26. }
27. }
29. p=b2\*((**double**)x+0.5)\*((**double**)x+0.5)+a2\*(y-1)\*(y-1)-a2\*b2;
30. **while**(y>=0){
31. xk=x\*k;
32. yk=y\*k;
33. point(xc+xk,yc+yk);
34. point(xc+xk,yc-yk);
35. point(xc-xk,yc+yk);
36. point(xc-xk,yc-yk);
37. y--;
38. py-=ta2;
39. **if**(p>0){
40. p+=a2-py;
41. }
42. **else**{
43. x++;
44. px+=tb2;
45. p-=a2-py+px;
46. }
47. }
48. }

The output of the above codes are shown in Figure 1.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |
|  |  |
| (e) | (f) |

Figure 1: (a) The grid (b) DDA Line (c) Bresenham Line (d) Mid-Point Circle (e) Bresenham Circle (f) Ellipse

**Filling Methods:**

1. **void** MainWindow::on\_set\_point\_3\_clicked()
2. {
3. **if**(ui->fill\_color->isChecked()){
4. p1.setX(ui->frame->x);
5. p1.setY(ui->frame->y);
6. }
7. }
9. **void** MainWindow::on\_Fill\_clicked()
10. {
11. **int** i=ui->comboBox\_4->currentIndex()+1;
12. **switch**(i){
13. **case** 1:
14. floodFill(p1);
15. **break**;
16. **case** 2:
17. boundaryFill(p1);
18. **break**;
19. /\*case 3:
20. scanLineFill(p1);
21. break;\*/
22. }
23. ui->frame->setPixmap(QPixmap::fromImage(img));
24. }

**Flood Fill:**

1. **void** MainWindow::floodFill(QPoint p){

4. **int** k=ui->grid\_size->value();
5. **int** xc=(p.x()/k)\*k+k/2;
6. **int** yc=(p.y()/k)\*k+k/2;
7. point(xc,yc,255,255,255);
8. floodFillRec(xc,yc,k,img.pixel(xc,yc),135,241,112);
9. }
11. **void** MainWindow::floodFillRec(**int** x, **int** y, **int** k, QRgb q, **int** r, **int** g, **int** b){
12. **if**(x<0 || x>img.width() || y<0 || y>img.height() || img.pixel(x,y)!=q)
13. **return**;
14. point(x,y,r,g,b);
15. floodFillRec(x+k,y,k,q,r,g,b);
16. floodFillRec(x,y+k,k,q,r,g,b);
17. floodFillRec(x-k,y,k,q,r,g,b);
18. floodFillRec(x,y-k,k,q,r,g,b);
19. }

**Boundary Fill:**

1. **void** MainWindow::boundaryFill(QPoint p){
2. **int** k=ui->grid\_size->value();
3. **int** xc=(p.x()/k)\*k+k/2;
4. **int** yc=(p.y()/k)\*k+k/2;
5. point(xc,yc,255,255,255);
6. boundaryFillRec(xc,yc,k,255,255,0,135,241,112);
7. }
9. **void** MainWindow::boundaryFillRec(**int** x, **int** y, **int** k, **int** rb, **int** gb, **int** bb, **int** rf, **int** gf, **int** bf){
10. QColor cur(img.pixel(x,y));
11. **if**(x<0 || x>img.width() || y<0 || y>img.height() || (cur.red()==rb && cur.green()==gb && cur.blue()==bb) || (cur.red()==rf && cur.green()==gf && cur.blue()==bf))
12. **return**;
13. point(x,y,rf,gf,bf);
14. boundaryFillRec(x+k,y,k,rb,gb,bb,rf,gf,bf);
15. boundaryFillRec(x,y+k,k,rb,gb,bb,rf,gf,bf);
16. boundaryFillRec(x-k,y,k,rb,gb,bb,rf,gf,bf);
17. boundaryFillRec(x,y-k,k,rb,gb,bb,rf,gf,bf);
18. }

