**计算机图形学实验**

**姓 名：白文强**

**学 号：20191060064**

**专 业：计算机科学与技术**

**教 师：钱文华**

实验五 填充算法

时间：2020年10月30日

地点：信息学院2202

1、实验内容

a.教材P66，填充六边形

b.使用opengl，用扫描线填充算法填充多边形

2、实验目的

验证扫描线填充算法，指定任意的多边形边数，填充多边形。

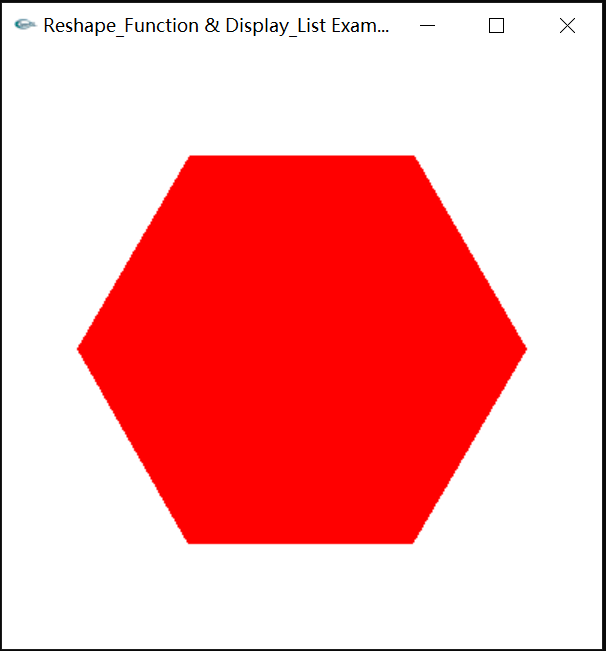
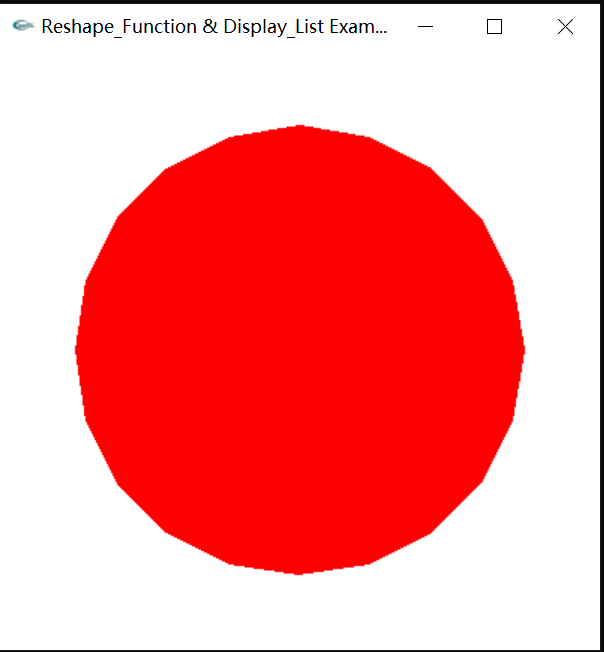
3、实验代码（扫描线）

