**模块5 曲线和曲面**

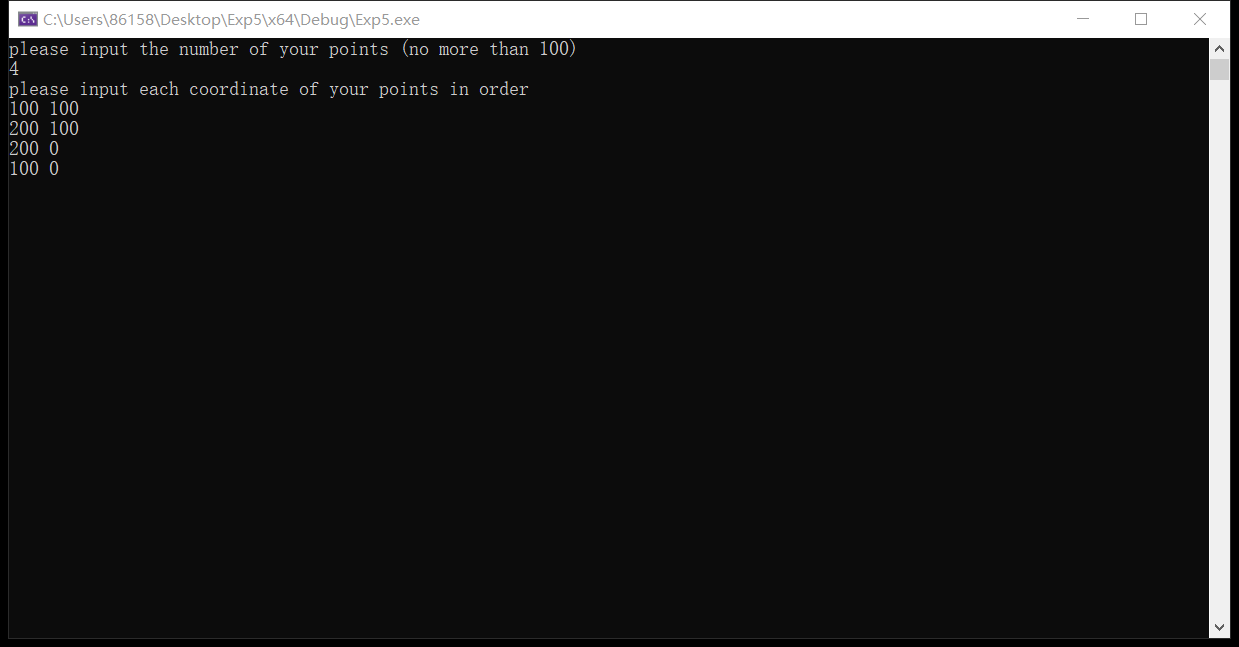
**一 实验目的**

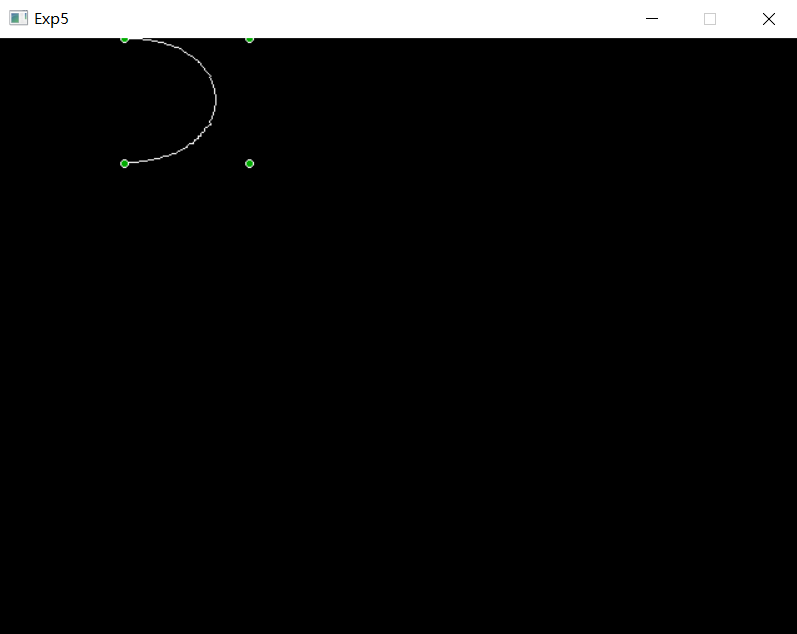
1. 编写曲线和曲面的算法

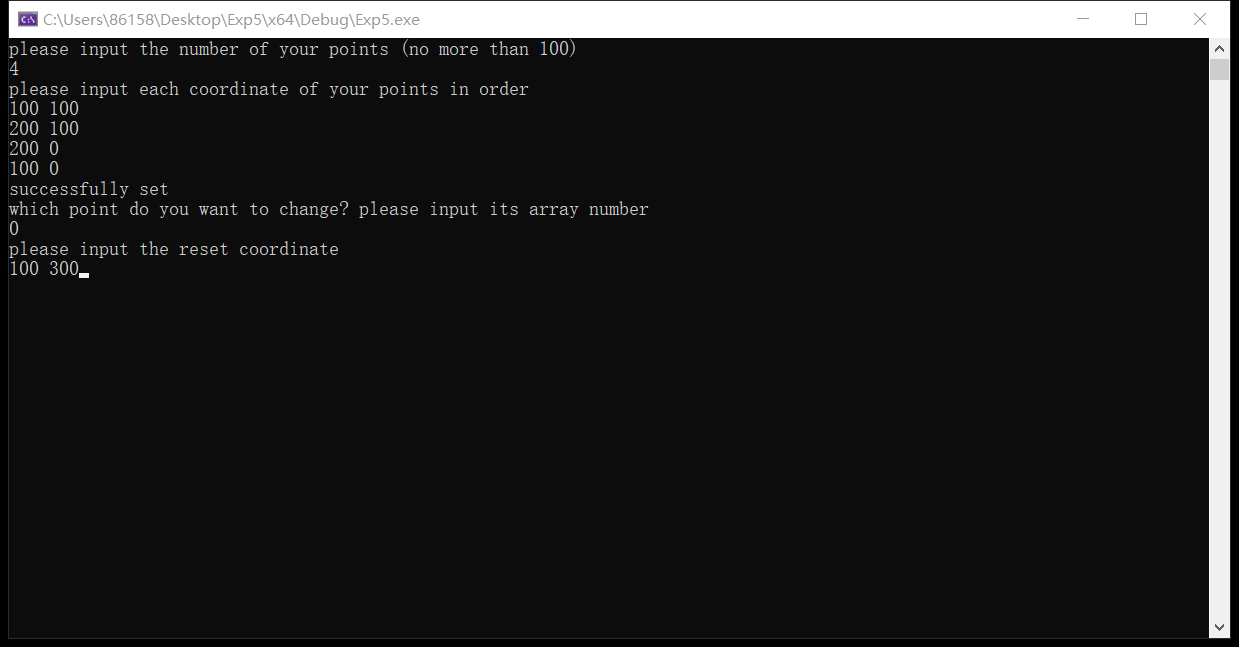
