要求详细介绍以下问题：

1、对设计题目进行分析，描述设计中所需基础知识和工具；
2、能对题目进行模块分解，描述每个模块的功能、具体实现原理；

1、设计题目分析

光学三维测量技术以现代光学为基础，融光电子学、图像处理、图形学等为一体，是科学技术飞速发展所催生出的现代测量技术。基于光栅投影的三维测量方法通过将一定规则的光栅条纹投影到物体表面，对获取到的条纹图像作为三维信息的载体加以分析，由视觉原理得到物体的表面信息，在医疗、航天、文物保护等领域有着重要的应用。光栅投影三维测量方法从原理上通过建立高度－相位的映射关系实现对三维重建，因此相位的计算至关重要。相位计算通常包括相位提取和相位解包裹两个步骤，首先仿真生成正弦条纹图，然后根据四步相移原理对相位进行提取，最后根据三频外差的原理对相位进行解包裹计算。仿真相机拍摄到的图像并计算最终提取的绝对相位，并根据实际光路图在程序中仿真出结构光路图。然后根据光栅三维投影重建中核心的高度-相位公式对物体进行三维重建，最终将相机的畸变模型考虑进去，判断实际重建情况并与理想情况进行对比。

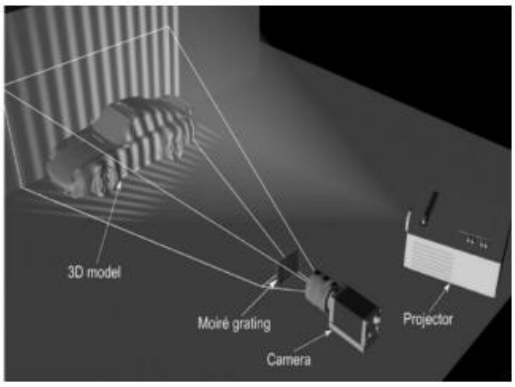


图 1 基于光栅投影原理的汽车三维重建

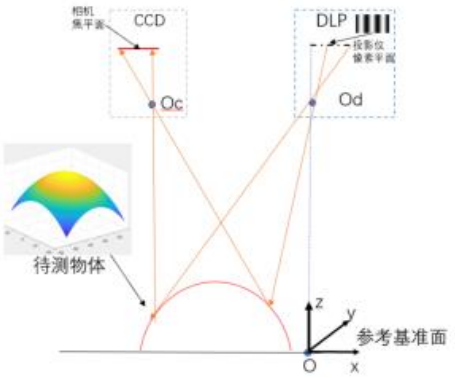


图 2 光栅投影测量系统结构示意图

题目的目的是全流程仿真出光栅投影三维重建的流程，通过模拟球面、投影的正弦条纹，结构光光路图，对模拟球体进行三维重建，并且通过三维点云的形式展现出重建结果。

2、任务模块分解

根据题目的要求我将该任务分成以下几个模块：

模块一：仿真生成正弦条纹，接着根据四步相移原理对相位进行提取，相移的原理就是通过采集多帧有一定相移的条纹图像来计算包含有被测物体表面三维信息的相位初值，通过查阅文献可知，N 步相移法对噪声有着最佳的抑制作用，因此在此采用四步相移法，接着通过多频外差原理进行相位解包，在此选用三频外差，完成初期的条纹仿真与相位提取和解算。

模块二：首先生成题目中所给的仿真球面，接着根据小孔成像模块模拟出光线与球面的交点以及反射光线与相机的交点，最终在空间中将相应点连接，形成空间的结构光路图。接着根据相位叠加的原理，计算出当光线投射到球面时正弦光的相位，并与原始的正弦光相位进行叠加。接着通过高斯低通滤波器将之前生成的原始正弦光和投射到球面的正弦光进行低通滤波，过滤掉高频噪声，再次运用四步相移法，获取多帧不同相位下的球面图片，接着通过多频外差法进行相位解算，分别解算出原始正弦光条纹和投射到球面的正弦光条纹的相位，然后将两个部分的相位进行相减，得到最后的相位，然后将相对相位化成绝对相位。最后画出解算后的绝对相位的分布图。

模块三：该模块的核心部分就是结构光三维重建中的核心高度-相位公式，根据上一个模块获得到的绝对相位计算高度，对待测物体进行三维重建，并在程序中将三维重建的点云显示出来。

模块四：该部分主要考虑到在相机拍摄过程中存在的畸变效应，将畸变系数代入到像素点坐标中，并最终根据高度-相位公式获取有畸变情况下的三维重建物体，并与无畸变情况下的物体进行比较。

