上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

# 课

# 程

# 论

# 文

**论文题目：** CT 重建软件报告

**学　　院：** 生物医学工程学院

**专　　业：** 生物医学工程

**姓　　名：** 丁昊妍

**学　　号：** 520021910379

# CT 重建软件报告

# 第一章 软件概述

如果用户运行环境里没有 Matlab，需先安装软件，大小为 1.47G。安装完后或已存在 Matlab 环境，软件只有 2.64MB。

该软件通过与用户交互，实现了图像的平行束和扇形束的滤波反投影重建的展示。

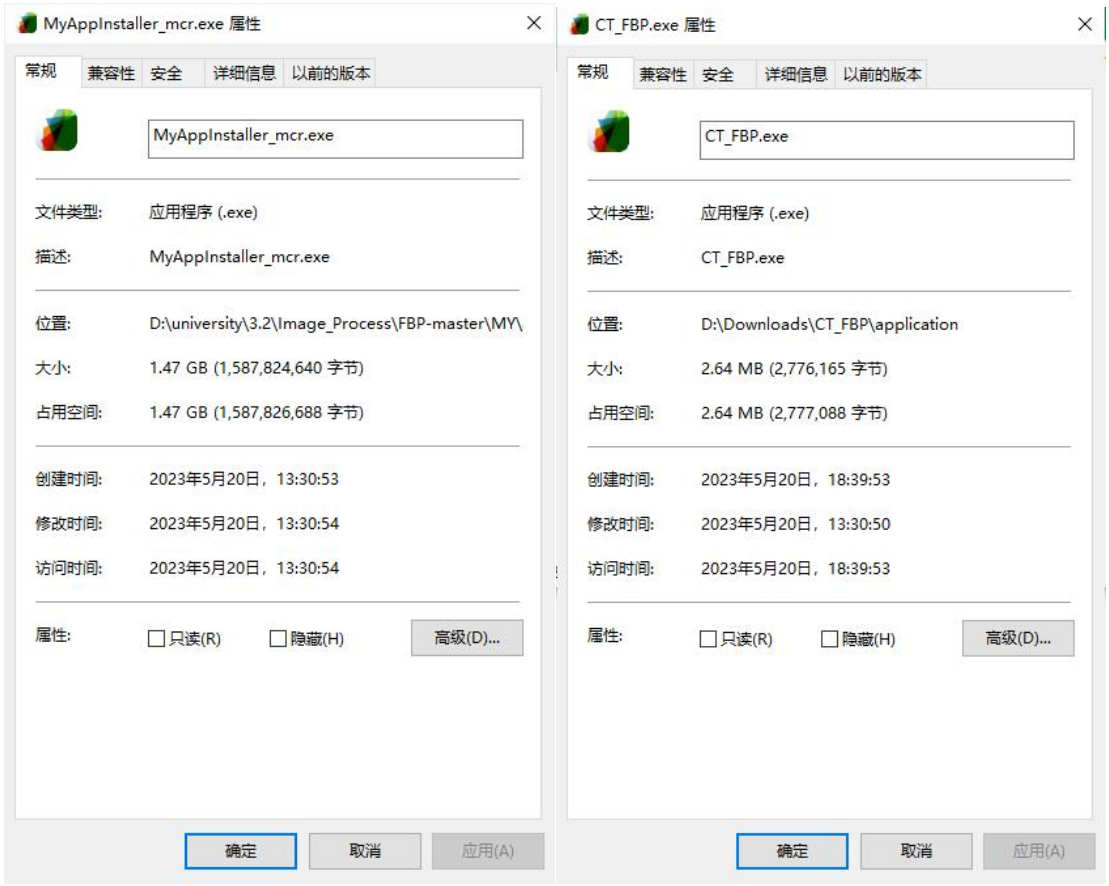



图 1 软件参数

若用户运行环境里有 Matlab，进入 for_redistribution_files_only 文件夹直接点击 CT_FBP.exe 进行运行。并可以直接阅读“运行软件”（2）。


图 2 可执行文件（有 Matlab）

若用户运行环境里没有 Matlab，可按照附录里的的“安装软件”进行安装，使用完后可以按照附录里的“卸载”软件进行卸载。


图 3 可执行文件（无 Matlab）

## 1、运行软件：

（1）用户运行环境里有 Matlab，直接点击“CT_FBP.exe”进行运行。若用户没有 Matlab，按上述步骤安装完软件，打开之前选择安装软件的文件夹，打开“application”文件夹。

