|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **绘图**  **10** | **图形变换、15** | **扫描转换15** | **消隐15** | **曲线曲面15** | **几何投影15** | **光照15** | **总分100** |
|  |  |  |  |  |  |  |  |

**江苏科技大学**

**计算机图形学研究报告**

**（2022/2023学年第2学期）**

**学生姓名： 陈余**

**学生学号： 212241807213**

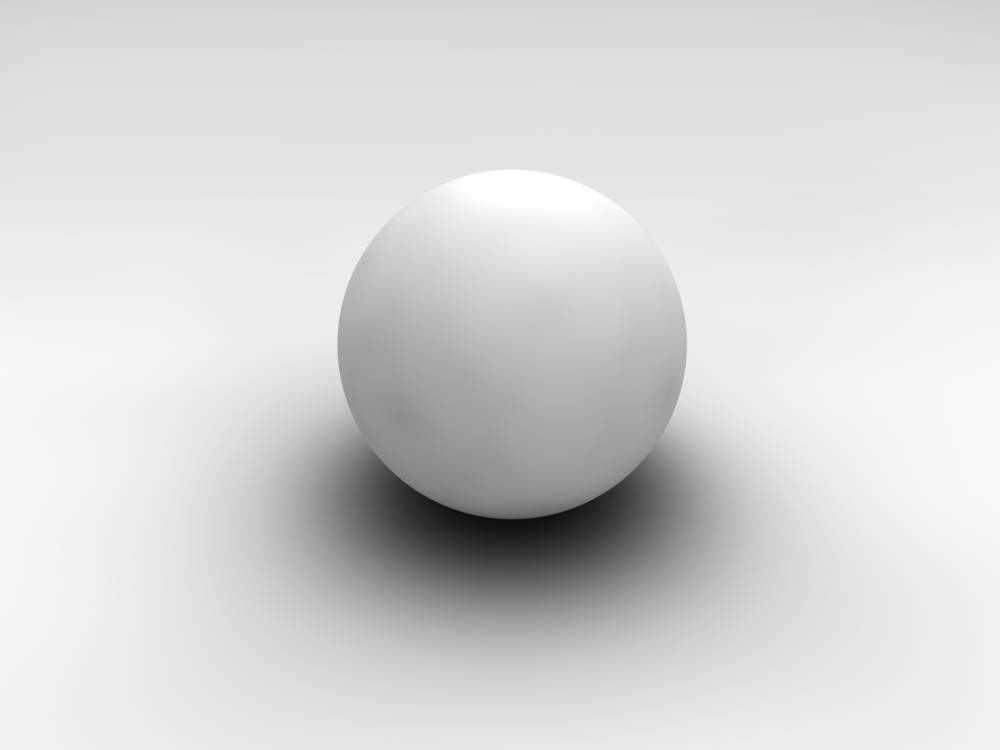
**院 系： 计算机学院**

**专 业： 计算机科学与技术专业**

2024 **年 6 月 30 日**

1. **题目要求：**

综合利用基本图元的扫描转换、填充算法、二维与三维变换、三维投影算法、自由曲线曲面算法、几何体消隐算法与光照模型，绘制下列图形。物体颜色和形状以及平面颜色不做限定



1. **函数/类模块功能清单与调用关系**

所开发完成软件由下列模块组成。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 函数头 | 功能和参数说明 | 调用函数 |
| 1 | void ReadPoint(); | 读取球上的点 |  |
| 2 | void ReadFace(); | 读取点构成的所有小面 | ReadPoint |
| 3 | void divid(); | 按每一个面进行处理 | ReadFace，ligth |
| 4 | Void ligth(double p[3][3]); | 调用函数处理，并进行消隐最后进行填充 | Environment，diffuse，mirror，Project，fillup，countFX |
| 5 | void environment(); | 环境光 |  |
| 6 | void diffuse(double v[3], int index); | 漫反射 | countFX |
| 7 | void mirror(double v[3], int index); | 镜面反射 | countFX |
| 8 | void Project(double p[3]); | 透视投影 | Init |
| 9 | void fillup(); | 双线性插值填充 |  |
| 10 | void countFX(double v[3][3], int index); | 计算光强 |  |
| 11 | void Init(); | 计算透视投影中的参数 |  |
| 12 | afx\_msg void OnBnClickedButton1(); | 按钮加初始化部分数据 | Init，ReadPoint，ReadFace，divid |