二、算法流程

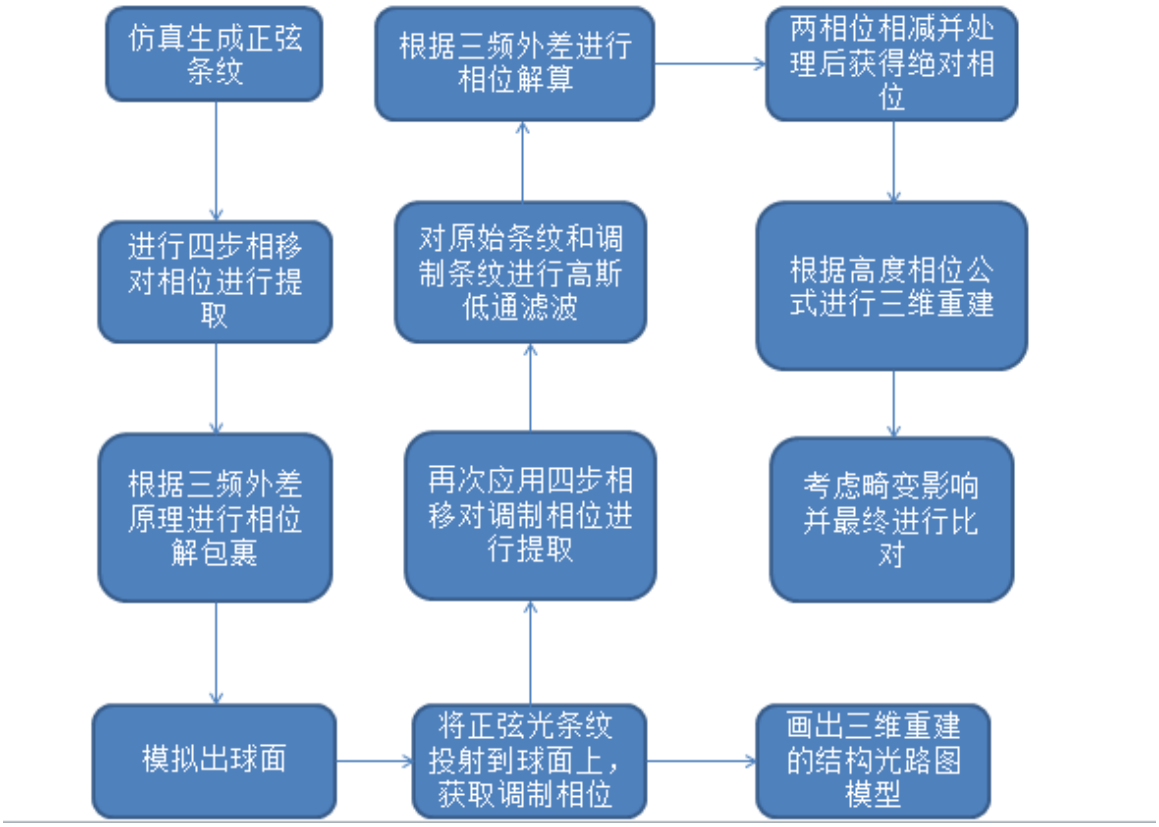
要求详细介绍以下问题：

1、绘制整个题目实现的流程图；

2、绘制各模块实现的流程图；

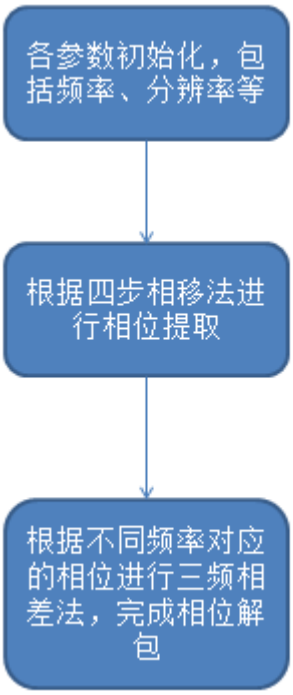
3、对算法流程进行适当说明。

1.整个题目的算法流程图如下：

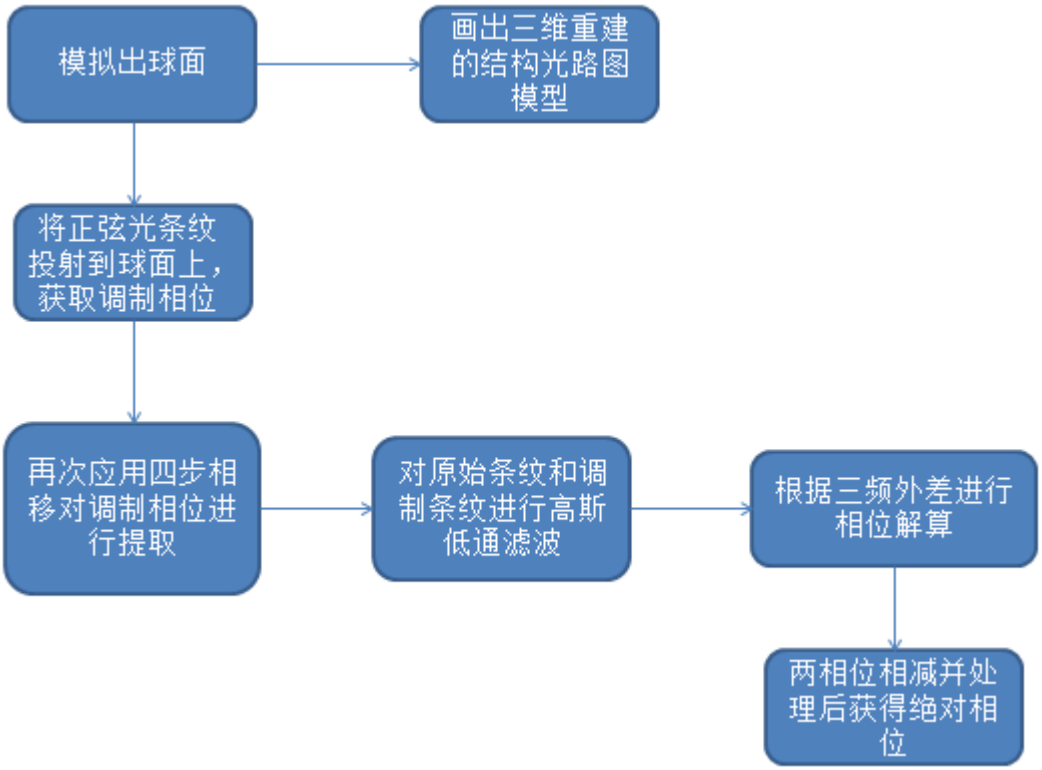


2. 各个模块的算法流程图：

模块一流程图：

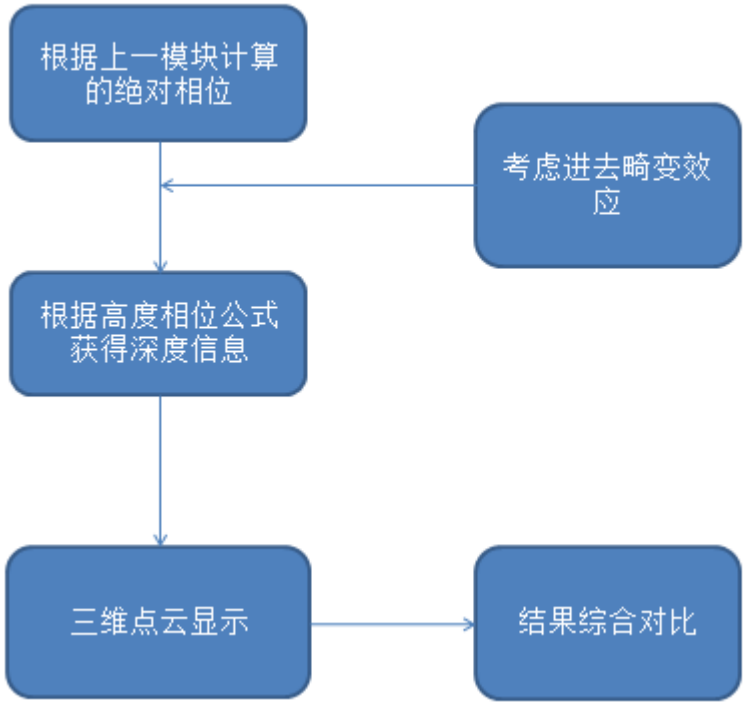


模块二流程图：



模块三算法流程图：

模块四算法流程图：



3.对算法流程的说明：

本算法整体采用了模块化架构思想，将显示、解相位、提取相位、画出结构图等

模块进行分离，通过函数之间调用的方式来实现各功能的实现，由于时间的一些原因，仍有一些优化的细节没有做好，但是整体上实现了题目的要求与功能实现。根据题目的流程进行了程序模块的设计，整体上程序的复用性高，通过更改一些关键部分参数即可实现整体程序的迁移运行。

三、实现过程

要求详细介绍以下问题：

1、各模块实现的函数说明，包括函数功能、函数输入输出等；
2、编程所需工具简单介绍，及主要操作步骤说明，并配以操作界面图；

1.各模块实现的函数说明

模块一函数说明：

程序请见 module1.m 文件

输入：分辨率、频率以及相位提取数 N 的大小

输出：每种频率的相位主值，相差，多频相差，相位解算结果

使用 cell，cos 等常用函数，利用分块矩阵存储不同相位和频率的图片，用 mesh 函数实现最终的解算相差彩色三维显示。

模块二函数说明：

程序structuregraph.m surface_pattern_simulation.m filter_parttern.m parse_pattern.m 等