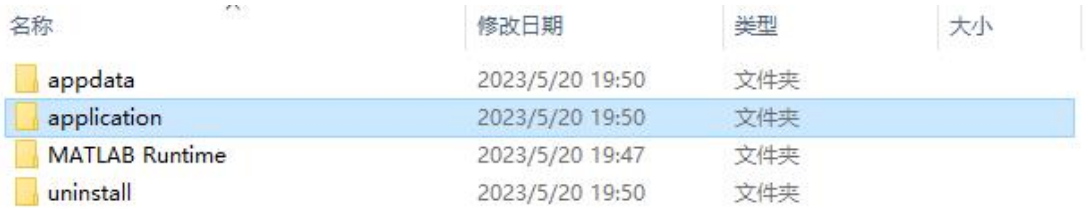



图 4 运行软件所在文件夹

（2）直接点击“CT_FBP.exe”进行运行。

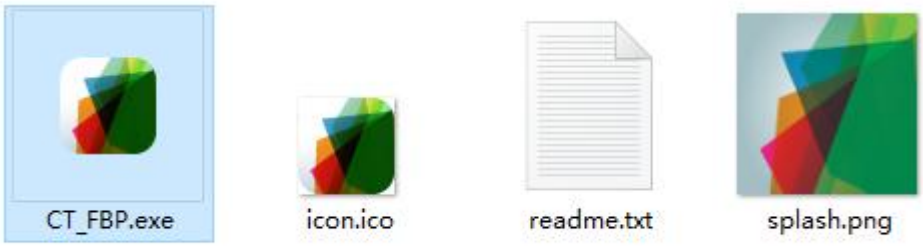



图 5 可执行文件

（3）进入软件内部。

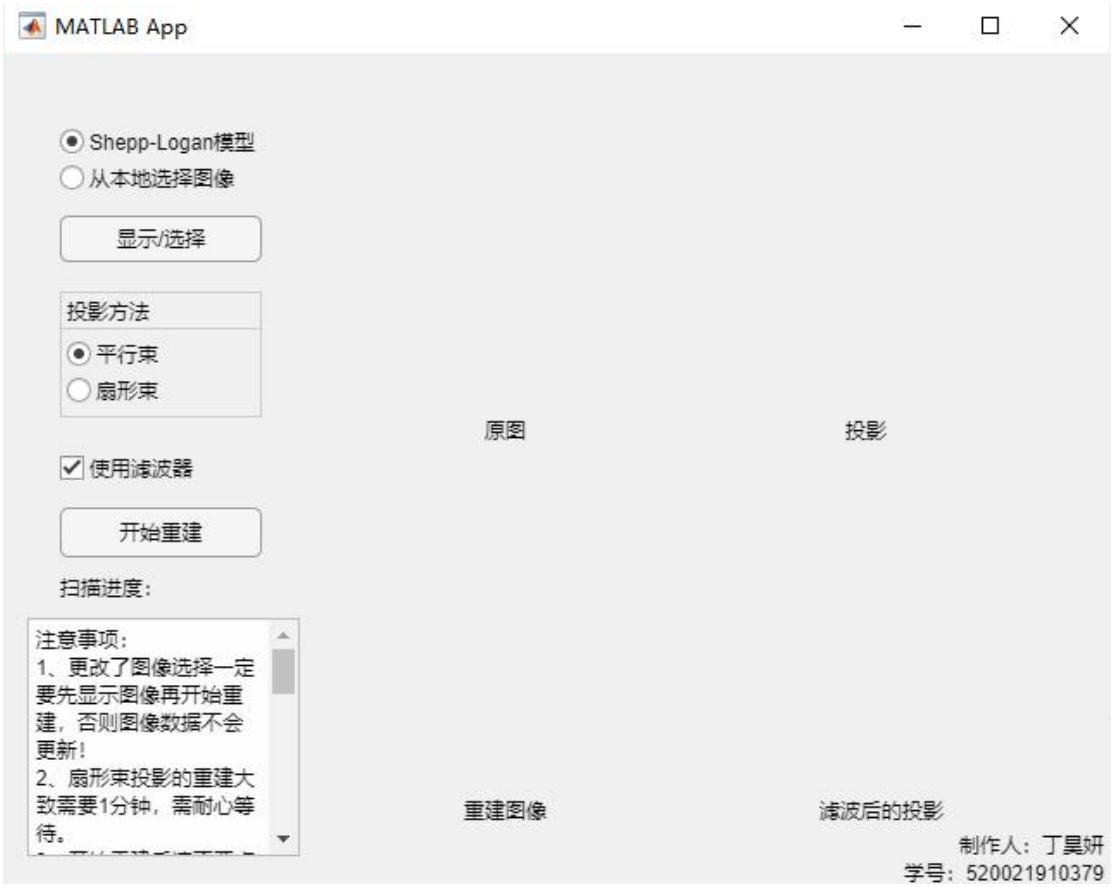



图 6 软件内部

## 2、使用软件：

软件界面展示如下：

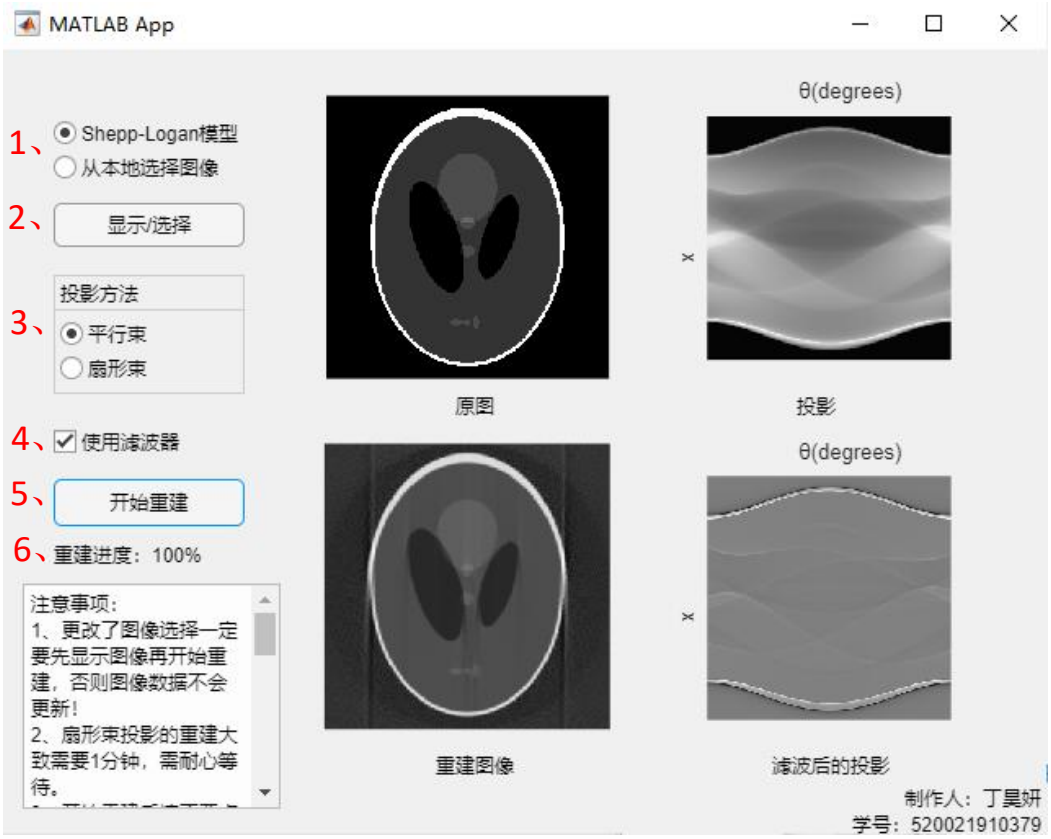



图 7 软件展示（shepp-logan 模型）

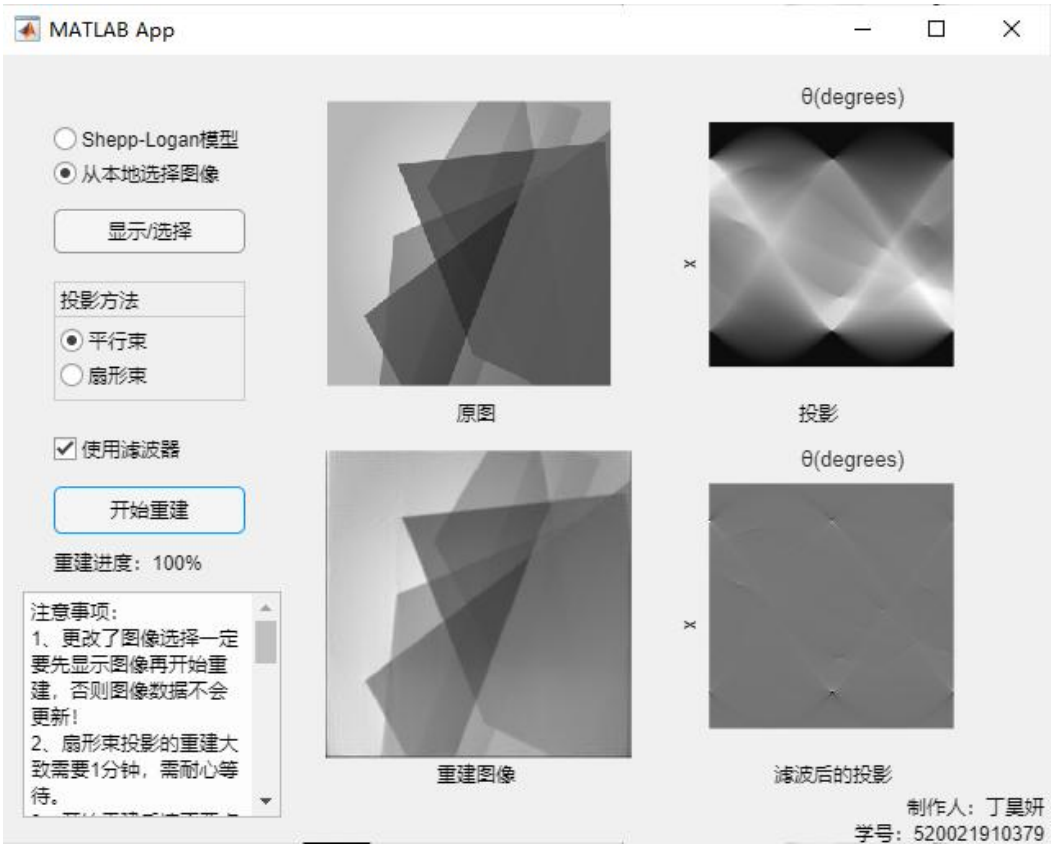



图 8 软件展示（本地图片）

**功能：**

1、可以选择 Shepp-Logan 模型或从本地选择图像（支持*.png,*.jpg 格式）

2、选择完后要点击“显示/选择”按钮将图像显示出来。（注意导入的图像为了方便和重建图像对比，会自动转为黑白显示，并且考虑到重建速度和显示美观，对所有图片的大小都进行了规格统一，一致自动更改为 256×256 的大小。）

2、投影方法可以选择平行束投影或扇形束投影。

3、勾选“使用滤波器”，会展示滤波后的投影，并在滤波后反投影。取消勾选，展示直接反投影的重建图像。

4、点击“开始重建”会进行投影展示和重建展示。

5、会实时显示扫描进度和重建进度。当扫描进度到达 100%会显示投影图像，当重建进度到达 100%会显示重建图像。

**建议使用/测试步骤：**

1、选择 Shepp-Logan 模型或从本地选择图像。

2、点击“显示/选择”按钮，若是 Shepp-Logan 模型，则会在“原图”上方位置直接显示出 Shepp-Logan 模型，若是从本地选择图片，会进入系统，可以挑选图片，选择完成后会在“原图”上方位置显示出选择的图片的黑白形式。

3、选择投影方法（默认为平行束投影）

4、选择是否勾选“使用滤波器”（默认勾选）。

5、点击“开始重建”，此时开始重建。当扫描和重建完成后会显示图像。

6、完成后可以更换图像，或更换投影方法，或改变滤波器勾选重新开始重建。

**注意事项：**

1、更改了图像选择一定要先显示图像再开始重建，否则图像数据不会更新。

2、扇形束投影的重建大致需要 1 分钟，需耐心等待。

3、开始重建后尽量不要点击其他按钮，否则可能影响结果。

# 第二章 代码架构

## 1、文件架构

app 文件夹包含集成的 App，使用步骤如上。CT_FBP(with GUI)文件夹是含 GUI 界面的代码，即 App 的源代码。CT_FBP(without GUI)文件夹是不包含 GUI 只有功能实现的代码。

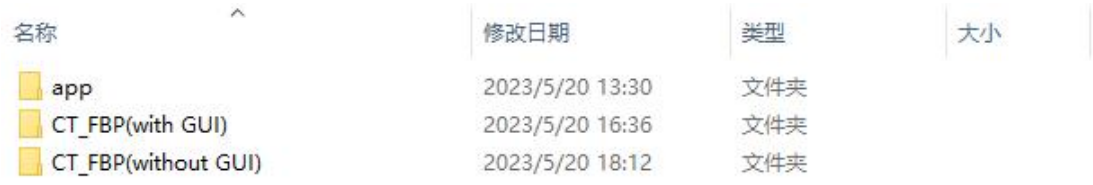



图 9 文件架构

整体使用 MATLAB R2020b 进行功能实现，使用 MATLAB 自带的 APP Designer 进行 GUI 界面设计。

开发界面展示如下：

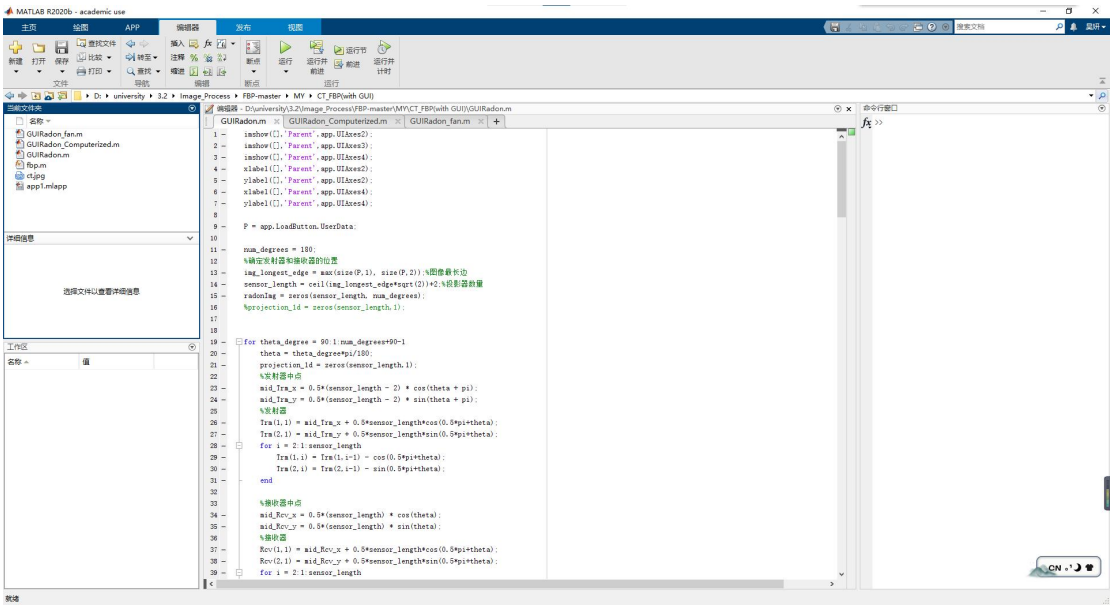


图 10 Matlab 开发界面

APP Designer 界面，可以点击“代码视图”查看代码，也可以直接点击“运行”进行使用。

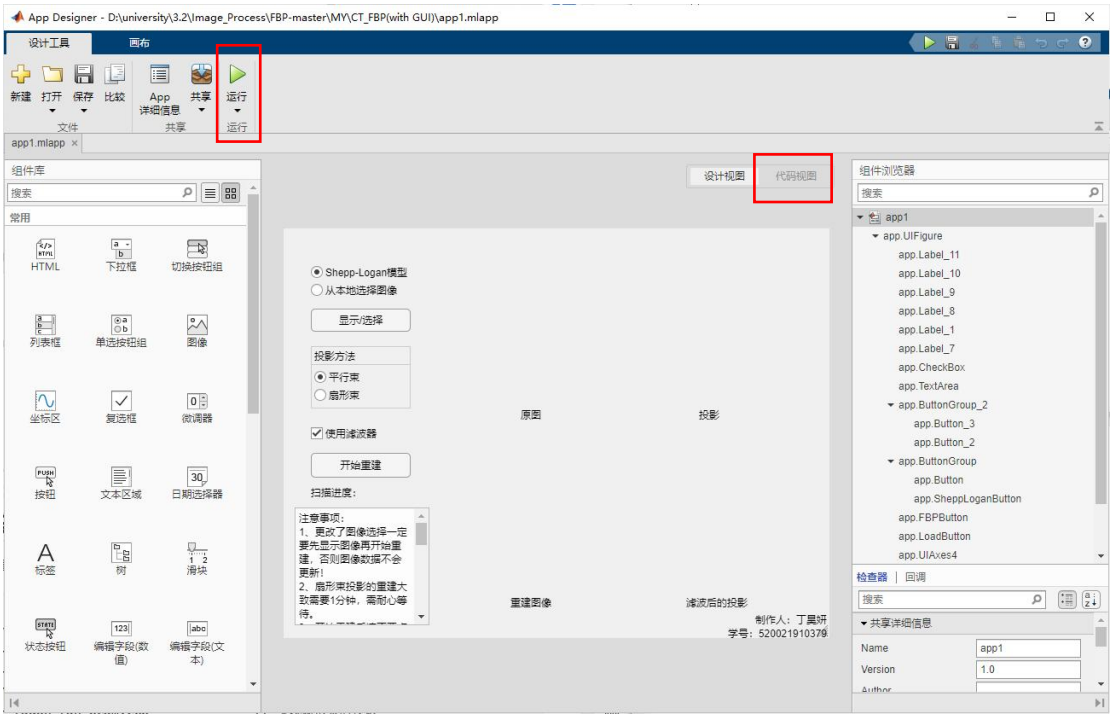



图 11 APP Designer 开发界面

## 2、CT_FBP(with GUI)

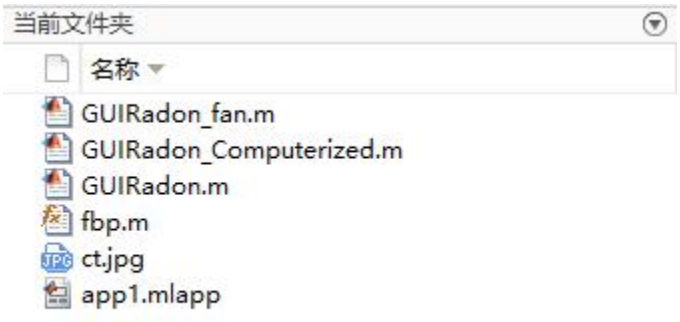



图 12 CT_FBP(with GUI)代码框架

app1.mlapp 是 GUI 界面开发的代码。针对平行束：GUIRadon_Computerized.m 实现解析法投影，GUIRadon.m 实现通用的数字积分法投影，fbp.m 实现滤波和反投影。GUIRadon_fan.m 实现扇形束的投影和滤波反投影。ct.jpg 可以用来测试一般图像的投影和重建。

