

计算几何实验报告

200810301 数学三班 李岳锴

目录

第 1 章 直线的扫描转换	3
1.1 实验目的及意义	3
1.2 算法流程	3
1.2.1 中点画线算法的算法流程	3
1.2.2 Bresenham 画线算法的算法流程	4
1.3 结果展示	5
1.3.1 中点画线算法的结果展示	5
1.3.2 Bresenham 画线算法的结果展示	6
1.4 小结	6
第 2 章 圆的扫描转换	7
2.1 实验目的及意义	7
2.2 算法流程	7
2.2.1 中点画圆算法的算法流程	8
2.2.2 Bresenham 画圆算法的算法流程	8
2.3 结果展示	10
2.3.1 中点画圆算法的结果展示	10
2.3.2 Bresenham 画圆算法的结果展示	10
2.4 小结	11
第 3 章 B 样条曲线	12
3.1 实验目的及意义	12
3.2 算法原理	12
3.2.1 结果展示	12

3.3 小结	13
------------------	----

第 1 章 直线的扫描转换

1.1 实验目的及意义

直线的扫描转换算法是计算机图形学中最基础的算法之一，它是用于在屏幕上绘制直线的一种算法。其主要目的是通过计算直线上的像素点，实现在计算机屏幕上绘制直线的功能。

直线的扫描转换算法可以将直线的起点和终点坐标作为输入，然后计算直线上的每一个像素点，并将其绘制在屏幕上，从而实现绘制直线的功能。该算法的意义在于为计算机图形学提供了基础的绘图工具，为更复杂的图形算法的实现提供了基础。

在本次实验中，我们将通过编写程序，实现直线的扫描转换算法，并通过实验观察、比较和分析该算法在不同输入条件下的性能表现。通过实验可以加深对该算法的理解，并掌握基本的计算机图形学绘图技能。同时，也可以通过实验对比不同算法的实现过程，如中点画线算法、Bresenham 画线算法等，加深对计算机图形学的认识和理解。

1.2 算法流程

1.2.1 中点画线算法的算法流程

Algorithm 1 中点画线算法

输入： 起点坐标 (x_0, y_0) ，终点坐标 (x_1, y_1) ，指定颜色 `color`

输出： 绘制线段的像素点

- 1: 计算起点和终点之间的水平、竖直和斜率差值： $\Delta x \leftarrow x_1 - x_0$, $\Delta y \leftarrow y_1 - y_0$,
 $dx \leftarrow |\Delta x|$, $dy \leftarrow |\Delta y|$
- 2: 确定水平方向： $sx \leftarrow \text{sign}(\Delta x)$ ，竖直方向： $sy \leftarrow \text{sign}(\Delta y)$
- 3: **if** $dy > dx$ **then**
- 4: 交换水平和竖直方向： $dx \leftarrow |\Delta y|$, $dy \leftarrow |\Delta x|$, $\text{swap}(sx, sy)$
- 5: 设置初始判别式： $d \leftarrow 2dx - dy$
- 6: **else**
- 7: 设置初始判别式： $d \leftarrow 2dy - dx$
- 8: **end if**
- 9: 绘制起点： `PlotPixel` (x_0, y_0, color)
- 10: $(x, y) \leftarrow (x_0, y_0)$

```

11: for  $i \leftarrow 1$  to  $dx$  do
12:   if  $d \geq 0$  then
13:     更新判别式:  $d \leftarrow d - 2dx$ 
14:     更新竖直方向:  $y \leftarrow y + sy$ 
15:   end if
16:   更新水平方向:  $x \leftarrow x + sx$ 
17:   更新判别式:  $d \leftarrow d + 2dy$ 
18:   绘制像素点:  $\text{PlotPixel}(x, y, \text{color})$ 
19: end for

```

1.2.2 Bresenham 画线算法的算法流程

Algorithm 2 Bresenham 画线算法

输入: 起点坐标 (x_0, y_0) , 终点坐标 (x_1, y_1) , 指定颜色 color

输出: 绘制线段的像素点

```

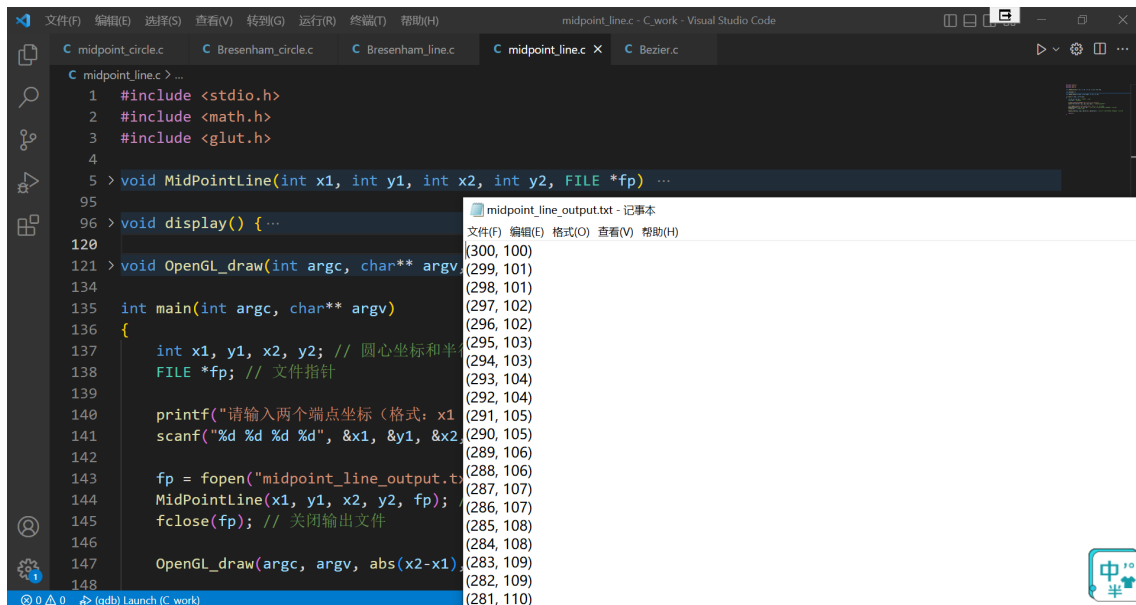
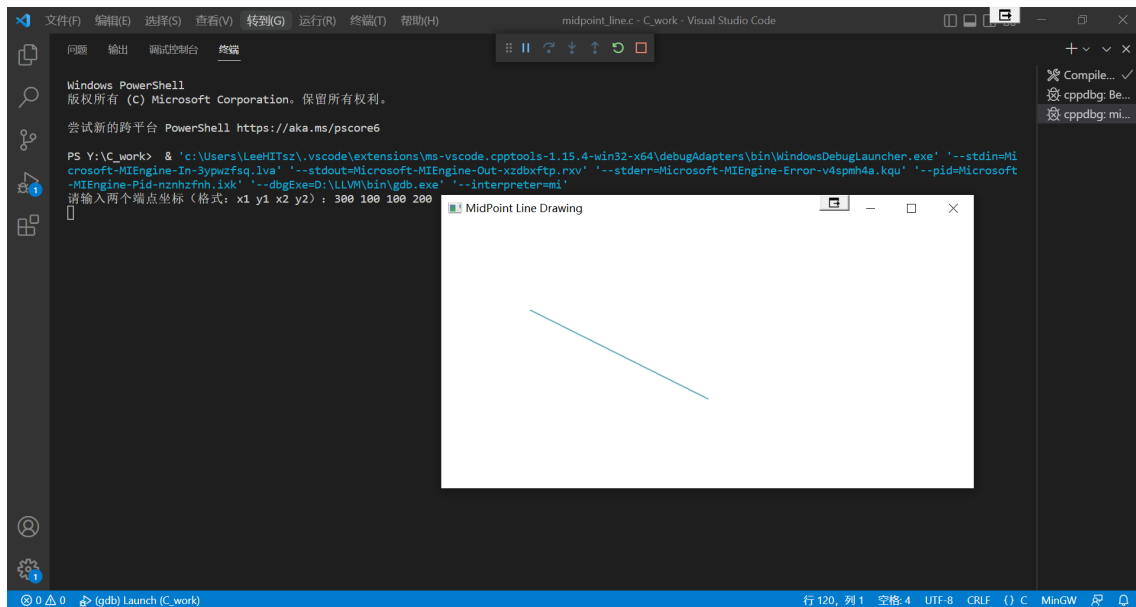
1: 计算起点和终点之间的水平、竖直和斜率差值:  $\Delta x \leftarrow x_1 - x_0$ ,  $\Delta y \leftarrow y_1 - y_0$ 
2: 确定水平方向步进值:  $sx \leftarrow \text{sign}(\Delta x)$ 
3: 确定竖直方向步进值:  $sy \leftarrow \text{sign}(\Delta y)$ 
4: 计算水平和竖直方向差值的绝对值:  $dx \leftarrow |\Delta x|$ ,  $dy \leftarrow |\Delta y|$ 
5: 初始化判别式:  $d \leftarrow 2dy - dx$ 
6: 初始化误差调整值:  $\text{error} \leftarrow 0$ 
7: 绘制起点:  $\text{PlotPixel}(x_0, y_0, \text{color})$ 
8:  $(x, y) \leftarrow (x_0, y_0)$ 
9: while  $x \neq x_1$  or  $y \neq y_1$  do
10:  if  $d \geq 0$  then
11:    调整竖直方向步进值:  $y \leftarrow y + sy$ 
12:    更新判别式:  $d \leftarrow d - 2dx$ 
13:  end if
14:  调整水平方向步进值:  $x \leftarrow x + sx$ 
15:  更新判别式:  $d \leftarrow d + 2dy$ 
16:  绘制像素点:  $\text{PlotPixel}(x, y, \text{color})$ 
17: end while