输入：模拟球面的各种参数，相机以及投影仪的参数，频率的种类，相机光心以及投影仪光心之间的距离以及他们距离标准平面的距离。

输出：完整的光路结构图以及待测物体的仿真结构，未调制时的条纹，调制后的条纹，最终解算后的绝对相位。

通过plot3函数绘制出整体光路图，通过surf函数以及`shading interp`实现对于仿真球面的绘制，通过cos等常用函数完成相位的调制、解调与最终计算，通过

fspecial产生高斯低通滤波器。

模块三函数说明：

输入：计算好的绝对相位

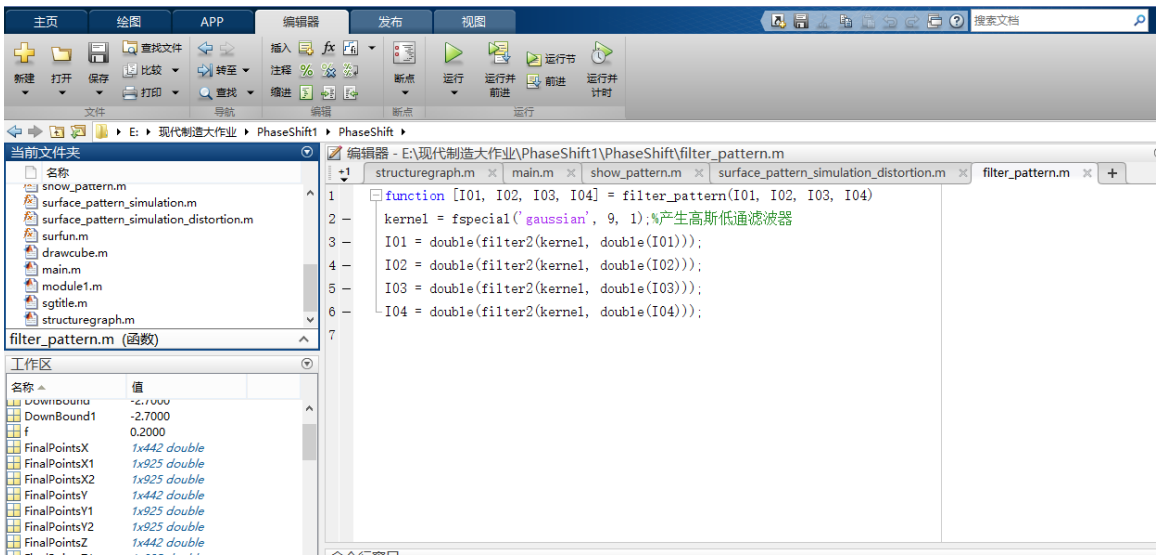
输出：与相位对应的高度信息

该部分主要根据目前已有的结构光三维重建中的高度-相位公式进行处理。

模块四函数说明：

同前几个模块相比，该部分主要是增加了输入 xy 时的畸变模型，其余并无太大改动。输入是绝对相位值，输出是深度信息与三维点云图。

编程采用工具均为 MATLAB：



四、结果及分析

要求详细介绍以下问题：

1、针对各个任务，分别列出具体处理结果和相关图示；

任务一的说明：

根据题目中所给的生成不同频率正弦光波对应的公式，进行如下程序设计：

$$I_i(x,y)=a(x,y)+b(x,y)\cos\left[2\pi f_0 x+\varphi(x,y)+2\pi(i-1)/N\right]$$

```
for i = 1:3
    for  j = 0:3
        for k = 1:width
           C{i,j+1}(:,k) =
0.5+0.5*cos(2*pi*k*freq(i)/width+2*pi*j/N);
        end
    end
end
```

接着根据四步相移的原理公式去计算每种频率对应的相位主值：

数学原理如下：

$$\phi(u,v)=\arctan\left(\frac{I_4(u,v)-I_2(u,v)}{I_1(u,v)-I_3(u,v)}\right)$$

程序实现如下：

```
for i = 1:3
     I1 = C{i,1};
     I2 = C{i,2};
     I3 = C{i,3};
     I4 = C{i,4};
      for g = 1:heigth
          for k = 1:width
            if     I4(g,k)==I2(g,k)&&I1(g,k)>I3(g,k)
                   phi{i,1}(g,k)=0;
            elseif I4(g,k)==I2(g,k)&&I1(g,k)<I3(g,k)
                   phi{i,1}(g,k)=pi;
            elseif I1(g,k)==I3(g,k)&&I4(g,k)>I2(g,k)
                   phi{i,1}(g,k)=pi/2;
            elseif I1(g,k)==I3(g,k)&&I4(g,k)<I2(g,k)
                   phi{i,1}(g,k)=3*pi/2;
            elseif I1(g,k)<I3(g,k)

phi{i,1}(g,k)=atan((I4(g,k)-I2(g,k))./(I1(g,k)-I3(g,k)))+
pi;
```

```
            elseif I1(g,k)>I3(g,k)&&I4(g,k)>I2(g,k)

phi{i,1}(g,k)=atan((I4(g,k)-I2(g,k))./(I1(g,k)-I3(g,k)));
            elseif I1(g,k)>I3(g,k)&&I4(g,k)<I2(g,k)

phi{i,1}(g,k)=atan((I4(g,k)-I2(g,k))./(I1(g,k)-I3(g,k)))+
2*pi;
            end
          end
     end
end
```

注意在程序中由于考虑到 arctan 函数会遇到几个特殊位置，因此通过 ifelse 判断进行相位主值计算。

计算多频相差的原理如下：

$$\phi_{ij}(x,y)=\begin{cases}\phi_i(x,y)-\phi_j(x,y) & \phi_i(x,y)>\phi_j(x,y)\\ \phi_i(x,y)-\phi_j(x,y)+2\pi & others\end{cases}$$

相应的程序实现为：

```
for g = 1:heigth
    for k = 1:width
        if PH1(g,k)>PH2(g,k)
            PH12(g,k) = PH1(g,k)-PH2(g,k);
        else
            PH12(g,k) = PH1(g,k)+2*pi-PH2(g,k);
        end
        if PH2(g,k)>PH3(g,k)
            PH23(g,k) = PH2(g,k)-PH3(g,k);
        else
            PH23(g,k) = PH2(g,k)+2*pi-PH3(g,k);
        end
    end
end
```

计算最终相差的程序如下：

```
for g = 1:heigth
    for k = 1:width
        if PH12(g,k)>PH23(g,k)
            PH123(g,k) = PH12(g,k)-PH23(g,k);
```

```
        else
            PH123(g,k) = PH12(g,k)+2*pi-PH23(g,k);
        end
    end
end
mesh(PH123);
```

任务二的说明：

根据球面参数进行球面绘制：

```
function z = surfun(R,x,y)
    z = sqrt(R*R - (x + 32).^2 - (y - 20).^2);
end
```

模拟正弦条纹投射到表面时：

```
var   = 2 * pi * f.* ones(h, 1) * x;
var_d = 2 * pi * f * d.* Z / len;
```

Z为根据球面公式计算出的纵坐标值

原始光栅：

```
I01 = 1 + cos(var);
I02 = 1 + cos(var + pi / 2);
I03 = 1 + cos(var + pi);
I04 = 1 + cos(var + 3 * pi / 2);
```

变形光栅：

```
I01d = 1 + cos(var + var_d);
I02d = 1 + cos(var + pi/2 + var_d);
I03d = 1 + cos(var + pi + var_d);
I04d = 1 + cos(var + 3 * pi / 2 + var_d);
```

依据的数学原理为：

$$\begin{cases} I_1(u,v) = a(u,v) + b(u,v)\cos(\varphi(u,v)) \\ I_2(u,v) = a(u,v) + b(u,v)\cos(\varphi(u,v) + \pi/2) \\ I_3(u,v) = a(u,v) + b(u,v)\cos(\varphi(u,v) + \pi) \\ I_4(u,v) = a(u,v) + b(u,v)\cos(\varphi(u,v) + 3\pi/2) \end{cases}$$