### 3、CT_FBP(without GUI)

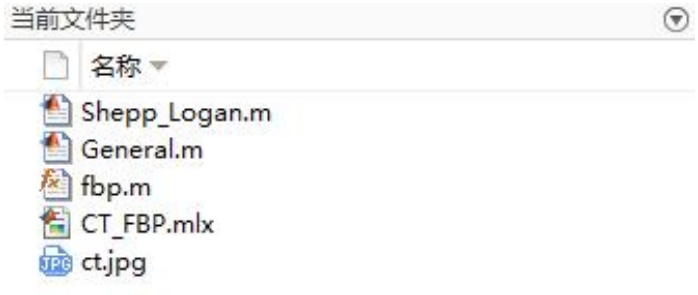



图 13 CT_FBP(without GUI)代码框架

Shepp_Logan.m 生成 Shepp_Logan 模型并实现解析法投影，随后调用 fbp.m 函数进行滤波和反投影，并进行扇形束的投影和滤波反投影。General.m 读取一般图像（默认为文件夹自带的 ct.jpg）后实现数字积分法投影，随后调用 fbp.m 函数进行滤波和反投影，并进行扇形束的投影和滤波反投影。

CT_FBP.mlx 是实时脚本文件，包含上述功能的**代码以及运行结果**，可以直接查看。

# 第三章 算法介绍

# 1、读入图像

## 1.1 Shepp-Logan 模型图

相关代码在 CT_FBP(with GUI)文件夹中的 app1.mlapp，在 CT_FBP(without GUI)文件夹中的 Shepp_Logan.m。

通过 Matlab 自带的 phantom 函数生成 Shepp-Logan 模型图。相关代码如下：

```
clc;clear;
M=256;
[P, E] = phantom('Modified Shepp-Logan', M); % 创建一个 shepp-logan 模型，原图像
figure
imshow(P) %原图像
title('原图');
```

生成的尺寸为 256*256 的 Shepp-Logan 模型如下图所示：


图 14 Shepp-Logan 模型图

## 1.2 一般图像

相关代码在 CT_FBP(with GUI)文件夹中的 GUIRadon_Computerized.m，在 CT_FBP(without GUI)文件夹中的 General.m。

因为考虑到后续投影和重建速度（像素点越多投影和重建越慢），以及在 GUI 界面中显示的美观统一性，所以将所有输入图像都转位 256*256 的格式。同时因为投影值用的是灰度值，重建的是黑白图片，所以在输入时要进行转换，将 rgb 值转换为灰度值，而为了和重建图片对比，所以最开始显示原图的时候，显示的是转换后的图像。相关代码如下：

```
[filename,filepath] = uigetfile({'*.png;*.jpg'}, 'Select File to Open');
fullname = [filepath, filename];
if fullname ~= 0                    %#ok<BDSCI>
    P = imread(fullname);
    P = imresize(P, [256,256]);
    P = rgb2gray(P);
    P = im2double(P); %转换成double
    imshow(P,'Parent',app.UIAxes);
    app.LoadButton.UserData = P;
end
```

```
M=256;
%显示图像，选择imtool工具可有效调节窗宽窗位
P = imread('ct.jpg');%使用默认图片进行测试，可以更换其他图片
P = imresize(P, [M,M]);
P = rgb2gray(P);
P = im2double(P); %转换成double
figure
imshow(P);
title('原图');
```

读入的图像（使用提供的默认图片）显示如下：

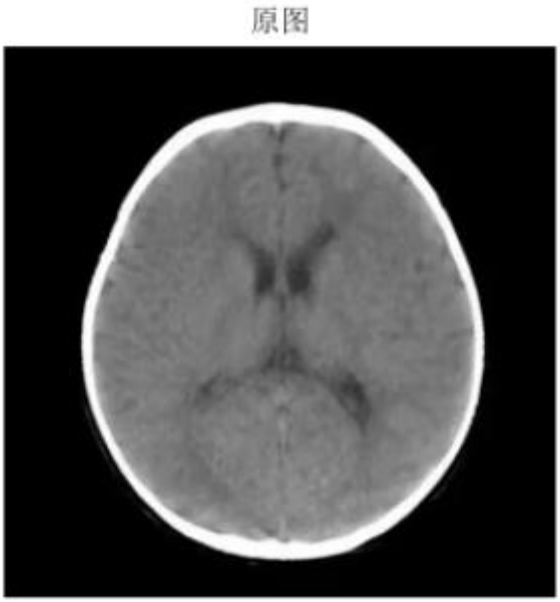



图 15 本地图片

# 2、投影

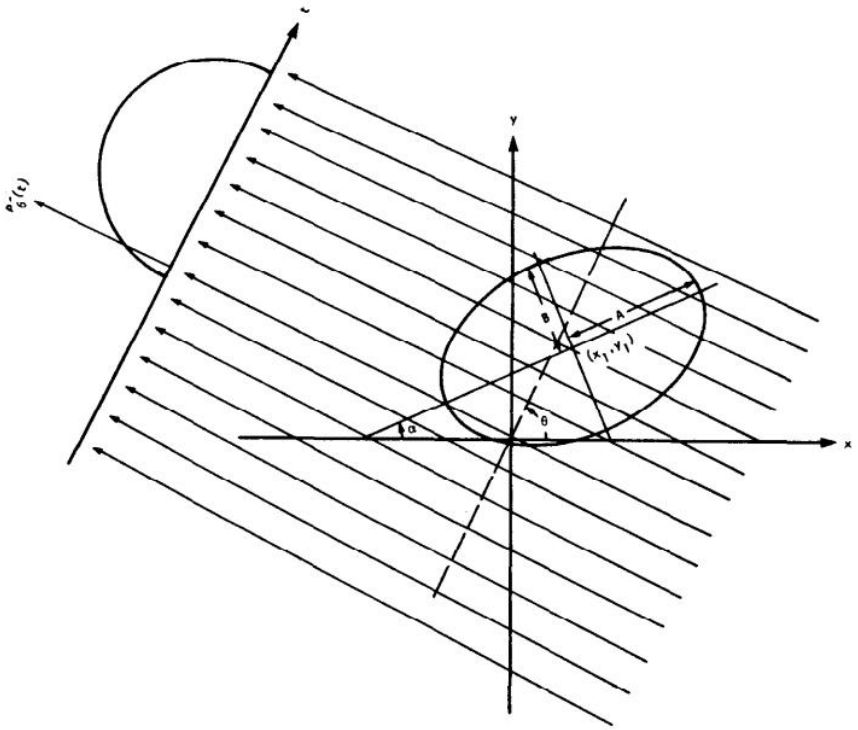
## 2.1 平行束

### 2.1.1 解析法投影

解析法投影的公式（椭圆投影公式）主要参考：Avinash C. Kak, and Malcolm Slaney,“Principles of Computerized Tomographic Imaging”, IEEE Press, 1999.
https://engineering.purdue.edu/~malcolm/pct/， Chapter 3. Algorithms for Reconstruction with Nondiffracting Sources.
相关代码在 CT_FBP(with GUI)文件夹中的 GUIRadon_Computerized.m，在 CT_FBP(without GUI)文件夹中的 Shepp_Logan.m。

根据上述参考书中的公式，可以算出椭圆各个角度的投影值。公式如下：
对于标准椭圆：

$$f(x, y)=\begin{cases}\rho & \text{for } \dfrac{x^2}{A^2}+\dfrac{y^2}{B^2}\le 1 \quad \text{(inside the ellipse)}\\ 0 & \text{otherwise} \qquad\quad \text{(outside the ellipse).}\end{cases}$$



It is easy to show that the projections of such a function are given by

$$P_\theta(t)=\begin{cases}\dfrac{2\rho AB}{a^2(\theta)}\sqrt{a^2(\theta)-t^2} & \text{for } |t|\le a(\theta)\\ 0 & |t|>a(\theta)\end{cases} \tag{5}$$

where $a^2(\theta) = A^2\cos^2\theta + B^2\sin^2\theta$. Note that $a(\theta)$ is equal to the projection half-width as shown in Fig. 3.5(a).

Now consider the ellipse described above centered at $(x_1, y_1)$ and rotated by an angle $\alpha$ as shown in Fig. 3.5(b). Let $P'(\theta, t)$ be the resulting projections. They are related to $P_\theta(t)$ in (5) by

$$P_\theta(t)=P_{\theta-\alpha}(t-s\cos(\gamma-\theta)) \tag{6}$$

where $s = \sqrt{x_1^2 + y_1^2}$ and $\gamma = \tan^{-1}(y_1/x_1)$.

图 16 解析投影法的参考资料

因此，计算 Shepp_Logan 模型的投影的自定义函数核心代码如下：

```
randon_row=M;
radonImg_Computerized=double(zeros(randon_row,180));
for i = 1:10 %计算每个椭圆
    rou = E(i,1);%灰度值
    a = E(i,2)*M/2;%椭圆横轴
    b = E(i,3)*M/2;%椭圆竖轴
    m = E(i,4)*M/2;%椭圆中心x
    n = E(i,5)*M/2;%椭圆中心y
    alpha = E(i,6)*pi/180;%椭圆横轴和x轴夹角
    for theta_degree =0:1:179 %计算每个角度
        theta = theta_degree*pi/180;
        a_theta_2 = a*a*((cos(theta-alpha))^2) + b*b*((sin(theta-alpha))^2);
        s = sqrt(m*m +n*n);
        if n == 0
            if m >= 0
                gamma =0;
            else
                gamma = pi;
            end
        else
            if m < 0
                gamma = atan(n/m) + pi;%二三象限
            else
                gamma = atan(n/m);%一四象限
            end
        end

        for j = 0:1:randon_row-1 %计算每个P
            t0 = j-0.5*randon_row;
            t = t0 -s*cos(gamma-theta);
            if t*t <= a_theta_2
                radonImg_Computerized(j+1,theta_degree+1) = radonImg_Computerized(j+1,theta_degree+1)...
                    + ((2*255*rou*a*b)/a_theta_2)*sqrt(a_theta_2 - t*t);
            end
        end
    end
end
```

解析法生成的投影图如下，为了比例好看，调整为也是 256*256 的格式。

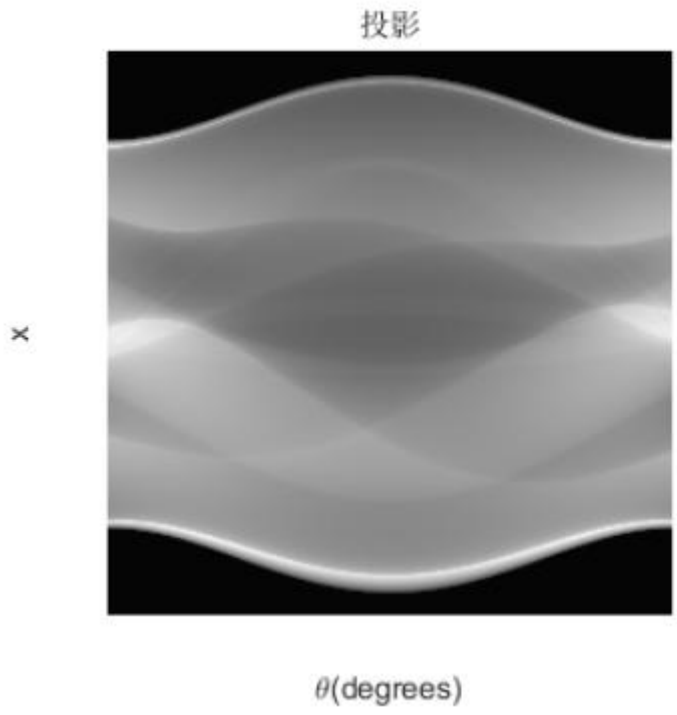



图 17 解析法投影图

## 2.1.2 数字积分法投影

数字积分法投影主要参考文献：Siddon, Robert L. "Fast calculation of the exact radiological path for a three-dimensional CT array." Medical physics 12.2 (1985): 252-255.
相关代码在 CT_FBP(with GUI)文件夹中的 GUIRadon.m，在 CT_FBP(without GUI)文件夹中的 General.m。