1. **源代码**

**.h文件**

// 实现

protected:

HICON m\_hIcon;

// 生成的消息映射函数

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

double P[37][72][3];//存储初始的点

double F[36][72][4][3];//存储每个面的四个顶点

double P1[4][3], Pdown[4][3], Pright[4][3], Pdownright[4][3];//表示循环使用的四个方格的四个点

double inx[3];//表示后面index顺序

double N1[3], Nd[3], Nr[3], Ndr[3];//四个面的法向量

double Lv[3];//光矢量

double v1[3];//视点坐标

double Uv2[3];//未单位化的视矢量

double Uv[3];//单位视矢量

double I1[3] = { 1.0,1.0,1.0 };//环境光强

double I2[3] = { 1.0,1.0,1.0 };//漫反射光强

double I3[3] = { 1.0,1.0,1.0 };//镜面反射光强

double k1[3] = { 0.012,0.012,0.175 };//环境光反射率

double k2[3] = { 0.041,0.041,0.614 };//漫反射率

double k3[3] = { 0.527,0.527,0.728 };//镜面反射率

double k4[9];//透视投影变换系数

double R = 300, Fei = 90, Thta = 90, D = 800;//视点参数

double n = 5;//高光指数

double aa[3] = { 0,0,500 };//点光源坐标(右上)

//衰减因子（（1，0，0）则为不衰减）

double cc[3] = { 0.65,0.00002,0.000001 };

double L[3][3];//存放三个点的光强

double L1[3];//某个点的光强

double ScreenP[2];//表示屏幕坐标系

double Point[3][2];//存放透视投影的点

public:

void ReadPoint();//去取点

void ReadFace();//读入面坐标

void divid();//逐面计算

void ligth(double p[3][3], bool First);//计算光强

void environment();//环境光

void diffuse(double v[3], int index);//漫反射

void mirror(double v[3], int index);//镜面反射

void Project(double p[3]);//透视投影

void fillup();//双线性插值

void countFX(double v[3][3], int index);//计算法向量

void Init();//透视变换所需要的系数

afx\_msg void OnBnClickedButton1();

};

**.cpp文件**

HCURSOR CtttttDlg::OnQueryDragIcon()

{

return static\_cast<HCURSOR>(m\_hIcon);

}

void CtttttDlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // 用于绘制的设备上下文

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// 使图标在工作区矩形中居中

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// 绘制图标

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CPaintDC dc(this); // 用于绘制的设备上下文

CRect rect;

GetClientRect(rect);

dc.FillSolidRect(rect, RGB(255, 255, 255)); //设置为浅蓝色背景

CDialogEx::OnPaint();

}

}

//当用户拖动最小化窗口时系统调用此函数取得光标

//显示。

//获取点

void CtttttDlg::ReadPoint()

{

double afa, beta, PI = 3.1414926;

double r = 100;

for (int i = 0; i < 37; i++)

{

afa = i \* 5 \* PI / 180;

for (int j = 0; j < 72; j++)

{

beta = j \* 5 \* PI / 180;

P[i][j][0] = r \* sin(afa) \* cos(beta);

P[i][j][1] = r \* sin(afa) \* sin(beta);

P[i][j][2] = r \* cos(afa);

}

}

}

//读入面坐标

void CtttttDlg::ReadFace()

{

for (int i = 0; i < 36; i++)

{

for (int j = 0; j < 72; j++)

{

for (int m = 0; m < 3; m++) {

F[i][j][0][m] = P[i][j][m];

F[i][j][1][m] = P[i + 1][j][m];

}

if (j == 71)

{

for (int m = 0; m < 3; m++) {

F[i][j][2][m] = P[i + 1][0][m];

F[i][j][3][m] = P[i][0][m];

}

}

else

{

for (int m = 0; m < 3; m++) {

F[i][j][2][m] = P[i + 1][j + 1][m];

F[i][j][3][m] = P[i][j + 1][m];

