**西南交通大学2023－2024学年第(一)学期期末考试**

**班 级** **学 号** **姓 名**

**密封装订线**  **密封装订线**  **密封装订线**

课程代码 SCAI003412 课程名称 计算机图形学 考试时间 120**分钟**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 题号 | 一 | 二 | 三 | 四 | 五 | 总成绩 |
| 得分 |  |  |  |  |  |  |

评阅老师：

一． 简答题（共20分）

1. 答：（共6分）
2. 图形生成算法的研究，包括基本图形的生成，比如直线，圆/圆弧，多边形，曲线/曲面，多面体，自然现象/自然景物等。（2分）
3. 图形变换算法的研究，包括图形在平面或者空间中的变换，各种视窗之间的转换，平面和空间裁剪，消隐等。（2分）
4. 图形显示技术研究，包括光照、材质、纹理、阴影、透明、光线追踪等技术的研究。（2分）
5. 答：(共4分)
6. 表示方法不同：图形是由基本几何体（直线，点，圆、曲线、三角形等）构成的实体，同时具有几何属性和视觉属性。图像是由很多像素点构成的点阵信息。（1分）
7. 生成方法不同：图形是通过计算机算法生成的，而图像是通过照相机，摄像机等扫描设备或图像生成软件制作而成。（1分）
8. 研究侧重点不同：图形学主要研究如何使用计算机表示几何体，构建几何模型、如何通过建立数学模型或者算法把真实的或者想象的物体显示出来。图像处理主要研究如何将一种图像处理成另一种图像，包括图像增强、复原、解析和理解、编码、压缩、匹配，识别等。（2分）
9. 答：（共10分）

GPU渲染流水线主要包括以下几个部分：

1. 输入装配阶段：该阶段主要是从内存中读入相关的顶点和索引，从而生成几何图形的基本要素。（1分）
2. 顶点着色阶段：其主要完成顶点转换，光照等各种形式的运算。包括将顶点从局部坐标系转换到齐次裁剪坐标系中。（1分）
3. 外壳着色阶段：根据输入的三角形和过一系列的控制点生成新的三角形。（1分）
4. 曲面细分阶段：曲面细分就是将一个大的三角形分解成若干个小的三角形，这样使得显示的图形更加精细。此前需要在CPU中实现这样的操作，现在已可通过GPU来进行处理。（1分）
5. 域着色阶段：对输入的四边形采用横向+纵向，输入的三角形采用重心分割的方式进行分割，形成更多的小三角形。（1分）
6. 几何着色阶段：几何着色并不是必须的，几何着色的好处是可以选择的对输入的点进行筛选，然后输出相应的构成几何图形的基本要素。同时几何着色可以输出一系列的点到内存空间中，以便后续进行绘制。几何着色阶段由几何着色器来完成。（1分）
7. 光栅化阶段：光栅化是一种将几何图元变为二维图像的过程。该过程包含了两部分的工作。第一部分工作：决定窗口坐标中的哪些整型栅格区域被基本图元占用；第二部分工作：分配一个颜色值和一个深度值到各个区域以便进行消隐操作。光栅化的目的，是找出一个几何单元（比如线段或三角形）所覆盖的像素，该部分由硬件完成。（2分）
8. 像素着色阶段：像素着色需要通过编写相应的GPU程序来完成。由于在光栅化阶段确定了相应的点及属性，通过这些点及相应的属性可以在GPU中计算除相应像素的颜色，光照的影响，阴影等处理效果。（2分）

二． 计算题（共30分）

1. Y= 16或17 。（2分）求解过程如下：（4分）

计算方法如下：

d的初值为： 其中和均大于0， 由于m>1,因此  
 (1)d<0时，y=y+1,d=d+2; (2)d>=0时，y=y+1,x=x+1,d=d+

计算步骤如下：

|  |  |  |  |
| --- | --- | --- | --- |
| d | y | x |  |
| -10 | 11 | 10 | 0 |
| 0 | 12 | 11 | -10 |
| -10 | 13 | 11 | 0 |
| 0 | 14 | 12 | -10 |
| -10 | 15 | 12 | 0 |
| 0 | 16 | 13 | -10 |
| -10 | 17 | 13 | 0 |

1. Y= 15或-5 。（2分）求解过程如下：（4分）

根据算法计算相应变量的值，如下表所示。

将圆心平移到（0,0）位置，半径为r,d的初值为2\*(1-r)。当d<0时dk=2\*(d+y)-1;当d>0时，dk=2\*(d-x)+1。根据题目要求，计算到x=2为止，也就是下表的X=3时截止。计算过程如下：

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | d | dk |
| 0 | 10 | -18 | -17 |
| 1 | 10 | -15 | -11 |
| 2 | 10 | -10 | -1 |
| 3 | 10 | -3 | -1 |

