目录

I.确定方向——选择对象 1

II.明确要求——划分面片和选择运动 1

III.研究原理——自建函数 2

1.通过控制点计算插值点 2
2.三维图形的基本变换 2
3.自建函数 3

IV.拉取、变换控制点 4

V.效果展示 5

VI.总结 5

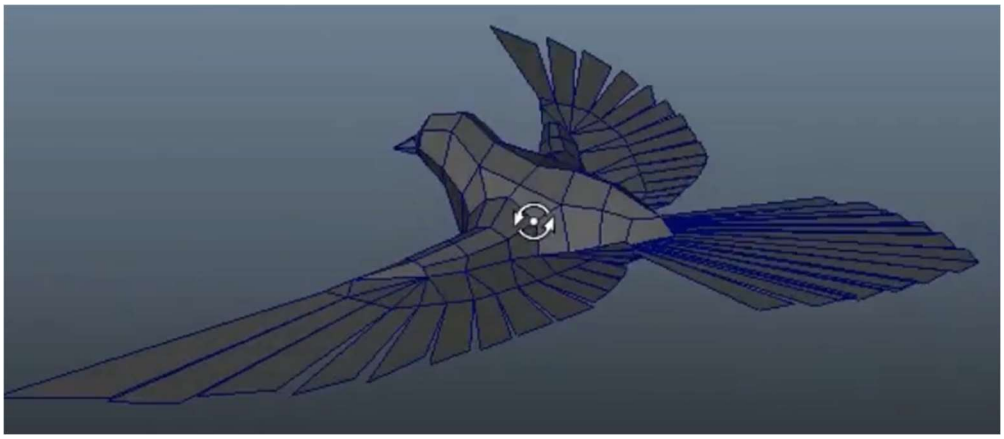
VII.附录： 6

1.两大主程序 6
generate_gif 6
A_Workflow_documentation 8
2.新建面片程序 9
C1 9
ns 10
xC1 10
xns 11
3.可视化程序 11
see_bezier 11
see_fbezier 12
only_see 12
bezier_func 12
control_pointslook 13
4.变换矩阵程序 13
anyrotationT 13
rotation 15
anyTranslationT 15
equal_scaleT 15
scaleT 16
Transformation1to3 16
Transformation3to1 17
Yzduichen 17

确定方向——选择对象

经过长时间讨论，我们小组选择了“飞翔之鸟”作为我们的建模对象，象征了青年勇于奋前、追求独立自由的精神特征。

因为现代生活的限制，我们很少观察飞鸟形态，为此要找用于参考的图片，最终选择了 1 张照片、3 张别人建模的截图作为参考，如下：



明确要求——划分面片和选择运动

根据《大作业说明》，基本要求为：

1. 实物对象复杂度要求：动画至少包含两个实物对象或者一个实物对象的面片数量大于 10 个（至少 3 x 10 个矩阵）
2. 动作要求：至少包含两个基本的几何变换动作

我们选择鸟作为建模对象，考虑到身体的对称性，我们只确定左半部分身体的控制点，右半部分直接对称生成，将面片这样划分：基面片背部一个，尾部、颈部、头部各一个，缝合下面用 3 个，喙部、膀部、羽部、尾羽各一个。

我们小组将运动分解为两部分：翅膀的扇动和身体的盘旋。

一个面片需要 4×4 控制点矩阵，一个矩阵包含三层，分别是 X,Y,Z 坐标值。面片的控制点需要存储，为了方便调用和修改，我们使用 matlab 的元胞 cell 变量类型，变量名选为 p，创造 n×3 的数据结构，n 行代表 n 个控制点矩阵，3 列代表三坐标；为了保存数据，将元胞保存到 afm.mat 中，保险与方便失误后倒退。