}

}

}

}

}

//对于每一面依次化成三角形计算

void CtttttDlg::divid()

{

for (int i = 0; i < 35; i++)

{

for (int j = 0; j < 72; j++)

{

for (int m = 0; m < 4; m++)

{

P1[m][0] = F[i][j][m][0];

P1[m][1] = F[i][j][m][1];

P1[m][2] = F[i][j][m][2];

Pdown[m][0] = F[i + 1][j][m][0];

Pdown[m][1] = F[i + 1][j][m][1];

Pdown[m][2] = F[i + 1][j][m][2];

if (j == 71)

{

Pright[m][0] = F[i][0][m][0];

Pright[m][1] = F[i][0][m][1];

Pright[m][2] = F[i][0][m][2];

Pdownright[m][0] = F[i + 1][0][m][0];

Pdownright[m][1] = F[i + 1][0][m][1];

Pdownright[m][2] = F[i + 1][0][m][2];

}

else

{

Pright[m][0] = F[i][j + 1][m][0];

Pright[m][1] = F[i][j + 1][m][1];

Pright[m][2] = F[i][j + 1][m][2];

Pdownright[m][0] = F[i + 1][j + 1][m][0];

Pdownright[m][1] = F[i + 1][j + 1][m][1];

Pdownright[m][2] = F[i + 1][j + 1][m][2];

}

}

bool First = true;

double Pt[3][3];//四边形划分为两个三角形面片

for (int t = 0; t < 3; t++) {

Pt[0][t] = P1[0][t]; Pt[1][t] = P1[1][t]; Pt[2][t] = P1[3][t];

}

inx[0] = 1, inx[1] = 2, inx[2] = 4;

ligth(Pt, First);//绘制上三角形面片

First = false;

for (int t = 0; t < 3; t++) {

Pt[0][t] = P1[1][t]; Pt[1][t] = P1[2][t]; Pt[2][t] = P1[3][t];

}

inx[0] = 2, inx[1] = 3, inx[2] = 4;

ligth(Pt, First);//绘制下三角形面片

First = true;

}

}

}

//计算法向量

void CtttttDlg::countFX(double v[3][3], int index)

{

int x[3], y[3];

double mm[3];

for (int k = 0; k < 3; k++) {

x[k] = v[0][k] - v[1][k];

y[k] = v[0][k] - v[2][k];

}

//求出法向量

mm[0] = x[1] \* y[2] - x[2] \* y[1];

mm[1] = x[2] \* y[0] - x[0] \* y[2];

mm[2] = x[0] \* y[1] - x[1] \* y[0];

//单位化

double m = sqrt(mm[0] \* mm[0] + mm[1] \* mm[1] + mm[2] \* mm[2]);

for (int k = 0; k < 3; k++) {

mm[k] = (double)mm[k] / m;

}

for (int i = 0; i < 3; i++) {

if (index == 1)

N1[i] = mm[i];

else if (index == 2)

Nd[i] = mm[i];

else if (index == 3)

Ndr[i] = mm[i];

else

Nr[i] = mm[i];

}

}

//计算光强

void CtttttDlg::ligth(double p[3][3], bool First)

{

//计算四个点各自的法向量

double v[3][3];

for (int i = 0; i < 3; i++) {

v[i][0] = P1[i][0];

v[i][1] = P1[i][1];

v[i][2] = P1[i][2];

}

countFX(v, 1);

for (int i = 0; i < 3; i++) {

v[i][0] = Pdown[i][0];

v[i][1] = Pdown[i][1];

v[i][2] = Pdown[i][2];

}

countFX(v, 2);

for (int i = 0; i < 3; i++) {

v[i][0] = Pright[i][0];

v[i][1] = Pright[i][1];

v[i][2] = Pright[i][2];

}

countFX(v, 4);

for (int i = 0; i < 3; i++) {

v[i][0] = Pdownright[i][0];

v[i][1] = Pdownright[i][1];

v[i][2] = Pdownright[i][2];

}

countFX(v, 3);

//单位化光矢量

double m = sqrt(aa[0] \* aa[0] + aa[1] \* aa[1] + aa[2] \* aa[2]);

for (int i = 0; i < 3; i++) {

Lv[i] = (double)aa[i] / m;