**Scanline Fill:**

1. #define maxHt 1800
2. #define maxWd 1000
3. #define maxVer 10000
5. **typedef** **struct** edgebucket
6. {
7. **int** ymax;   //max y-coordinate of edge
8. **float** xofymin;  //x-coordinate of lowest edge point updated only in aet
9. **float** slopeinverse;
10. }EdgeBucket;
12. **typedef** **struct** edgetabletup
13. {
14. // the array will give the scanline number
15. // The edge table (ET) with edges entries sorted
16. // in increasing y and x of the lower end
17. **int** countEdgeBucket;    //no. of edgebuckets
18. EdgeBucket buckets[maxVer];
19. }EdgeTableTuple;
21. EdgeTableTuple EdgeTable[maxHt], ActiveEdgeTuple;
23. // Scanline Function
24. **void** MainWindow::initEdgeTable()
25. {
26. **int** i;
27. **for** (i=0; i<maxHt; i++)
28. {
29. EdgeTable[i].countEdgeBucket = 0;
30. }
32. ActiveEdgeTuple.countEdgeBucket = 0;
33. }
35. **void** insertionSort(EdgeTableTuple \*ett)
36. {
37. **int** i,j;
38. EdgeBucket temp;
40. **for** (i = 1; i < ett->countEdgeBucket; i++)
41. {
42. temp.ymax = ett->buckets[i].ymax;
43. temp.xofymin = ett->buckets[i].xofymin;
44. temp.slopeinverse = ett->buckets[i].slopeinverse;
45. j = i - 1;
47. **while** ((temp.xofymin < ett->buckets[j].xofymin) && (j >= 0))
48. {
49. ett->buckets[j + 1].ymax = ett->buckets[j].ymax;
50. ett->buckets[j + 1].xofymin = ett->buckets[j].xofymin;
51. ett->buckets[j + 1].slopeinverse = ett->buckets[j].slopeinverse;
52. j = j - 1;
53. }
54. ett->buckets[j + 1].ymax = temp.ymax;
55. ett->buckets[j + 1].xofymin = temp.xofymin;
56. ett->buckets[j + 1].slopeinverse = temp.slopeinverse;
57. }
58. }

61. **void** storeEdgeInTuple (EdgeTableTuple \*receiver,**int** ym,**int** xm,**float** slopInv)
62. {
63. (receiver->buckets[(receiver)->countEdgeBucket]).ymax = ym;
64. (receiver->buckets[(receiver)->countEdgeBucket]).xofymin = (**float**)xm;
65. (receiver->buckets[(receiver)->countEdgeBucket]).slopeinverse = slopInv;
67. insertionSort(receiver);
69. (receiver->countEdgeBucket)++;
71. }
73. **void** storeEdgeInTable (**int** x1,**int** y1, **int** x2, **int** y2)
74. {
75. **float** m,minv;
76. **int** ymaxTS,xwithyminTS, scanline;
78. **if** (x2==x1)
79. {
80. minv=0.000000;
81. }
82. **else**
83. {
84. m = ((**float**)(y2-y1))/((**float**)(x2-x1));
86. **if** (y2==y1)
87. **return**;
89. minv = (**float**)1.0/m;
90. }
92. **if** (y1>y2)
93. {
94. scanline=y2;
95. ymaxTS=y1;
96. xwithyminTS=x2;
97. }
98. **else**
99. {
100. scanline=y1;
101. ymaxTS=y2;
102. xwithyminTS=x1;
103. }
104. storeEdgeInTuple(&EdgeTable[scanline],ymaxTS,xwithyminTS,minv);
105. }
107. **void** removeEdgeByYmax(EdgeTableTuple \*Tup,**int** yy)
108. {
109. **int** i,j;
110. **for** (i=0; i< Tup->countEdgeBucket; i++)
111. {
113. **if** (Tup->buckets[i].ymax == yy)
114. {
115. **for** ( j = i ; j < Tup->countEdgeBucket -1 ; j++ )
116. {
117. Tup->buckets[j].ymax =Tup->buckets[j+1].ymax;
118. Tup->buckets[j].xofymin =Tup->buckets[j+1].xofymin;
119. Tup->buckets[j].slopeinverse = Tup->buckets[j+1].slopeinverse;
120. }
121. Tup->countEdgeBucket--;
122. i--;
123. }
124. }
125. }
127. **void** updatexbyslopeinv(EdgeTableTuple \*Tup)
128. {
129. **int** i;
131. **for** (i=0; i<Tup->countEdgeBucket; i++)
132. {
133. (Tup->buckets[i]).xofymin =(Tup->buckets[i]).xofymin + (Tup->buckets[i]).slopeinverse;
134. }
135. }