研究原理——自建函数

用到的数学原理分为两部分：通过控制点计算插值点、三维图形的基本变换。

通过控制点计算插值点

控制点矩阵为 P，插值点数据为 S，细密程度由 u、w 的取值数量决定。

每取一次 u、w，就能算出一个插值点，并存储起来；当 u、w 全部选值都算后，一个通过控制点计算出来的插值点数据就出来了。

为了便于计算，将 P 分为三层，分别是某一点的 x、y、z 坐标值，分别通过上述步骤计算得到插值点的对应坐标数据。

$$\mathbf{S}(u,w) = UM_zPM_z^TW^T$$

$$\mathbf{S}(u,w) = [u^3 \quad u^2 \quad u \quad 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}P\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}w^3\\ w^2\\ w\\ 1\end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix}\mathbf{p}_{00} & \mathbf{p}_{01} & \mathbf{p}_{02} & \mathbf{p}_{03}\\ \mathbf{p}_{10} & \mathbf{p}_{11} & \mathbf{p}_{12} & \mathbf{p}_{13}\\ \mathbf{p}_{20} & \mathbf{p}_{21} & \mathbf{p}_{22} & \mathbf{p}_{23}\\ \mathbf{p}_{30} & \mathbf{p}_{31} & \mathbf{p}_{32} & \mathbf{p}_{33}\end{bmatrix}$$

$$M_z = M_z^T$$

三维图形的基本变换

将所有点的表示化为齐次坐标矩阵形式，

$$\begin{pmatrix}x_1 & y_1\\ x_2 & y_2\\ x_3 & y_3\\ x_4 & y_4\end{pmatrix} \Rightarrow \begin{pmatrix}x_1 & y_1 & 1\\ x_2 & y_2 & 1\\ x_3 & y_3 & 1\\ x_4 & y_4 & 1\end{pmatrix}$$

然后通过公式套入计算变换后的点的坐标数据。

$$V^* = V \cdot T$$

V：变换前图形的控制点齐次坐标矩阵

V*：变换后图形的控制点齐次坐标矩阵

T：变换矩阵

变换矩阵 T 如下：

变换名称	变换矩阵	说明

比例变换	$T=\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	a，e，i：分别是 x，y，z 方向的比例因子
等比例变换	$T=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}$	s:全图的比例因子
平移变换	$T=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix}$	1，m，n：分别是 x，y，z 方向的平移量
旋转变换 x 轴	$T=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	a 是绕 x 轴旋转角，逆时针为正顺时针为负
旋转变换 y 轴	$T=\begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	β是绕 y 轴旋转角，逆时针为正，顺时针为负
旋转变换 z 轴	$T=\begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	y 是绕 z 轴旋转角,逆时针为正,顺时针为负

自建函数

为了方便建模和变换，提高代码的复用性，我们选择了先建立工具，再去建模，根据工作过程中的问题改善工具这样的路线。最终建立了如下代码和函数，具体代码会放入附录：

see_bezier：调用 bezier_func()通过元胞变量 p 的控制点数据计算插值点数据并可视化，还调用 control_pointslook()控制点可视化，方便后续调整控制点位置。

only_see()：相比 see_bezier，只能看见一个面片。

see_fbezie：相比 see_bezier，去掉了控制点可视化

control_pointslook()：控制点可视化，可调整颜色

ns()：newslice，方便生成 C1 连续面片，借助 C1()依托某一面片在某一方向生成面片，并为了方便观察所有控制点位置，将自由的两行控制点数据向外展开。