}

//求出视矢量并单位化

for (int i = 0; i < 3; i++) {

Uv2[i] = v1[i] - p[0][i];

}

m = sqrt(Uv2[0] \* Uv2[0] + Uv2[1] \* Uv2[1] + Uv2[2] \* Uv2[2]);

for (int i = 0; i < 3; i++) {

Uv[i] = (double)Uv2[i] / m;

}

//求光强

double vv[3];

for (int i = 0; i < 3; i++) {

//加入环境光

environment();

//加入漫反射

vv[0] = p[i][0];

vv[1] = p[i][1];

vv[1] = p[i][2];

diffuse(vv, inx[i]);

//加入镜面反射

mirror(vv, inx[i]);

for (int j = 0; j < 3; j++) {

L[i][j] = L1[j];

//清空

L1[j] = 0;

}

}

//透视投影

for (int k = 0; k < 3; k++)

{

double v[3];

for (int m = 0; m < 3; m++) {

v[m] = p[k][m];

}

Project(v);

Point[k][0] = ScreenP[0];

Point[k][1] = ScreenP[1];

}

double ss = Uv[0] \* N1[0] + Uv[1] \* N1[1] + Uv[2] \* N1[2];

if (ss >= 0)//根据数量积正负消隐

{

fillup();

}

}

//环境光

void CtttttDlg::environment()

{

for (int i = 0; i < 3; i++) {

L1[i] += (double)k1[i] \* I1[i];

}

}

//漫反射

void CtttttDlg::diffuse(double v[3], int index)

{

double d = sqrt((aa[0] - v[0]) \* (aa[0] - v[0]) + (aa[1] - v[1]) \* (aa[1] - v[1]) + (aa[2] - v[2]) \* (aa[2] - v[2]));

double xx = min(1, (double)1.0 / (cc[0] + cc[1] \* d + cc[2] \* d \* d));

//计算L·N

double mm = 0, N2[3];

for (int i = 0; i < 3; i++) {

if (index == 1)

N2[i] = N1[i];

else if (index == 2)

N2[i] = Nd[i];

else if (index == 3)

N2[i] = Ndr[i];

else

N2[i] = Nr[i];

}

for (int i = 0; i < 3; i++) {

mm += N2[i] \* Lv[i];

}

for (int i = 0; i < 3; i++) {

L1[i] += (double)k2[i] \* I2[i] \* max(mm, 0);

}

}

//镜面反射

void CtttttDlg::mirror(double v[3], int index)

{

double d = sqrt(abs(static\_cast<long long>((aa[0] - v[0]) \* (aa[0] - v[0]) + (aa[1] - v[1]) \* (aa[1] - v[1]) + (aa[2] - v[2]) \* (aa[2] - v[2]))));

double xx = min(1, (double)1.0 / (cc[0] + cc[1] \* d + cc[2] \* d \* d));

//计算H向量

double hh[3];

hh[0] = (Uv2[0] + aa[0], hh[1] = Uv2[1] + aa[1], hh[2] = Uv2[2] + aa[2]) / 2.0;

//单位化

double jj = sqrt(abs(static\_cast<long long>(hh[0] \* hh[0] + hh[1] \* hh[1] + hh[2] \* hh[2])));

for (int i = 0; i < 3; i++) {

hh[i] = hh[i] / jj;

}

//求max(H·M，0)

double ll = 0, N2[3];

for (int i = 0; i < 3; i++) {

if (index == 1)

N2[i] = N1[i];

else if (index == 2)

N2[i] = Nd[i];

else if (index == 3)

N2[i] = Ndr[i];

else

N2[i] = Nr[i];

}

for (int i = 0; i < 3; i++) {

ll += N2[i] \* hh[i];

}

double uu = max(0, ll);

if (uu > 0) {

//高光指数为5

double kk = 1;

for (int i = 0; i < 5; i++) {

kk = kk \* uu;

}

for (int i = 0; i < 3; i++) {

L1[i] += (double)k3[i] \* I3[i] \* kk;

}

}

else {}

}

//透视投影

void CtttttDlg::Project(double p[3])