在圆心坐标为xc=5,yc=5时，考虑到圆的对称性，在x=2时y=15或-5。

1. t=0时曲线上点的坐标为 （2,1.5） 。（1分）

t=0.5时曲线上点的坐标为 （2.938,1.417） 。（1分）

t=0.8时曲线上点的坐标为 （3.344,1.111） 。（1分）

计算表达式如下：（3分）

=

++

1. 点在世界坐标系中的位置坐标为： （1.366,3,4.366） 。（6分）
2. 交点的坐标分别为：（-1.714,0）(0,2.667)(4.714,10)(5,10.44) 。（2分）

顶点的编码为： 0101 ，的编码为： 1010 。（2分）

交点的编码分别为： (0100) (0000) (0000) (1000) 。（2分）

三． 绘图题（共10分）

1. 填充过程如下（给出每一步对应的填充结果）：（共5分）



待填充多边形（原图）



第1次填充后的结果（1分） 第2次填充后的结果（1分）



第3次填充后的结果（1分） 第4次填充后的结果（1分）

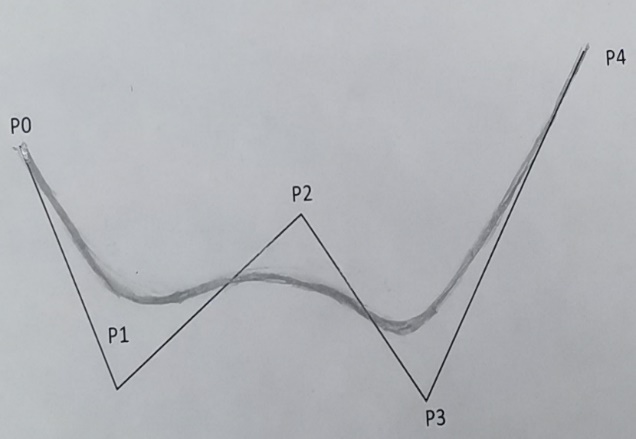


第5次填充后的结果（0.5分） 第6次填充后的结果（0.5分）

1. 如下图所示。（共5分）



t=0.1, 0.5, 0.8时对应的曲线上的点（3分）



采用De Casteljau求出的连续的Bezier曲线（2分）

四． 分析与设计题（共30分）

1. 本题共15分。

（1）顶点的数据结构定义如下（5分）

struct Vertex{

float x,y,z; //顶点的坐标

float nx,ny,nz; //顶点的法向

float u,v; //顶点的u,v坐标

};

struct Vertex V[24];