## 思路

数字积分法的总思路为：

- 把二维图像看作水平线和竖直线组成的网格。
- 遍历所有角度，求每个角度下的所有发射器和接收器。
- 在确定好发射器和接收器后，发射器与接收器的连线和网格会形成一系列交点。
- 定义$\alpha = \frac{\text{发射器到交点的距离}}{\text{发射器到接收器的距离}}$，其中发射器到交点的距离通过 x 方向和 y 方向分别计算。与竖线相交形成的交点放入$\{\alpha_x\}$，与横线相交的交点放入$\{\alpha_y\}$，最后融合成一个集合后进行升序排列。
- 两个相邻的 α 值对应了射线穿过的某个像素格的两个边，所以根据相邻的 α 值的平均值，计算穿过某个像素格的路径长度——比例*总长度，以此确定像素格的位置，由此求得像素格对应的灰度值。
- 最后投影值= ∑（各个像素格的路径长度*密度值）

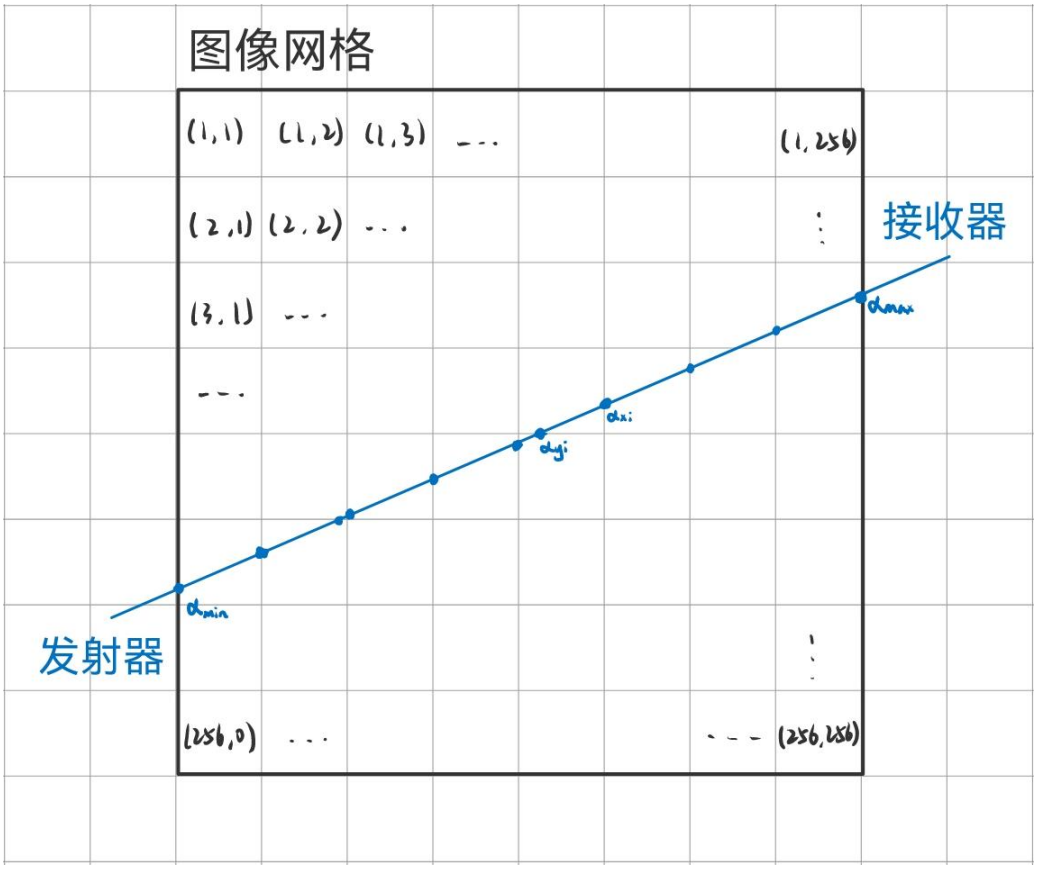



图 18 数字积分法思路

## 确定发射器和接收器的位置

为了使发射器和接收器的长度覆盖到整张图片，考虑 45° 和 135° 投影时数据的长度达到的最大值。

先根据角度确定当前发射器和接收器的中点位置，再确定该角度下的所有发射器和接收器。相关代码如下：

```
theta = theta_degree*pi/180;
projection_1d = zeros(sensor_length,1);
%发射器中点
mid_Trm_x = 0.5*(sensor_length - 2) * cos(theta + pi);
mid_Trm_y = 0.5*(sensor_length - 2) * sin(theta + pi);
%发射器
Trm(1,1) = mid_Trm_x + 0.5*sensor_length*cos(0.5*pi+theta);
Trm(2,1) = mid_Trm_y + 0.5*sensor_length*sin(0.5*pi+theta);
for i = 2:1:sensor_length
    Trm(1,i) = Trm(1,i-1) - cos(0.5*pi+theta);
    Trm(2,i) = Trm(2,i-1) - sin(0.5*pi+theta);
end

%接收器中点
mid_Rcv_x = 0.5*(sensor_length) * cos(theta);
mid_Rcv_y = 0.5*(sensor_length) * sin(theta);
%接收器
Rcv(1,1) = mid_Rcv_x + 0.5*sensor_length*cos(0.5*pi+theta);
Rcv(2,1) = mid_Rcv_y + 0.5*sensor_length*sin(0.5*pi+theta);
for i = 2:1:sensor_length
    Rcv(1,i) = Rcv(1,i-1)-cos(0.5*pi+theta);
    Rcv(2,i) = Rcv(2,i-1)-sin(0.5*pi+theta);
end
```

## 确定 α 范围

把 x-y 直角坐标系的原点放在图像中心，则图像范围为：x ∈（-0.5*图像原宽，0.5*图像原宽），y∈（-0.5*图像原长，0.5*图像原长）。

注意计算范围时候要考虑没有交点的情况，在代码中我用变量 flag_intersection 来记录这种情况。

α 范围公式如下：

$$\alpha_{\min} = \max\{0, \min[\alpha_x(1), \alpha_x(N_x)], \min[\alpha_y(1), \alpha_y(N_y)], \min[\alpha_z(1), \alpha_z(N_z)]\},$$

$$\alpha_{\max} = \min\{1, \max[\alpha_x(1), \alpha_x(N_x)], \max[\alpha_y(1), \alpha_y(N_y)], \max[\alpha_z(1), \alpha_z(N_z)]\},$$

相关代码封装在了 Caculate_range_alpha(i,Trm,Rcv,P)函数中，如下：

```
function [min_alpha, max_alpha, flag_intersection]  = Caculate_range_alpha(i,Trm,Rcv,P)%计算alpha范围
    %确定图像位置
    P_col = size(P,2);
    P_row = size(P,1);
    img_x_min = -0.5*P_col;
    img_y_min = -0.5*P_row;
    img_x_max = img_x_min + P_col;
    img_y_max = img_x_min + P_row;
    min_alpha_x = 1.0;
    min_alpha_y = 1.0;
    max_alpha_x = 0.0;
    max_alpha_y = 0.0;
    if (Rcv(1,i)-Trm(1,i))~=0 %不竖直
        min_alpha_x = min((img_x_min - Trm(1,i)) / (Rcv(1,i) - Trm(1,i)),...
            (img_x_max - Trm(1,i)) / (Rcv(1,i) - Trm(1,i)));
        max_alpha_x = max((img_x_min - Trm(1,i)) / (Rcv(1,i) - Trm(1,i)),...
            (img_x_max - Trm(1,i)) / (Rcv(1,i) - Trm(1,i)));
    end
    if ((Rcv(2,i) - Trm(2,i)) ~= 0)%不平行
        min_alpha_y = min((img_y_min - Trm(2,i)) / (Rcv(2,i) - Trm(2,i)),...
            (img_y_max - Trm(2,i)) / (Rcv(2,i) - Trm(2,i)));
        max_alpha_y = max((img_y_min - Trm(2,i)) / (Rcv(2,i) - Trm(2,i)),...
            (img_y_max - Trm(2,i)) / (Rcv(2,i) - Trm(2,i)));
    end

    min_alpha = max([min_alpha_x, min_alpha_y, 0.0]);
    max_alpha = min([max_alpha_x, max_alpha_y, 1.0]);

    flag_intersection = 1;
    if(min_alpha >= max_alpha)
        flag_intersection = 0;
    end
end
```

## 确定索引 m，n 范围

m，n 范围的原理公式（以 n 为例）如下：

$(X_2 - X_1) \geqslant 0$：

$$n_{min} = N_x - \frac{X_{plane}(N_x) - \alpha_{min}(X_2 - X_1) - X_1}{d_x}，\ n_{max} = -1 + \frac{X_1 + \alpha_{max}(X_2 - X_1) - X_{plane}(1)}{d_x}$$

$(X_2 - X_1) \leqslant 0$：

$$n_{min} = N_x - \frac{X_{plane}(N_x) - \alpha_{max}(X_2 - X_1) - X_1}{d_x}，\ n_{max} = -1 + \frac{X_1 + \alpha_{min}(X_2 - X_1) - X_{plane}(1)}{d_x}$$

相关代码封装在了 Calculate_range_index_image(i, Trm, Rcv, P, min_alpha, max_alpha)函数中，如下：

```
function [intersection_n_min, intersection_n_max, intersection_m_min, intersection_m_max] = ...
    Calculate_range_index_image(i, Trm, Rcv, P, min_alpha, max_alpha)
    %确定图像位置
    P_col = size(P,2);
    P_row = size(P,1);
    img_x_min = -0.5*P_col;
    img_y_min = -0.5*P_row;
    img_x_max = img_x_min + P_col;
    img_y_max = img_x_min + P_row;

    if Trm(1,i) < Rcv(1,i)
        intersection_n_min = (P_col) - (img_x_max - min_alpha * (Rcv(1,i) - Trm(1,i)) - Trm(1,i));
        intersection_n_max = -1 + (Trm(1,i) + max_alpha * (Rcv(1,i) - Trm(1,i)) - img_x_min);
    else
        intersection_n_min = (P_col) - (img_x_max - max_alpha * (Rcv(1,i) - Trm(1,i)) - Trm(1,i));
        intersection_n_max = -1 + (Trm(1,i) + min_alpha * (Rcv(1,i) - Trm(1,i)) - img_x_min);
    end

    %intersection_n_min = floor(intersection_n_min);
    %intersection_n_max = ceil(intersection_n_max);

    if Trm(2,i) < Rcv(2,i)
        intersection_m_min = (P_col) - (Trm(2,i) + max_alpha * (Rcv(2,i) - Trm(2,i)) - img_y_min);
        intersection_m_max = img_y_max - min_alpha * (Rcv(2,i) - Trm(2,i)) - Trm(2,i) - 1;
    else
        intersection_m_min = (P_col) - (Trm(2,i) + min_alpha * (Rcv(2,i) - Trm(2,i)) - img_y_min);
        intersection_m_max = img_y_max - max_alpha * (Rcv(2,i) - Trm(2,i)) - Trm(2,i) - 1;
    end

    %intersection_m_min = floor(intersection_m_min);
    %intersection_m_max = ceil(intersection_m_max);

end
```

## 确定 α 元素

根据已知的索引范围，对符合索引范围内的整数进行遍历，就获得了对应范围内所有的 α 值。把两个方向内的 α 合并起来，升序排列，就可以获得最终的集合。要注意的是当探测器和接收器的连线方向和 x 轴或 y 轴平行时，要特殊分类。