{

double ViewP[3];//观察坐标系

ViewP[0] = k4[1] \* p[0] - k4[3] \* p[1];//观察坐标系的三维坐标

ViewP[1] = -k4[7] \* p[0] - k4[8] \* p[1] + k4[2] \* p[2];

ViewP[2] = -k4[5] \* p[0] - k4[6] \* p[1] - k4[4] \* p[2] + R;

ScreenP[0] = (double)D \* ViewP[0] / ViewP[2];//屏幕坐标系的二维坐标

ScreenP[1] = D \* ViewP[1] / ViewP[2];

}

//双线性插值

void CtttttDlg::fillup()

{

CDC\* pDC = GetDC();

int lightX = 500;

int lightY = 0;

//下面开始使用线性表实现有效边表填充

/\*

\* 将边表中的信息分别用不同的数组存储

\* 每个边的信息联系通过下标联系，即同一下标的各个信息数组表示一个边的

\* 边表填充时通过y\_min和y\_max的左闭右开来判断哪一个边是有效边，同时在有效边使用后各自的x\_min自动加k(这里的k为x变化率)

\* 记录下的点按照x的大小重新排序，且填充时，只按下表的[0,1][2,3]等区间填充

\* 其中注意水平的线段（即两点的y相等）就直接不计入

\*/

//记录边的最值和x变化率和对应起点的下标（方便计算光强）

double y\_min[3], y\_max[3], x\_min[3];

int lo[3];

double kk[3];

int count1 = 0;//记录有效边表的数量

for (int i = 0; i < 3; i++) {

int j = (i + 1) % 3;

if (abs(static\_cast<long long>(Point[i][1] - Point[j][1])) > 1.1) {

kk[count1] = (double)(Point[i][0] - Point[j][0]) / (Point[i][1] - Point[j][1]);

y\_min[count1] = Point[i][1] < Point[j][1] ? Point[i][1] : Point[j][1];

x\_min[count1] = Point[i][1] < Point[j][1] ? Point[i][0] : Point[j][0];

y\_max[count1] = Point[i][1] > Point[j][1] ? Point[i][1] : Point[j][1];

lo[count1] = i;

count1++;

}

else {

//水平的线就直接一次线性插值

double x\_min = Point[i][0] < Point[j][0] ? Point[i][0] : Point[j][0];

double x\_max = Point[i][0] > Point[j][0] ? Point[i][0] : Point[j][0];

double Ls[3];//记录光强

for (int k = x\_min; k <= x\_max; k++) {

for (int m = 0; m < 3; m++) {

Ls[m] = L[i][m] \* (k - Point[j][0]) / (Point[i][0] - Point[j][0]) + L[j][m] \* (k - Point[i][0]) / (Point[j][0] - Point[i][0]);

}

pDC->SetPixel(k + 400, 300 - Point[i][1], RGB(Ls[0] \* 255, Ls[1] \* 255, Ls[2] \* 255));

}

}

}

//找出扫描线算法的范围

double Y\_min, Y\_max;

Y\_min = min(Point[0][1], Point[1][1]);

Y\_min = min(Y\_min, Point[2][1]);

Y\_max = max(Point[0][1], Point[1][1]);

Y\_max = max(Y\_max, Point[2][1]);

COLORREF shadowColor = RGB(100, 100, 100); // 阴影颜色为灰色，您可以根据需要调整

//开始扫描

for (double i = Y\_min; i <= Y\_max; i++) {

double ax[3]; int acount = 0;//保存交点的x值

int lo2[3];//保存交点边的起始点的下标

for (int j = 0; j < count1; j++) {

if (y\_min[j] <= i && y\_max[j] > i) {

ax[acount] = x\_min[j];

lo2[acount++] = lo[j];

x\_min[j] += kk[j];

}

}

//对交点的x进行排序

double t;

int t2;

for (int w = 0; w < acount - 1; w++) {

if (ax[w] > ax[w + 1]) {

t = ax[w + 1];

t2 = lo2[w + 1];

ax[w + 1] = ax[w];

lo2[w + 1] = lo2[w];

ax[w] = t;

lo2[w] = t2;