xns()：由于会调整母面片（子面片依托母面片生成）的边坐标，这样同一条边就不再重合，为方便快速调整为 C1 连续，用 ns 简化出了 xns。

C1()：计算相邻边满足 C1 连续的控制点数据。还有 xC1

yzduichen：专门为生成关于 yz 平面对称数据创建的简易代码。

bezier_func()：计算一个控制点矩阵的插值数据，可控制细密程度。

anyrotationT()：旋转。

anyTranslationT()：平移。

rotationT()：关于 z 轴旋转。

equal_scaleT()：等比例缩放。

scaleT()：某方向缩放。

Transformation3to1：三坐标分离式数据转化为齐次坐标数据。

Transformation1to3：齐次坐标数据转化为三坐标分离式数据。

A_Workflow_documentation：为了更好理解流程，创建了《工作流程文档》，不过未能保持更新，对于外人用处已经不大了。

generate_gif：运动部分的代码，外加生成 gif 代码。

guiji：根据一段运动，自动生成返回原来位置和对称的运动，可根据参数调整对称运动幅度的大小。

拉取、变换控制点

基面片即第一个面片的控制点需要手动输入。后面的面片根据已存在的面片先生成出来。

改变面片形状有两种办法：一是手动拉取控制点，二是通过变换函数。

变换函数不能实现复杂的变换，但快速简便，我们组用在了将喙变换为膀的过程和调整躯体之外部分的（喙、膀、羽、尾羽）相对位置上。

手动拉取虽然繁琐，但可以拉出复制曲面，是定型的必要操作。

我们组的建立流程如下：

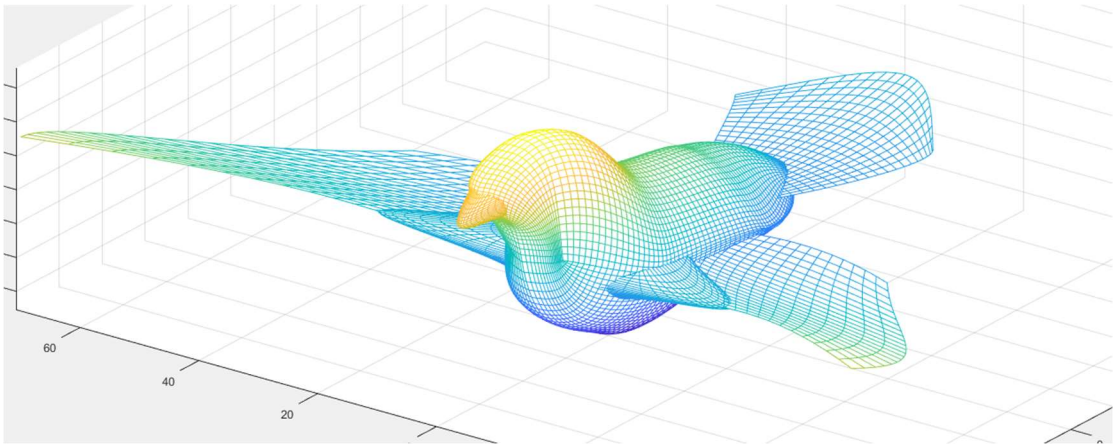
1. 手写输入第一张面片：背部，并拉到合适形状。
2. 生成尾部，然后颈部，头部，手动改变点坐标，达到满意的形状。
3. 使用三个面片缝合下面并拉到合适形状。
4. 创建喙部面片（锥形）利用变换函数调整形状并插入对应位置。
5. 复制喙部面片并将其变换成膀部面片插入对应位置。
6. 创建羽部面片插入对应位置，并拉到合适形状。
7. 创建尾羽面片插入对应位置，并拉到合适形状。

至于运动部分的控制点改变：

1. 扇动部分预先生成 6 个羽部控制点加 4 个膀部控制点的每一帧数据，然后在后面的循环中逐次调用。

2. 身体则是用了基本变换中的平移和绕轴旋转，在每次循环中，将所有面片数据都通过基本变换函数调整位置，最终将整个鸟的 22 个面片都变换。

效果展示



总结

这次的作业加深了对控制点的理解，并且训练了在工作中应用 matlab 的能力，较好地完成了大作业的任务，更重要的是建立了一套贝塞尔曲面表面建模的完整工具，可以在将来使用，独立自主。