1. #include <windows.h>
2. #include <GL/glut.h>
3. #include <stdio.h>
4. #include <stdlib.h>
6. #define NULL 0
7. #define WINDOW\_HEIGHT 400
8. #define WINDOW\_WIDTH 400
9. **struct** dcPt {
10. **int** x;
11. **int** y;
12. };
14. **typedef** **struct** tEdge {
15. **int** yUpper;
16. **float** xIntersect, dxPerScan;
17. **struct** tEdge \* next;
18. } Edge;
20. **void** setPixel(GLint x, GLint y) {
21. glBegin(GL\_POINTS);
22. glVertex2i(x, y);
23. glEnd();
24. }
26. /\* Inserts edge into list in order of increasing xIntersect field. \*/
27. **void** insertEdge(Edge \* list, Edge \* edge) {
28. Edge \* p, \* q = list;
29. p = q->next;
30. **while**(p != NULL) {
31. **if**(edge->xIntersect < p->xIntersect)
32. p = NULL;
33. **else** {
34. q = p;
35. p = p->next;
36. }
37. }
38. edge->next = q->next;
39. q->next = edge;
40. }
41. /\* For an index, return y-coordinate
42. of next nonhorizontal line \*/
43. **int** yNext(**int** k, **int** cnt, dcPt \* pts) {
44. **int** j;
45. **if**((k + 1) > (cnt - 1))
46. j = 0;
47. **else**
48. j = k + 1;
49. **while**(pts[k].y == pts[j].y)
50. **if**((j + 1) > (cnt - 1))
51. j = 0;
52. **else**
53. j++;
54. **return** (pts[j].y);
55. }
56. /\* Store lower-y coordinate and inverse slope for each edge.  Adjust and store upper-y coordinate for edges that are the lower member of a monotically increasing or decreasing pair of edges \*/
57. **void** makeEdgeRec
58. (dcPt lower, dcPt upper, **int** yComp, Edge \*  edge, Edge \* edges[]) {
59. edge->dxPerScan =
60. (**float**)(upper.x - lower.x) / (upper.y - lower.y);
61. edge->xIntersect = lower.x;
62. **if**(upper.y < yComp)
63. edge->yUpper = upper.y - 1;
64. **else**
65. edge->yUpper = upper.y;
66. insertEdge(edges[lower.y], edge);
67. }
68. **void** buildEdgeList(**int** cnt, dcPt \* pts, Edge \* edges[]) {
69. Edge \* edge;
70. dcPt v1, v2;
71. **int** i, yPrev = pts[cnt - 2].y;
72. v1.x = pts[cnt - 1].x;
73. v1.y = pts[cnt - 1].y;
74. **for**(i = 0; i < cnt; i++) {
75. v2 = pts[i];
76. **if**(v1.y != v2.y) {                 /\* nonhorizontal line \*/
77. edge = (Edge \*) malloc(**sizeof**(Edge));
78. **if**(v1.y < v2.y)                  /\* up-going edge      \*/
79. makeEdgeRec(v1, v2, yNext(i, cnt, pts), edge, edges);
80. **else**                             /\* down-going edge    \*/
81. makeEdgeRec(v2, v1, yPrev, edge, edges);
82. }
83. yPrev = v1.y;
84. v1 = v2;
85. }
86. }
87. **void** buildActiveList(**int** scan, Edge \* active, Edge \* edges[]) {
88. Edge \* p, \* q;
89. p = edges[scan]->next;
90. **while**(p) {
91. q = p->next;
92. insertEdge(active, p);
93. p = q;
94. }
95. }
96. **void** fillScan(**int** scan, Edge \* active) {
97. Edge \* p1, \* p2;
98. **int** i;
99. p1 = active->next;
100. **while**(p1) {
101. p2 = p1->next;
102. **for**(i = p1->xIntersect; i < p2->xIntersect; i++)
103. setPixel((**int**) i, scan);
104. p1 = p2->next;
105. }
106. }
107. **void** deleteAfter(Edge \* q) {
108. Edge \* p = q->next;
109. q->next = p->next;
110. free(p);
111. }
113. /\* Delete completed edges. Update 'xIntersect' field for others \*/
114. **void** updateActiveList(**int** scan, Edge \* active) {
115. Edge \* q = active, \* p = active->next;
116. **while**(p)
117. **if**(scan >= p->yUpper) {
118. p = p->next;
119. deleteAfter(q);
120. } **else** {
121. p->xIntersect = p->xIntersect + p->dxPerScan;
122. q = p;
123. p = p->next;
124. }
125. }
126. **void** resortActiveList(Edge \* active) {
127. Edge \* q, \* p = active->next;
128. active->next = NULL;
129. **while**(p) {
130. q = p->next;
131. insertEdge(active, p);
132. p = q;
133. }
134. }
135. **void** scanFill(**int** cnt, dcPt \* pts) {
136. Edge \*edges[WINDOW\_HEIGHT], \* active;
137. **int** i, scan;
138. **for**(i = 0; i < WINDOW\_HEIGHT; i++) {
139. edges[i] = (Edge \*) malloc(**sizeof**(Edge));
140. edges[i]->next = NULL;
141. }
142. buildEdgeList(cnt, pts, edges);
143. active = (Edge \*) malloc(**sizeof**(Edge));
144. active->next = NULL;
145. **for**(scan = 0; scan < WINDOW\_HEIGHT; scan++) {
146. buildActiveList(scan, active, edges);
147. **if**(active->next) {
148. fillScan(scan, active);
149. updateActiveList(scan, active);
150. resortActiveList(active);
151. Sleep(10); //停顿10毫秒，放慢填充速度，便于观看填充过程
152. glFlush();
153. }
154. }
155. /\* Free edge records that have been malloc'ed ... \*/
156. }