当正弦光照射到球体表面时，只需要将原始相位加上调制相位，通过四步相移法就可以获得调制后的相位和图像

模拟相机拍摄到的光栅照片显示函数：

```
function [] = show_pattern(I01, I02, I03, I04, suptitle)
subplot(2, 2, 1);  imshow(I01); title('I01');
subplot(2, 2, 2);  imshow(I02); title('I02');
subplot(2, 2, 3);  imshow(I03); title('I03');
subplot(2, 2, 4);  imshow(I04); title('I04');
```

```
 pause;
end
```

对调制后的相位进行高斯低通滤波，目的是为了过滤掉其中的高频噪声与干扰

```
function [I01, I02, I03, I04] = filter_pattern(I01, I02, I03,
I04)
kernel = fspecial('gaussian', 9, 1);
I01 = double(filter2(kernel, double(I01)));
I02 = double(filter2(kernel, double(I02)));
I03 = double(filter2(kernel, double(I03)));
I04 = double(filter2(kernel, double(I04)));
```

相位解算的原理与第一问相同，但是为了更好的封装程序，在这一个任务中设计了相位解算函数parse_pattern

```
function [phase] = parse_pattern(I01, I02, I03, I04)
[width, height] = size(I01);

phase = zeros(300, 300);
for w = 1: width
    for h = 1: height
        s = abs(atan((I02(w, h) - I04(w, h)) / (I01(w, h) - I03(w,
h))));
        if(I02(w, h) >= I04(w, h))     % sin(var) < 0
            if(I01(w, h) <= I03(w, h)) % cos(var) > 0
                phase(w, h) = -s;
            else
                phase(w, h) = s - pi;
            end
        else                          % sin(var) > 0
            if(I01(w, h) <= I03(w, h)) % cos(var) > 0
                phase(w, h) = s;
            else
                phase(w, h) = pi - s;
            end
        end
    end
end
end
```

将原始相位与调制相位通过unwrap函数进行相减，目的是为了保证相位不发生突变，反应出真实的相位变化。

```
phase = unwrap(phase1 - phase0);
```

任务三的说明：

高度-相位公式如下：

$$X = \frac{\theta - \theta_0}{2\pi}\lambda_0$$

$$PP' = \frac{l(\theta_A - \theta_B)}{(\theta_A - \theta_B) + 2\pi d / \lambda_0}$$

```
H = -p * len * phase./ (p * phase +2 * pi * d);
```

任务四的说明：

考虑畸变模型：

$$\begin{bmatrix}\overline{x}\\ \overline{y}\end{bmatrix} = \begin{bmatrix}d(k_1,k_2,r^2)x\\ d(k_1,k_2,r^2)y\end{bmatrix}$$

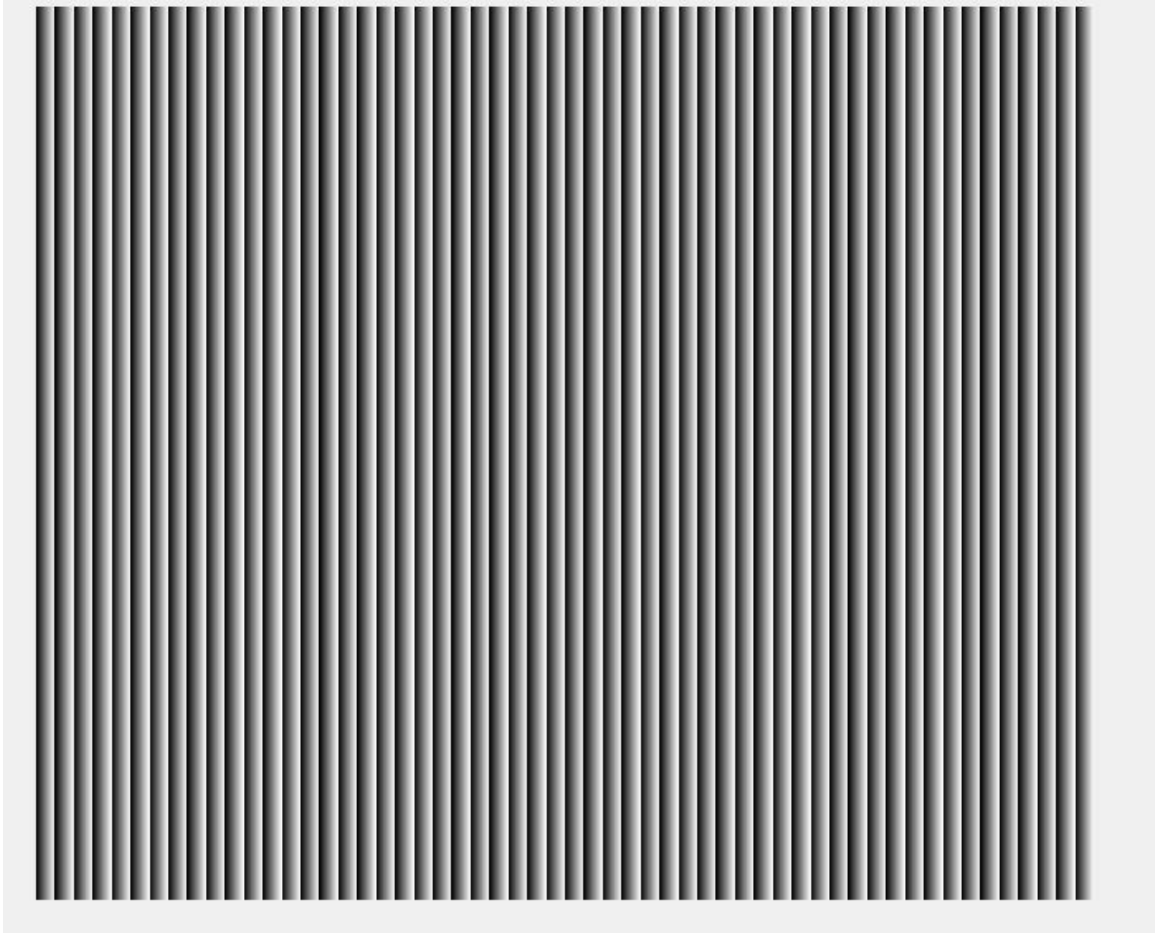
$$d(k_1,k_2,r^2) = 1 + (k_1 + k_2 r^2), r^2 = x^2 + y^2$$

```
X1=X*(1+(0.02+0.04*(X.*X+Y.*Y)));
Y1=Y*(1+(0.02+0.04*(X.*X+Y.*Y)));
Z = 0.4*sqrt(100*100-(X1-150).^2-(Y1-150).^2);
```