138. **void** MainWindow::scanLineFill()
139. {
141. **int** i, j, x1, ymax1, x2, ymax2, FillFlag = 0, coordCount;
143. **for** (i=0; i<maxHt; i++)
144. {
145. **for** (j=0; j<EdgeTable[i].countEdgeBucket; j++)
146. {
147. storeEdgeInTuple(&ActiveEdgeTuple,EdgeTable[i].buckets[j].
148. ymax,EdgeTable[i].buckets[j].xofymin,
149. EdgeTable[i].buckets[j].slopeinverse);
150. }
152. removeEdgeByYmax(&ActiveEdgeTuple, i);
154. insertionSort(&ActiveEdgeTuple);
155. j = 0;
156. FillFlag = 0;
157. coordCount = 0;
158. x1 = 0;
159. x2 = 0;
160. ymax1 = 0;
161. ymax2 = 0;
162. **while** (j<ActiveEdgeTuple.countEdgeBucket)
163. {
164. **if** (coordCount%2==0)
165. {
166. x1 = (**int**)(ActiveEdgeTuple.buckets[j].xofymin);
167. ymax1 = ActiveEdgeTuple.buckets[j].ymax;
168. **if** (x1==x2)
169. {
170. **if** (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))
171. {
172. x2 = x1;
173. ymax2 = ymax1;
174. }
176. **else**
177. {
178. coordCount++;
179. }
180. }
181. **else**
182. {
183. coordCount++;
184. }
185. }
186. **else**
187. {
188. x2 = (**int**)ActiveEdgeTuple.buckets[j].xofymin;
189. ymax2 = ActiveEdgeTuple.buckets[j].ymax;
191. FillFlag = 0;
192. **if** (x1==x2)
193. {
194. **if** (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))
195. {
196. x1 = x2;
197. ymax1 = ymax2;
198. }
199. **else**
200. {
201. coordCount++;
202. FillFlag = 1;
203. }
204. }
205. **else**
206. {
207. coordCount++;
208. FillFlag = 1;
209. }
210. **if**(FillFlag)
211. {
212. QPoint px,py;
213. px.setX(x1);px.setY(i);
214. py.setX(x2);py.setY(i);
215. drawBresenham(px,py);
216. }
217. }
218. j++;
219. }
220. updatexbyslopeinv(&ActiveEdgeTuple);
221. }
222. vertex\_list.clear();
223. }

Output of filling methods are shown in Figure 2.

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |
| --- |
|  |
|  |

Figure 2: (a) Flood Fill (b) Boundary Fill (c) Scanline Fill

**Transformation:**

**Translation:**

1. **void** MainWindow::translate\_object(**int** xx,**int** yy){
3. on\_pushButton\_clicked();
4. on\_show\_grid\_clicked();
6. **int** l = vertices.size();
7. **int** i;
9. **for**(i=0;i<l;i++){
10. QPoint p = vertices[i];
11. p.setX(p.x()+xx);
12. p.setY(p.y()+yy);
14. }
16. **for**(i=0;i<l-1;i++){
17. drawBresenham(vertices[i], vertices[i+1]);
18. }
20. drawBresenham(vertices[0], vertices[l-1]);
22. }
24. **void** MainWindow::on\_draw\_ellipse\_2\_clicked()
25. {
26. flag = **true**;
27. }
29. **void** MainWindow::on\_Draw\_2\_clicked()
30. {
31. **int** l = vertices.size();
32. **int** i;
34. **for**(i=0;i<l-1;i++){
35. drawBresenham(vertices[i], vertices[i+1]);
36. }
38. drawBresenham(vertices[0], vertices[l-1]);
39. }
41. **void** MainWindow::on\_Draw\_3\_clicked()
42. {
43. **int** x = ui->hor\_trans->value();
44. **int** y = ui->Ver\_trans->value();
45. **int** k = ui->grid\_size->value();
46. **int** xx = x\*k;
47. **int** yy = y\*k;
48. **int** l = vertices.size();
49. **int** i;

52. on\_pushButton\_clicked();
53. on\_show\_grid\_clicked();
54. **for**(i=0;i<l;i++){
55. QPoint p = vertices[i];
56. p.setX(p.x()+xx);
57. p.setY(p.y()-yy);
58. vertices[i] = p;
60. }
62. **for**(i=0;i<l-1;i++){
63. drawBresenham(vertices[i], vertices[i+1]);
64. }
66. drawBresenham(vertices[0], vertices[l-1]);
67. }