```

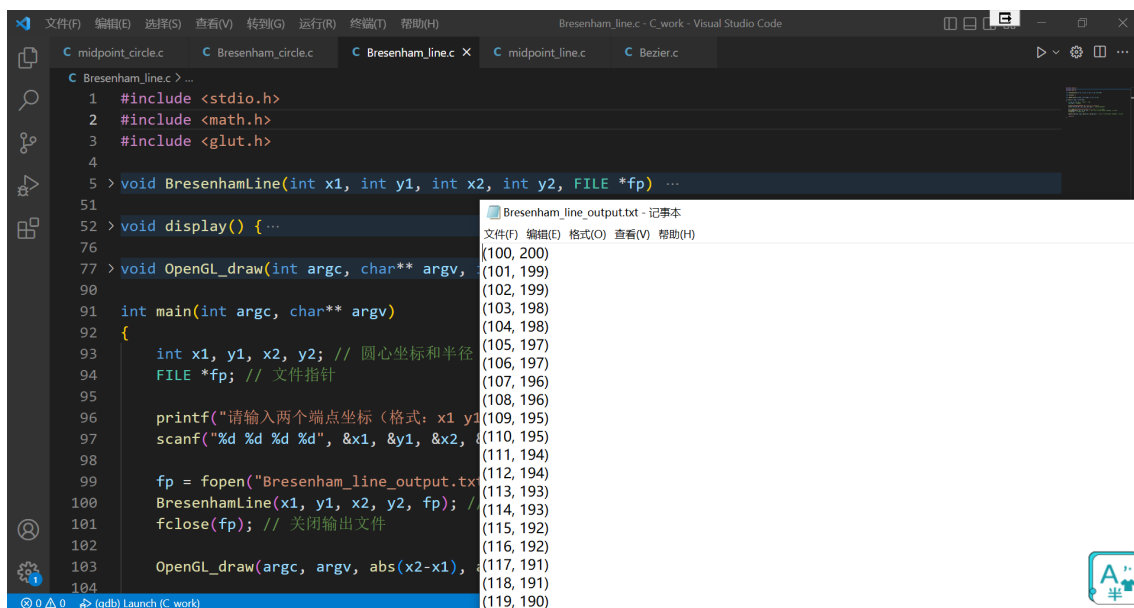
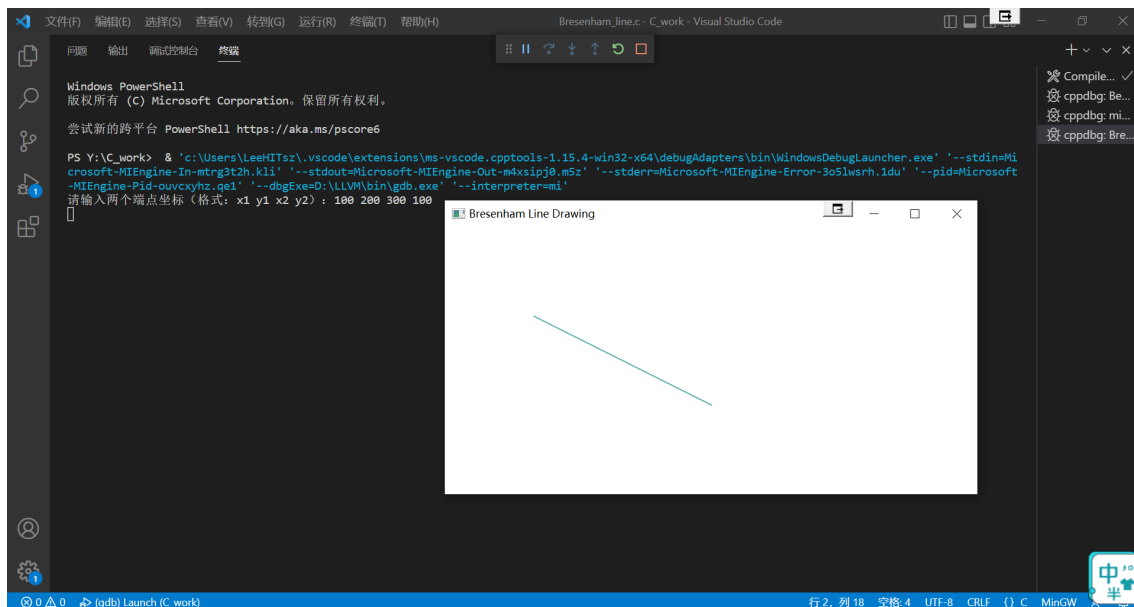
1.3 结果展示

在本次实验中，我们基于 C 语言，实现了中点画线算法以及 Bresenham 画线算法，并借助 OpenGL 库进行了可视化。具体结果如以下截图所示：

1.3.1 中点画线算法的结果展示



1.3.2 Bresenham 画线算法的结果展示



1.4 小结

中点画线算法和 Bresenham 画线算法都是常用的用于绘制直线的算法。它们的核心思想是通过选择适当的像素点来近似绘制直线，以提高绘制效率。

中点画线算法的基本思想是从直线的起点到终点逐步绘制像素点，并根据当前点的位置和与直线的距离来决定下一个要绘制的像素点。算法中通过判别式来判断下一个像

素点的位置，并根据判别式的值进行步进调整。中点画线算法对直线的斜率没有限制，可以适用于任意斜率的直线。然而，在一些特殊情况下，如直线接近水平或竖直方向时，该算法可能会出现不连续或漏点的情况。