**二 实验内容**

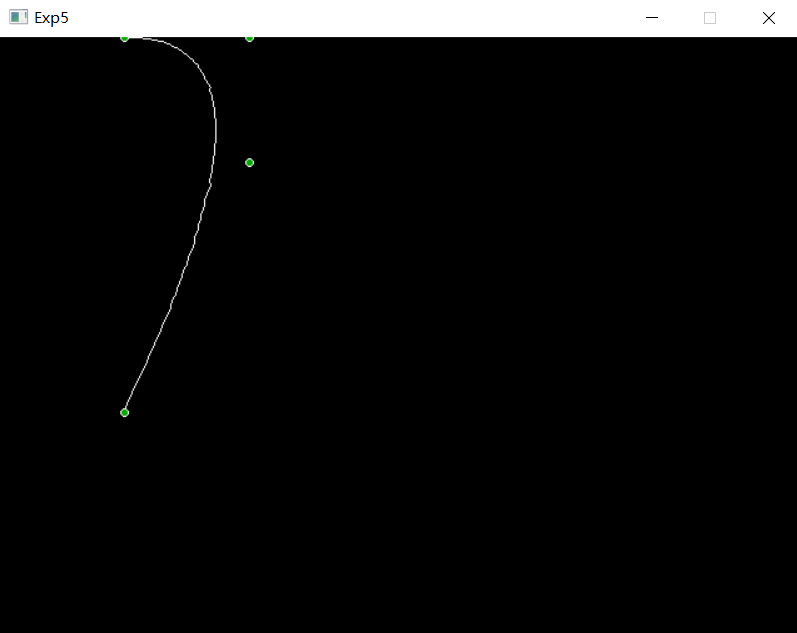
1：绘制Bezier曲线，并采用自行设计输入和交互修改数据点的方式。

实验结果如下图所示：

第一步：输入特征多边形的顶点个数，并按照顺序输入顶点的坐标。

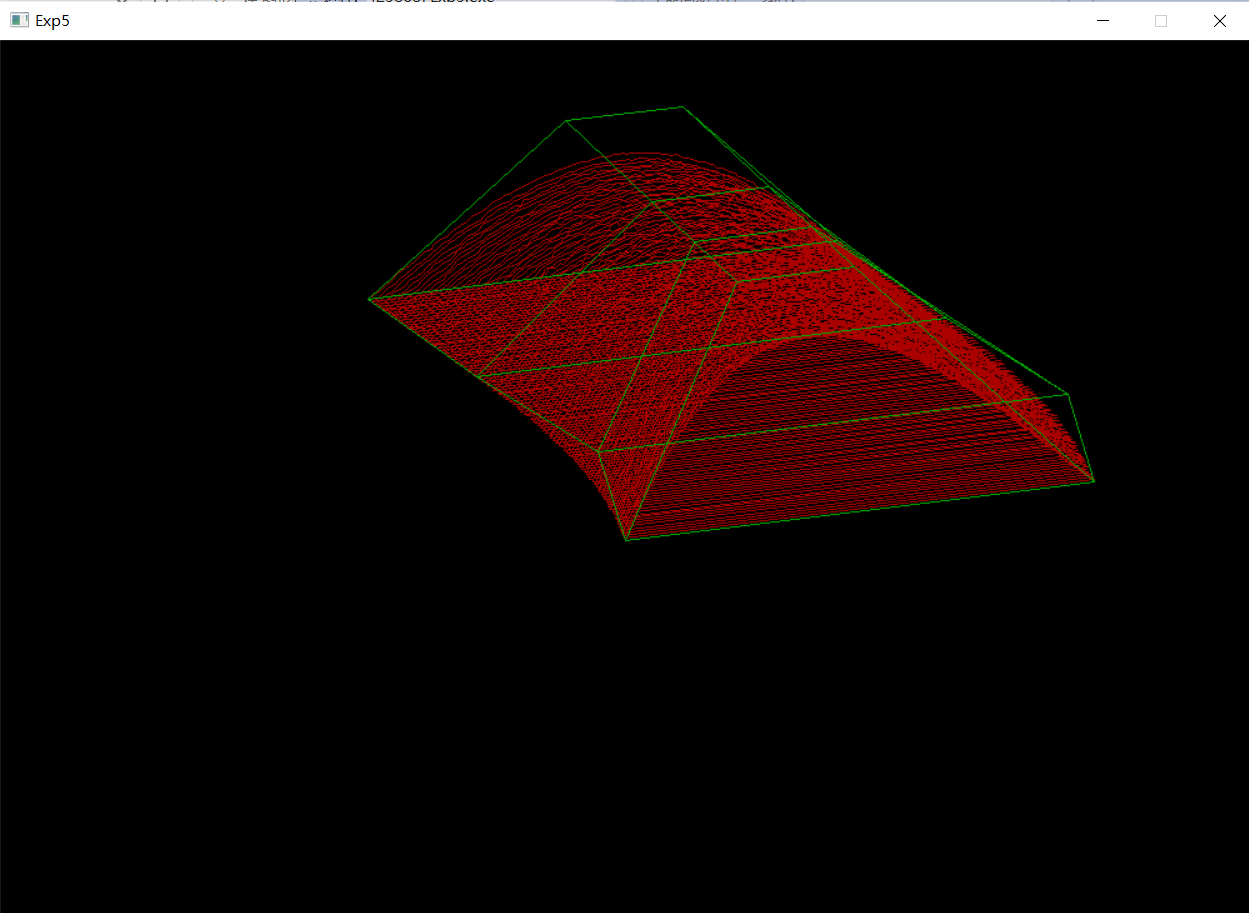