**Reflection about Y axis:**

1. **void** MainWindow::on\_Draw\_4\_clicked()
2. {
4. on\_pushButton\_clicked();
5. on\_show\_grid\_clicked();
6. **int** i;
7. **int** wid = img.width();
8. //int len = img.height();
10. //QVector<QPoint> temp;
11. **int** l = vertices.size();
12. **for**(i=0;i<l;i++){
13. QPoint p = vertices[i];
14. p.setX(wid - p.x());
15. p.setY(p.y());
16. //temp.push\_back(p);
17. vertices[i] = p;
19. }
21. **for**(i=0;i<l-1;i++){
22. drawBresenham(vertices[i], vertices[i+1]);
23. }
25. drawBresenham(vertices[0], vertices[l-1]);
26. }

**Reflection about X axis:**

1. **void** MainWindow::on\_Draw\_5\_clicked()
2. {
4. on\_pushButton\_clicked();
5. on\_show\_grid\_clicked();
6. **int** i;
7. //int wid = img.width();
8. **int** len = img.height();
10. //QVector<QPoint> temp;
11. **int** l = vertices.size();
12. **for**(i=0;i<l;i++){
13. QPoint p = vertices[i];
14. p.setX(p.x());
15. p.setY( len - p.y());
16. //  temp.push\_back(p);
17. vertices[i] = p;
19. }
21. **for**(i=0;i<l-1;i++){
22. drawBresenham(vertices[i], vertices[i+1]);
23. }
25. drawBresenham(vertices[0], vertices[l-1]);
26. }

**Scaling:**

1. **void** MainWindow::on\_Draw\_6\_clicked()
2. {
3. on\_pushButton\_clicked();
4. on\_show\_grid\_clicked();
6. **int** x = ui->scaleratio->value();
7. **int** y = ui->scaleratio\_2->value();
8. **int** i;
9. **int** wid = img.width();
10. wid = wid/2;
11. **int** len = img.height();
12. len = len/2;
14. //QVector<QPoint> temp;
15. **int** l = vertices.size();
16. **for**(i=0;i<l;i++){
17. QPoint p = vertices[i];
18. p.setX((x\*(p.x() - wid)) + wid);
19. p.setY((y\*(p.y() - len)) + len);
20. //temp.push\_back(p);
21. vertices[i] = p;
23. }
25. **for**(i=0;i<l-1;i++){
26. drawBresenham(vertices[i], vertices[i+1]);
27. }
29. drawBresenham(vertices[0], vertices[l-1]);
31. }

**Shear:**

1. **void** MainWindow::on\_Draw\_7\_clicked()
2. {
4. on\_pushButton\_clicked();
5. on\_show\_grid\_clicked();
7. **int** x = ui->scaleratio->value();
8. **int** y = ui->scaleratio\_2->value();
9. **int** i;
10. **int** wid = img.width();
11. wid = wid/2;
12. **int** len = img.height();
13. len = len/2;
15. //   QVector<QPoint> temp;
16. **int** l = vertices.size();
17. **for**(i=0;i<l;i++){
18. QPoint p = vertices[i];
20. **int** xx = p.x() - wid;
21. **int** yy = len - p.y();
22. p.setX(xx+yy\*x + wid);
23. p.setY(2\*len - (yy+xx\*y + len));
24. vertices[i] = p;
25. // temp.push\_back(p);
27. }
29. **for**(i=0;i<l-1;i++){
30. drawBresenham(vertices[i], vertices[i+1]);
31. }
33. drawBresenham(vertices[0], vertices[l-1]);
34. }

**Reflection about a line:**

1. **void** MainWindow::on\_Draw\_8\_clicked()
2. {
4. on\_pushButton\_clicked();
5. on\_show\_grid\_clicked();
7. **int** wid = img.width();
8. **int** len = img.height();
9. wid = wid/2;
10. len = len/2;
12. **int** x1 = p1.x() - wid;
13. **int** x2 = p2.x() - wid;
14. **int** y1 = len - p1.y();
15. **int** y2 = len - p2.y();
17. **int** a,b,c;
19. a = y2-y1;
20. b = x1-x2;
21. c = x1\*(y1-y2) - y1\*(x1-x2);
23. **int** i;