Bresenham 画线算法是一种针对整数斜率的直线绘制算法，主要用于提高直线绘制的效率。它利用直线的斜率来选择合适的像素点，从而避免了浮点运算，提高了绘制速度。**Bresenham** 算法使用了判别式和误差调整值来决定下一个要绘制的像素点，并根据当前点与理想直线的差值进行调整。该算法只能处理整数斜率的直线，对于浮点斜率的直线需要进行近似处理。

总体而言，中点画线算法是一种通用的直线绘制算法，适用于任意斜率的直线，但在某些情况下可能会出现不连续或漏点的问题。而 **Bresenham** 画线算法则是一种优化的直线绘制算法，适用于整数斜率的直线，具有更高的绘制效率。根据具体的应用场景和要求，可以选择适合的算法来绘制直线。

第 2 章 圆的扫描转换

2.1 实验目的及意义

圆的扫描转换算法也是计算机图形学中最基本的算法之一，它是用于在屏幕上绘制圆的一种算法。其主要目的是通过计算圆上的像素点，实现在计算机屏幕上绘制圆的功能。

与直线的扫描转换算法相比，圆的扫描转换算法需要更多的计算量和绘制时间。圆的扫描转换算法可以将圆心和半径作为输入，然后计算圆上的每一个像素点，并将其绘制在屏幕上，从而实现绘制圆的功能。该算法的意义在于为计算机图形学提供了基础的绘图工具，为更复杂的图形算法的实现提供了基础。

在本次实验中，我们将通过编写程序，实现圆的扫描转换算法，并通过实验观察、比较和分析该算法在不同输入条件下的性能表现，如圆的半径、绘制速度等。通过实验可以加深对该算法的理解，并掌握基本的计算机图形学绘图技能。同时，也可以通过实验对比不同算法的实现过程，如中点画圆算法、**Bresenham** 画圆算法等，加深对计算机图形学的认识和理解。

2.2 算法流程

2.2.1 中点画圆算法的算法流程

Algorithm 3 中点画圆算法

输入： 圆心坐标 (x_c, y_c) ，半径 r ，指定颜色 color

输出： 绘制圆的像素点

- 1: 初始化: $(x_0, y_0) \leftarrow (0, r)$, $d \leftarrow 1 - r$
 - 2: 绘制起始像素点:
 $\text{glVertex2i}(x_c, y_c + r, \text{color})$, $\text{glVertex2i}(x_c, y_c - r, \text{color})$, $\text{glVertex2i}(x_c + r, y_c, \text{color})$,
 $\text{glVertex2i}(x_c - r, y_c, \text{color})$
 - 3: $(x, y) \leftarrow (x_0, y_0)$
 - 4: **while** $x < y$ **do**
 - 5: **if** $d < 0$ **then**
 - 6: $d \leftarrow d + 2x + 3$
 - 7: $x \leftarrow x + 1$
 - 8: **else**
 - 9: $d \leftarrow d + 2(x - y) + 5$
 - 10: $x \leftarrow x + 1$
 - 11: $y \leftarrow y - 1$
 - 12: **end if**
 - 13: 绘制对称点:
 $\text{glVertex2i}(x_c + x, y_c + y, \text{color})$, $\text{glVertex2i}(x_c - x, y_c + y, \text{color})$, $\text{glVertex2i}(x_c +$
 $x, y_c - y, \text{color})$, $\text{glVertex2i}(x_c - x, y_c - y, \text{color})$, $\text{glVertex2i}(x_c + y, y_c + x, \text{color})$,
 $\text{glVertex2i}(x_c - y, y_c + x, \text{color})$, $\text{glVertex2i}(x_c + y, y_c - x, \text{color})$, $\text{glVertex2i}(x_c -$
 $y, y_c - x, \text{color})$
 - 14: **end while**
-

2.2.2 Bresenham 画圆算法的算法流程

Algorithm 4 Bresenham 画圆算法

输入： 圆心坐标 (x_c, y_c) ，半径 r ，指定颜色 color

输出： 绘制圆的像素点

- 1: 初始化: $(x_0, y_0) \leftarrow (0, r)$, $\Delta \leftarrow 2(1 - r)$

2: 绘制起始像素点:

$\text{glVertex2i}(x_c, y_c + r, \text{color}), \text{glVertex2i}(x_c, y_c - r, \text{color}), \text{glVertex2i}(x_c + r, y_c, \text{color}),$
 $\text{glVertex2i}(x_c - r, y_c, \text{color})$

3: $(x, y) \leftarrow (x_0, y_0)$

4: **while** $y > 0$ **do**

5: **if** $\Delta < 0$ **then**

6: **if** $\delta_{HD} < 0$ **then**

7: $x \leftarrow x + 1$

8: $\Delta \leftarrow \Delta + 2x + 1$

9: **else**

10: $x \leftarrow x + 1$

11: $y \leftarrow y - 1$

12: $\Delta \leftarrow \Delta + 2(x - y + 1)$

13: **end if**

14: **else if** $\Delta > 0$ **then**

15: **if** $\delta_{DV} < 0$ **then**

16: $x \leftarrow x + 1$

17: $y \leftarrow y - 1$

18: $\Delta \leftarrow \Delta + 2(x - y + 1)$

19: **else**

20: $y \leftarrow y - 1$

21: $\Delta \leftarrow \Delta - 2y + 1$

22: **end if**

23: **else**

24: $x \leftarrow x + 1$

25: $y \leftarrow y - 1$

26: $\Delta \leftarrow \Delta + 2(x - y + 1)$

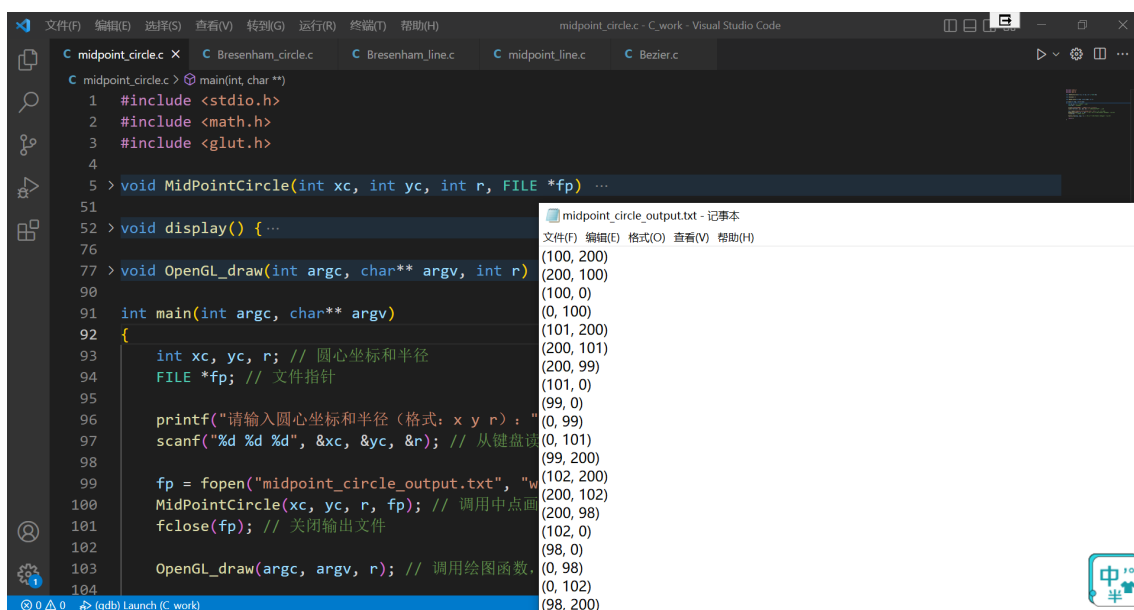
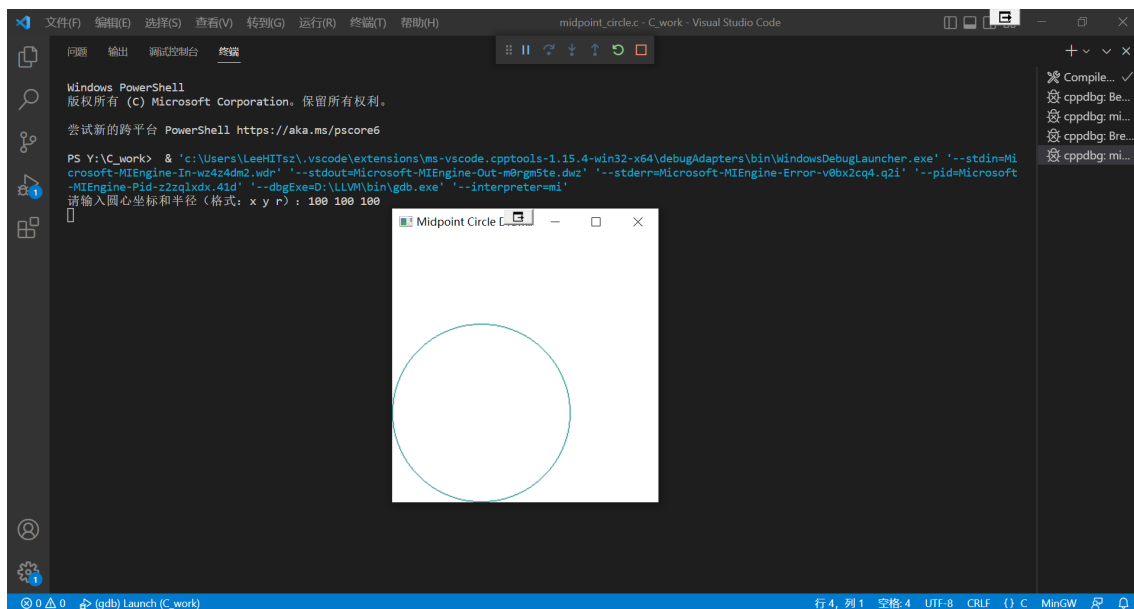
27: **end if**