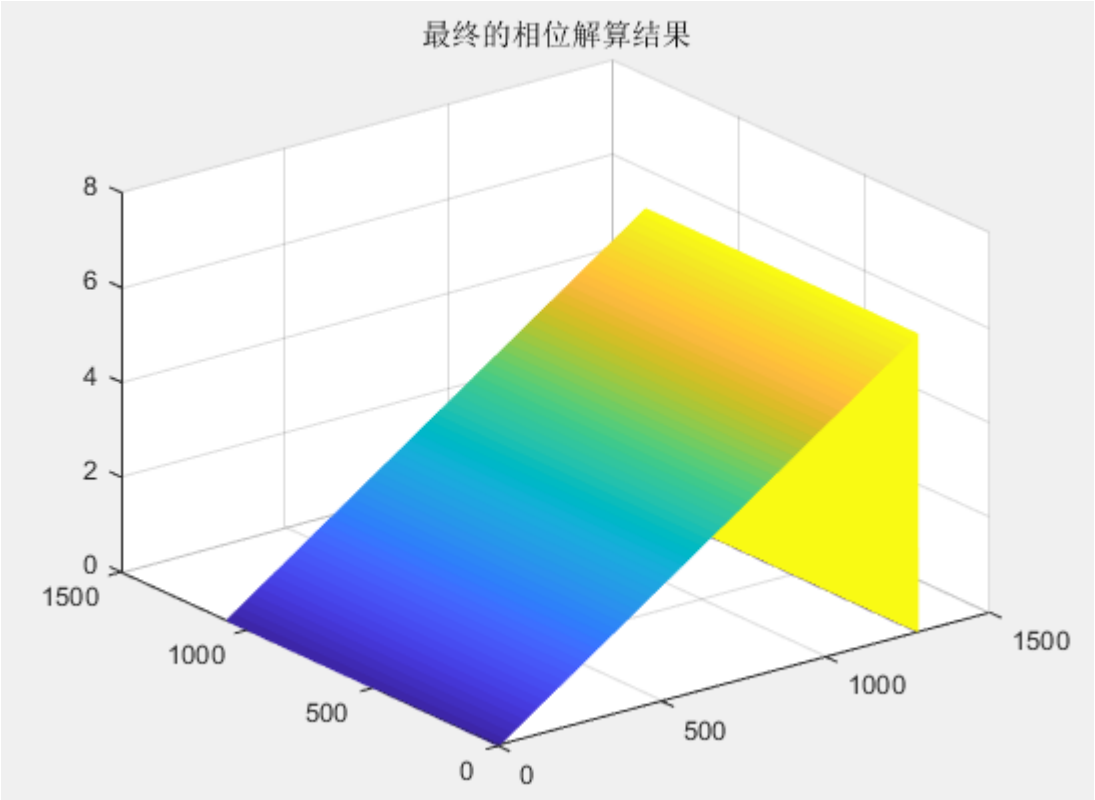
考虑畸变模型的程序如上所示

2、对结果进行分析说明，如必要可配以图表说明；

任务一结果：



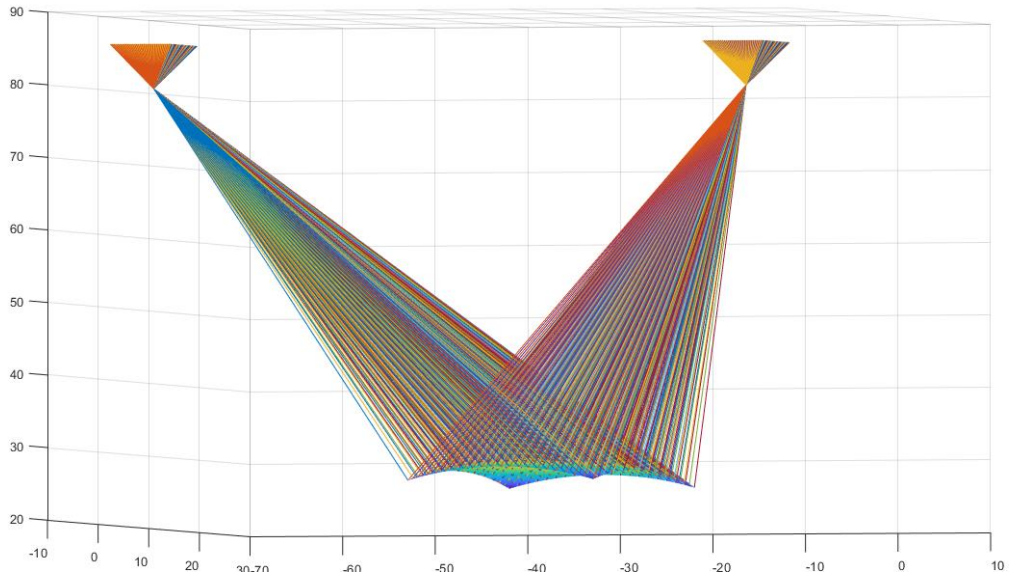
仿真出的正弦条纹图：



上图为最终的相位解算结果。可以看到程序最终解算的相位符合实际要求

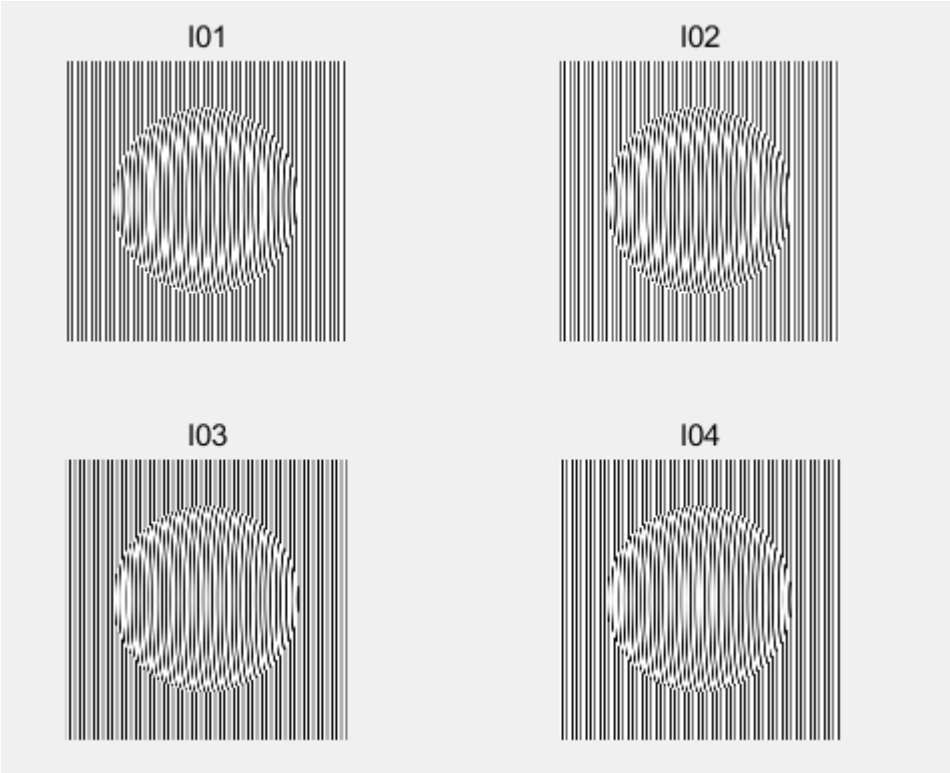