第二步：点击左键生成bezier曲线（白色部分）和多边形顶点（绿色部分）。

第三步：点击右键修改顶点，首先输入需要修改的顶点在数组中的坐标（即顺序个数-1），此处以（100，100）为例，将其修改为（100，300）。

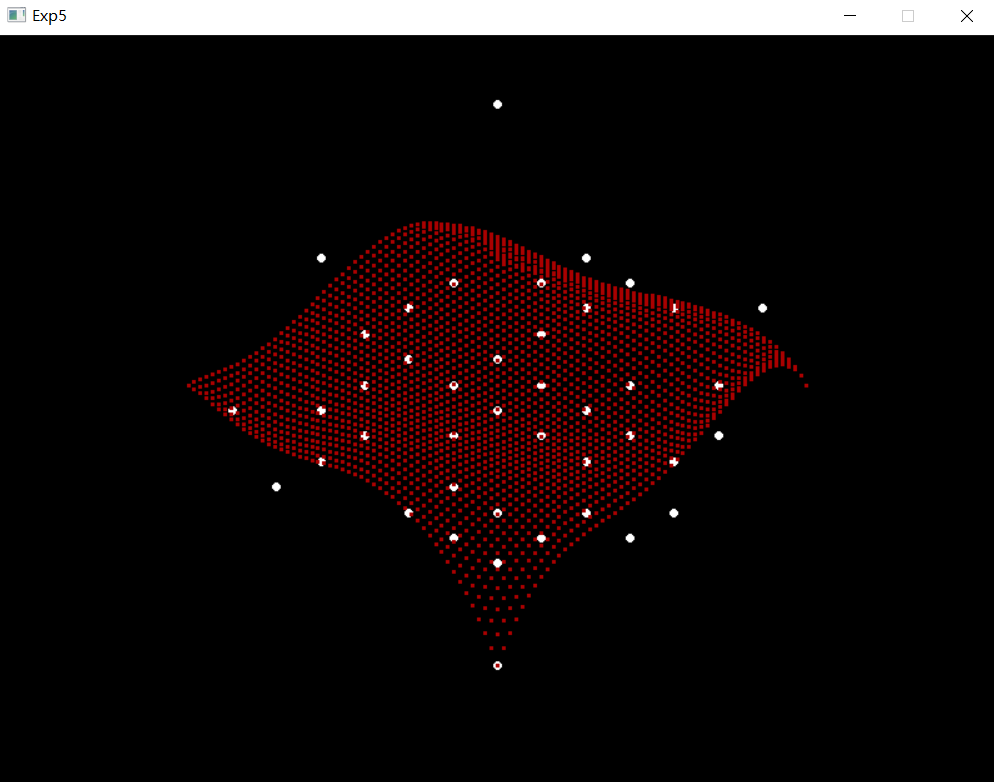