28: 绘制其他四分圆域的对称点

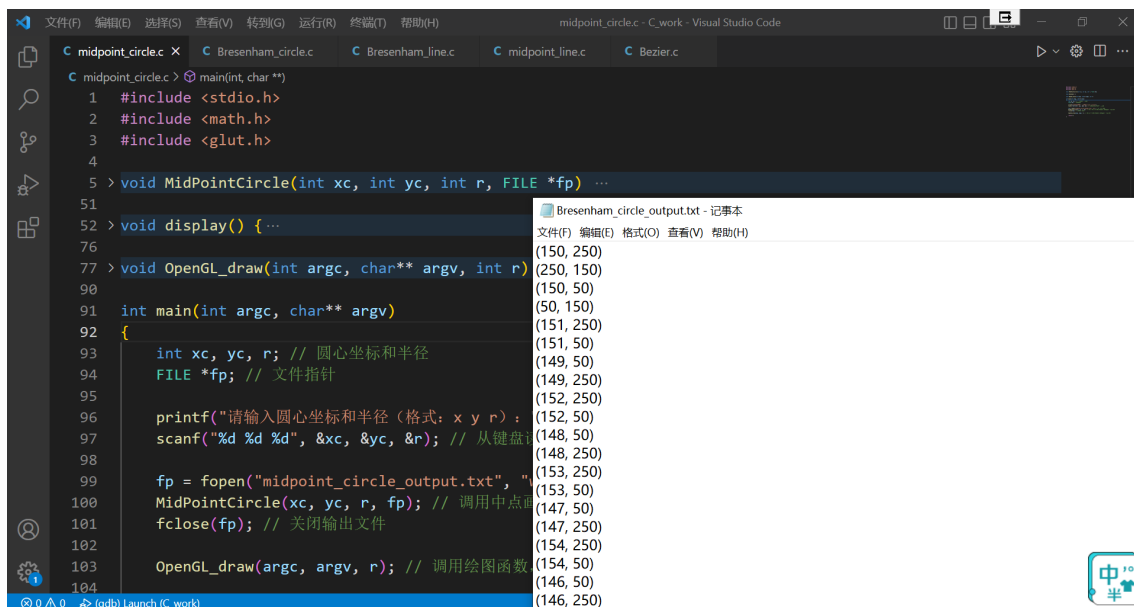
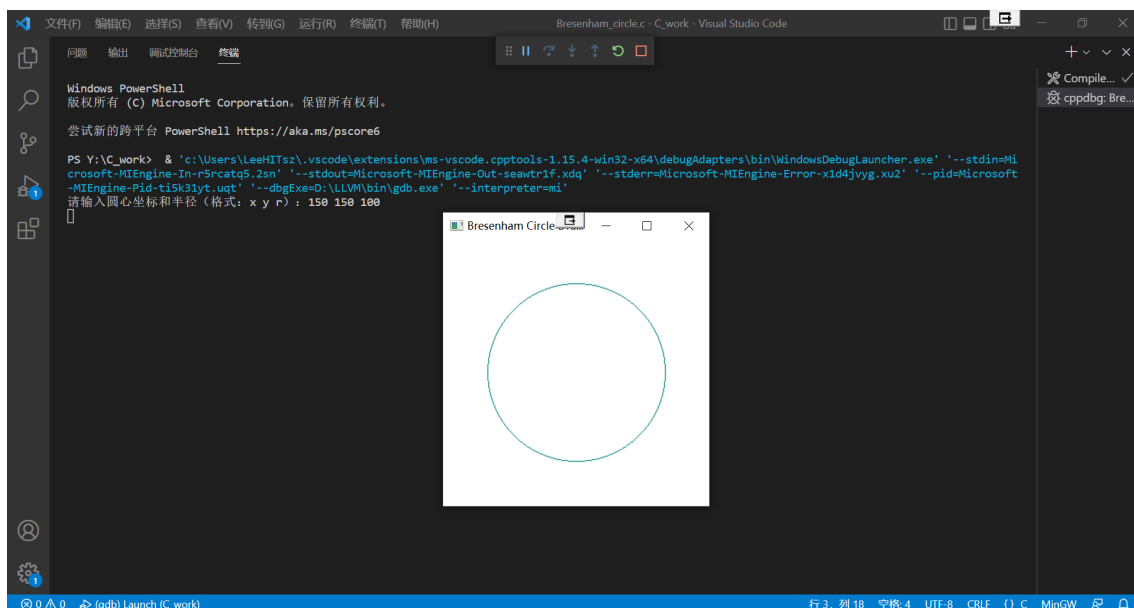
29: **end while**

2.3 结果展示

2.3.1 中点画圆算法的结果展示



2.3.2 Bresenham 画圆算法的结果展示



2.4 小结

中点画圆算法和 Bresenham 画圆算法都是常用的用于绘制圆的算法。它们的核心思想是通过选择适当的像素点来近似绘制圆形，以提高绘制效率。

中点画圆算法的基本思想是从圆的起点开始逐步绘制像素点，并根据当前点的位置和与圆的距离来决定下一个要绘制的像素点。算法利用一个判别式来判断下一个像素点的位置，并根据判别式的值进行步进调整。中点画圆算法适用于绘制任意大小的圆，但在某些情况下，绘制结果可能出现不完美的圆形。

Bresenham 画圆算法是一种优化的圆绘制算法，通过利用圆的对称性和像素点之间

的关系来提高绘制效率。该算法根据圆的四分对称性，只绘制一个四分圆，然后通过对称操作将其复制到其他三个四分圆的位置。**Bresenham** 画圆算法利用一个判别式来决定下一个要绘制的像素点，并根据判别式的值进行步进调整。该算法适用于绘制整数半径的圆，对于浮点半径的圆需要进行近似处理。