贴图会让作品更加精美，但也无法展示贝塞尔曲线的特点，因此没有贴图，背景和音乐也是如此。

附录：

两大主程序

generate_gif

% 不知原因，需要先有 gif 文件，才能写入，因此运行一次，出错暂停，再执行一次 imwrite(I, map, filename, 'gif', 'Loopcount', inf, 'DelayTime', 0.3);，就可以了

```
clc;clear all;
load('afb.mat');    %加载控制点数据
filename = 'cfzj.gif';  % 保存文件名

%set(gcf, 'visible', 'off');                % 不显示窗口

%调整控制点
s = 0.8;    $调整参数
wingz1 = linspace(8,45,100);    wingz1 = guiji(wingz1,s);
wingz2 = linspace(4.5,30,100);    wingz2 = guiji(wingz2,s);
wingz3 = linspace(-1.28734077588658,20,100);    wingz3 = guiji(wingz3,s);
wingz4 = linspace(-3.46457607500707,15,100);    wingz4 = guiji(wingz4,s);
wingz5 = linspace(0,30,100);    wingz5 = guiji(wingz5,s);
wingz6 = linspace(-3.5,20,100);    wingz6 = guiji(wingz6,s);

wingx1 = linspace(-70,-65,100);    wingx1 = guiji(wingx1,s);
wingx2 = linspace(-75,-45,100);    wingx2 = guiji(wingx2,s);
wingx3 = linspace(-55,-40,100);    wingx3 = guiji(wingx3,s);
wingx4 = linspace(-40,-30,100);    wingx4 = guiji(wingx4,s);
wingx5 = linspace(-35,-25,100);    wingx5 = guiji(wingx5,s);
wingx6 = linspace(-26.8972568484771,-20,100);    wingx6 = guiji(wingx6,s);


armz1 = linspace(1.77178654752536,35,100);    armz1 = guiji(armz1,0.65);
armz2 = linspace(1.77178654752536,35,100);    armz2 = guiji(armz2,0.65);
armz3 = linspace(-1.83380306553770,31.5,100);    armz3 = guiji(armz3,0.65);
armz4 = linspace(-1.83380306553770,31.5,100);    armz4 = guiji(armz4,0.65);


angle = 0;    %旋转角度
```

```
for i=1:10:400     %有问题，是以翅膀的扇动来划分的，还有身体移动

    figure(i);
    set(gcf,'position',[0,0,1600,900],'color','w');     %确定框的大小

    % 翅膀运动部分
    p{10,3}(1,1)=wingz1(i);
    p{10,3}(2,1)=wingz2(i);
    p{10,3}(3,1)=wingz3(i);
    p{10,3}(4,1)=wingz4(i);
    p{10,3}(1,2)=wingz5(i);
    p{10,3}(2,2)=wingz6(i);

    p{10,1}(1,1)=wingx1(i);
    p{10,1}(2,1)=wingx2(i);
    p{10,1}(3,1)=wingx3(i);
    p{10,1}(4,1)=wingx4(i);
    p{10,1}(1,2)=wingx5(i);
    p{10,1}(2,2)=wingx6(i);

    p{9,3}(2,2)=armz1(i);
    p{9,3}(2,3)=armz2(i);
    p{9,3}(3,2)=armz3(i);
    p{9,3}(3,3)=armz4(i);

    q = p;     %复制形态数据
    yzduichen;   %计算对称点

    angle = angle + 1 ;
    %身体运动
    for nana = 1:100

[p{nana,1},p{nana,2},p{nana,3}]=anyTranslationT(p{nana,1},p{nana,2},p{nana,3},80,0,0);
        [p{nana,1},p{nana,2},p{nana,3}]=rotationT(p{nana,1},p{nana,2},p{nana,3},angle);

    end

    see_fbezier;
    view(-150,20);
    grid on;axis equal;
    frame = getframe(gcf);     %得到一帧，gcf 获得当前视窗句柄
    im = frame2im(frame);          %转化为图片  制作 gif 文件，图像必须是 index 索引图像

    [I, map] = rgb2ind(im, 256);     %rgb 变成 ind
```

```
        if i == 1
            imwrite(I, map, filename, 'gif', 'Loopcount', inf, 'DelayTime', 0.1);
        else
            imwrite(I, map, filename, 'gif', 'WriteMode', 'append', 'DelayTime', 0.1);
        end

        %pause();
    close all;              %close(gcf);

    p = q;
end

                                ----------其中的 guiji----------
function I=guiji(wing,s)
%根据一段运动，自动生成返回原来位置和对称的运动，可根据参数 s 调整对称运动幅度的大小。
    if(~exist('s','var'))
    s = 1;  % 如果未出现该变量，则对其进行赋值
    end

    I = [wing fliplr(wing)];    %原路返回
    I = [I (-wing+2*wing(1)).*s];    %关于初始位置对称的运动控制点坐标
    I = [I fliplr(-wing+2*wing(1)).*s];    %上一行的返回

end
```