第四步：通过敲击回车键，可观察修改后的bezier曲线和顶点。

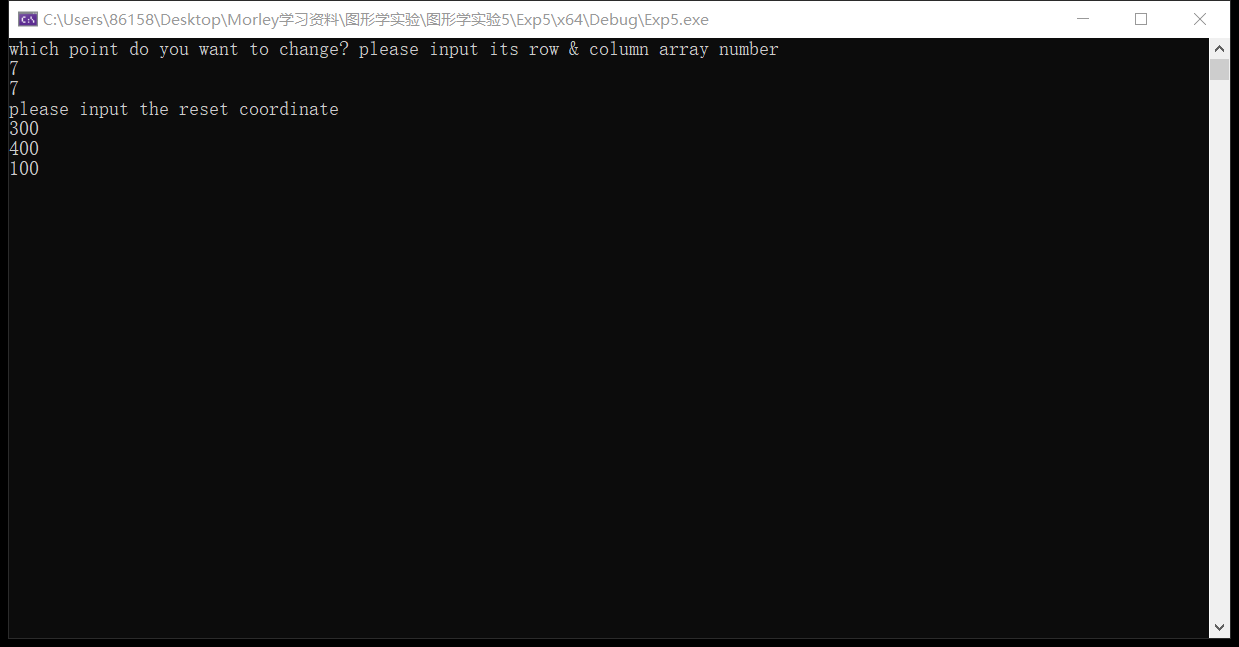
第五步：如果需要继续修改顶点坐标，用户可以继续点击右键，重复上述操作。此处不再赘述。如果需要退出图像绘制界面，用户可以通过敲击esc键，即可退出本实验界面。

