**计算机图形学**

**实习报告**

班级： 191174

姓名： 牟鑫一

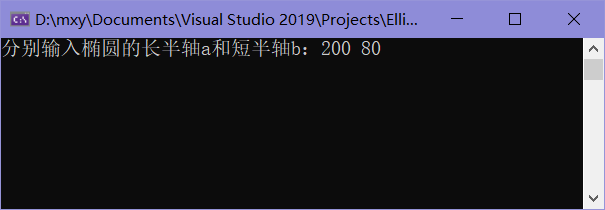
学号： 20161001764

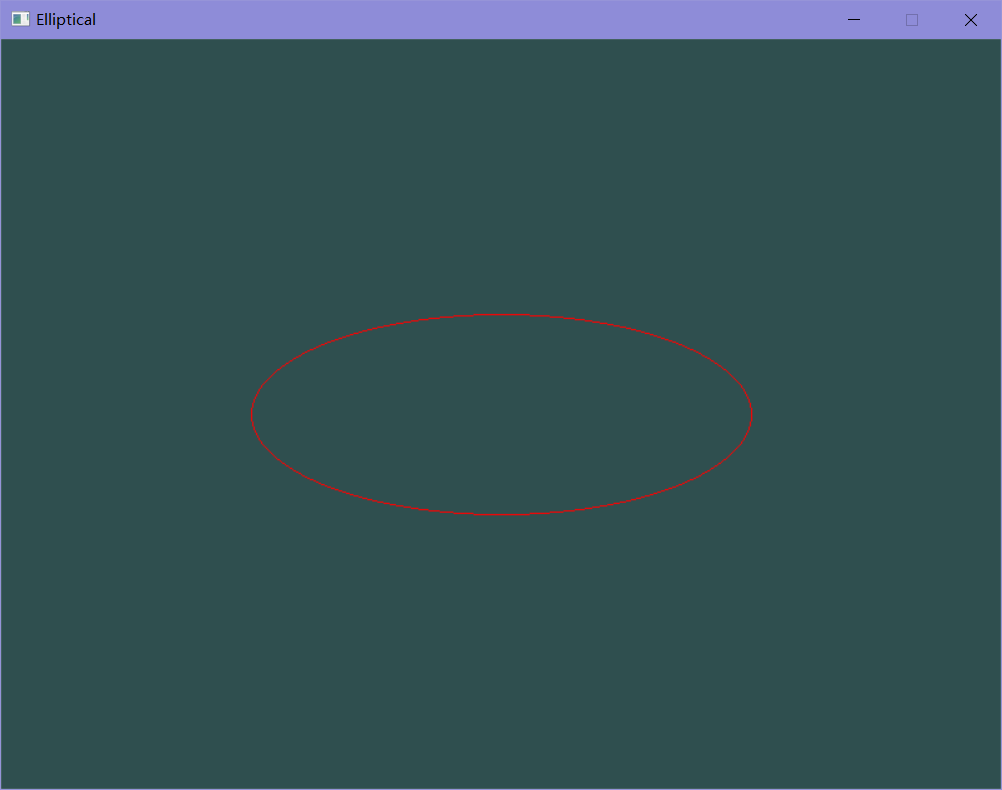
日期： 2019/6/9

1. **椭圆的扫描转换**
2. **题目**

椭圆的扫描转换，用Bresenham算法画椭圆

1. **源代码**
2. #include <iostream>
3. #include <conio.h>
4. #include <graphics.h>
6. **using** **namespace** std;
8. **void** MidBresenhamEllipse(**int** a, **int** b, **int** p1, **int** p2, **int** color)
9. {
10. **int** x, y;
11. **float** d1, d2;
12. x = 0;
13. y = b;
14. d1 = b \* b + a \* a \* (-b + 0.25);
15. putpixel(x + p1, y + p2, color);
16. putpixel(x + p1, -y + p2, color);
17. putpixel(-x + p1, y + p2, color);
18. putpixel(-x + p1, -y + p2, color);
19. **while** (b \* b \* (x + 1) < a \* a \* (y - 0.5))
20. {
21. **if** (d1 <= 0)
22. {
23. d1 += b \* b \* (2 \* x + 3);
24. x++;
25. }
26. **else**
27. {
28. d1 += b \* b \* (2 \* x + 3) + a \* a \* (-2 \* y + 2);
29. x++;
30. y--;
31. }
32. putpixel(x + p1, y + p2, color);
33. putpixel(x + p1, -y + p2, color);
34. putpixel(-x + p1, y + p2, color);
35. putpixel(-x + p1, -y + p2, color);
36. }
37. d2 = b\*b\*(x + 0.5)\*(x + 0.5) + a\*a\*(y - 1)\*(y - 1) - a\*a\*b\*b;
38. **while** (y > 0)
39. {
40. **if** (d2 <= 0)
41. {
42. d2 += b \* b \* (2 \* x + 2) + a \* a \* (-2 \* y + 3);
43. x++;
44. y--;
45. }
46. **else**