总体而言，中点画圆算法是一种通用的圆绘制算法，适用于任意大小的圆，但可能出现不完美的圆形。而 **Bresenham** 画圆算法则是一种优化的圆绘制算法，适用于整数半径的圆，具有更高的绘制效率。根据具体的应用场景和要求，可以选择适合的算法来绘制圆形。

第 3 章 B 样条曲线

3.1 实验目的及意义

B 样条曲线 (B-Spline Curve) 是一种平滑的曲线，广泛应用于计算机图形学、CAD/CAM、数值分析等领域。B 样条曲线由一系列控制点和节点构成，通过对节点和控制点的调整，可以灵活地控制曲线的形状和光滑度。

B 样条曲线的构造方法是在给定节点序列的情况下，利用递推关系式计算出每个节点对应的基函数，然后将基函数加权叠加得到曲线。B 样条曲线的优点是具有局部控制性，即曲线的形状只受到控制点局部范围内的影响；并且具有光滑性，即曲线在控制点处有一阶连续性。因此，B 样条曲线比其他曲线更适合于进行插值和逼近问题的求解，同时也更适合于进行曲面生成和形状设计等应用。

3.2 算法原理

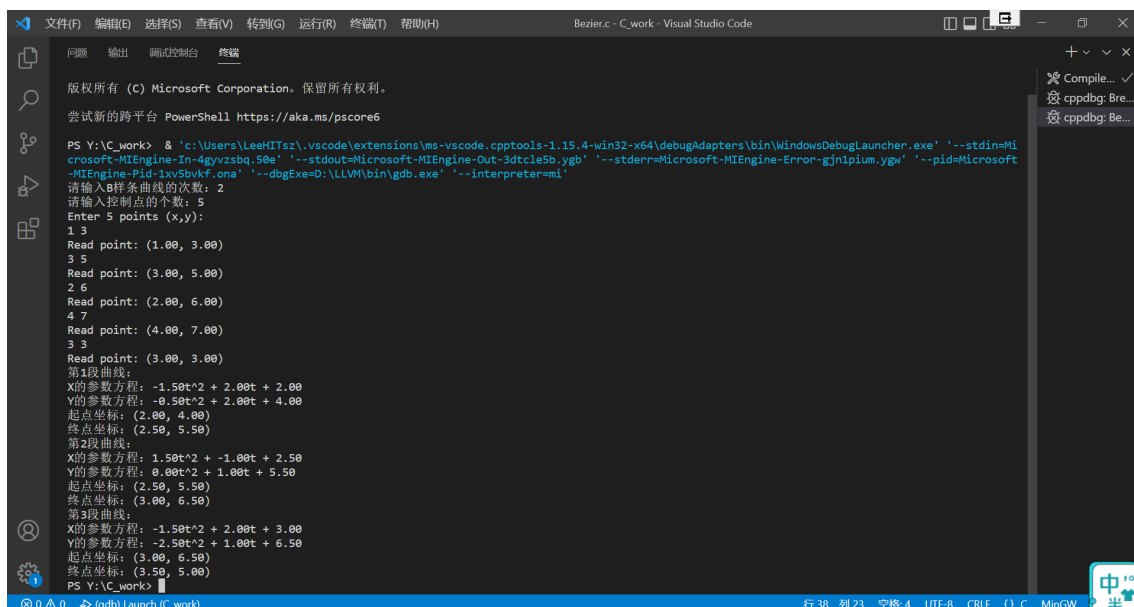
二次 B 样条曲线：

$$P(t) = \frac{1}{2}(t-1)^2 P_0 + \frac{1}{2}(-2t^2 + 2t + 1) P_1 + \frac{1}{2}t^2 P_2 \quad 0 \leq t \leq 1$$

三次 B 样条曲线：

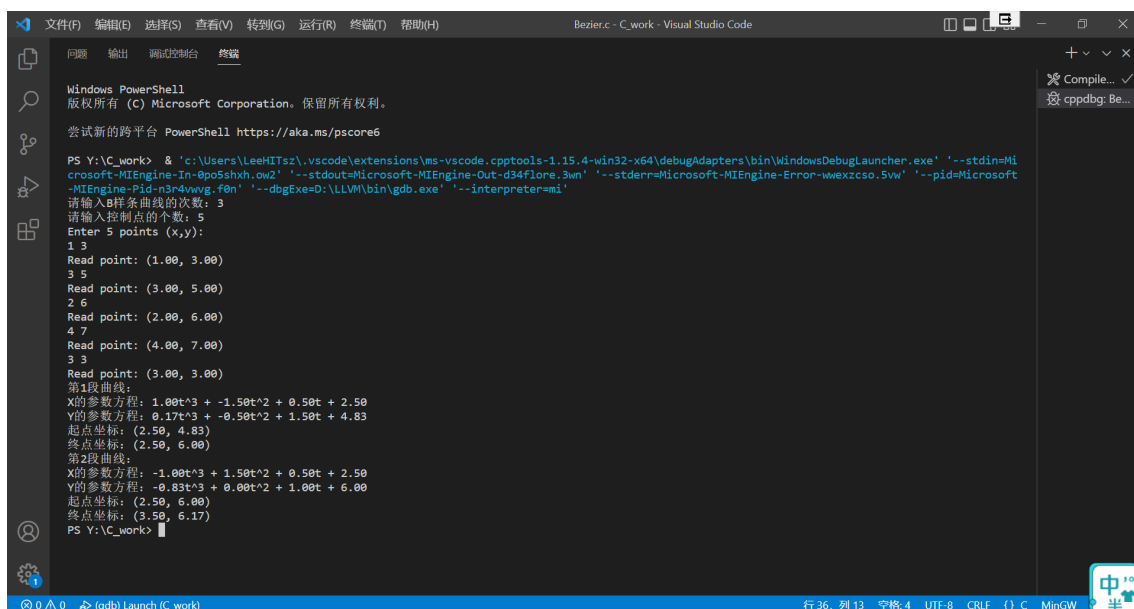
$$P(t) = \frac{1}{6}(-t^3 + 3t^2 - 3t + 1) P_0 + \frac{1}{6}(3t^3 - 6t^2 + 4) P_i + \frac{1}{6}(-3t^3 + 3t^2 + 3t + 1) P_2 + \frac{1}{6}t^3 P_3 \quad 0 \leq t \leq 1.$$