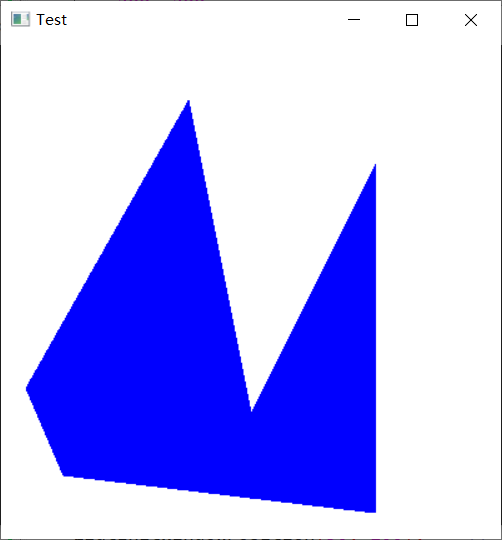
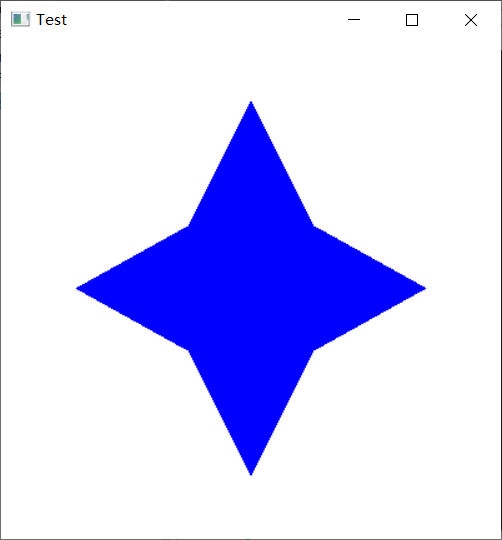
159. **void** init(**void**) {
160. glClearColor(1.0, 1.0, 1.0, 0.0);   // Set display-window color to white.
161. glMatrixMode(GL\_PROJECTION);        // Set projection parameters.
162. gluOrtho2D(0.0, WINDOW\_WIDTH, 0.0, WINDOW\_HEIGHT);
163. }
165. **void** myDraw(**void**) {
166. dcPt pts[] = {
167. 60, 200,
168. 150, 150,
169. 200, 50,
170. 250, 150,
171. 340, 200,
172. 250, 250,
173. 200, 350,
174. 150, 250
175. };
176. glClear(GL\_COLOR\_BUFFER\_BIT);
177. glColor3f(0.0, 1.0, 1.0);
178. scanFill(8, pts);  // 第一个参数为填充图元的顶点数，第二个参数为顶点坐标数组
179. }
181. **int** main(**int** argc, **char**\*\* argv) {
182. glutInit(&argc, argv);                          // Initialize GLUT.
183. glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);    // Set display mode.
184. glutInitWindowPosition(50, 100);    // Set top-left display-window position.
185. glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);       // Set display-window width and height.
186. glutCreateWindow("Test");  // Create display window.
187. init();                              // Execute initialization procedure.
188. glutDisplayFunc(myDraw);        // Send graphics to display window.
189. glutMainLoop();                      // Display everything and wait.
190. **return** 0;
191. }

4、实验结果

a(1). a(2).

b：



5、实验总结

多边形填充过程一般可以分为四个步骤

　　（1）求交：计算扫描线与多边形各边的交点；

　　（2）排序：把所有交点按照递增的顺序进行排序；

　　（3）交点配对：1与2， 3与4等配对处理，每对代表扫描线与多边形的一个相交的区间；

　　（4）区间填充：把这些相交的区间内的像素设置成多边形颜色，填充区间之外的像素设置背景色。

当扫描线与多边形相交时，需要考虑交点的取舍问题，遵循以下原则：

　　① 如果另外两个顶点都高于扫描线，则取这个顶点算取2次，即此顶点可以填充，已经作为填充区间；

　　② 如果另外两个顶点都低于扫描线，则这个交点算取0个，即此顶点不进行填充；

　　③ 如果另外两个顶点一个在扫描线之上，另一个在扫描线之下，则这个交点算取1次，需要与另一个交点进行配对组成填充区间。

通过本次实验，对扫描线填充算法有了更加深刻的理解，自己的代码能力也有了很大的提升。