}

}

//求出光强

double Id[3][3];//交点的光强

for (int k = 0; k < acount; k++) {

for (int j = 0; j < 3; j++) {

Id[k][j] = (double)L[lo2[k]][j] \* ((i - Point[(lo2[k] + 1) % 3][1]) / (Point[lo2[k]][1] - Point[(lo2[k] + 1) % 3][1])) + L[(lo2[k] + 1) % 3][j] \* ((i - Point[lo2[k]][1]) / (-Point[lo2[k]][1] + Point[(lo2[k] + 1) % 3][1]));

}

}

//求出扫描线上的点的光强并设置颜色

double If[3];//扫描线上的点的光强

for (int j = 0; j < acount - 1; j += 2)

for (double k = ax[j]; k <= ax[j + 1]; k++) {

// 检查当前像素是否在光源背后

if (lightX < ax[j] && lightY < i) {

// 如果在光源背后，则将当前像素颜色设置为阴影颜色

pDC->SetPixel(k + 400, 300 - i, shadowColor);

}

else {

// 否则，根据光照计算当前像素颜色

for (int jj = 0; jj < 3; jj++) {

If[jj] = (double)Id[j][jj] \* ((k - ax[j + 1]) / (ax[j] - ax[j + 1])) + Id[j + 1][jj] \* ((k - ax[j]) / (ax[j + 1] - ax[j]));

}

pDC->SetPixel(k + 400, 300 - i, RGB(If[0] \* 255, If[1] \* 255, If[2] \* 255));

}

}

acount = 0;

}

ReleaseDC(pDC);

}

//透视投影系数

void CtttttDlg::Init()

{

double PI = 3.1415926;

k4[1] = sin(PI \* Thta / 180);

k4[2] = sin(PI \* Fei / 180);

k4[3] = cos(PI \* Thta / 180);

k4[4] = cos(PI \* Fei / 180);

k4[5] = k4[3] \* k4[2];

k4[6] = k4[1] \* k4[2];

k4[7] = k4[3] \* k4[4];

k4[8] = k4[1] \* k4[4];

}

void CtttttDlg::OnBnClickedButton1()

{

//镜面反射光初始化

for (int i = 0; i < 3; i++) {

I3[i] = I3[i] - I1[i] \* k1[i] - I2[i] \* k2[i];

}

//得系数

Init();

//初始化视点

v1[0] = R \* k4[5], v1[1] = R \* k4[6], v1[2] = R \* k4[4];

ReadPoint();

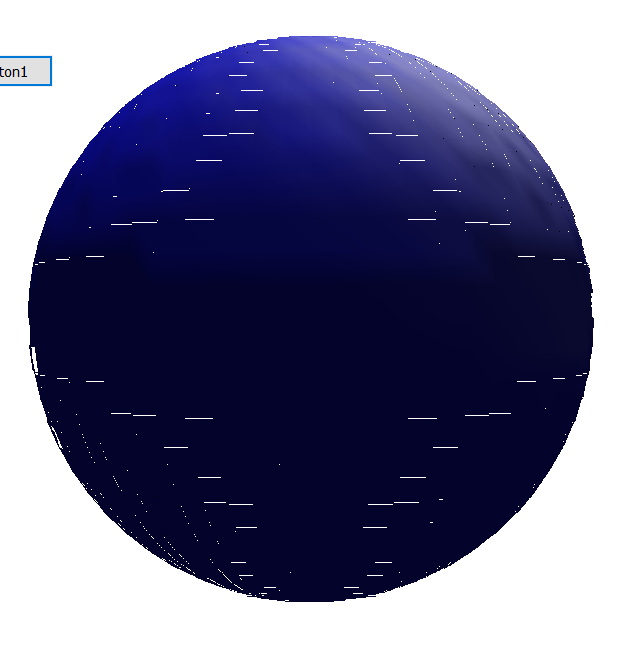
ReadFace();

divid();

// TODO: 在此添加控件通知处理程序代码

}

1. **运行结果截图**



1. **本门课程学习体会与心得**

在做课设中感到了计算的复杂性，有时看起来很简单的一个算法，实行起来中会出现很多难点，需要自己去上网或者用一定的数据结果来简化才能实现。所以在以后的学习，我会时常以很认真的态度去对待每一个问题，并解决。