3.2.1 结果展示



```
PS Y:\C_work> & 'c:\Users\LeeHITsz\.vscode\extensions\ms-vscode.cpptools-1.15.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-4gyvzsbq.50e' '--stdout=Microsoft-MIEngine-Out-3dtcle5b.ygb' '--stderr=Microsoft-MIEngine-Error-gjn1plum.ygw' '--pid=Microsoft-MIEngine-Pid-1xv5bvkf.ona' '--dbgExe=D:\LLVM\bin\gdb.exe' '--interpreter=mi'

请输入B样条曲线的次数: 2
请输入控制点的个数: 5
Enter 5 points (x,y):
1 3
Read point: (1.00, 3.00)
3 5
Read point: (3.00, 5.00)
2 6
Read point: (2.00, 6.00)
4 7
Read point: (4.00, 7.00)
3 3
Read point: (3.00, 3.00)
第1段曲线:
X的参数方程: -1.50t^2 + 2.00t + 2.00
Y的参数方程: -0.50t^2 + 2.00t + 4.00
起点坐标: (2.00, 4.00)
终点坐标: (2.50, 5.50)
第2段曲线:
X的参数方程: 1.50t^2 + -1.00t + 2.50
Y的参数方程: 0.00t^2 + 1.00t + 5.50
起点坐标: (2.50, 5.50)
终点坐标: (3.00, 6.50)
第3段曲线:
X的参数方程: -1.50t^2 + 2.00t + 3.00
Y的参数方程: -2.50t^2 + 1.00t + 6.50
起点坐标: (3.00, 6.50)
终点坐标: (3.50, 5.00)
PS Y:\C_work>
```



```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS Y:\C_work> & 'c:\Users\LeeHITsz\.vscode\extensions\ms-vscode.cpptools-1.15.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-0po5shxh.ow2' '--stdout=Microsoft-MIEngine-Out-d34flore.3wn' '--stderr=Microsoft-MIEngine-Error-wwezcs0.5vw' '--pid=Microsoft-MIEngine-Pid-n3r4vwwg.f0n' '--dbgExe=D:\LLVM\bin\gdb.exe' '--interpreter=mi'

请输入B样条曲线的次数: 3
请输入控制点的个数: 5
Enter 5 points (x,y):
1 3
Read point: (1.00, 3.00)
3 5
Read point: (3.00, 5.00)
2 6
Read point: (2.00, 6.00)
4 7
Read point: (4.00, 7.00)
3 3
Read point: (3.00, 3.00)
第1段曲线:
X的参数方程: 1.00t^3 + -1.50t^2 + 0.50t + 2.50
Y的参数方程: 0.17t^3 + -0.50t^2 + 1.50t + 4.83
起点坐标: (2.50, 4.83)
终点坐标: (2.50, 6.00)
第2段曲线:
X的参数方程: -1.00t^3 + 1.50t^2 + 0.50t + 2.50
Y的参数方程: -0.83t^3 + 0.00t^2 + 1.00t + 6.00
起点坐标: (2.50, 6.00)
终点坐标: (3.50, 6.17)
PS Y:\C_work>
```

3.3 小结

B样条曲线作为一种平滑的曲线，具有以下优点：

1. 具有局部控制性：B样条曲线的形状只受到控制点局部范围内的影响，可以对曲线进行局部调整和修改，而不会影响到整个曲线的形状。
2. 具有光滑性：B样条曲线在控制点处有一阶连续性，可以生成光滑的曲线，同时也能在控制点处产生平滑的曲率。
3. 支持高阶拟合：B样条曲线可以通过增加节点的数量来实现高阶拟合，能够逼近比较复杂的曲线形状。

4. 支持多种拓扑结构：B 样条曲线可以通过调整节点的位置和权值来实现多种拓扑结构，如开放曲线、封闭曲线、环面、球面和扭曲曲面等。

然而，B 样条曲线也存在以下缺点：

1. 对初始控制点的敏感性：B 样条曲线对初始控制点的位置和数量比较敏感，控制点的位置和数量不当会导致曲线出现奇异点或偏离期望形状。

2. 逼近效果不如 Bezier 曲线好：相对于 Bézier 曲线，B 样条曲线对于曲线形状的逼近效果不如 Bezier 曲线好，因为 B 样条曲线的形状受到了节点的影响，可能导致曲线的形状发生变化。

3. 计算复杂度较高：B 样条曲线的计算复杂度较高，特别是在高阶曲线的情况下，需要进行大量的计算和存储，对于计算资源和存储空间的要求比较高。

4. 模板选择困难：选择适当的节点和权值模板对于 B 样条曲线的构造至关重要，但是模板的选择和调整比较困难，需要一定的经验和知识储备。

综上所述，B 样条曲线具有局部控制性、光滑性和支持高阶拟合等优点，但也存在对初始控制点敏感、逼近效果不如 Bezier 曲线好、计算复杂度较高和模板选择困难等缺点。因此，在应用 B 样条曲线时需要综合考虑其优缺点，选择合适的方法和技巧来解决具体问题。

附录

中点画线算法代码

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <glut.h>
4
5  void MidPointLine(int x1, int y1, int x2, int y2, FILE *fp)
6  {
7      int dx = x2 - x1;
8      int dy = y2 - y1;
9      int a = -dy;
10     int b = dx;
11     if (b < 0)
12     {
```

```

13     a = -a;
14     b = -b;
15 }
16 double m = fabs(1.0 * dy / dx);
17 int sx = (x1 > x2) ? -1 : 1;
18 int sy = (y1 > y2) ? -1 : 1;
19 int d = (m > 1) ? 2 * sy * b + sx * a : 2 * sx * a + sy * b;
20 int x = x1;
21 int y = y1;
22
23 while (1)
24 {
25     fprintf(fp, "(%d, %d)\n", x, y);
26     if (x == x2 && y == y2)
27     {
28         break;
29     }
30
31     if (d <= 0)
32     {
33         if (m > 1)
34         {
35             if (sy > 0)
36             {
37                 y += sy;
38                 d += 2 * b * sy;
39             }
40             else
41             {
42                 x += sx;
43                 y += sy;
44                 d += 2 * (b * sy + a * sx);
45             }
46         }

```

```

47     else
48     {
49         if (sy > 0)
50         {
51             x += sx;
52             y += sy;
53             d += 2 * (b * sy + a * sx);
54         }
55         else
56         {
57             x += sx;
58             d += 2 * a * sx;
59         }
60     }
61 }
62 else
63 {
64     if (m > 1)
65     {
66         if (sy >= 0)
67         {
68             x += sx;
69             y += sy;
70             d += 2 * (b * sy + a * sx);
71         }
72         else
73         {
74             y += sy;
75             d += 2 * b * sy;
76         }
77     }
78     else
79     {
80         if (sy >= 0)

```



```

81         {
82             x += sx;
83             d += 2 * a * sx;
84         }
85         else
86         {
87             x += sx;
88             y += sy;
89             d += 2 * (b * sy + a * sx);
90         }
91     }
92 }
93 }
94 }
95
96 void display() {
97
98     glClear(GL_COLOR_BUFFER_BIT);
99     glColor3f(0.0, 0.5, 0.5);
100    glBegin(GL_POINTS);
101
102    FILE *fp;
103    int x, y;
104
105    fp = fopen("midpoint_line_output.txt", "r"); // 打开输出文件
106    if (fp == NULL)
107    {
108        printf("Error: Failed to open output file.\n");
109        return;
110    }
111    while (fscanf(fp, "%d, %d\n", &x, &y) == 2) // 依次读取每个坐标点
112    {
113        glVertex2i(x, y);
114    }

```