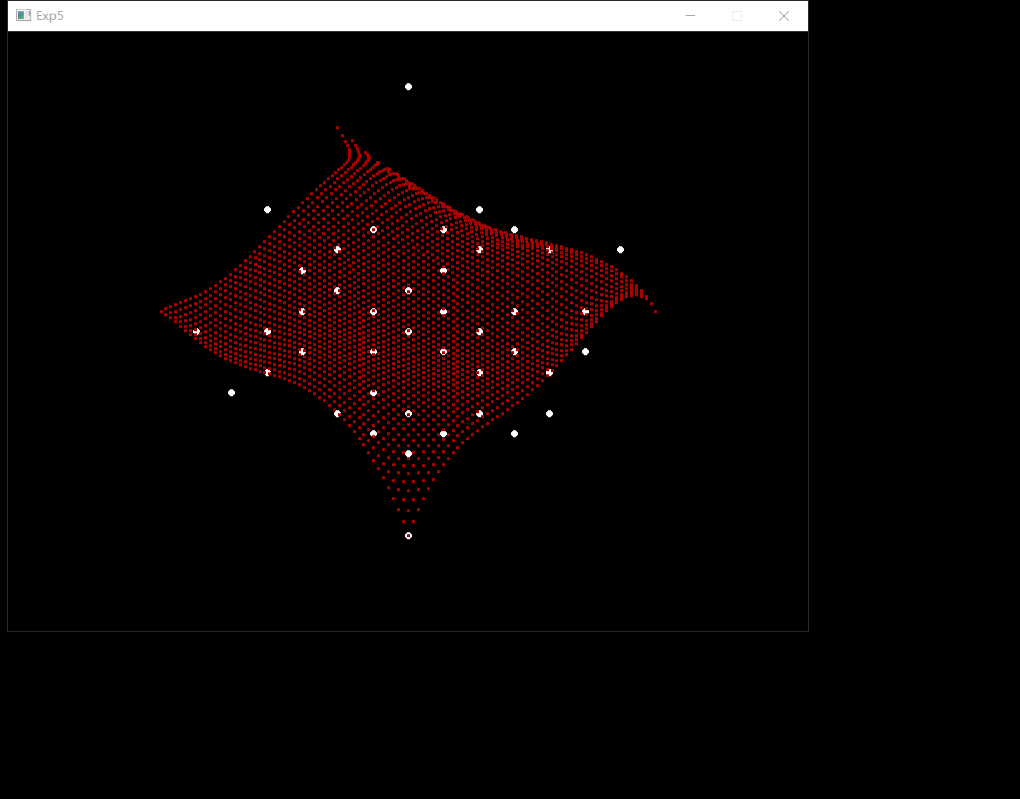
2：绘制Bezier曲面，双三次类型。

实验结果如下图所示：

2.1：绘制Bezier曲面，并采用自行设计输入和交互修改数据点的方式，改进。

实验结果如下图所示：

修改数据点：

新贝塞尔曲面：

**三 程序说明**

最终的实验代码如下表所示：

|  |
| --- |
| 1题 |
| //////////////////////////////////////////////////////  // 程序名称：Bezier曲线  // 功 能：绘制Bezier曲线，并采用自行设计输入和交互修改数据点的方式。  // 编译环境：VS2019，EasyX\_20220116  // 作 者：夏婉可<2020301010225><1597493790@qq.com>  // 最后修改：2022-4-28  #include <graphics.h>  #include <conio.h>  #include <iostream>  #include <math.h>  using namespace std;  //特征多边形的特征  int n;  int px[100], py[100];  //初始化特征多边形  void initialize() {  cout << "please input the number of your points (no more than 100)" << endl;  cin >> n;  cout << "please input each coordinate of your points in order" << endl;  for (int i = 0; i < n; i++) {  cin >> px[i] >> py[i];  }  cout << "successfully set" << endl;  }  //递归  int Factorial(int nn) {  if (nn == 0 || nn == 1) return 1;  else return nn \* Factorial(nn - 1);  }  //绘制bezier曲线  void Bezier() {  int nn = n - 1;  //line(px[0], py[0], px[1], py[1]);  double i = 0, n1, n2;  n1 = Factorial(nn);  moveto(px[0], py[0]);  POINT point;  //在[0,1]内循环  for (; i <= 1; i += 0.01) {  point.x = 0;  point.y = 0;  for (int j = 0; j <= nn; j++) {  n2 = (n1 / (Factorial(j) \* Factorial(nn - j))) \* pow(i, j) \* pow(1 - i, nn - j);  point.x += n2 \* px[j];  point.y += n2 \* py[j];  }  lineto(point.x, point.y);  }  lineto(px[n - 1], py[n - 1]);  //描绘用户定义的点  for (int ii = 0; ii < n; ii++) {  setfillcolor(GREEN);  fillcircle(px[ii], py[ii], 3);  }  }  //主函数  int main() {  //初始化  initialize();  initgraph(640, 480);  ExMessage m;  //绘制图像  while (1) {  m = getmessage(EX\_MOUSE | EX\_KEY);  switch (m.message) {  //右键修改坐标信息  case WM\_RBUTTONDOWN:  closegraph();  int arr, newx, newy;  //输入数组中的坐标，进行修改  cout << "which point do you want to change? please input its array number" << endl;  while (1) {  cin >> arr;  if (arr >= n) {  cout << "input error, try again" << endl;  }  else {  break;  }  }  cout << "please input the reset coordinate" << endl;  cin >> newx >> newy;  px[arr] = newx;  py[arr] = newy;  cout << "successfully reset, please wait a second to see a new graph" << endl;  initgraph(640, 480);  //左键绘制  case WM\_LBUTTONDOWN:  Bezier();  //退出图形化界面  case WM\_KEYDOWN:  if (m.vkcode == VK\_ESCAPE) {  return 0;  }  }  }  \_getch();  closegraph();  return 0;  } |
| 2题 |
| //////////////////////////////////////////////////////  // 程序名称：Bezier曲面  // 功 能：绘制Bezier曲面，双三次类型。  // 编译环境：VS2019，EasyX\_20220116  // 作 者：夏婉可<2020301010225><1597493790@qq.com>  // 最后修改：2022-5-9  #include <graphics.h>  #include <conio.h>  #include <iostream>  #include <cmath>  using namespace std;  #define pi 3.1415926  #define MAX 20  //三维点类  class Point3D {  public:  double x, y, z;  Point3D(int x = 0, int y = 0, int z = 0) {  this->x = x;  this->y = y;  this->z = z;  };  void operator=(Point3D& a) {  x = a.x;  y = a.y;  z = a.z;  };  Point3D operator\*(double a) {  return Point3D(a \* x, a \* y, a \* z);  };  Point3D operator+(Point3D a) {  return Point3D(x + a.x, y + a.y, z + a.z);  };  void operator+=(Point3D a) {  x += a.x;  y += a.y;  z += a.z;  };  };  //求阶乘  long int Factorial(int n) {  int i, sum = 1;  if (n == 0) {  return 1;  }  for (i = 2; i <= n; i++) {  sum \*= i;  }  return sum;  }  //Bernstein函数  double Bernstein(int i, int n, double t) {  return (double)Factorial(n) / Factorial(i) / Factorial(n - i) \* pow(t, i) \* pow(1 - t, n - i);  }  //Bezier曲面求解  void BezierCurve(Point3D p[][MAX], int n, int m, int nd, int md, double sita, double fai) {  double hu = 1.0 / nd;  double hv = 1.0 / md;  double u = 0, v = 0;  int i, j, k, l;  sita = sita \* pi / 180;  fai = fai \* pi / 180;  int\*\* pB = new int\* [nd + 1];  for (i = 0; i <= nd; i++) {  pB[i] = new int[2 \* md + 2];  }  Point3D ptemp(0, 0, 0);  for (i = 0; i <= nd; i++, u += hu) {  v = 0;  for (j = 0; j <= 2 \* md + 1; j += 2, v += hv) {  ptemp.x = 0;  ptemp.y = 0;  ptemp.z = 0;  for (k = 0; k <= n; k++) {  for (l = 0; l <= m; l++) {  ptemp += p[k][l] \* Bernstein(k, n, u) \* Bernstein(l, m, v);  }  }  pB[i][j] = ptemp.x \* cos(sita) - ptemp.y \* sin(sita) + 500;  pB[i][j + 1] = -ptemp.x \* sin(sita) \* sin(fai) - ptemp.y \* cos(sita) \* sin(fai) + ptemp.z \* cos(fai) + 400;  }  drawpoly(md + 1, pB[i]);  }  for (i = 0; i < nd + 1; i++) {  delete[]pB[i];  }  delete[]pB;  }  //画控制多边形  void drawControlPoly\_3D(Point3D p[][MAX], int n, int m, double sita, double fai, int color) {  int\*\* p2d = new int\* [n + 1];  int i, j;  setcolor(color);  sita = sita \* pi / 180;  fai = fai \* pi / 180;  for (i = 0; i < n + 1; i++) {  p2d[i] = new int[2 \* m + 4];  }  char str[80];  //3D->2D  for (i = 0; i <= n; i++) {  for (j = 0; j <= m; j++) {  p2d[i][2 \* j] = p[i][j].x \* cos(sita) - p[i][j].y \* sin(sita) + 500;  p2d[i][2 \* j + 1] = -p[i][j].x \* sin(sita) \* sin(fai) - p[i][j].y \* cos(sita) \* sin(fai) + p[i][j].z \* cos(fai) + 400;  }  p2d[i][2 \* j] = p2d[i][0];  p2d[i][2 \* j + 1] = p2d[i][1];  drawpoly(m + 2, p2d[i]);  }  for (j = 0; j <= 2 \* m + 1; j += 2) {  moveto(p2d[0][j], p2d[0][j + 1]);  for (i = 1; i <= n; i++) {  lineto(p2d[i][j], p2d[i][j + 1]);  }  }  for (i = 0; i < n + 1; i++) {  delete[]p2d[i];  }  delete[]p2d;  }  //双几次曲面  void BezierCurve\_Shuang(Point3D p[][MAX], int pn, int pm, int nd, int md, double sita, double fai, int shuangjici) {  int i, j, k, l;  Point3D point[MAX][MAX];  for (i = 0; i < pn - shuangjici; i += shuangjici) {  for (j = 0; j < pm - shuangjici; j += shuangjici) {  for (k = 0; k <= shuangjici; k++) {  for (l = 0; l <= shuangjici; l++) {  point[k][l] = p[i + k][j + l];  }  }  BezierCurve(point, shuangjici, shuangjici, nd, md, sita, fai);  }  }  }  //主函数  int main() {  int n;  double fai, sita;  //固定角度了。  