相关代码如下：

```
%求解alpha_x, alpha_y
if mod(theta_degree, 90)~=0
    %求alpha_x
    sizeof_alpha_x = round(intersection_n_max-intersection_n_min+1);
    alpha_x = zeros(sizeof_alpha_x,1);
    for k = 0:1:sizeof_alpha_x-1
        x_position = intersection_n_min + k - size(P,2)/2;
        alpha_x(k+1) = (x_position -Trm(1,i))/(Rcv(1,i)-Trm(1,i));
    end
    %求alpha_y
    sizeof_alpha_y = round(intersection_m_max-intersection_m_min+1);
    alpha_y = zeros(sizeof_alpha_y,1);
    for k = 0:1:sizeof_alpha_y-1
        y_position = size(P,1) / 2 - (intersection_m_min + k);
        alpha_y(k+1) = (y_position -Trm(2,i))/(Rcv(2,i)-Trm(2,i));
    end
    %合并
    sizeof_alpha = sizeof_alpha_x + sizeof_alpha_y;
    alpha = [alpha_x; alpha_y];
    alpha = sort(alpha);%, alpha + sizeof_alpha);%升序排序
elseif mod(theta_degree, 180) == 0%平行x轴
    %求alpha_x
    sizeof_alpha_x = round(intersection_n_max-intersection_n_min+1);
    alpha_x = zeros(sizeof_alpha_x,1);
    for k = 0:1:sizeof_alpha_x-1
        x_position = intersection_n_min + k - size(P,2)/2;
        alpha_x(k+1) = (x_position -Trm(1,i))/(Rcv(1,i)-Trm(1,i));
    end
    %求alpha
    sizeof_alpha = sizeof_alpha_x;
    alpha = alpha_x;
    alpha = sort(alpha);%, alpha + sizeof_alpha);
else
    %求alpha_y
    sizeof_alpha_y = round(intersection_m_max-intersection_m_min+1);
    alpha_y = zeros(sizeof_alpha_y,1);
    for k = 0:1:sizeof_alpha_y-1
        y_position = size(P,1) / 2 - (intersection_m_min + k);
        alpha_y(k+1) = (y_position -Trm(2,i))/(Rcv(2,i)-Trm(2,i));
    end
    sizeof_alpha = sizeof_alpha_y;
    alpha = alpha_y;
    alpha = sort(alpha);%, alpha + sizeof_alpha);
end

projection_1d(i) = Calculate_projection_oneValue(i, Trm, Rcv, P, sizeof_alpha, alpha);
```

## 计算投影值

- 取集合{α}内相邻元素相减，乘上探测器和接收器之间的长度，即可计算出这一段路径长度
- 对应的像素标号根据中间位置$\alpha=(1/2)*(\alpha_m+\alpha_{m-1})$计算得到。
- 投影值=∑（各个像素格的路径长度*密度值）

相关代码封装在了 Calculate_projection_oneValue(i, Trm, Rcv, P, sizeof_alpha, alpha)函数中，如下：

```
function projection = Calculate_projection_oneValue(i, Trm, Rcv, P, sizeof_alpha, alpha)
    total_TR_length = sqrt((Trm(1,i)-Rcv(1,i))^2+(Trm(2,i)-Rcv(2,i))^2);
    projection = 0.0;
    P_col = size(P,2);
    P_row = size(P,1);
    for k=2:1:sizeof_alpha
        length_little_route = (alpha(k)-alpha(k-1))*total_TR_length;
        mid_alpha = 0.5*(alpha(k)+alpha(k-1));
        M_pixel(1) = Trm(1,i) + mid_alpha*(Rcv(1,i)-Trm(1,i));
        M_pixel(2) = Trm(2,i) + mid_alpha*(Rcv(2,i)-Trm(2,i));

        n_pixel = floor(M_pixel(1) + 0.5 * P_col);
        n_pixel = max(0, n_pixel);
        n_pixel = min(n_pixel, P_col - 1);

        m_pixel = floor(0.5 * P_row - M_pixel(2));
        m_pixel = max(0, m_pixel);
        m_pixel = min(m_pixel, P_row - 1);

        grayscale = P(m_pixel+1, n_pixel+1);
        projection = projection + length_little_route * grayscale;
    end
end
```

## 2.2 扇形束

扇形束的思路（椭圆投影公式）主要参考：Avinash C. Kak, and Malcolm Slaney, “Principles of Computerized Tomographic Imaging”, IEEE Press, 1999.
https://engineering.purdue.edu/~malcolm/pct/， Chapter 3. Algorithms for Reconstruction with Nondiffracting Sources.
相关代码在 CT_FBP(with GUI)文件夹中的 GUIRadon_fan.m，在 CT_FBP(without GUI)文件夹中的 Shepp_Logan.m 和 General.m。

模型如下图，s 是发射器，D2 是接收器，O 是物体中心，物体中心 O 到发射器 s 的距离为 D。各接收器与发射器相交的直线在过 D'2 线上的各点也是等距的，发射器与 y 轴的夹角为β，发射器过物体中心的线与扇形射线的夹角为γ， θ 是射线垂线与 x 轴的夹角，t 是射线垂线与原点的距离（求 θ 和 t 是为了将射线写成 xcos θ +ysin θ =t 的形式）。

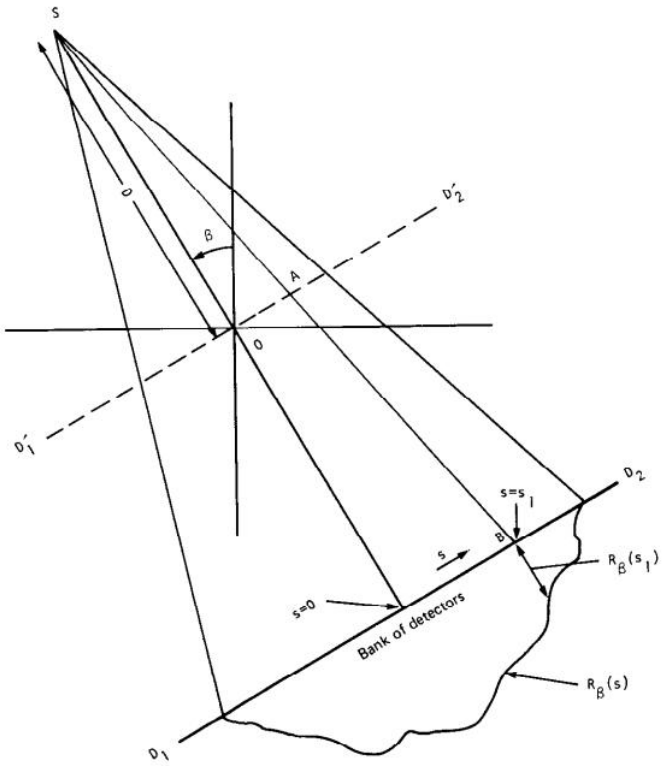



图 19 扇形束投影模型

根据书中的公式可以得到：

$$t = s\cos\gamma \qquad \theta = \beta + \gamma$$

$$t = \frac{sD}{\sqrt{D^2 + s^2}} \qquad \theta = \beta + \tan^{-1}\frac{s}{D}$$

根据 t 即可算出像素点的位置，再通过线性差值即可获得所需所有的像素点。

相关代码封装在了 Projection(img, beta,distance,rayNum,dS)如下：

```
function proj = Projection(img, beta,distance,rayNum,dS)%正投影过程
    proj = zeros(rayNum, length(beta));
    [M,~] = size(img);
    M2 = (M-1)/2;
    [X, Y] = meshgrid(-M2 : 1 : M2); %图片坐标系重定位，以图像中心为(0,0)
    ray_x = -(rayNum-1)/2 * dS:dS:(rayNum-1)/2 * dS;%各光线在物体中心坐标系中与射线源中心线垂直线上的各值，dS为各探测器在该线上的间隔
    CY = -M2*sqrt(2):1:M2*sqrt(2);%垂直x轴线采样点的y,图像为正方形且采样间隔为1，因此设采样点数为根号2*M+1
    CX = zeros([1,length(CY)]);%垂直x轴线采样点的x
    for i = beta%360角全投影,默认逆时针旋转
        proj_1 = zeros(rayNum,1);%一个角度的投影
        r = deg2rad(i);%角度转弧度
        CX_1 = CX.*cos(-r)+CY.*sin(-r);
        CY_1 = CY.*cos(-r)-CX.*sin(-r);%射线源旋转i度后采样点跟着旋转
        for j = 1:length(ray_x)%每个角度有ray_x根线
            gamma = atan(ray_x(j)/distance);%gamma角，即射线与穿过源射线的夹角
            theta = r+gamma;%cos(theta)*x+sin(theta)*y=t中的theta
            t = (distance*ray_x(j))/sqrt(distance*distance+ray_x(j)*ray_x(j));%cos(theta)*x+sin(theta)*y=t中的t
            bias_x = t*cos(theta);%扇形束采样点的x偏置
            bias_y = t*sin(theta);%扇形束采样点的y偏置
            CX_real = CX_1.*cos(-gamma)+CY_1.*sin(-gamma)+bias_x;
            CY_real = CY_1.*cos(-gamma)-CX_1.*sin(-gamma)+bias_y;%采样点旋转gamma角并加上偏置得到射线采样点
            inter = interp2(X, Y, img, CX_real, CY_real, 'linear',0)';%线性插值得到各采样点的值,如果可以最好不要使用interp函数，很慢
            proj_1(j) = sum(inter,1);
        end
        proj(:,i+1)=proj_1;
        %str = ['扫描进度: ',num2str(round(i/359*100)),'%'];
        %disp(str)
    end
end
```

投影图如下：

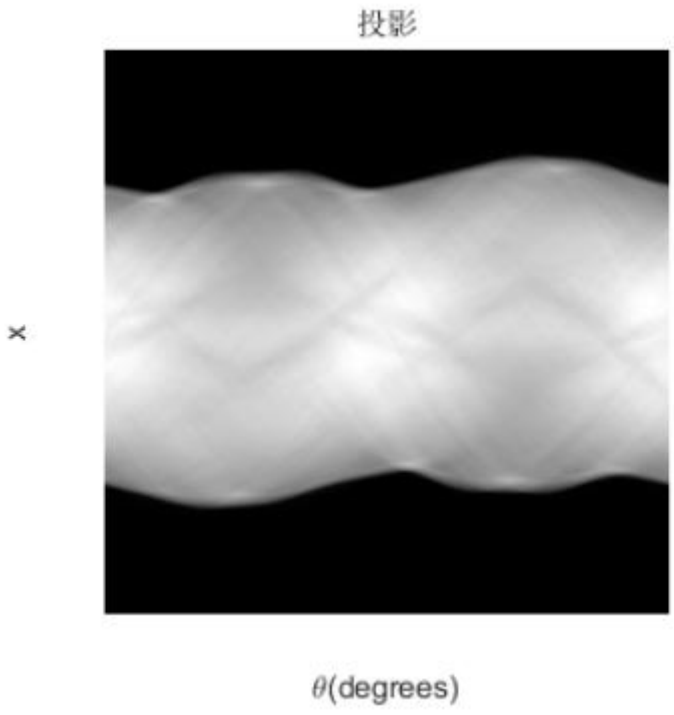



图 20 扇形束投影图

# 3、滤波

相关代码在 fbp.m 中。

通过查询了解到有时域和频域滤波两个方向。我在时域滤波器选择的是 sinc 低通滤波，通过与投影图进行卷积获得滤波后的数据，该滤波器公式如下：

$$\mathrm{Filter(t)} = 0.0085 * (\frac{\sin(\pi t)}{2\pi t} - \frac{(\sin(\pi\frac{t}{2}))^2}{4(\pi\frac{t}{2})^2})$$