```

115     fclose(fp); // 关闭输出文件
116
117     glEnd();
118     glFlush();
119 }
120
121 void OpenGL_draw(int argc, char** argv, int lx, int ly)
122 {
123     glutInit(&argc, argv);
124     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
125     glutInitWindowSize(3*lx, 3*ly); // 展示3lx*3ly大小的图形窗口
126     glutCreateWindow("MidPoint Line Drawing");
127     glClearColor(1.0, 1.0, 1.0, 1.0);
128     glMatrixMode(GL_PROJECTION);
129     gluOrtho2D(0, 3*lx, 0, 3*ly); //以左下角为坐标轴原点，绘制位于第一象限的像素
130     glutDisplayFunc(display);
131     glutMainLoop();
132     return;
133 }
134
135 int main(int argc, char** argv)
136 {
137     int x1, y1, x2, y2; // 圆心坐标和半径
138     FILE *fp; // 文件指针
139
140     printf("请输入两个端点坐标（格式：x1 y1 x2 y2）：");
141     scanf("%d %d %d %d", &x1, &y1, &x2, &y2); // 从键盘读取端点坐标
142
143     fp = fopen("midpoint_line_output.txt", "w"); // 打开输出文件
144     MidPointLine(x1, y1, x2, y2, fp); //
        调用中点画圆函数，并将输出文件指针传递给函数
145     fclose(fp); // 关闭输出文件
146
147     OpenGL_draw(argc, argv, abs(x2-x1), abs(y2-y1)); //

```

调用绘图函数，并将输出文件指针传递给函数

148

149 return 0;

150 }

Bresenham 画线算法代码

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <glut.h>
4
5  void BresenhamLine(int x1, int y1, int x2, int y2, FILE *fp)
6  {
7      int dx = x2 - x1;
8      int dy = y2 - y1;
9      double m = fabs(1.0 * dy / dx);
10     int sx = (x1 < x2) ? 1 : -1;
11     int sy = (y1 < y2) ? 1 : -1;
12     int e = (m > 1) ? 2 * abs(dx) - abs(dy) : 2 * abs(dy) - abs(dx);
13     int x = x1;
14     int y = y1;
15
16     while (1)
17     {
18         fprintf(fp, "(%d, %d)\n", x, y);
19         if (x == x2 && y == y2) {
20             break;
21         }
22
23         if (e < 0)
24         {
25             if (m > 1)
26                 {
```

```

27         y += sy;
28         e += 2 * abs(dx);
29     }
30     else
31     {
32         x += sx;
33         e += 2 * abs(dy);
34     }
35 }
36 else
37 {
38     x += sx;
39     y += sy;
40     if (m > 1)
41     {
42         e += 2 * abs(dx) - 2 * abs(dy);
43     }
44     else
45     {
46         e += 2 * abs(dy) - 2 * abs(dx);
47     }
48 }
49 }
50 }
51
52 void display() {
53
54     glClear(GL_COLOR_BUFFER_BIT);
55     glColor3f(0.0, 0.5, 0.5);
56     glBegin(GL_POINTS);
57
58     FILE *fp;
59     int x, y;
60

```

```

61     fp = fopen("Bresenham_line_output.txt", "r"); // 打开输出文件
62     if (fp == NULL)
63     {
64         printf("Error: Failed to open output file.\n");
65         return;
66     }
67     while (fscanf(fp, "(%d, %d)\n", &x, &y) == 2) // 依次读取每个坐标点
68     {
69         glVertex2i(x, y);
70     }
71     fclose(fp); // 关闭输出文件
72
73     glEnd();
74     glFlush();
75 }
76
77 void OpenGL_draw(int argc, char** argv, int lx, int ly)
78 {
79     glutInit(&argc, argv);
80     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
81     glutInitWindowSize(3*lx, 3*ly); // 展示3lx*3ly大小的图形窗口
82     glutCreateWindow("Bresenham Line Drawing");
83     glClearColor(1.0, 1.0, 1.0, 1.0);
84     glMatrixMode(GL_PROJECTION);
85     gluOrtho2D(0, 3*lx, 0, 3*ly); //以左下角为坐标轴原点，绘制位于第一象限的像素
86     glutDisplayFunc(display);
87     glutMainLoop();
88     return;
89 }
90
91 int main(int argc, char** argv)
92 {
93     int x1, y1, x2, y2; // 圆心坐标和半径
94     FILE *fp; // 文件指针

```

```

95
96     printf("请输入两个端点坐标（格式：x1 y1 x2 y2）：");
97     scanf("%d %d %d %d", &x1, &y1, &x2, &y2); // 从键盘读取端点坐标
98
99     fp = fopen("Bresenham_line_output.txt", "w"); // 打开输出文件
100    BresenhamLine(x1, y1, x2, y2, fp); //
        调用中点画圆函数，并将输出文件指针传递给函数
101    fclose(fp); // 关闭输出文件
102
103    OpenGL_draw(argc, argv, abs(x2-x1), abs(y2-y1)); //
        调用绘图函数，并将输出文件指针传递给函数
104
105    return 0;
106 }

```

中点画圆算法代码

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <glut.h>
4
5  void MidPointCircle(int xc, int yc, int r, FILE *fp)
6  {
7      int x = 0, y = r; // 初始化初始点
8      int d = 1 - r; // 初始化判别式
9
10     while (y >= x) // 只在1/8圆弧中计算
11     {
12         if (x == 0) // 如果是在y轴上，输出4个对称点
13         {
14             fprintf(fp, "(%d, %d)\n", xc, yc + y);
15             fprintf(fp, "(%d, %d)\n", xc + y, yc);
16             fprintf(fp, "(%d, %d)\n", xc, yc - y);

```

```

17         fprintf(fp, "(%d, %d)\n", xc - y, yc);
18     }
19     else if (x == y) // 如果是在对角线上, 输出4个对称点并结束
20     {
21         fprintf(fp, "(%d, %d)\n", xc + x, yc + y);
22         fprintf(fp, "(%d, %d)\n", xc + x, yc - y);
23         fprintf(fp, "(%d, %d)\n", xc - x, yc - y);
24         fprintf(fp, "(%d, %d)\n", xc - y, yc + y);
25         break;
26     }
27     else // 否则输出8个对称点
28     {
29         fprintf(fp, "(%d, %d)\n", xc + x, yc + y);
30         fprintf(fp, "(%d, %d)\n", xc + y, yc + x);
31         fprintf(fp, "(%d, %d)\n", xc + y, yc - x);
32         fprintf(fp, "(%d, %d)\n", xc + x, yc - y);
33         fprintf(fp, "(%d, %d)\n", xc - x, yc - y);
34         fprintf(fp, "(%d, %d)\n", xc - y, yc - x);
35         fprintf(fp, "(%d, %d)\n", xc - y, yc + x);
36         fprintf(fp, "(%d, %d)\n", xc - x, yc + y);
37     }
38
39     if (d < 0) // 如果判别式小于0, 取下一个像素点
40     {
41         d += 2*x + 3;
42     }
43     else // 否则取下一个像素点, 并将判别式更新
44     {
45         d += 2*(x - y) + 5;
46         y--; // y坐标减1
47     }
48     x++; // x坐标加1
49 }
50 }

```

```

51
52 void display() {
53
54     glClear(GL_COLOR_BUFFER_BIT);
55     glColor3f(0.0, 0.5, 0.5);
56     glBegin(GL_POINTS);
57
58     FILE *fp;
59     int x, y;
60
61     fp = fopen("midpoint_circle_output.txt", "r"); // 打开输出文件
62     if (fp == NULL)
63     {
64         printf("Error: Failed to open output file.\n");
65         return;
66     }
67     while (fscanf(fp, "(%d, %d)\n", &x, &y) == 2) // 依次读取每个坐标点
68     {
69         glVertex2i(x, y);
70     }
71     fclose(fp); // 关闭输出文件
72
73     glEnd();
74     glFlush();
75 }
76
77 void OpenGL_draw(int argc, char** argv, int r)
78 {
79     glutInit(&argc, argv);
80     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
81     glutInitWindowSize(3*r, 3*r); // 展示3r*3r大小的图形窗口
82     glutCreateWindow("Midpoint Circle Drawing");
83     glClearColor(1.0, 1.0, 1.0, 1.0);
84     glMatrixMode(GL_PROJECTION);

```



```

85     gluOrtho2D(0, 3*r, 0, 3*r); //以左下角为坐标轴原点，绘制位于第一象限的像素
86     glutDisplayFunc(display);
87     glutMainLoop();
88     return;
89 }
90
91 int main(int argc, char** argv)
92 {
93     int xc, yc, r; // 圆心坐标和半径
94     FILE *fp; // 文件指针
95
96     printf("请输入圆心坐标和半径（格式：x y r）：");
97     scanf("%d %d %d", &xc, &yc, &r); // 从键盘读取圆心坐标和半径
98
99     fp = fopen("midpoint_circle_output.txt", "w"); // 打开输出文件
100    MidPointCircle(xc, yc, r, fp); //
        调用中点画圆函数，并将输出文件指针传递给函数
101    fclose(fp); // 关闭输出文件
102
103    OpenGL_draw(argc, argv, r); // 调用绘图函数，并将输出文件指针传递给函数
104
105    return 0;
106 }

```

Bresenham 画圆算法代码

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <glut.h>
4
5  void BresenhamCircle(int xc, int yc, int r, FILE *fp)
6  {
7      int x = 0, y = r; // 初始化初始点