A_Workflow_documentation

```
% 先将要建模的物体划分面片，并标记编号。

% 创建新的模型时，请先修改下方变量（有----------的），运行一次后，再根据下面流程进行面片修改。

% 填写第一张面片控制点：
% p{1,1}=[;;];
% p{1,2}=[;;];
% p{1,3}=[;;];

% 创建新面片（默认 C1 连续），在命令行运行：ns(j,i,direction)，还有改进的地方
% j 是贴着的面片矩阵，i 是创建第几张面片，direction 方向，对着胸的 填 0 ，逆时针依次加 1 。
% xns
```

```
% 调整控制点剩下两行的坐标值
% p{i,1}(,) = ;    第 i 个面片，x 坐标，选择点

% 查看效果：see_bezier    关于 yz 平面对称：see_bezierd        only_see(i)
% 不满意继续调整

% 保存数据（非常重要，否则数据会很容易丢失）
% save afb.mat p
% 而后创建新的面片

% 如果出错，返回上一步工作环境：load('first.mat')

clear;

global p;

num = 100;  %面片数量----------------------

p=cell(num,3);   %生成所有面片所需要的控制点矩阵，一行三个矩阵 x,y,z，分别是三坐标的，总共 num 行
for i=1:num
    for j=1:3
        p{i,j}=ones(4);
    end
end

un = 20;  %x 方向插值数量--------------------
wn = 20;  %y 方向插值数量----------------------
```

新建面片程序

C1

```
function [Qx,Qy,Qz]=C1(Px,Py,Pz,direction)
% 根据输入的三坐标矩阵，和方向，返回满足 C1 的三坐标矩阵，多余的根据算法展开
% 先统一视角：matlab 默认视图，对着胸的为 0，往左转，依次是 1、2、3
    Qx=ones(4); Qy=Qx;  Qz=Qx;

    alpha=1; % 默认 C1 连续
    Px=rot90(Px,direction);    Py=rot90(Py,direction);    Pz=rot90(Pz,direction);  %将要处理的边都对着胸

    Qz(1,:)=Pz(4,:);    Qy(1,:)=Py(4,:);    Qx(1,:)=Px(4,:);  %同一条边上的点一样
```

```
    Qz(2,:)=(Pz(4,:)-Pz(3,:))*alpha+Qz(1,:);  %再后一条边-同边=……
    Qy(2,:)=(Py(4,:)-Py(3,:))*alpha+Qy(1,:);
    Qx(2,:)=(Px(4,:)-Px(3,:))*alpha+Qx(1,:);


    if rem(direction,2) == 0
        Qx(3,:)=Qx(2,:);  Qy(3,:)=Qy(2,:)+Qy(2,:)-Qy(1,:);  Qz(3,:)=(Qz(2,:)+Qz(1,:))/2;  %自动生成另外两行数据，方便查看.x 继承第二行的
        Qx(4,:)=Qx(2,:);  Qy(4,:)=Qy(3,:)+Qy(2,:)-Qy(1,:);  Qz(4,:)=(Qz(3,:)+Qz(2,:))/2;   %y 以前两行插值作为步长，继续下去;     z 以前两行平均值
    end

    if rem(direction,2) == 1
        Qy(3,:)=Qy(2,:);  Qx(3,:)=Qx(2,:)+Qx(2,:)-Qx(1,:);  Qz(3,:)=(Qz(2,:)+Qz(1,:))/2;  %左边和右边的，因为旋转后，x、y 轴互换，所以也要换
        Qy(4,:)=Qy(2,:);  Qx(4,:)=Qx(3,:)+Qx(2,:)-Qx(1,:);  Qz(4,:)=(Qz(3,:)+Qz(2,:))/2;
    end
    Qx=rot90(Qx,-direction);  Qy=rot90(Qy,-direction);  Qz=rot90(Qz,-direction);  %似乎不必要，矩阵中点的坐标都是对的，只是位置相对坐标轴不一样

end
```