sita = 20;  fai = 160;  n = 3;  //顶点集合  Point3D p[][MAX] = {  {Point3D(0,0,0),Point3D(150,150,150),Point3D(250,150,150),Point3D(400,0,0)},  {Point3D(50,200,0),Point3D(150,250,150),Point3D(250,250,150),Point3D(450,200,0)},  {Point3D(20,400,0),Point3D(150,350,150),Point3D(250,350,150),Point3D(420,400,0)},  {Point3D(0,600,0),Point3D(150,550,150),Point3D(250,550,150),Point3D(400,600,0)}  };  /\*  cout << "please input sita and fai" << endl;  cin >> sita >> fai;  cout << "please input your n" << endl;  cin >> n;  \*/  //初始化界面  initgraph(1000, 700);  setcolor(RED);  //Bezier曲面  BezierCurve\_Shuang(p, 4, 4, 100, 100, sita, fai, n);  //画控制多边形  drawControlPoly\_3D(p, 3, 3, sita, fai, GREEN);  \_getch();  closegraph();  return 0;  } |
| 2题改进版本 |
| //////////////////////////////////////////////////////  // 程序名称：Bezier曲面  // 功 能：绘制Bezier曲面，并采用自行设计输入和交互修改数据点的方式。  // 编译环境：VS2019，EasyX\_20220116  // 作 者：夏婉可<2020301010225><1597493790@qq.com>  // 最后修改：2022-5-13  #include <iostream>  #include <graphics.h>  #include <conio.h>  #include <math.h>  using namespace std;  #define PI 3.1415926  //定义 N \* N 的控制点数组  const int N = 8;  float controlPoints[N][N][3] ={  { {-150, -150, 100}, {-100, -150, 0}, {-50, -150, 0}, {0, -150, 50}, {50, -150, 0}, {100, -150, 0}, {150, -150, -100}, {200, -150, 0} },  { {-150, -100, 0}, {-100, -100, 50}, {-50, -100, -100}, {0, -100, 0}, {50, -100, 0}, {100, -100, 100}, {150, -100, 0}, {200, -100, -50} },  { {-150, -50, 0}, {-100, -50, 0}, {-50, -50, 50}, {0, -50, 0}, {50, -50, 0}, {100, -50, 0}, {150, -50, -50}, {200, -50, 0} },  { {-150, 0, -50}, {-100, 0, -100}, {-50, 0, 0}, {0, 0, 150}, {50, 0, 0}, {100, 0, -100}, {150, 0, -50}, {200, 0, -50} },  { {-150, 50, 0}, {-100, 50, 0}, {-50, 50, -100}, {0, 50, 0}, {50, 50, 50}, {100, 50, 0}, {150, 50, 0}, {200, 50, 50} },  { {-150, 100, 50}, {-100, 100, 0}, {-50, 100, 0}, {0, 100, -50}, {50, 100, -50}, {100, 100, 50}, {150, 100, 0}, {200, 100, 0} },  { {-150, 150, 0}, {-100, 150, 100}, {-50, 150, -100}, {0, 150, 0}, {50, 150, 0}, {100, 150, 0}, {150, 150, -150}, {200, 150, 50} },  { {-150, 200, 0}, {-100, 200, 0}, {-50, 200, 50}, {0, 200, -50}, {50, 200, 0}, {100, 200, -100}, {150, 200, 0}, {200, 200, 50} }  };  //计算贝塞尔基函数的C值  int C(int n, int k) {  if (k > n) {  return 0;  }  int c = 1;  for (int i = 0; i < k; i++) {  c \*= (n - i);  c /= (i + 1);  }  return c;  }  //勾勒函数  void display(float x, float y, float z) {  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400), int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300), RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400) + 1, int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300), RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400) - 