展示如下：

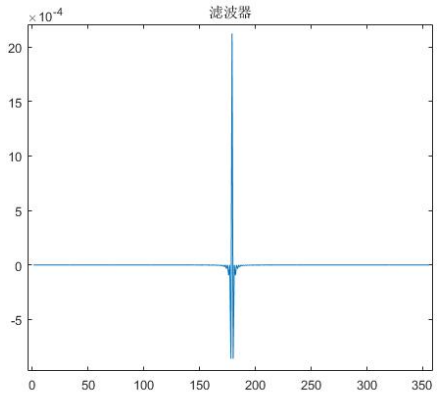



图 21 滤波器图

频域上的滤波器有 R-L 滤波器和 Hamming 窗。将投影图进行傅里叶变换后乘以滤波器，再通过傅里叶反变换回来取实数部分即可获得滤波后的数据。

将这几种进行组合滤波，重建图展示如下，可以发现效果最好的是只用频域上的 R-L 滤波器和 Hamming 窗，但是为了展示滤波后的投影图所以要有时域上的投影。所以发现单独时域时图像与原图有一定差距，而时域滤波加上频域的两个滤波器发现滤波程度过高导致重建图像锐化过高并损失了灰度值的准确性，因此选择了时域 sinc 滤波和频域 hamming 窗滤波结合。滤波完后为了美观统一性，也调整为 256*256 的尺寸进行展示。

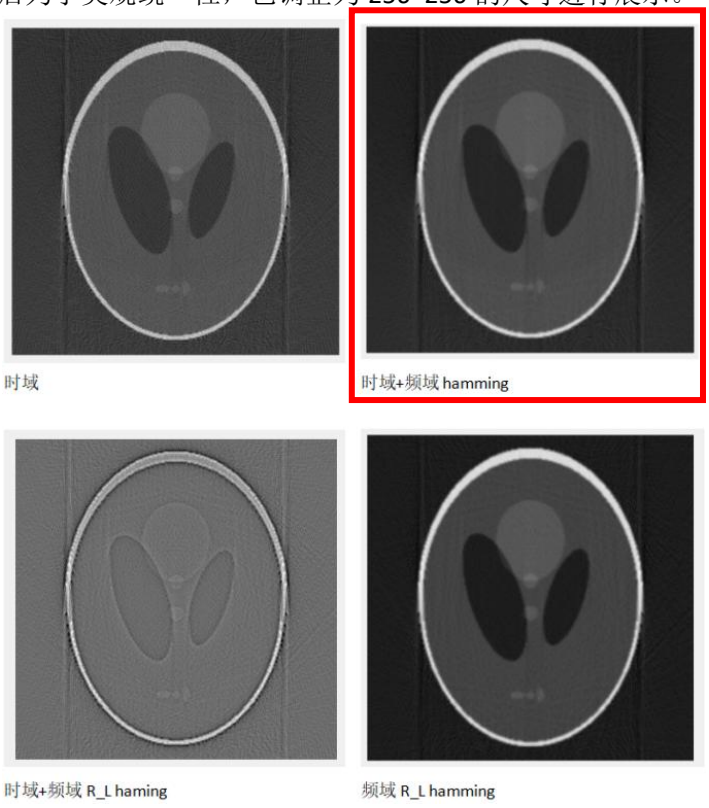



图 22 滤波器组合

相关代码和反投影代码一起封装在了 fbp(projection,filter_choose, sensor_length,pixelsize) 函数内，滤波代码如下（以平行束为例，扇形束同理）：

```
%和滤波器卷积
dS=1;
rayNum = sensor_length;
ray_x=-(rayNum-1)/2 * dS:dS:(rayNum-1)/2 * dS;
ray_x=ray_x';
l=length(ray_x);

N=M;
t = linspace(-N/2,N/2-1,N);%生成线性间距向量
filt = 0.0085*(sinc(t)/2-sinc(t/2).^2/4);%得到滤波的卷积核

p = projection;
projection0 = zeros(l+2*N,180);
projection0(N+1:l+N,:) = p;
for n = 1:180
    temp(:,n) = conv(projection0(:,n),filt); %#ok<SAGROW>
end
projection0=temp(round(3*M/2:3*M/2+l),:);%投影滤波
projection0_show = imresize(projection0, [256,256]);
imshow(projection0_show,[],'Parent',app.UIAxes4)
xlabel('\theta(degrees)','Parent',app.UIAxes4);
ylabel('x','Parent',app.UIAxes4);
drawnow
%卷积结束

%Img = zeros(M);
%Detectorsize = detector_size;
%[ImgRow,ImgCol] = size(Img);
width = 2^nextpow2(size(projection0,1))*16;
pro_fft = fft(projection0,width);

%R_L = (1/Detectorsize)*[0:(width/2), width/2-1:-1:1]'/width; %1/2tau,tau为采样间距
filter_Hamming = 0.54-0.46*cos([-width/2+1:0,1:width/2]'.*(2*pi/(width)));
pro_filter = zeros(width,size(projection0,2));
for i = 1:1:size(projection0,2)
    pro_filter(:,i)=pro_fft(:,i).*filter_Hamming;
    %pro_filter(:,i) = pro_fft(:,i).*R_L.*filter_Hamming;
end
pro_ifft = real(ifft(pro_filter));
```

滤波后的投影图如下：

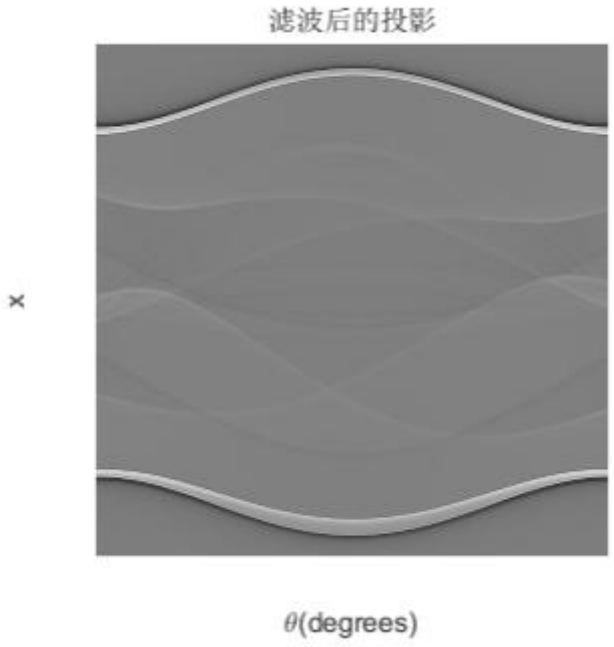



图 23 平行束滤波后的投影

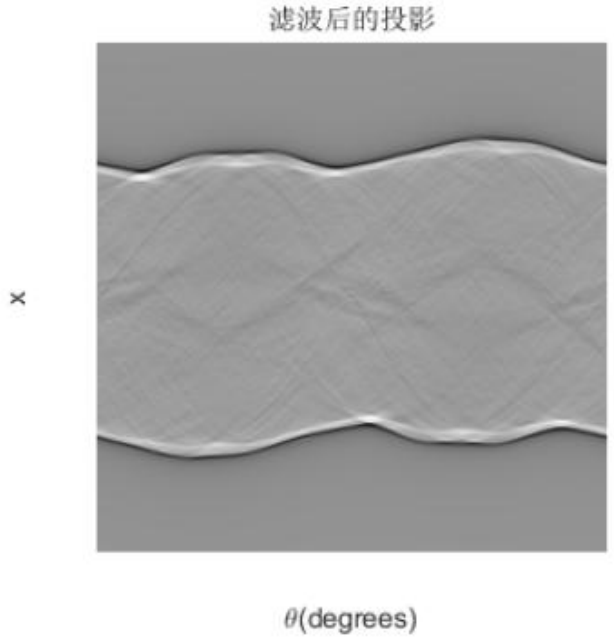



图 24 扇形束滤波后的投影

# 4、反投影

## 4.1 平行束

使用像素驱动法的反投影算法，具体过程是遍历总投影角度，即投影图像的列数。随后遍历像素格点，确定像素中心的坐标，然后确定该位置的投影值。最后遍历完要对所有像素值除以反投影的次数。

相关代码封装在了 fbp(projection,filter_choose, sensor_length,pixelsize)函数中，代码如下：

```
for i = 1:1:size(projection,2)
    for c = 1:ImgCol
        for r = 1:ImgRow
            x = (c-(ImgCol+1)/2).*pixelsize(1);
            y = ((ImgRow+1)/2-r).*pixelsize(2);
            s = x*cos((i-1)*pi/180)+y*sin((i-1)*pi/180);
            s = s/Detectorsize + N_d/2 + 1;
            n = floor(s);
            if n > 0 && n <= N_d
                q = pro_ifft(n,i);
                Img(r,c) = Img(r,c)+q;
            end
        end
    end
    %str = ['重建进度: ',num2str(round(i/size(projection,2)*100)),'%'];
    %disp(str);
end
Img = Img*pi/size(projection,2);
```

滤波反投影后重建图如下：

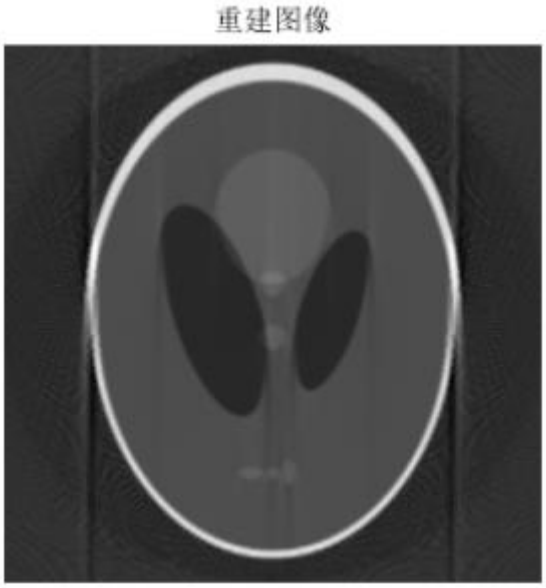



图 25 平行束滤波反投影图

# 4.2 扇形束

## 思路

- 根据参考书上的思路和公式，将投影数据乘以权重$\frac{D}{\sqrt{D^2+\sqrt{s^2}}}$。
- 卷积滤波后，遍历总投影角度，即投影图像的列数。
- 遍历像素格点，确定像素中心的坐标，将穿过该像素点的所有射线进行累加，公式如下：

$$f(x,y)=\Delta B\sum_{i=1}^{M}\frac{1}{U^2(x,y,\beta_i)}Q_{\beta_i}(s')$$

- 最终累计后得到重建结果。

  相关代码封装在了 FanBeamFBP(sizeM,sizeN,beta,R,rayNum,dS,distance,filter_choose)函数中，核心代码如下：

```
recon = zeros([M,N]);%重建图像
M2 = (M-1)/2;
dbeta = deg2rad(1);
for i = beta%射线源旋转每一个角度
    r = deg2rad(i);%角度转弧度
    for x = 1:M
        for y = 1:N%对于重建图像中的每一个点
            sp = distance*((y-M2-1)*cos(r)+(x-M2-1)*sin(r))/(distance+(y-M2-1)*sin(r)-(x-M2-1)*cos(r));
            %得到s'，这里的推导可以试着自己推一推，过(x,y)和射线源的点和D'2(该线随β变化而变化)的交点与0连接线段的长度。
            % 再注意：这段循环中的x和y并不是正常的x和y，x表示行数而y表示列数，与我们习惯中的x和y的意义相反，
            % 因此推得公式的x，y要互换
            if (sp >= -(1-1)/2)&&(sp<=(1-1)/2)%若s 没有超出探测器的范围
                isp = ceil(sp); sp0 = isp-sp;
                proj_sp = (1-sp0)*R(isp+(1-1)/2+1,i+1)+sp0*R(isp+(1-1)/2,i+1);
                %根据s 的值用他前后的射线投影为他插值，这里没用interp1，能够很大程度上加快运行速度
                recon(x,y) = recon(x,y)+dbeta*(1/2*power(U(distance,(y-M2-1),(x-M2-1),r),2))*proj_sp;%重建图像上的像素点
            end
        end
    end
    %str = ['重建进度: ',num2str(round(i/359*100)),'%'];
    %disp(str)
end
```