ns

```
function song=ns(j,i,direction)
% new sheet，slice  新的面片
    global p;

    [p{i,1},p{i,2},p{i,3}]=C1(p{j,1},p{j,2},p{j,3},direction);

end
```

xC1

```
function [Qx,Qy,Qz]=xC1(Px,Py,Pz,Xx,Xy,Xz,direction)
% 只修改相关联的边

    Qx=Xx; Qy=Xy;  Qz=Xz;

    alpha=1; % 默认 C1 连续
    Px=rot90(Px,direction);    Py=rot90(Py,direction);    Pz=rot90(Pz,direction);  %将要处理的边都对着胸
    Qx=rot90(Qx,direction);    Qy=rot90(Qy,direction);    Qz=rot90(Qz,direction);
```

```
    Qz(1,:)=Pz(4,:);    Qy(1,:)=Py(4,:);    Qx(1,:)=Px(4,:);  %同一条边上的点一样
    Qz(2,:)=(Pz(4,:)-Pz(3,:))*alpha+Qz(1,:);  %再后一条边-同边=……
    Qy(2,:)=(Py(4,:)-Py(3,:))*alpha+Qy(1,:);
    Qx(2,:)=(Px(4,:)-Px(3,:))*alpha+Qx(1,:);

    Qx=rot90(Qx,-direction);  Qy=rot90(Qy,-direction);  Qz=rot90(Qz,-direction);  %似乎不必要，矩阵中点的坐标都是对的，只是位置相对坐标轴不一样

end
```

xns

```
function song=xns(j,i,direction)
%% 只修改相关联的边,有缺陷，当两个面都倒转了，就不行了
% xiu new sheet，slice  修已有的面片
    global p;

    [p{i,1},p{i,2},p{i,3}]=xC1(p{j,1},p{j,2},p{j,3},p{i,1},p{i,2},p{i,3},direction);

end
```

可视化程序

see_bezier

```
for i=1:100    %这个限制了，每次都要手动修改
    [Sx,Sy,Sz]=bezier_func(p{i,1},p{i,2},p{i,3},20,20);  %先计算插值点

    control_pointslook(p{i,1},p{i,2},p{i,3});  %画控制点的图
    mesh(Sx,Sy,Sz);  %画插值点的图

    % 标注是第几个面片
    xzhou=sum(sum(p{i,1}))/numel(p{i,1});
    yzhou=sum(sum(p{i,2}))/numel(p{i,2});
    zzhou=max(max(p{i,3}))+sum(sum(p{i,3}))/numel(p{i,3});
    neirong = num2str(i);

    text(xzhou,yzhou,zzhou,neirong,'FontSize',10)
    xlabel('x');  ylabel('y');  zlabel('z');

    axis equal;
```

```
end
```

see_fbezier

```
% 不带控制点的

for i=1:100     %这个限制了，每次都要手动修改
    [Sx,Sy,Sz]=bezier_func(p{i,1},p{i,2},p{i,3},20,20);   %先计算插值点

    mesh(Sx,Sy,Sz);   %画插值点的图

    xlabel("x");   ylabel("y");   zlabel("z");

    hold on;
end

axis equal;
```

only_see

```
function song=only_see(i)
%i 是第几个面片
    global p;

    [Sx,Sy,Sz]=bezier_func(p{i,1},p{i,2},p{i,3},20,20);   %先计算插值点

    control_pointslook(p{i,1},p{i,2},p{i,3});   %画控制点的图
    mesh(Sx,Sy,Sz);   %画插值点的图

    xlabel("x");   ylabel("y");   zlabel("z");
    axis equal;

end
```