1, int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300), RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400), int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300) + 1, RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400), int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300) - 1, RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400) + 1, int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300) + 1, RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400) + 1, int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300) - 1, RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400) - 1, int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300) + 1, RED);  putpixel(int(0.7071 \* x - 0.7071 \* y + 0.5f + 400) - 1, int(-0.4082 \* x - 0.4082 \* y + 0.8165 \* z + 0.5f + 300) - 1, RED);  }  //Bezier曲面绘制  void Bezier() {  //绘制控制点  for (int i = 0; i < N - 1; i++) {  for (int j = 0; j < N - 1; j++) {  fillcircle(int(0.7071 \* controlPoints[i][j][0] - 0.7071 \* controlPoints[i][j][1] + 0.5f + 400), int(-0.4082 \* controlPoints[i][j][0] - 0.4082 \* controlPoints[i][j][1] + 0.8165 \* controlPoints[i][j][2] + 0.5f + 300), 3);  }  }  // 绘制贝塞尔曲面  const int nPoints = 50; // 每行/列计算的点数  float uStep = 1.0f / nPoints;  float vStep = 1.0f / nPoints;  for (float u = 0; u < 1; u += uStep) {  for (float v = 0; v < 1; v += vStep) {  float x = 0, y = 0, z = 0;  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  // 计算贝塞尔基函数的值  float basisU = powf(1 - u, N - 1 - i) \* powf(u, i) \* float(C(N - 1, i));  float basisV = powf(1 - v, N - 1 - j) \* powf(v, j) \* float(C(N - 1, j));  float basis = basisU \* basisV;  // 使用贝塞尔基函数的值和控制点计算曲面上的点  x += basis \* controlPoints[i][j][0];  y += basis \* controlPoints[i][j][1];  z += basis \* controlPoints[i][j][2];  }  }  // 将曲面上的点连接起来  display(x, y, z);  }  }  }  //主函数  int main(){  initgraph(800, 600);  Bezier();  ExMessage m;  while (1) {  m = getmessage(EX\_MOUSE | EX\_KEY);  switch (m.message) {  case WM\_RBUTTONDOWN:  //右键修改坐标信息  closegraph();  int row, column, newx, newy, newz;  //输入数组中的坐标，进行修改  cout << "which point do you want to change? please input its row & column array number" << endl;  while (1) {  cin >> row;  cin >> column;  if (row >= N) {  cout << "row number input error, try again" << endl;  }  else if (column >= N) {  cout << "column number input error, try again" << endl;  }  else {  break;  }  }  cout << "please input the reset coordinate" << endl;  cin >> newx >> newy >> newz;  controlPoints[row][column][0] = newx;  controlPoints[row][column][1] = newy;  controlPoints[row][column][2] = newz;  cout << "successfully reset, please wait a second to see a new graph" << endl;  initgraph(800, 600);  case WM\_LBUTTONDOWN:  //左键绘制  Bezier();  case WM\_KEYDOWN:  //退出图形化界面  if (m.vkcode == VK\_ESCAPE) {  return 0;  }  }  }    \_getch();  closegraph();  return 0;  } |