* 1. 顶点缓冲区中每个顶点的值如下：（6分）

V[0]=Vertex{-1.0,1.0，-1.0,0.0,0.0,-1.0,0.0,0.0}; //A点 前面

V[1]=Vertex{1.0,1.0,-1.0,0.0,0.0,-1.0,0.2,0.0}; //D点 前面

V[2]=Vertex{1.0,-1.0,-1.0,0.0,0.0,-1.0,0.2,0.5}; //C点 前面

V[3]=Vertex{-1.0,-1.0,-1.0,0.0,0.0,-1.0,0.0,0.5}; //B点 前面

V[4]=Vertex{-1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.5}; //E点 后面

V[5]=Vertex{1.0,1.0,1.0,0.0,0.0,1.0,0.2,0.5}; //H点 后面

V[6]=Vertex{1.0,-1.0,1.0,0.0,0.0,1.0,0.2,1.0}; //G点 后面

V[7]=Vertex{-1.0,-1.0,1.0,0.0,0.0,1.0,0.0,1.0}; //F点 后面

V[8]=Vertex{-1.0,1.0,1.0,0.0,1.0,0.0,0.2,0.0}; //E点 上面

V[9]=Vertex{1.0,1.0,1.0,0.0,1.0,0.0,0.4,0.0}; //H点 上面

V[10]=Vertex{1.0,1.0,-1.0,0.0,1.0,0.0,0.4,0.5}; //D点 上面

V[11]=Vertex{-1.0,1.0,-1.0,0.0,1.0,0.0,0.2,0.5}; //A点 上面

V[12]=Vertex{-1.0,-1.0,1.0,0.0,-1.0,0.0,0.4,0.0}; //F点 下面

V[13]=Vertex{1.0,-1.0,1.0,0.0,-1.0,0.0,0.6,0.0}; //G点 下面

V[14]=Vertex{1.0,-1.0,-1.0,0.0,-1.0,0.0,0.6,0.5}; //C点 下面

V[15]=Vertex{-1.0,-1.0,-1.0,0.0,-1.0,0.0,0.4,0.5}; //B点 下面

V[16]=Vertex{-1.0,1.0,1.0,-1.0,0.0,0.0,0.6,0.0}; //E点 左边

V[17]=Vertex{-1.0,1.0,-1.0,-1.0,0.0,0.0,0.8,0.0}; //A点 左边

V[18]=Vertex{-1.0,-1.0,-1.0,-1.0,0.0,0.0,0.8,0.5}; //B点 左边

V[19]=Vertex{-1.0,-1.0,1.0,-1.0,0.0,0.0,0.6,0.5}; //F点 左边

V[20]=Vertex{1.0,1.0,-1.0,1.0,0.0,0.0,0.8,0.0}; //D点 右面

V[21]=Vertex{1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0}; //H点 右面

V[22]=Vertex{1.0,-1.0,1.0,1.0,0.0,0.0,1.0,0.5}; //G点 右面

V[23]=Vertex{1.0,-1.0,-1.0,1.0,0.0,0.0,1.0,0.5}; //C点 右面

索引缓冲区中的索引值如下：（4分）

{0,1,3,3,1,2,//前面 4,5,7,7,5,6,//后面 8,9,11,11,9,12,//上面

12,13,15,15,13,14,//下面 16,17,19,19,17,18 //左面 20,21,23,23,21,22 //右面}

1. 本题共15分。

（1）算法描述如下：（共6分）

输入：（1分）

待裁剪线段P1P2,其中P1（x1,y1），P2(x2,y2),裁剪多边形ABCD，其中A(XL,YT), B(XR,YT), C(XR,YB), D(XL,YB)。

输出：（1分）

裁剪后的线段P3P4。

算法：（4分）

设d1=-(x2-x1),d2=x2-x1,d3=-(y2-y1),d4=y2-y1,q1=x1-XL,q2=XR-x1,q3=y1-YB,q4=YT-y1, t1=q1/d1,t2=q2/d2,t3=q3/d3,t4=q4/d4,算法步骤如下：

1. 如果XL<=x1,x2<=XR且YB<=y1,y2<=YT,则线段在区域内，直接输出线段P1P2，退出
2. 如果XL<=x1<=XR,YB<=y1<=YT,则输出左端点；如果XL<=x2<=XR,YB<=y2<=YT,则输出右端点。
3. 如果di=0, qi<0,其中i∈[1,4], 则说明该线段完全在边界之外，线段将舍去，退出；否则执行（2）；
4. 如果d1<0，rn1=t1,rn2=t2；反之,rn1=t2,rn2=t1;
5. 计算左端点:xa = x1 + d2 \* rn1,ya=y1+d4\*rn1;计算右端点：xb = x1 + d2 \* rn2，yb=y1+d4\*rn2;
6. 如果d3<0，rn1=t3,rn2=t4；反之,rn1=t4,rn2=t3;
7. 计算左端点:xc = x1 + d2 \* rn1,yc=y1+d4\*rn1;计算右端点：xd = x1 + d2 \* rn2，yd=y1+d4\*rn2;
8. 根据x1,xa,xb,xc,xd,x2将线段分割成若干段，判断每段线段是否在完全在区域内，即可得到区域内的线段。

(2)求解过程如下：（9分）

先计算d1=-14,d2=14,d3=4,d4=-4,q1=-2,q2=12,q3=2,q4=3,t1=,t2=,t3=,t4=-。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 第几步 | rn1 | rn2 | xa | ya | xb | yb | xc | yc | xd | yd |
| 5 |  |  | 0 |  | 10 |  | 0 | 0 | 0 | 0 |
| 7 | - |  | 0 |  | 10 |  |  | 5 | 5 | 0 |

排序结果：(-,5)（-2，2）（0，）（5,0）(10,-)

结合对应的矩形区域，得到区域内的线段的坐标为：（0，）（5,0）

五． 程序实现题（共10分）

1. 程序如下：（10分）

#include <iostream>

using namespace std;

#define NUM 1000

int main()

{

double k[4];

int p[4][2];

int cnt = 0;

double t = 0;

double t30,t20,t31,t21;

int pt[NUM][2] = {0,0};

cin >> p[0][0] >> p[0][1]>> p[1][0] >> p[1][1]>> p[2][0] >> p[2][1]>> p[3][0] >> p[3][1];

while (cnt<NUM) {

t20 = t \* t;

t30 = t \* t20;

t21 = (1 - t) \* (1 - t);

t31 = t21 \* (1 - t);

k[0] = t31;

k[1] = 3\*t\*t21;

k[2] = 3\*t20\*(1-t);

k[3] = t30;

for (int i = 0; i < 4; i++) {

pt[cnt][0] += k[i] \* p[i][0];

pt[cnt][1] += k[i] \* p[i][1];

}

t += 1.0 / NUM;

cnt++;

}

return 0;

}