bezier_func

```
function [Sx,Sy,Sz] = bezier_func(Px,Py,Pz,un,wn)
% Px,Py,Pz 是 16 个控制点的三坐标值
% un,wn 分别是“x，y”坐标轴上取多少个计算点-1
% [Sx,Sy,Sz]返回所有计算点的三坐标值

    Mz=[-1 3 -3 1;3 -6 3 0;-3 3 0 0;1 0 0 0];
```

```
    Sx = zeros(un+1,wn+1);   %解释在循环里
    Sy = zeros(un+1,wn+1);
    Sz = zeros(un+1,wn+1);

    for i=0:un
        for j=0:wn

            U = [(i/un)^3 (i/un)^2 (i/un) 1];   % 计算系数矩阵
            W = [(j/wn)^3 (j/wn)^2 (j/wn) 1];
            W = W';

            Sx(i+1,j+1)=U * Mz * Px * Mz * W;   %为了方便存储每个计算点的三坐标
            Sy(i+1,j+1)=U * Mz * Py * Mz * W;    %使用矩阵存储，这不是坐标，而是
            Sz(i+1,j+1)=U * Mz * Pz * Mz * W;    %在两方向上分别数第几个的点对应的值

        end
    end

end
```

control_pointslook

```
function song = control_pointslook(Px,Py,Pz,k)     %k 控制颜色
%------控制点框架可视化---------
    if(~exist('k','var'))
    k = 'b';   % 如果未出现该变量，则对其进行赋值
    end

    plot3(Px,Py,Pz,'bs','markerfacecolor',k);     %描点
    hold on;mesh(Px,Py,Pz,'edgecolor','k','facealpha',0);     %连线

    hold on;   %保持
end
```

变换矩阵程序

anyrotationT

```
function [Wx,Wy,Wz]=anyrotationT(Vx,Vy,Vz,axis,angle,x,y,z)   %z 是逆时针旋转的角度，若是顺时针，取负
```

```
% 三维旋转变换,输入的是三坐标分开矩阵的形式,，x、y、z 是真转轴和对应坐标轴的距离,用真转轴 - 对应坐标轴
    Wx = Vx;    %保持输入输出形式相同
    Wy = Vy;
    Wz = Vz;

    angle = angle * pi / 180;    %变换角度为 pi 的数值
    if axis == 1
    T1 = [1 0 0 0;0 1 0 0;0 0 1 0;-x -y -z 1];    %先平移，再旋转，再回去
    T0 = [1 0 0 0;0 cos(angle) sin(angle) 0;0 -sin(angle) cos(angle) 0; 0 0 0 1];    %变换矩阵
    T11 = [1 0 0 0;0 1 0 0;0 0 1 0;x y z 1];

    T = T1*T0*T11;

    V = Transformation3to1(Vx,Vy,Vz);    %三坐标分离式数据转化为齐次坐标数据
    W = V*T;  %计算结果
    [Wx,Wy,Wz]=Transformation1to3(W);  %相反
    end

    if axis == 2
    T1 = [1 0 0 0;0 1 0 0;0 0 1 0;-x -y -z 1];
    T0 = [cos(angle) 0 -sin(angle) 0;0 1 0 0; sin(angle) 0 cos(angle) 0; 0 0 0 1];    %变换矩阵
    T11 = [1 0 0 0;0 1 0 0;0 0 1 0;x y z 1];

    T = T1*T0*T11;    %总变换矩阵

    V = Transformation3to1(Vx,Vy,Vz);
    W = V*T;
    [Wx,Wy,Wz]=Transformation1to3(W);

    if axis == 3              %%%%%%%%%不起作用,
    T1 = [1 0 0 0;0 1 0 0;0 0 1 0;-x -y -z 1];
    T0 = [cos(angle) sin(angle) 0 0;-sin(angle) cos(angle) 0 0;0 0 1 0; 0 0 0 1];    %变换矩阵
    T11 = [1 0 0 0;0 1 0 0;0 0 1 0;x y z 1];

    T = T1*T0*T11;

    V = Transformation3to1(Vx,Vy,Vz);
    W = V*T;
    [Wx,Wy,Wz]=Transformation1to3(W);
    end
end
```