任务二结果：

仿真出的结构光路图：

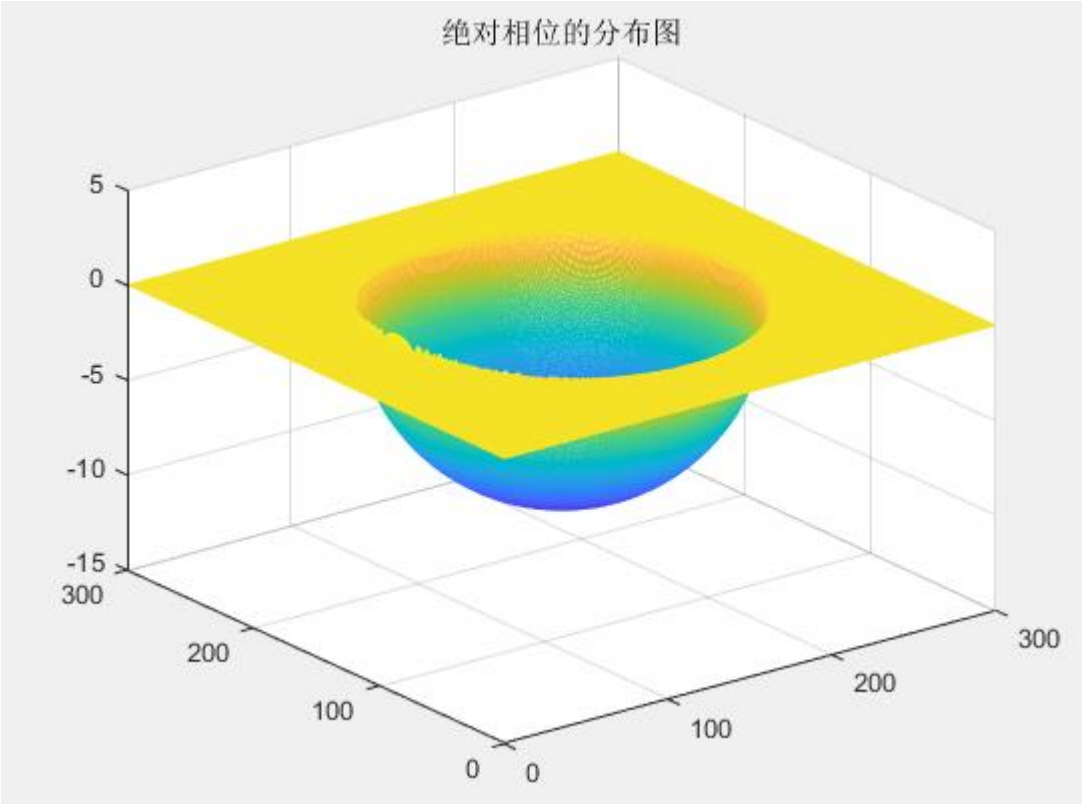


仿真出的拍摄到的球面的图像：

可以看到程序仿真出来的结构光路图与实际中基本一致。

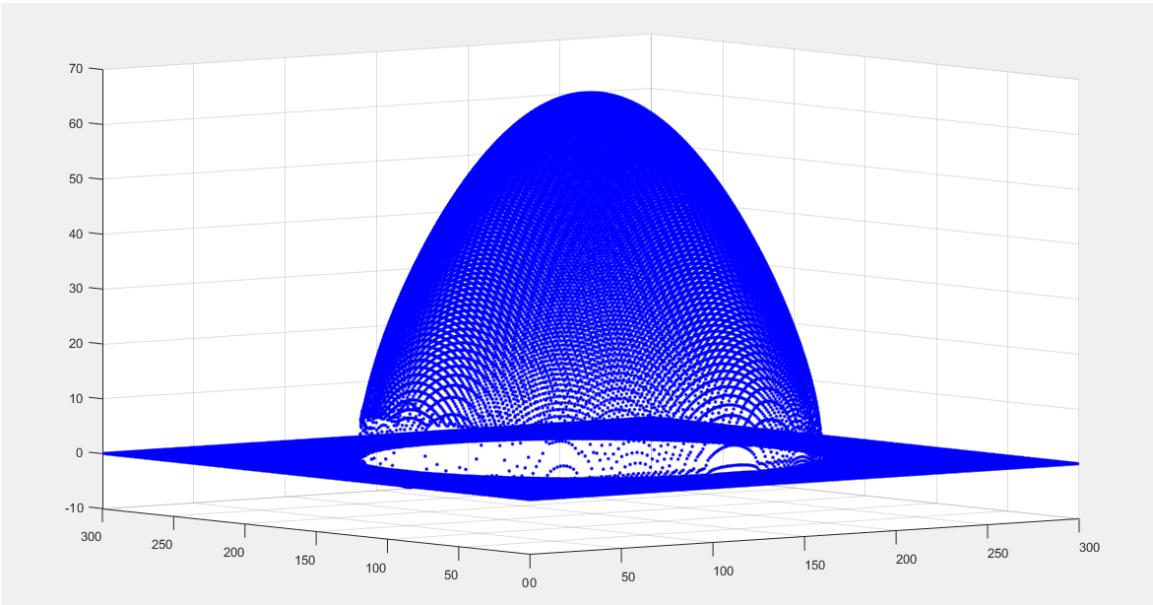


进行相位解算后的最终相位：

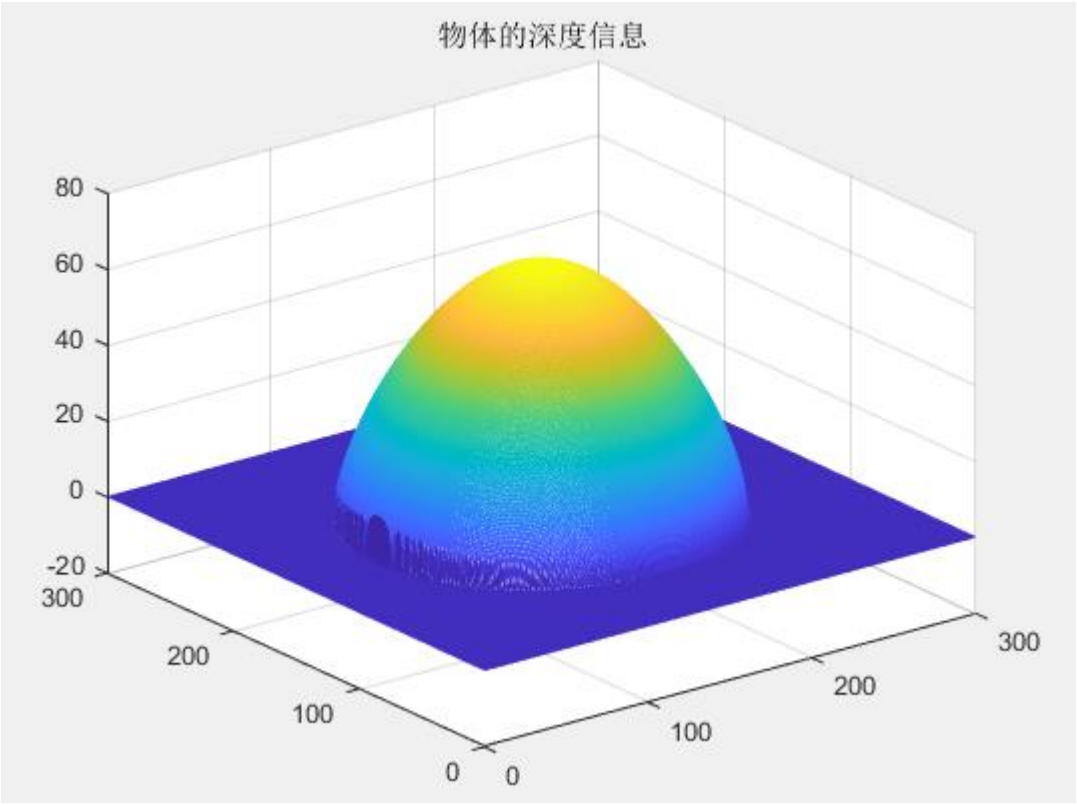