47. {
48. d2 += a \* a \* (-2 \* y + 3);
49. y--;
50. }
51. putpixel(x + p1, y + p2, color);
52. putpixel(x + p1, -y + p2, color);
53. putpixel(-x + p1, y + p2, color);
54. putpixel(-x + p1, -y + p2, color);
55. }
56. **return**;
57. }
59. **int** main(**int** argc, **char**\* argv[])
60. {
61. **int** a, b;
62. **int** p1, p2;
63. **int** color;
65. cout << "分别输入椭圆的长半轴a和短半轴b：";
66. cin >> a >> b;                      //输入长半轴、短半轴
67. //cout << "输入圆心坐标(p1,p2)：";
68. //cin >> p1 >> p2;
69. p1 = 400;
70. p2 = 300;
71. //cout << "输入颜色值：";
72. //cin >> color;
73. color = RGB(255, 0, 0);
75. initgraph(800, 600);                //初始化图形环境
76. setbkcolor(RGB(47, 79, 79));        //设置背景色
77. cleardevice();
78. MidBresenhamEllipse(a, b, p1, p2, color);   //画椭圆
79. \_getch();
80. closegraph();
81. **return** 0;
82. }
83. **测试**





1. **解题思路**

定义int型变量a、b分别保存长半轴和短半轴长，定义int型变量p1、p2用于保存圆心坐标(p1,p2)，定义int型变量color用于保存颜色值，具体颜色值可以通过RGB()函数获得。

需要手动输入长半轴长和短半轴长（20-200区间为佳），圆心坐标默认为(400,300)，通过initgraph(800, 600)函数初始化窗口大小为宽800高600，所以圆心坐标即窗口正中心。

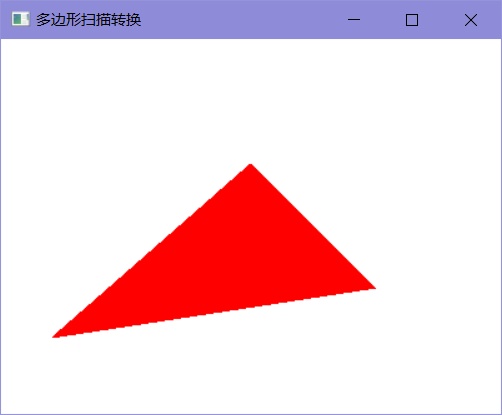
glut库我不太会，最终放弃，所以生成的窗口不像使用glut库生成的窗口一样可以调整窗口大小，所以，窗口大小限制了椭圆能显示的范围，即椭圆的长半轴不能超过400，短半轴不能超过300。

setbkcolor(RGB(47, 79, 79))函数设置窗口背景色为深灰色，color默认设置为红色，然后调用MidBresenhamEllipse()函数，传入长半轴长、短半轴长、圆心坐标、画笔颜色，画出椭圆。

