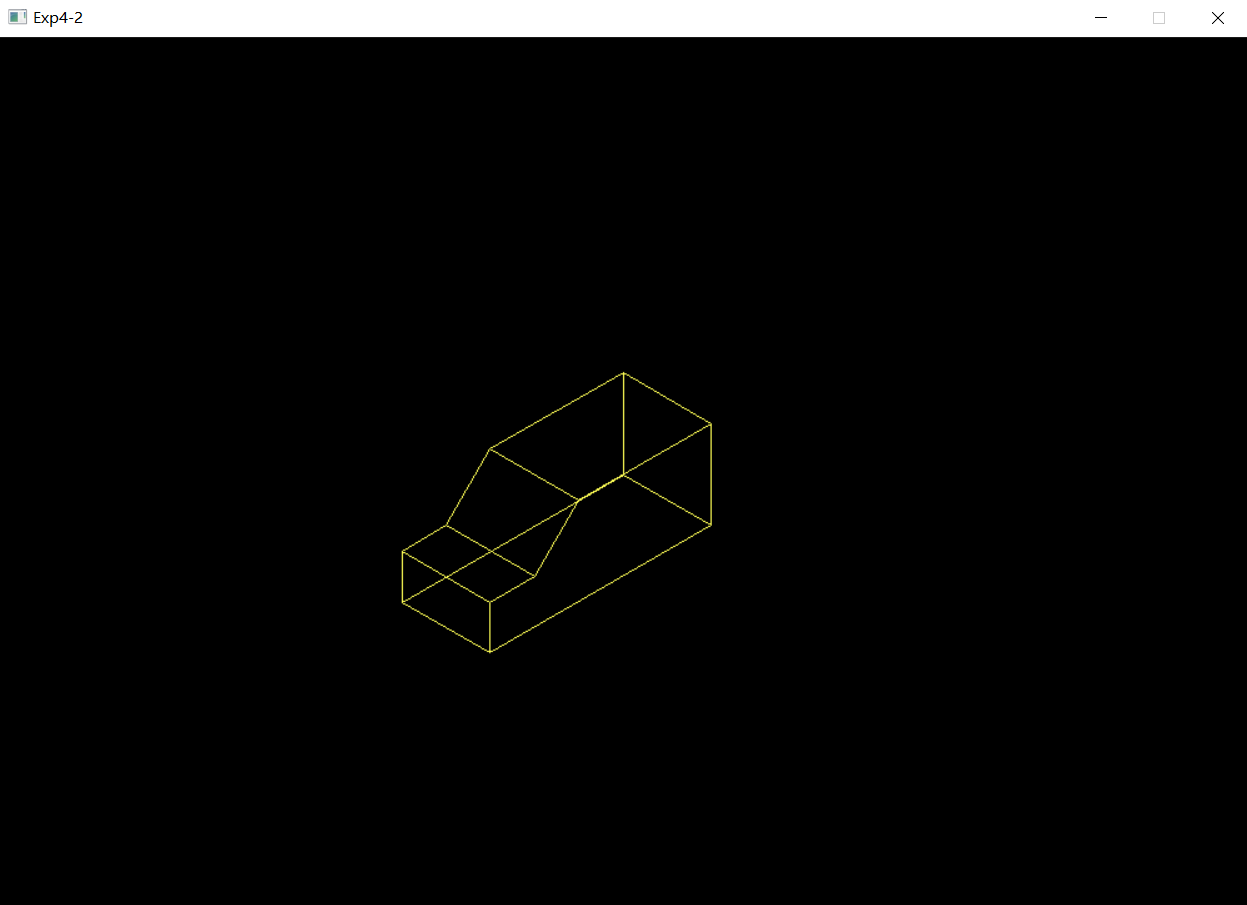
**模块4-2 三维图形投影和消隐**

**一 实验目的**

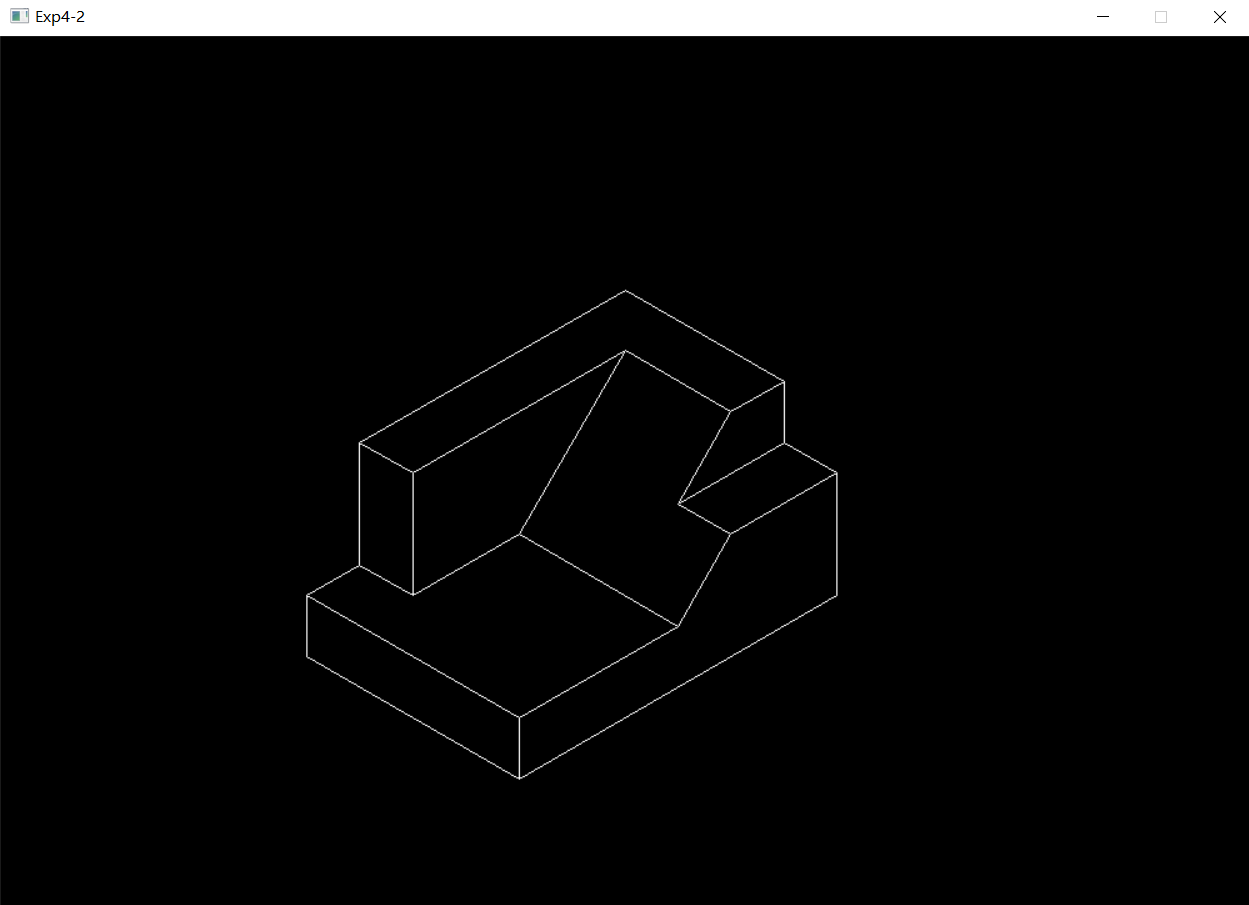
1. 编写三维图形各种变换的投影或消隐算法

**二 实验内容**

1：自行选择三维物体（不能选长方体），建立坐标系，给定点的三维坐标值，建立边表结构，完成正等轴测投影图。

实验结果如下图所示：

2：自行选择三维物体（不能选长方体），建立坐标系，给定点的三维坐标值，画出三维物体的消隐图。

实验结果如下图所示：

**三 程序说明**

最终的实验代码如下表所示：

|  |
| --- |
| 1题 |
| //////////////////////////////////////////////////////  // 程序名称：正等轴测投影图  // 功 能：实现预设三维物体的正等轴测投影图。  // 编译环境：VS2019，EasyX\_20220116  // 作 者：夏婉可<2020301010225><1597493790@qq.com>  // 最后修改：2022-4-14  #include <graphics.h>  #include <conio.h>  #include <iostream>  #include <math.h>  using namespace std;  //和分割线之间的距离  const int gap = 50;  //预设三维体的点坐标  int X[33] = { 0,0,0,0,0,0,0,0,100,100,0,100,100,100,100,100,100,100,100,0,0,0,0,100,100,100,100,0,0,0,0,100,100 };  int Y[33] = { 0,0,250,250,200,150,0,0,0,0,0,0,150,200,250,250,0,0,150,150,0,150,200,200,150,200,250,250,200,250,250,250,250 };  int Z[33] = { 100,0,0,50,50,100,100,0,0,100,100,100,100,50,50,0,0,100,100,100,100,100,50,50,100,50,50,50,50,50,0,0,50 };  //顶点总数  const int num = 33;  //正等轴测图绘制  void fun() {  POINT\* p = new POINT[num];  for (int i = 0; i < num; i++) {  p[i].x = 0.7071 \* X[i] - 0.7071 \* Y[i] + 500;  p[i].y = 0.4082 \* X[i] + 0.4082 \* Y[i] - 0.8165 \* Z[i] + 350;  }  for (int i = 0; i < num - 1; i++) {  setcolor(YELLOW);  line(p[i].x, p[i].y, p[i + 1].x, p[i + 1].y);  }  free(p);  }  int main() {  initgraph(1000, 700);  //调用函数  fun();  \_getch();  closegraph();  return 0;  } |