滤波反投影后重建图如下：

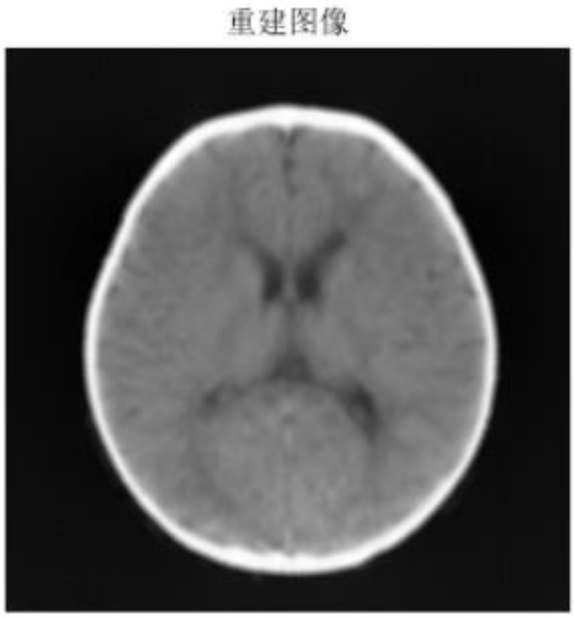



图 26 扇形束滤波反投影图

# 第四章 界面设计

使用 Matlab 自带的 APP Designer 进行设计。

## 1 代码介绍

## 1.1 所有对象

GUI 组件对象：

```
% Properties that correspond to app components
properties (Access = public)
    UIFigure          matlab.ui.Figure
    LoadButton        matlab.ui.control.Button
    FBPButton         matlab.ui.control.Button
    ButtonGroup       matlab.ui.container.ButtonGroup
    SheppLoganButton  matlab.ui.control.RadioButton
    Button            matlab.ui.control.RadioButton
    ButtonGroup_2     matlab.ui.container.ButtonGroup
    Button_2          matlab.ui.control.RadioButton
    Button_3          matlab.ui.control.RadioButton
    TextArea          matlab.ui.control.TextArea
    CheckBox          matlab.ui.control.CheckBox
    Label_7           matlab.ui.control.Label
    Label_1           matlab.ui.control.Label
    Label_8           matlab.ui.control.Label
    Label_9           matlab.ui.control.Label
    Label_10          matlab.ui.control.Label
    Label_11          matlab.ui.control.Label
    UIAxes            matlab.ui.control.UIAxes
    UIAxes2           matlab.ui.control.UIAxes
    UIAxes3           matlab.ui.control.UIAxes
    UIAxes4           matlab.ui.control.UIAxes
end
```

内部自定义变量：

P_choose0 是作为图像选择方法的参数，P_chooses 是通过“显示/选择”按钮更新后的图像选择方法参数，后面调用的都是这个参数。radonway_choose 是投影方式的选择参数，1 为平行束滤波，2 为扇形束滤波。filter_choose 是滤波器的选择参数，1 为选择，0 为不选择。

```
properties (Access = public)
    P_choose0 = 1;
    P_choose = 1 % 图像选择
    radonway_choose = 1; %投影方式选择
    filter_choose = 1; %滤波选择
end
```

内部自定义函数：

为了可以实时将投影进度和重建进度反应在 GUI 界面上，定义了 updateGUI 函数来更新进度值。

```
methods (Access = public)

    function updateGUI(app, i)
        app.Label_7.Text = i;
    end
end
```

## 1.2 用户交互：

显示/选择：

通过 P_choose0 的值来判断是进行 shepp-logan 模型显示，还是供用户选择图片，将选好的图片进行格式转换。并将图像信息更新至 LoadButton.UserData。

相关代码如下：

```
% Button pushed function: LoadButton
function LoadButtonPushed(app, event)
    if app.P_choose0 == 1
        M=256;
        [P,E] = phantom('Modified Shepp-Logan', M); % 创建一个 shepp-logan 模型，原图像
        imshow(P,'Parent',app.UIAxes);
        app.P_choose = 1;
        if app.radonway_choose == 1
            app.LoadButton.UserData = E;
        else
            app.LoadButton.UserData = P;
        end
    else
        app.P_choose = 2;
        [filename,filepath] = uigetfile({'*.png;*.jpg'}, 'Select File to Open');
        fullname = [filepath, filename];
        if fullname ~= 0                    %#ok<BDSCI>
            P = imread(fullname);
            P = imresize(P, [256,256]);
            P = rgb2gray(P);
            P = im2double(P); %转换成double
            imshow(P,'Parent',app.UIAxes);
            app.LoadButton.UserData = P;
        end
    end
end
```

开始重建：

如果是用平行束重建 shepp-logan 模型，则调用 GUIRadon_Computerized.m 文件，如果用平行束重建一般图像。则调用 GUIRadon.m 文件，如果用扇形束重建图像，则调用 GUIRadon_fan.m 文件。

相关代码如下：

```
% Button pushed function: FBPButton
function FBPButtonPushed(app, event)
    if app.radonway_choose == 1
        if app.P_choose == 1
            run("GUIRadon_Computerized.m")
        else
            run("GUIRadon.m")
        end
    else
        run("GUIRadon_fan.m")
    end
end
```

选择：

对图像选择的方式，投影的方式，滤波器的选择的代码如下：

```
% Selection changed function: ButtonGroup
function ButtonGroupSelectionChanged(app, event)
    selectedButton = app.ButtonGroup.SelectedObject;
    %选择模型或本地图像
    if selectedButton.Text == "Shepp-Logan模型"
        app.P_choose0 = 1;
    else
        app.P_choose0 = 2;
    end
end

% Selection changed function: ButtonGroup_2
function ButtonGroup_2SelectionChanged(app, event)
    selectedButton = app.ButtonGroup_2.SelectedObject;
    %选择平行束投影或扇形束投影
    if selectedButton.Text == "平行束"
        app.radonway_choose = 1;
    else
        app.radonway_choose = 2;
    end
end

% Value changed function: CheckBox
function CheckBoxValueChanged(app, event)
    value = app.CheckBox.Value;
    %是否选择滤波器，value=1是，=0否
    if value == 1
        app.filter_choose = 1;
    else
        app.filter_choose = 0;
    end
end
```

## 2 打包成独立桌面 APP

使用 Matlab 自带的 Application Compile 插件进行打包，同时下载对应的 runtime 作为软件运行环境。最后成功打包出了可以独立运行的 APP 和依靠 Matlab 运行的 APP。适配于不同环境的用户。