rotation

```
function [Wx,Wy,Wz]=rotationT(Vx,Vy,Vz,z)   %z 是逆时针旋转的角度，若是顺时针，取负
% 三维旋转变换,输入的是三坐标分开矩阵的形式，这个只放了绕 z 轴的
    Wx = Vx;
    Wy = Vy;
    Wz = Vz;

    z = z * pi / 180;

    T = [cos(z) sin(z) 0 0;-sin(z) cos(z) 0 0;0 0 1 0; 0 0 0 1];    %变换矩阵

    V = Transformation3to1(Vx,Vy,Vz);
    W = V*T;
    [Wx,Wy,Wz]=Transformation1to3(W);

end
```

anyTranslationT

```
function [Wx,Wy,Wz]=anyTranslationT(Vx,Vy,Vz,x,y,z)
% 三维平移变换
    Wx = Vx;
    Wy = Vy;
    Wz = Vz;

    T = [1 0 0 0;0 1 0 0;0 0 1 0;x y z 1];   %变换矩阵

    V = Transformation3to1(Vx,Vy,Vz);
    W = V*T;
    [Wx,Wy,Wz]=Transformation1to3(W);

end
```

equal_scaleT

```
function [Wx,Wy,Wz]=equal_scaleT(Vx,Vy,Vz,s)   %s 是全图的比例因子,小于 1 是放大
% 三维比例变换,输入的是三坐标分开矩阵的形式
    Wx = Vx;
    Wy = Vy;
    Wz = Vz;
```

```
    T = [1 0 0 0;0 1 0 0; 0 0 1 0;0 0 0 s];    %变换矩阵

    V = Transformation3to1(Vx,Vy,Vz);
    W = V*T;
    [Wx,Wy,Wz]=Transformation1to3(W);

end
```

scaleT

```
function [Wx,Wy,Wz]=scaleT(Vx,Vy,Vz,a,e,i)   %a,e,j 分别是 x,y,z 方向的比例因子
% 三维比例变换,输入的是三坐标分开矩阵的形式
    Wx = Vx;
    Wy = Vy;
    Wz = Vz;

    T = [a 0 0 0;0 e 0 0; 0 0 i 0;0 0 0 1];    %变换矩阵

    V = Transformation3to1(Vx,Vy,Vz);

    W = V*T;

    [Wx,Wy,Wz]=Transformation1to3(W);

end


% [p{i,1},p{i,2},p{i,3}]=scaleT(p{i,1},p{i,2},p{i,3},a,e,i);
```

Transformation1to3

```
function [T3x,T3y,T3z]=Transformation1to3(One1)
    %将齐次坐标形式   变为   三坐标分开矩阵的形式
    T3x = zeros(4,4);    %%%%%%%%%%有局限
    T3y = zeros(4,4);
    T3z = zeros(4,4);

    for j=1:16
        T3x(j)=One1(j,1)/One1(j,4);    %将 T3x 的数从上往下，从左往右依次赋予 One1
        T3y(j)=One1(j,2)/One1(j,4);
        T3z(j)=One1(j,3)/One1(j,4);
    end
end
```

Transformation3to1

```
function One1=Transformation3to1(T3x,T3y,T3z)
    %将三坐标分开矩阵的形式  变为  齐次坐标形式
    One1 = ones(16,4);    %%%%%%%%%%有局限
    for j=1:16
        One1(j,1)=T3x(j);    %将 T3x 的数从上往下，从左往右依次赋予 One1
        One1(j,2)=T3y(j);
        One1(j,3)=T3z(j);
    end
end
```

Yzduichen

```
% 计算 yz 对称后的 p

num = 100;    %%%%%%%%%%%%%%%%%这里有局限，需要手动修改

for i=1:num/2+1
    p{i+50,1}=-p{i,1};
    p{i+50,2}=p{i,2};
    p{i+50,3}=p{i,3};
end
```