26. **int** l = vertices.size();
27. **for**(i=0;i<l;i++){
28. QPoint p = vertices[i];
30. **int** x = p.x() - wid;
31. **int** y = len - p.y();
32. **int** xx = (x\*(b\*b - a\*a) - 2\*a\*(b\*y+c))/(a\*a + b\*b);
33. **int** yy = (y\*(a\*a - b\*b) - 2\*b\*(a\*x+c))/(a\*a + b\*b);
34. p.setX(xx + wid);
35. p.setY(len - yy);
36. vertices[i] = p;
37. }
39. **for**(i=0;i<l-1;i++){
40. drawBresenham(vertices[i], vertices[i+1]);
41. }
43. drawBresenham(vertices[0], vertices[l-1]);
44. drawBresenham(p1,p2);
45. }
47. **void** MainWindow::on\_draw\_ellipse\_3\_clicked()
48. {
49. flag = **false**;
50. // vertices.clear();
51. }
53. **void** MainWindow::on\_Draw\_9\_clicked()
54. {
55. flag = **false**;
56. vertices.clear();
57. }

**Rotation:**

1. **void** MainWindow::on\_Draw\_10\_clicked()
2. {
4. on\_pushButton\_clicked();
5. on\_show\_grid\_clicked();
7. **int** angle = ui->rotate\_point->value();
8. **int** x1 = p1.x();
9. **int** y1 = p1.y();
11. **float** rad = (3.14/180.0)\*angle;
12. **float** s = sin(rad);
13. **float** c = cos(rad);
14. **int** i;
15. **int** l = vertices.size();
16. **for**(i=0;i<l;i++){
17. QPoint p = vertices[i];
19. **int** x = p.x() - x1;
20. **int** y = y1 - p.y();
22. **int** xn = **int**(x\*1.0\*c - y\*1.0\*s);
23. **int** yn = **int**(y\*1.0\*c + x\*1.0\*s);
25. p.setX(xn + x1);
26. p.setY(y1 - yn);
27. vertices[i] = p;
28. }
29. **for**(i=0;i<l-1;i++){
30. drawBresenham(vertices[i], vertices[i+1]);
31. }
33. drawBresenham(vertices[0], vertices[l-1]);
35. }

The output for various transformation is given in Figure 3.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |
|  |  |
| (e) | (f) |

|  |  |
| --- | --- |
|  |  |
| (g) | (h) |

Figure 3: (a) The original drawing (b) Reflection about X axis (c) Reflection about Y axis (d) Reflection about a Line (e) Scaling (f) Shear (g) Translation (h) Rotation.