1. **多边形的扫描转换**
2. **题目**

多边形扫描转换，逐行扫描绘制多边形。

1. **源代码**
2. #include <iostream>
3. #include <vector>
4. #include<windows.h>
5. #include "GL/glut.h"
6. **using** **namespace** std;
8. //窗口宽高
9. **const** **int** windowWidth = 400;
10. **const** **int** windowHeight = 300;
12. //定义用于边表ET和活动边表AET的通用类Edge
13. **class** Edge
14. {
15. **public**:
16. **int** ymax;
17. **float** x;
18. **float** dx;
19. Edge\* next;
20. };
22. Edge\* ET[windowHeight];     //边表
23. Edge\* AET;                  //活动边表
25. //定义用于表示像素点坐标的类Point
26. **class** Point
27. {
28. **public**:
29. **int** x;
30. **int** y;
31. Point(**int** x, **int** y)
32. {
33. **this**->x = x;
34. **this**->y = y;
35. }
36. };
38. //三角形
39. vector<Point> vertices = { Point(40, 60), Point(200, 200), Point(300, 100) };
40. //四边形
41. //vector<Point> vertices = { Point(40, 40), Point(100, 240), Point(240, 220), Point(300, 100) };
43. **void** Pixelfill(GLint x, GLint y)
44. {
45. glColor3f(1.0f, 0.0f, 0.0f);    //设定颜色为红色
46. //glPointSize(3.0);             //点大小为3像素（减少生成时间）
47. glBegin(GL\_POINTS);
48. glVertex2i(x, y);
49. glEnd();
50. glFlush();          //强制刷新缓冲
51. }
53. //多边形扫描填充
54. **void** polygonScan()
55. {
56. //计算最高点的y坐标
57. **int** maxY = 0;
58. **for** (unsigned **int** i = 0; i < vertices.size(); i++)
59. {
60. **if** (vertices[i].y > maxY)
61. {
62. maxY = vertices[i].y;
63. }
64. }
65. //初始化ET和AET
66. Edge\* pET[windowHeight];
67. **for** (**int** i = 0; i < maxY; i++)
68. {
69. pET[i] = **new** Edge();
70. pET[i]->next = nullptr;
71. }
72. AET = **new** Edge();
73. AET->next = nullptr;
74. //清空显示窗口并设置画点颜色为红色
75. glClear(GL\_COLOR\_BUFFER\_BIT);
76. glColor3f(1.0, 0.0, 0.0);
77. glBegin(GL\_POINTS);
78. //建立边表ET
79. **for** (**int** i = 0; i < vertices.size(); i++)
80. {
81. //取出当前点1前后相邻的共4个点
82. **int** x0 = vertices[(i - 1 + vertices.size()) % vertices.size()].x;
83. **int** x1 = vertices[i].x;
84. **int** x2 = vertices[(i + 1) % vertices.size()].x;
85. **int** x3 = vertices[(i + 2) % vertices.size()].x;
86. **int** y0 = vertices[(i - 1 + vertices.size()) % vertices.size()].y;
87. **int** y1 = vertices[i].y;
88. **int** y2 = vertices[(i + 1) % vertices.size()].y;
89. **int** y3 = vertices[(i + 2) % vertices.size()].y;
90. //水平线直接舍弃
91. **if** (y1 == y2) **continue**;
92. //分别计算下端点y坐标、上端点y坐标、下端点x坐标和斜率倒数
93. **int** ymin = y1 > y2 ? y2 : y1;
94. **int** ymax = y1 > y2 ? y1 : y2;
95. **float** x = y1 > y2 ? x2 : x1;
96. **float** dx = (x1 - x2) \* 1.0f / (y1 - y2);
98. **if** (((y1 < y2) && (y1 > y0)) || ((y2 < y1) && (y2 > y3)))