| 2题 |
| //////////////////////////////////////////////////////  // 程序名称：消隐图  // 功 能：实现预设三维物体的消隐图。  // 编译环境：VS2019，EasyX\_20220116  // 作 者：夏婉可<2020301010225><1597493790@qq.com>  // 最后修改：2022-4-14  #include <graphics.h>  #include <conio.h>  #include <iostream>  #include <stdio.h>  #include <stdlib.h>  using namespace std;  //顶点表  float point[20][4] = {  {0,0,150,1},//0  {250,0,150,1},//1  {250,0,50,1},//2  {300,0,50,1},//3  {300,0,0,1},//4  {300,200,0,1},//5  {0,200,0,1},//6  {0,200,100,1},//7  {0,150,100,1},//8  {0,150,150,1},//9  {50,150,150,1},//10  {50,50,150,1},//11  {250,50,150,1},//12  {250,50,50,1},//13  {150,50,50,1},//14  {150,200,50,1},//15  {100,200,100,1},//16  {100,150,100,1},//17  {300,200,50,1},//18  {0,0,0,1}//19  };  //环表  int ring[72] = {  //parallel XOY  0,1,12,11,10,9,0,//visible  8,17,16,7,8,//visible  2,3,18,15,14,13,2,//visible  4,19,6,5,4,//invisible  //parallel XOZ  1,2,3,4,19,0,1,//invisible  12,13,14,11,12,//visible  10,17,8,9,10,//visible  5,6,7,16,15,18,5,//visible  //parallel YOZ  1,2,13,12,1,//visible  0,9,8,7,6,19,0,//invisible  3,4,5,18,3,//visible  //slope  11,10,17,16,15,14,11 //visible  };  //面表  int surface[12][2] = {  //parallel XOY  {0,6},{7,11},{12,18},{19,23},  //parallel XOZ  {24,30},{31,35},{36,40},{41,47},  //parallel YOZ  {48,52},{53,59},{60,64},  //slope  {65,71}  };  //矩阵乘法，a\*b=c，a[m][n]，b[n][q]，c[m][q]  void MatrixMultiplication(float a[20][4], float b[4][4], float c[20][4]) {  int i = 0, j = 0, k = 0;  for (i = 0; i < 20; i++) {  for (j = 0; j < 4; j++) {  c[i][j] = 0;  for (k = 0; k < 4; k++) {  c[i][j] = c[i][j] + a[i][k] \* b[k][j];  }  }  }  }  //主函数  int main() {  //float point[20][4]  //int ring[72]  //int surface[12][2]  int i, j, start, end;  //正等轴测向量  float zdz[4][4] = { {0.707,0,-0.408,0},{-0.707,0,-0.408,0},{0,0,0.8165,0},{0,0,0,1} };  //备份顶点集  float point1[20][4];  double x1, x2, x3, z1, z2, z3;  //E值  double valueE[12];  //矩阵乘法=>正等轴测  MatrixMultiplication(point, zdz, point1);  initgraph(1000, 700);  //移动x坐标和y坐标  for (i = 0; i < 20; i++) {  point1[i][0] = 500 - point1[i][0] \* 1.2;//220  point1[i][2] = 350 - point1[i][2] \* 1.2;//140  }  //求E的值  for (i = 0; i < 12; i++) {  j = surface[i][0];  x1 = point1[ring[j]][0];  x2 = point1[ring[j + 1]][0];  x3 = point1[ring[j + 2]][0];  z1 = point1[ring[j]][2];  z2 = point1[ring[j + 1]][2];  z3 = point1[ring[j + 2]][2];  valueE[i] = (z2 - z1) \* (x3 - x2) - (x2 - x1) \* (z3 - z2);  }  for (i = 0; i < 12; i++) {  if (valueE[i] >= 0) {  start = surface[i][0];  end = surface[i][1];  for (j = start; j < end; j++) {  line(point1[ring[j]][0], point1[ring[j]][2], point1[ring[j + 1]][0], point1[ring[j + 1]][2]);  }  }  }  \_getch();  closegraph();  return 0;  } |