**Clipping:**

**Cohen-Sutherland Line-Clipping:**

1. **const** **int** INSIDE = 0; // 0000
2. **const** **int** LEFT = 1;   // 0001
3. **const** **int** RIGHT = 2;  // 0010
4. **const** **int** BOTTOM = 4; // 0100
5. **const** **int** TOP = 8;    // 1000
6. **int** x\_max,x\_min,y\_max,y\_min;
8. **void** MainWindow::on\_Draw\_13\_clicked()
9. {
10. QPoint px,py;
11. **int** k=ui->grid\_size->value();
12. x\_max=ui->xmax->value()\*k+350;
13. x\_min=ui->xmin->value()\*k+350;
14. y\_max=350-ui->ymax->value()\*k;
15. y\_min=350-ui->ymin->value()\*k;
16. px.setX(x\_max);
17. px.setY(y\_max);
18. py.setX(x\_min);
19. py.setY(y\_max);
20. drawBresenham(px,py);
21. px.setX(x\_min);
22. px.setY(y\_min);
23. drawBresenham(px,py);
24. py.setX(x\_max);
25. py.setY(y\_min);
26. drawBresenham(px,py);
27. px.setX(x\_max);
28. px.setY(y\_max);
29. drawBresenham(px,py);
30. }
32. **int** MainWindow::computeCode(**double** x, **double** y)
33. {
34. // initialized as being inside
35. **int** code = INSIDE;
36. **if** (x < x\_min)       // to the left of rectangle
37. code |= LEFT;
38. **else** **if** (x > x\_max)  // to the right of rectangle
39. code |= RIGHT;
40. **if** (y > y\_min)       // below the rectangle
41. code |= BOTTOM;
42. **else** **if** (y < y\_max)  // above the rectangle
43. code |= TOP;
45. **return** code;
46. }
48. **int** MainWindow::x\_intersect(**int** x1, **int** y1, **int** x2, **int** y2,**int** x3, **int** y3, **int** x4, **int** y4)
49. {
50. y1=ui->frame->height()-y1+1;
51. y2=ui->frame->height()-y2+1;
52. y3=ui->frame->height()-y3+1;
53. y4=ui->frame->height()-y4+1;
54. **int** num = (x1\*y2 - y1\*x2) \* (x3-x4) -(x1-x2) \* (x3\*y4 - y3\*x4);
55. **int** den = (x1-x2) \* (y3-y4) - (y1-y2) \* (x3-x4);
56. **int** retx=num/den;
57. **return** retx;
58. }
59. **int** MainWindow::y\_intersect(**int** x1, **int** y1, **int** x2, **int** y2,**int** x3, **int** y3, **int** x4, **int** y4)
60. {
61. y1=ui->frame->height()-y1+1;
62. y2=ui->frame->height()-y2+1;
63. y3=ui->frame->height()-y3+1;
64. y4=ui->frame->height()-y4+1;
65. **int** num = (x1\*y2 - y1\*x2) \* (y3-y4) -(y1-y2) \* (x3\*y4 - y3\*x4);
66. **int** den = (x1-x2) \* (y3-y4) - (y1-y2) \* (x3-x4);
67. **int** rety= (ui->frame->height()-num/den+1);
68. **return** rety;
69. }
71. **void** MainWindow::on\_Draw\_11\_clicked()
72. {
73. **double** x1=vertices[0].x();
74. **double** y1=vertices[0].y();
75. **double** x2=vertices[1].x();
76. **double** y2=vertices[1].y();
77. **int** code1=computeCode(x1,y1);
78. **int** code2=computeCode(x2,y2);
79. **bool** accept = **false**;
80. **while** (**true**)
81. {
82. **if** ((code1 == 0) && (code2 == 0))
83. {
84. accept = **true**;
85. **break**;
86. }
87. **else** **if** (code1 & code2)
88. {
89. **break**;
90. }
91. **else**
92. {
93. **int** code\_out;
94. **int** x, y;
95. **if** (code1 != 0)
96. code\_out = code1;
97. **else**
98. code\_out = code2;
99. **if** (code\_out & TOP)
100. {
101. x = x1 + (**int**)((**double**)(x2 - x1) \*(**double**)(y\_max - y1) /(**double**)(y2 - y1));
102. y = y\_max;
103. }
104. **else** **if** (code\_out & BOTTOM)
105. {
106. x = x1 + (**int**)((**double**)(x2 - x1) \* (**double**)(y\_min - y1) / (**double**)(y2 - y1));
107. y = y\_min;
108. }
109. **else** **if** (code\_out & RIGHT)
110. {
111. y = y1 + (**int**)((**double**)(y2 - y1) \* (**double**)(x\_max - x1) / (**double**)(x2 - x1));
112. x = x\_max;
113. }
114. **else** **if** (code\_out & LEFT)
115. {
116. y = y1 + (**int**)((**double**)(y2 - y1) \* (**double**)(x\_min - x1) / (**double**)(x2 - x1));
117. x = x\_min;
118. }
119. **if** (code\_out == code1)
120. {
121. x1 = x;
122. y1 = y;
123. code1 = computeCode(x1, y1);
124. }
125. **else**
126. {
127. x2 = x;
128. y2 = y;
129. code2 = computeCode(x2, y2);
130. }
131. }
132. }
133. on\_pushButton\_clicked();
134. on\_show\_grid\_clicked();
135. on\_Draw\_13\_clicked();
137. **if** (accept)
138. {
139. QPoint px,py;
140. px.setX((**int**)x1);
141. px.setY((**int**)y1);
142. py.setX((**int**)x2);
143. py.setY((**int**)y2);
144. drawBresenham(px,py);
145. }
147. }