99. {
100. ymin++;
101. x += dx;
102. }
103. //创建新边，插入边表ET
104. Edge\* p = **new** Edge();
105. p->ymax = ymax;
106. p->x = x;
107. p->dx = dx;
108. p->next = pET[ymin]->next;
109. pET[ymin]->next = p;
110. }
111. //扫描线从下往上扫描，y坐标每次加1
112. **for** (**int** i = 0; i < maxY; i++) {
113. //取出ET中当前扫描行的所有边并按x的递增顺序插入AET
114. **while** (pET[i]->next)
115. {
116. //取出ET中当前扫描行表头位置的边
117. Edge\* pInsert = pET[i]->next;
118. Edge\* p = AET;
119. //在AET中搜索合适的插入位置
120. **while** (p->next)
121. {
122. **if** (pInsert->x > p->next->x)
123. {
124. p = p->next;
125. **continue**;
126. }
127. **if** (pInsert->x == p->next->x && pInsert->dx > p->next->dx)
128. {
129. p = p->next;
130. **continue**;
131. }
132. //找到位置
133. **break**;
134. }
135. //将pInsert从ET中删除，并插入AET的当前位置
136. pET[i]->next = pInsert->next;
137. pInsert->next = p->next;
138. p->next = pInsert;
139. }
140. //AET中的边两两配对并填色
141. Edge\* p = AET;
142. **while** (p->next && p->next->next)
143. {
144. **for** (**int** x = p->next->x; x < p->next->next->x; x++)
145. {
146. Pixelfill(x, i);
147. }
148. p = p->next->next;
149. }
150. p = AET;
151. **while** (p->next)
152. {
153. **if** (p->next->ymax == i)
154. {
155. Edge\* pDelete = p->next;
156. p->next = pDelete->next;
157. pDelete->next = nullptr;
158. **delete** pDelete;
159. }
160. **else**
161. {
162. p = p->next;
163. }
164. }
165. p = AET;
166. **while** (p->next)
167. {
168. p->next->x += p->next->dx;
169. p = p->next;
170. }
171. }
172. }
174. **int** main(**int** argc, **char**\*\* argv)
175. {
176. glutInit(&argc, argv);
177. glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);
178. glutInitWindowPosition(100, 100);                   //设置窗口位置
179. glutInitWindowSize(windowWidth, windowHeight);      //设置窗口大小
180. glutCreateWindow("多边形扫描转换");                    //创建窗口
181. glClearColor(1.0, 1.0, 1.0, 0.0);
182. glMatrixMode(GL\_PROJECTION);
183. gluOrtho2D(0.0, windowWidth, 0.0, windowHeight);    //定义一个二维图像投影矩阵
184. glutDisplayFunc(polygonScan);     //注册绘图函数polygonScan()
185. glutMainLoop();         //循环绘图
186. **return** 0;
187. }
188. **测试**



图片包含 名片

描述已自动生成

1. **解题思路**

glut不太会，这部分主要还是参考的网上的代码，花了挺多时间在配置环境上，这些天配置了glut、freeglut、glew、glfw等OpenGL相关的库，有些也不太懂，只是想那么按着教程装上了，各个库的配置也不尽相同，有的是直接下载然后把相应的文件拷贝到对应的目录下就行，有的下载下来后需要先编译，然后才拷贝到对应路径下，也算是学了点东西。

这个程序能画一般的多边形，但是目前还不能画凹多边形（画出来是错误的）。

多边形的扫描转换需要输入多边形的顶点坐标，根据顶点坐标算出最高点的纵坐标作为逐行扫描的上界（由下往上逐行扫描）。没有做动态输入顶点坐标的函数，所以需要在代码里面具体给出某一多边形的顶点坐标。

还需要继续学习以下使用glut库作图，关于glut的教程似乎不多，很多地方理解的不透彻，然后可以再完善一下这个程序。