任务三结果：

最终根据高度相位公式重建的球面的三维点云图像：



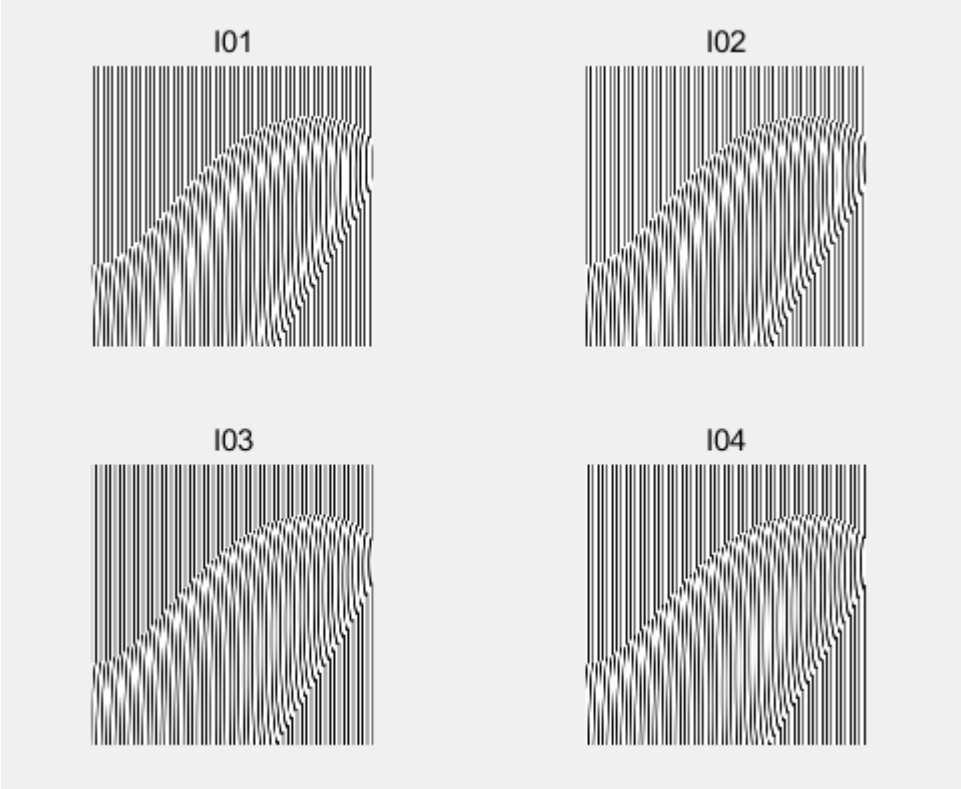
通过 mesh 函数实现最终效果：



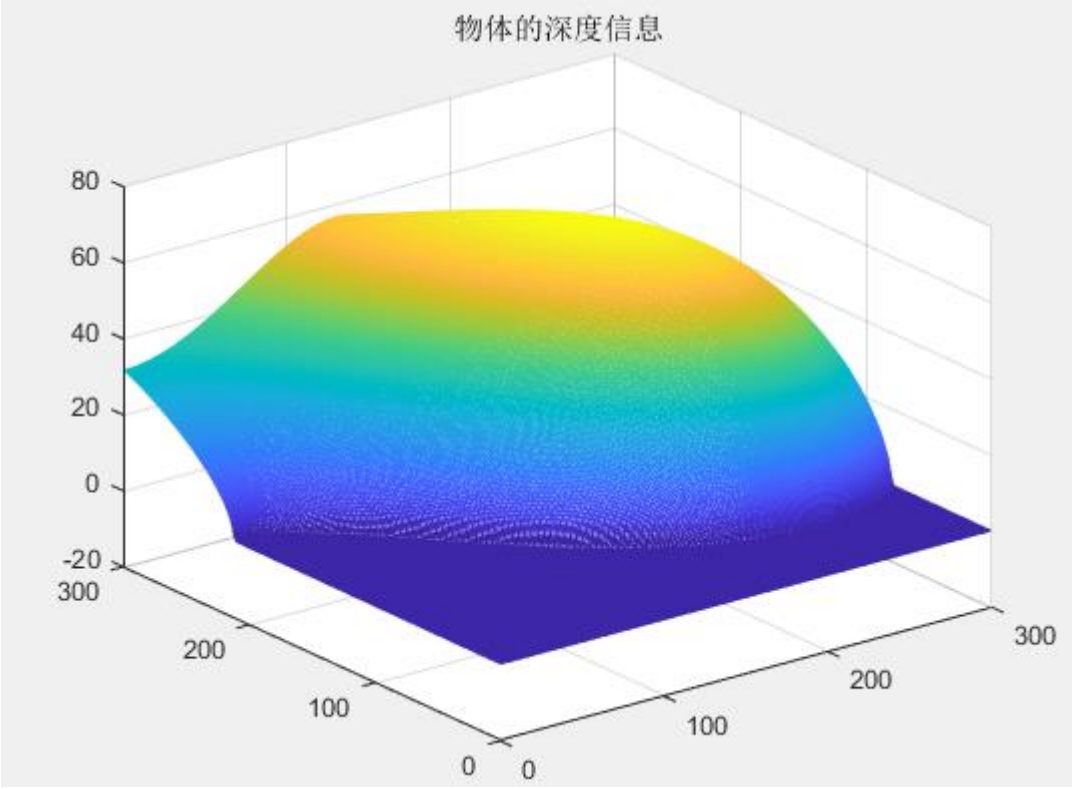
可以看出此时结构光三维投影基本重建出球体的基本形貌，除了边缘部分有一些瑕疵，其余部分效果较好