**Sutherland-Hodgman's polygon-clipping**

1. **void** MainWindow::clip(**int** x1, **int** y1, **int** x2, **int** y2)
2. {
3. **int** poly\_size=vertices.size()-1;
4. **int** new\_poly\_size = 0;
5. std::vector<std::pair<**int**,**int**> > new\_points;
6. **for** (**int** i = 0; i < poly\_size; i++)
7. {
8. **int** k = (i+1) % poly\_size;
9. **int** ix = vertices[i].x(), iy = vertices[i].y();
10. **int** kx = vertices[k].x(), ky = vertices[k].y();
11. **int** i\_pos,k\_pos;
12. **if**(x2==x1 && ix>x1) i\_pos=1;
13. **else** **if**(x2==x1 && ix<x1) i\_pos=-1;
14. **else** **if**(y2==y1 && iy<y1) i\_pos=1;
15. **else** i\_pos=-1;
16. **if**(x2==x1 && kx>x1) k\_pos=1;
17. **else** **if**(x2==x1 && kx<x1) k\_pos=-1;
18. **else** **if**(y2==y1 && ky<y1) k\_pos=1;
19. **else** k\_pos=-1;
20. **if**(y1>y2||x1>x2)
21. {
22. i\_pos=(-1)\*i\_pos;
23. k\_pos=(-1)\*k\_pos;
24. }
25. **if** (i\_pos >= 0  && k\_pos >= 0)
26. {
27. new\_points.push\_back(std::make\_pair(kx,ky));
28. new\_poly\_size++;
29. }
30. **else** **if** (i\_pos < 0  && k\_pos >= 0)
31. {
32. new\_points.push\_back(std::make\_pair(x\_intersect(x1,y1, x2, y2, ix, iy, kx, ky),
33. y\_intersect(x1,y1, x2, y2, ix, iy, kx, ky)));
34. new\_poly\_size++;
35. new\_points.push\_back(std::make\_pair(kx,ky));
36. new\_poly\_size++;
37. }
38. **else** **if** (i\_pos >= 0  && k\_pos < 0)
39. {
41. new\_points.push\_back(std::make\_pair(x\_intersect(x1,y1, x2, y2, ix, iy, kx, ky),
42. y\_intersect(x1,y1, x2, y2, ix, iy, kx, ky)));
43. new\_poly\_size++;
44. }
45. **else**
46. {
47. }
48. }
49. poly\_size = new\_poly\_size;
50. vertices.clear();
51. **for** (**int** i = 0; i < new\_points.size(); i++)
52. {
53. QPoint p;
54. p.setX(new\_points[i].first);
55. p.setY(new\_points[i].second);
56. vertices.push\_back(p);
57. }
58. **if**(poly\_size>0){
59. QPoint pp;
60. pp.setX(new\_points[0].first);
61. pp.setY(new\_points[0].second);
62. vertices.push\_back(pp);
63. }
64. }
65. **void** MainWindow::suthHodgClip()
66. {
68. clip(x\_min,y\_max,x\_min,y\_min); //Left
69. **if**(vertices.size()>0)
70. clip(x\_min,y\_min,x\_max,y\_min); //Bottom
71. **if**(vertices.size()>1)
72. clip(x\_max,y\_min,x\_max,y\_max); //Right
73. **if**(vertices.size()>1)
74. clip(x\_max,y\_max,x\_min,y\_max); //Top
75. on\_pushButton\_clicked();
76. on\_show\_grid\_clicked();
77. on\_Draw\_13\_clicked();
78. **int** l=vertices.size();
79. QPoint px,py;
80. **int** k;
81. **if**(l>1){
82. **for**(**int** i=0;i<l-1;i++){
83. k=(i+1)%l;
84. px.setX(vertices[i].x());
85. px.setY(vertices[i].y());
86. py.setX(vertices[k].x());
87. py.setY(vertices[k].y());
88. drawBresenham(px,py);
89. }
91. }
93. }
95. **void** MainWindow::on\_Draw\_12\_clicked()
96. {
97. suthHodgClip();
98. }

Output for the above two clipping algorithm is shown in Figure 4.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |
|  |  |
| (c) | (d) |

Figure 4: (a) The straight line and the clipping window (b) The straight line after clipping (c) The polygon and the clipping window (d) The polygon after clipping

**Bezier Curve:**

1. **void** MainWindow::on\_Fill\_2\_clicked()
2. {
3. **double** xu = 0.0 , yu = 0.0 , u = 0.0 ;
4. **int** i = 0 ;
5. **for**(u = 0.0 ; u <= 1.0 ; u += 0.0001)
6. {
7. xu = pow(1-u,3)\*vertices[0].x()+3\*u\*pow(1-u,2)\*vertices[1].x()+3\*pow(u,2)\*(1-u)\*vertices[2].x()+pow(u,3)\*vertices[3].x();
8. yu = pow(1-u,3)\*vertices[0].y()+3\*u\*pow(1-u,2)\*vertices[1].y()+3\*pow(u,2)\*(1-u)\*vertices[2].y()+pow(u,3)\*vertices[3].y();
9. point((**int**)xu , (**int**)yu);
10. }
11. }

Output for Bazier curve is shown in Figure 5.

|  |
| --- |
|  |
|  |

Figure 5: (a) Bazier curve