```

```

8     int delta = 2*(1 - r); // 初始化判别式
9
10    while (y > 0) // 只在1/4圆弧中计算
11    {
12        if (x == 0) // 如果是在y轴上，输出4个坐标轴上的点
13        {
14            fprintf(fp, "(%d, %d)\n", xc, yc + y);
15            fprintf(fp, "(%d, %d)\n", xc + y, yc);
16            fprintf(fp, "(%d, %d)\n", xc, yc - y);
17            fprintf(fp, "(%d, %d)\n", xc - y, yc);
18        }
19        else // 否则输出4个对称点
20        {
21            fprintf(fp, "(%d, %d)\n", xc + x, yc + y);
22            fprintf(fp, "(%d, %d)\n", xc + x, yc - y);
23            fprintf(fp, "(%d, %d)\n", xc - x, yc - y);
24            fprintf(fp, "(%d, %d)\n", xc - x, yc + y);
25        }
26
27        if (delta < 0) // 判别式小于0时，考虑delta_HD的符号
28        {
29            int delta_HD = 2 * (delta + y) - 1;
30            if (delta_HD >= 0)
31            {
32                delta += 2 * (++x - (--y) + 1);
33            }
34            else
35            {
36                delta += 2 * (++x) + 1;
37            }
38        }
39        else if (delta > 0) // 判别式大于0时，考虑delta_DV的符号
40        {
41            int delta_DV = 2 * (delta - x) - 1;

```

```

42         if (delta_DV < 0)
43         {
44             delta += 2 * (++x - (--y) + 1);
45         }
46         else
47         {
48             delta += (-2) * (--y) + 1;
49         }
50     }
51     else
52     {
53         delta += 2 * (++x - (--y) + 1);
54     }
55 }
56 }
57
58 void display() {
59
60     glClear(GL_COLOR_BUFFER_BIT);
61     glColor3f(0.0, 0.5, 0.5);
62     glBegin(GL_POINTS);
63
64     FILE *fp;
65     int x, y;
66
67     fp = fopen("Bresenham_circle_output.txt", "r"); // 打开输出文件
68     if (fp == NULL)
69     {
70         printf("Error: Failed to open output file.\n");
71         return;
72     }
73     while (fscanf(fp,("(%d, %d)\n", &x, &y) == 2) // 依次读取每个坐标点
74     {
75         glVertex2i(x, y);

```

```

76     }
77     fclose(fp); // 关闭输出文件
78
79     glEnd();
80     glFlush();
81 }
82
83 void OpenGL_draw(int argc, char** argv, int r)
84 {
85     glutInit(&argc, argv);
86     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
87     glutInitWindowSize(3*r, 3*r); // 展示3r*3r大小的图形窗口
88     glutCreateWindow("Bresenham Circle Drawing");
89     glClearColor(1.0, 1.0, 1.0, 1.0);
90     glMatrixMode(GL_PROJECTION);
91     gluOrtho2D(0, 3*r, 0, 3*r); //以左下角为坐标轴原点，绘制位于第一象限的像素
92     glutDisplayFunc(display);
93     glutMainLoop();
94     return;
95 }
96
97 int main(int argc, char** argv)
98 {
99     int xc, yc, r; // 圆心坐标和半径
100     FILE *fp; // 文件指针
101
102     printf("请输入圆心坐标和半径（格式：x y r）：");
103     scanf("%d %d %d", &xc, &yc, &r); // 从键盘读取圆心坐标和半径
104
105     fp = fopen("Bresenham_circle_output.txt", "w"); // 打开输出文件
106     BresenhamCircle(xc, yc, r, fp); //
        调用中点画圆函数，并将输出文件指针传递给函数
107     fclose(fp); // 关闭输出文件
108

```

```

109     OpenGL_draw(argc, argv, r); // 调用绘图函数，并将输出文件指针传递给函数
110
111     return 0;
112 }

```

B 样条曲线算法代码

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void Bspline(int m, int n, float* X, float* Y)
5  {
6      if (m==2)
7      {
8          for(int i=0; i<n-m; i++)
9          {
10             printf("第%d段曲线: \n", i+1);
11             printf("X的参数方程: %.2ft^2 + %.2ft + %.2f\n",
12                 X[i]/2-X[i+1]+X[i+2]/2, X[i+1]-X[i], X[i]/2+X[i+1]/2);
13             printf("Y的参数方程: %.2ft^2 + %.2ft + %.2f\n",
14                 Y[i]/2-Y[i+1]+Y[i+2]/2, Y[i+1]-Y[i], Y[i]/2+Y[i+1]/2);
15             printf("起点坐标: (%.2f, %.2f)\n", X[i]/2+X[i+1]/2, Y[i]/2+Y[i+1]/2);
16             printf("终点坐标: (%.2f, %.2f)\n", X[i+1]/2+X[i+2]/2,
17                 Y[i+1]/2+Y[i+2]/2);
18         }
19         return;
20     }
21     else if(m==3)
22     {
23         for(int i=0; i<n-m; i++)
24         {
25             printf("第%d段曲线: \n", i+1);
26             printf("X的参数方程: %.2ft^3 + %.2ft^2 + %.2ft + %.2f\n",

```

```

        -X[i]/6+X[i+1]/2-X[i+2]/2+X[i+3]/6,
26      X[i]/2-X[i+1]+X[i+2]/2, X[i+2]/2-X[i]/2,
        X[i]/6+2*X[i+1]/3+X[i+2]/6);
27      printf("Y的参数方程: %.2ft^3 + %.2ft^2 + %.2ft + %.2f\n",
        -Y[i]/6+Y[i+1]/2-Y[i+2]/2+Y[i+3]/6,
28      Y[i]/2-Y[i+1]+Y[i+2]/2, Y[i+2]/2-Y[i]/2,
        Y[i]/6+2*Y[i+1]/3+Y[i+2]/6);
29      printf("起点坐标: (%.2f, %.2f)\n", X[i]/6+2*X[i+1]/3+X[i+2]/6,
        Y[i]/6+2*Y[i+1]/3+Y[i+2]/6);
30      printf("终点坐标: (%.2f, %.2f)\n", X[i+1]/6+2*X[i+2]/3+X[i+3]/6,
        Y[i+1]/6+2*Y[i+2]/3+Y[i+3]/6);
31    }
32    return;
33  }
34 }
35
36 int main() {
37
38     int m; // 设置B样条的次数
39     printf("请输入B样条曲线的次数: ");
40     scanf("%d", &m);
41
42     int n; // 设置控制点的个数
43     printf("请输入控制点的个数: ");
44     scanf("%d", &n);
45
46     float *x_coords = (float *)malloc(n * sizeof(float));
47     float *y_coords = (float *)malloc(n * sizeof(float));
48
49     printf("Enter %d points (x,y):\n", n);
50     for (int i = 0; i < n; i++)
51     {
52         scanf("%f %f", &x_coords[i], &y_coords[i]);
53         // 处理读取到的坐标点

```

```
54     printf("Read point: (%.2f, %.2f)\n", x_coords[i], y_coords[i]);
55 }
56
57 Bspline(m, n, x_coords, y_coords);
58
59 free(x_coords);
60 free(y_coords);
61
62 return 0;
63 }
```