任务四结果：

相机拍摄到的图像：



带有径向畸变的物体深度信息：



可以看到加入畸变模型后，相机的成像出现了一定程度的扭曲，

对最后的重见效果产生了较大影响，因此对相机进行标定是非常必须的。

附录-源代码

请附以上实现的源代码，包括主程序、模块（函数）程序等，并对代码做相应注释。

```
% ######### 01  参数  #########
w = 300;
h = 300;
f     = 1 / 5;                          %  一个光栅空间频率
p     = 1 / f;                           %  投影光栅的空间周期
len = 100;                               %  相机的镜头中心到参考面的距离
d     = 20;                              %  投影中心到相机中心的距离

% ######### 02  仿真  #########
[I01, I02, I03, I04, I01d, I02d, I03d, I04d] = surface_pattern_simulation_distortion(w, h, f, d, len);

% ######### 03  显示  #########
show_pattern(I01, I02, I03, I04, "原始条纹");
show_pattern(I01d, I02d, I03d, I04d, "调制条纹");

% ######### 04  滤波  #########
[I01, I02, I03, I04] = filter_pattern(I01, I02, I03, I04);
[I01d, I02d, I03d, I04d] = filter_pattern(I01d, I02d, I03d, I04d);

% ######### 05  解相位  #########
phase0 = parse_pattern(I01, I02, I03, I04);
phase1 = parse_pattern(I01d, I02d, I03d, I04d);
%  两个部分相位的相减，得到最后的相位，再从相对相位得到绝对相位
phase = unwrap(phase1 - phase0);
```

```
%  画出绝对相位的分布图
figure; mesh(phase); title('绝对相位的分布图');

% ######### 06  获得高度  #########
H = -p * len * phase./ (p * phase +2 * pi * d);
figure;
mesh(H);
title('物体的深度信息');
hold on;
% x1 = 1:w;
% y1 = 1:h;
% [X1,Y1] = meshgrid(x1,y1);
% X1transform = X1(:)';
% Y1transform = Y1(:)';
% Htransform = H(:)';
% scatter3(X1transform,Y1transform,Htransform,'.','b');
% % plot3(X1transform,Y1transform,Htransform,'.','MarkerSize',0.5,'color',[rand rand
rand])
% %close all; clear; clc;

function [] = show_pattern(I01, I02, I03, I04, suptitle)
%  光栅图片
subplot(2, 2, 1);    imshow(I01); title('I01');
subplot(2, 2, 2);    imshow(I02); title('I02');
subplot(2, 2, 3);    imshow(I03); title('I03');
subplot(2, 2, 4);    imshow(I04); title('I04');
  pause;
end

clc;clear;
CameraCenter = [-64,0,80];
ProjectorCenter = [0,0,80];
SphereCenter = [-32,20,0];
Radius = 30;
```

```
Distance = 6;
Width = 1280;
Height = 1080;
PixelLength = 0.005;
LeftBound = CameraCenter(1) - 0.5*Width*PixelLength;
RightBound = CameraCenter(1) + 0.5*Width*PixelLength;
DownBound = CameraCenter(2) - 0.5*Height*PixelLength;
UpBound = CameraCenter(2) + 0.5*Height*PixelLength;
x1 = LeftBound:0.2:RightBound;
y1 = DownBound:0.2:UpBound;
[X1,Y1] = meshgrid(x1,y1);
Z1 = zeros(28,33);
Z1(:,:) = CameraCenter(3)+Distance;

LeftBound1 = ProjectorCenter(1) - 0.5*Width*PixelLength;
RightBound1 = ProjectorCenter(1) + 0.5*Width*PixelLength;
DownBound1 = ProjectorCenter(2) - 0.5*Height*PixelLength;
UpBound1 = ProjectorCenter(2) + 0.5*Height*PixelLength;
x2 = LeftBound1:0.2:RightBound1;
y2 = DownBound1:0.2:UpBound1;
[X2,Y2] = meshgrid(x2,y2);
Z2 = zeros(28,33);
Z2(:,:) = ProjectorCenter(3)+Distance;

figure;%create a new window
set(gcf,'color','white');
set(gcf,'outerposition',get(0,'screensize'));
x = -42:1:-22;
y = 10:1:30;
[X,Y] = meshgrid(x,y);
Z = surfun(Radius,X,Y);
Xtransform = X(:)';
Ytransform = Y(:)';
Ztransform = Z(:)';
```

```
X1transform = X1(:)';
Y1transform = Y1(:)';
Z1transform = Z1(:)';
X2transform = X2(:)';
Y2transform = Y2(:)';
Z2transform = Z2(:)';
FinalPointsX = [CameraCenter(1),Xtransform];
FinalPointsY = [CameraCenter(2),Ytransform];
FinalPointsZ = [CameraCenter(3),Ztransform];
FinalPointsX1 = [CameraCenter(1),X1transform];
FinalPointsY1 = [CameraCenter(2),Y1transform];
FinalPointsZ1 = [CameraCenter(3),Z1transform];
FinalPointsX2 = [ProjectorCenter(1),X2transform];
FinalPointsY2 = [ProjectorCenter(2),Y2transform];
FinalPointsZ2 = [ProjectorCenter(3),Z2transform];
surf(X,Y,Z);
shading interp;
hold on;
for i = 2:442
    ProcessX = [FinalPointsX(1),FinalPointsX(i)];
    ProcessY = [FinalPointsY(1),FinalPointsY(i)];
    ProcessZ = [FinalPointsZ(1),FinalPointsZ(i)];
    plot3(ProcessX,ProcessY,ProcessZ);
    hold on;
    ProjectorX = [ProjectorCenter(1),FinalPointsX(i)];
    ProjectorY = [ProjectorCenter(2),FinalPointsY(i)];
    ProjectorZ = [ProjectorCenter(3),FinalPointsZ(i)];
    plot3(ProjectorX,ProjectorY,ProjectorZ);
    hold on;
end
hold on;
for j = 1:925
    PixelX = [FinalPointsX1(1),FinalPointsX1(j)];
    PixelY = [FinalPointsY1(1),FinalPointsY1(j)];
```

```
    PixelZ = [FinalPointsZ1(1),FinalPointsZ1(j)];
    plot3(PixelX,PixelY,PixelZ);
    hold on;
end
hold on;
for j = 1:925
    PixelX1 = [FinalPointsX2(1),FinalPointsX2(j)];
    PixelY1 = [FinalPointsY2(1),FinalPointsY2(j)];
    PixelZ1 = [FinalPointsZ2(1),FinalPointsZ2(j)];
    plot3(PixelX1,PixelY1,PixelZ1);
    hold on;
end

function [I01, I02, I03, I04, I01d, I02d, I03d, I04d] =
surface_pattern_simulation_distortion(w, h, f, d, len)

x = 1:w;
y = 1:h;
[X, Y] = meshgrid(x,y);

X1=X*(1+(0.02+0.04*(X.*X+Y.*Y)))/1000000;
Y1=Y*(1+(0.02+0.04*(X.*X+Y.*Y)))/1000000;
Z = 0.4*sqrt(100*100-(X1-150).^2-(Y1-150).^2);
Z(find(angle(Z))) = 0;
figure; mesh(x, y, Z); title('实际物体的三维轮廓');

%--------------第二步，调制相位产生--------------------
var      = 2 * pi * f.* ones(h, 1) * x; %  原始相位
var_d = 2 * pi * f * d.* Z/ len;        %  调制相位

mkdir imgs;
%  原始光栅
I01 = 1 + cos(var);
I02 = 1 + cos(var + pi / 2);
```

```
I03 = 1 + cos(var + pi);
I04 = 1 + cos(var + 3 * pi / 2);
%  变形光栅
I01d = 1 + cos(var + var_d);
I02d = 1 + cos(var + pi/2 + var_d);
I03d = 1 + cos(var + pi + var_d);
I04d = 1 + cos(var + 3 * pi / 2 + var_d);
end


function [phase] = parse_pattern(I01, I02, I03, I04)
[width, height] = size(I01);
%相位矩阵的初始化
phase = zeros(300, 300);
for w = 1: width
    for h = 1: height
        s = abs(atan((I02(w, h) - I04(w, h)) / (I01(w, h) - I03(w, h))));
        if(I02(w, h) >= I04(w, h))          % sin(var) < 0
            if(I01(w, h) <= I03(w, h)) % cos(var) > 0
                phase(w, h) = -s;           %  第四象限
            else
                phase(w, h) = s - pi;    %  第三象限
            end
        else                                        % sin(var) > 0
            if(I01(w, h) <= I03(w, h)) % cos(var) > 0
                phase(w, h) = s;             %  第一象限
            else
                phase(w, h) = pi - s;    %  第二象限
            end
        end
    end
end
end
```

```
function [I01, I02, I03, I04, I01d, I02d, I03d, I04d] = surface_pattern_simulation(w, h,
f, d, len)
x = 1:w;
y = 1:h;
[X, Y] = meshgrid(x, y);
Z = 0.4*sqrt(100*100-(X-150).^2-(Y-150).^2);
Z(find(angle(Z))) = 0;
figure; mesh(x, y, Z); title('实际物体的三维轮廓');

%--------------第二步，调制相位产生--------------------
var      = 2 * pi * f.* ones(h, 1) * x; %  原始相位
var_d = 2 * pi * f * d.* Z / len;        %  调制相位

mkdir imgs;
%  原始光栅
I01 = 1 + cos(var);
I02 = 1 + cos(var + pi / 2);
I03 = 1 + cos(var + pi);
I04 = 1 + cos(var + 3 * pi / 2);
%  变形光栅
I01d = 1 + cos(var + var_d);
I02d = 1 + cos(var + pi/2 + var_d);
I03d = 1 + cos(var + pi + var_d);
I04d = 1 + cos(var + 3 * pi / 2 + var_d);
end
```