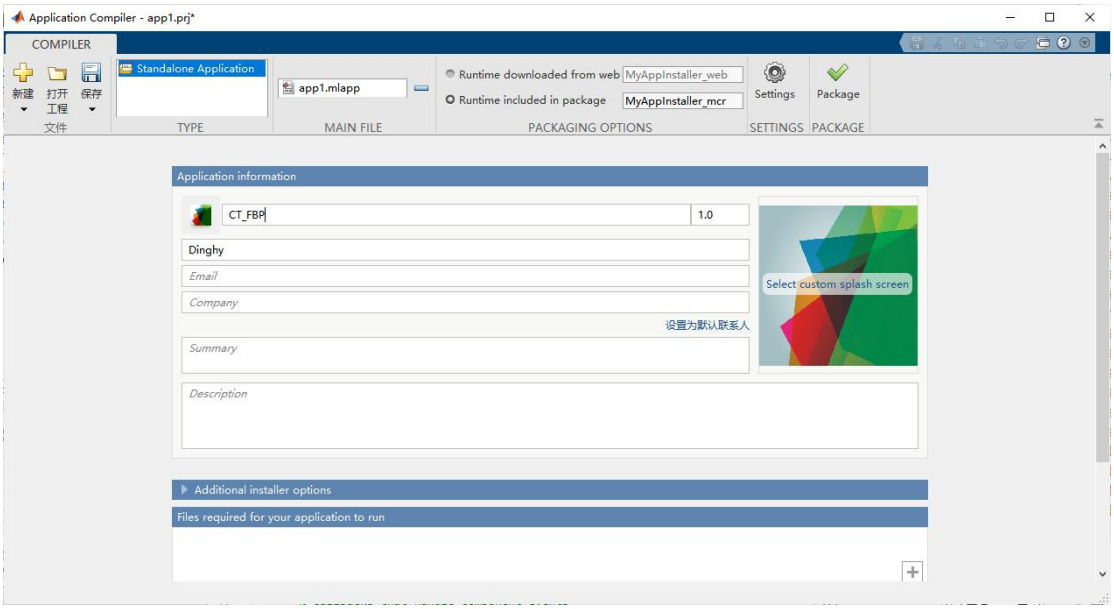



图 27 打包界面

# 第五章 成果展示

## 1 平行束

### 解析法

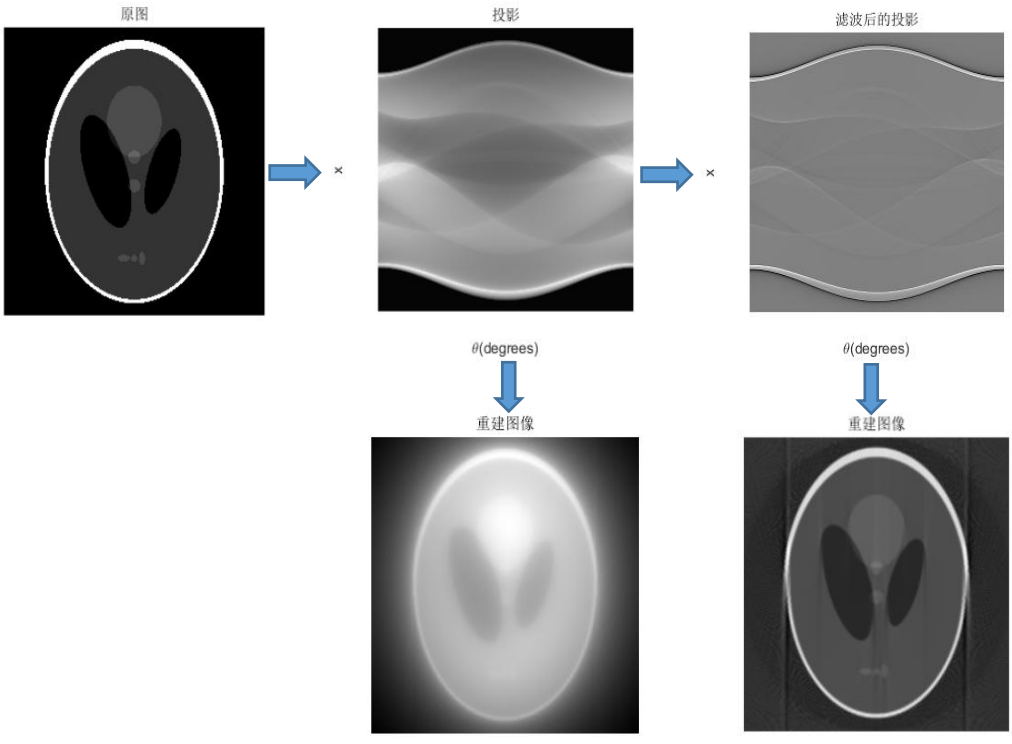



图 28 解析法成果

数字积分法

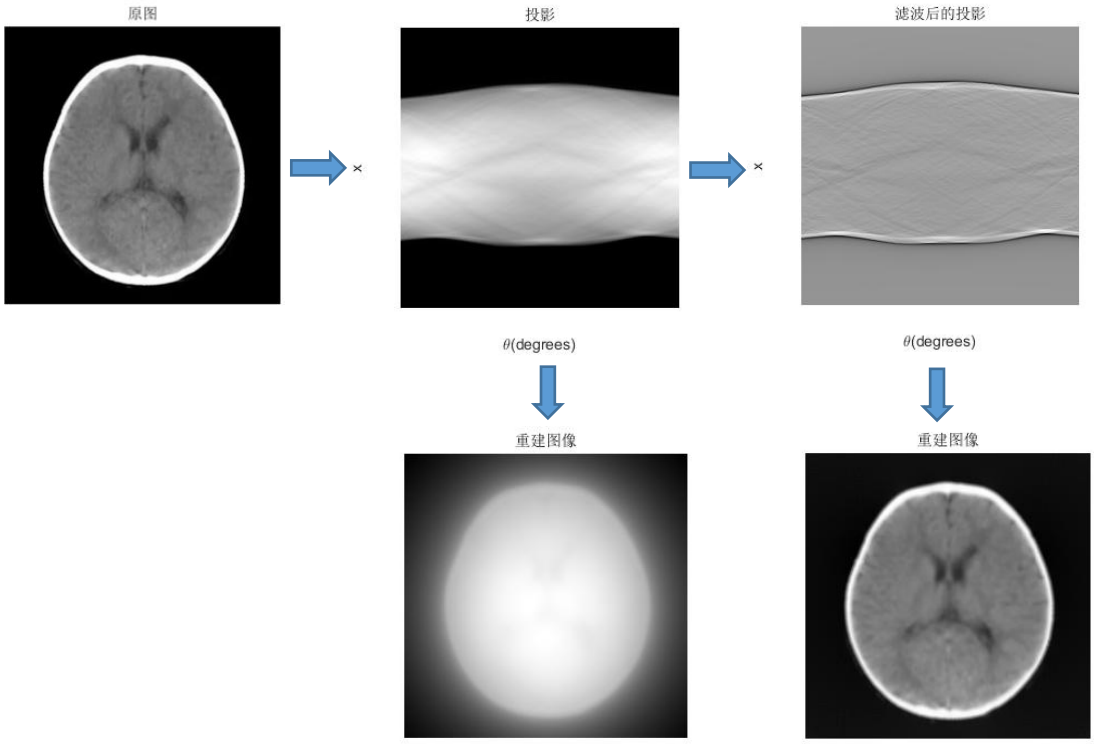



图 29 数字积分法成果

## 2 扇形束

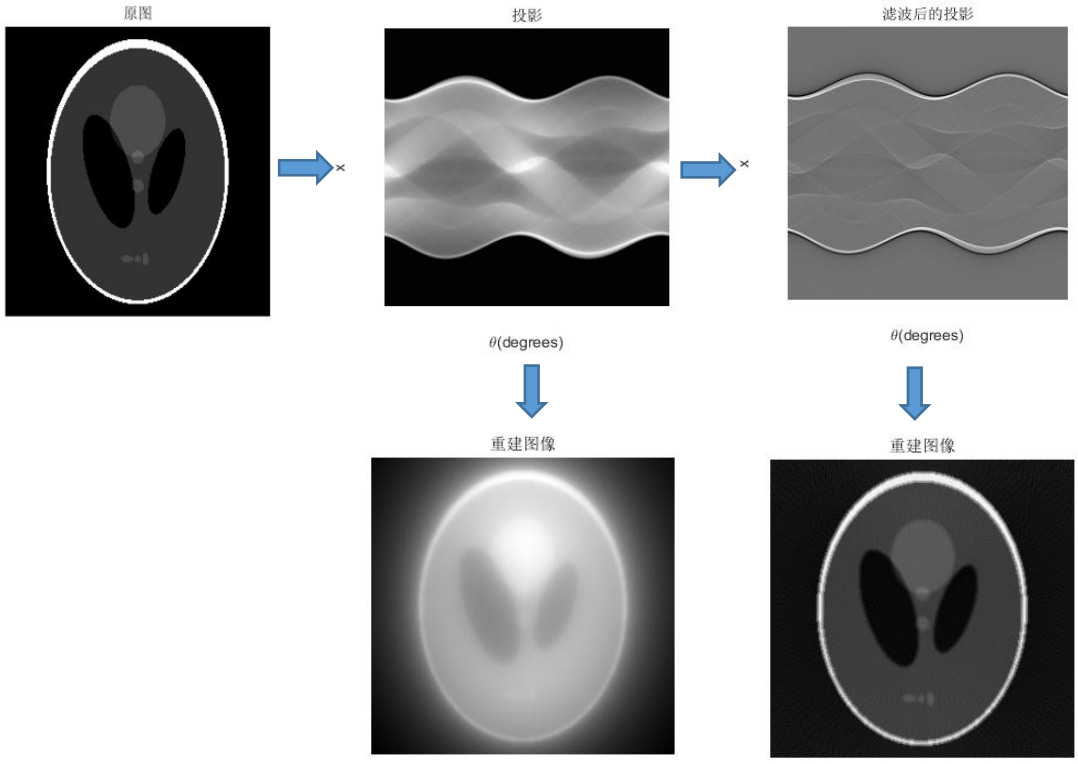



图 30 扇形束成果

## 3 成果评价

可以看到上面都将图像较好地重建出来了，并且加了滤波后的重建图片要远优于未加滤波器的重建图像，加了滤波后的重建图像与原图更相近。

# 第六章 软件完善

## 1 滤波器改进

可以看到滤波的效果不是特别理想，在图片边缘会有切线，并且图片整体会比原图的颜色淡一些，所以还可以继续改进滤波器。并且实际 CT 重建算法的滤波中，可能会对特定频率进行增强或抑制，以此突出目标区域。

## 2 时间效率的提高

在扇形束投影时候线性差值算法用到了 Matlab 自带的 interp 函数，所以运行较慢，投影大致需要 1 分钟左右。对更大的图像所需时间会更高，所以考虑到时间，将所有图片统一为 256*256 格式。后续可以改进算法，这样也可以对更大的图片进行投影重建。

## 3 参数选择的多样

一些参数在编程的时候为了方便，直接固定了值，例如投影的角度范围，图片的大小，后续可以对这些参数进行调整，实现供用户选择的功能。

## 4 锥形束的实现

锥形束是在三维上的扩展，后续可以在原有基础上，进一步扩展锥形束的功能。

# 第七章 项目收获

## 1 CT 重建知识

通过本次大作业项目，我对**滤波反投影算法**有了更加深入的了解。在这个过程中，我发现了很多有趣的东西，并学到了很多新的技能。首先，我通过文献查阅和网上资料搜索，逐步了解了各种算法。我对 shepp-logan 模型的解析算法投影，对一般图像的数字积分法投影，以及反投影的算法有了更加深入的了解。我还了解到了不同类型的滤波器，例如 Ram-Lak 滤波器和 Sinc 滤波器。这些滤波器的作用都是去除图像中的噪声和伪影，从而提高图像的质量和精度。但它们在不同的情况下具有不同的优缺点，需要根据实际应用进行选择。我还了解到了扇形束的投影反投影算法，这对我非常有帮助。

在了解滤波反投影算法的基础上，我对 **CT 技术**也有了更深入的了解。我了解到 CT 技术是一种非破坏性的成像技术，可以用于检测物体内部的结构和组成。CT 扫描利用 X 射线穿过物体的不同程度来获取图像信息，然后通过滤波反投影算法进行图像重建。我了解到 CT 技术在医学、工业和科学研究等领域都有广泛的应用，可以用于诊断疾病、检测材料缺陷、研究材料结构等方面。同时我了解到在实际中，目前 CT 技术除了普通的基于滤波的反投影算法，还有基于迭代的重建算法，并且现在逐步向基于深度学习的重建算法发展，所以目前重建算法还是有很大的发展空间。

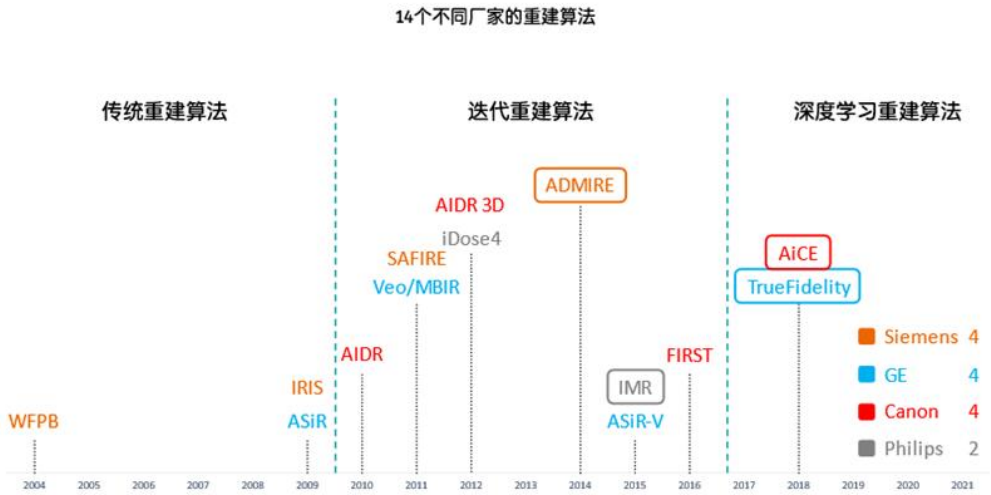



图 31 CT 重建算法

# 2 代码编写能力

其次，我在使用 Matlab 编写代码时也获得了很多的收获，提高了自己的代码能力。

在**将公式用代码实现**方面，Matlab 是一种非常强大的数学计算软件，它提供了许多内置函数和工具箱来实现各种数学公式和算法。比如生成 shepp-logan 模型可以直接使用 Matlab 提供的 phatom 函数，进行傅里叶变换时也可以直接使用 fft 函数。但是难点主要在于先理解参考文献里的思路和公式，并将其用代码实现。在编写代码的时候还要特别注意**边界处理和特殊条件的处理**，因为很多理论参考是用公式表示的，但将其转换为代码实现时候，可能会有特殊情况。比如在实现投影时，思路是直接计算相交的 α 值，但是在代码编写时，要注意可能有的边缘射线是不和图像相交的。还有当射线和 x 轴或 y 轴平行时也要特殊考虑。而在参考文献里面这些都是不需要考虑的。

在**调试**方面，Matlab 提供了许多有用的调试工具，例如断点、变量监视器和调用堆栈跟踪器等。但调试是一个需要耐心和时间的过程，尤其是在处理复杂代码时。虽然 Matlab 提供了许多有用的调试工具，但找到和解决问题仍然需要一定的耐心和技巧。例如，我当时在写数字积分法的代码时候，投影出的图和用 radon()函数投影出的理想图像总会多一些奇怪的噪声，我检查了所有的函数变量细节和思路但是并没有找到问题，所以只能通过调试，一步步跟踪代码运行情况。而这个算法里面又有很多循环，如果一步步循环是很麻烦的事情，经过搜索了解到可以在循环中设置条件语句来跟踪中途某次循环产生的问题。经过调试，后面终于发现是因为在升序排序的时候，使用 rand()函数，但是没有重新赋值，所以并不会改变原始数组，所以当合并 $\alpha_x$ 和 $\alpha_y$ 的时候，并没有被升序排序，因此在后续计算中会产生错误的投影值。经过这些调试和尝试，我终于解决了这个问题，投影出了理想的图片。

在 **GUI 界面实现**方面，Matlab 提供了一个称为 APP Design（图形用户界面开发环境）的可视化开发环境，使得创建 GUI 界面变得更加容易。

最后，在**软件打包**方面，Matlab 提供了一个称为 MATLAB Compiler 的工具箱，可以将 Matlab 代码编译成独立应用程序或可执行文件，并使其在没有安装 Matlab 的计算机上运行。为了保险起见，我找了几个没有 matlab 的朋友，在他们的电脑上都测试成功，说明这份可执行文件是可以独立运行的。

最终将这些预期功能实现出来，项目完成后，我不仅获得了知识上的提升，编程技能上的提升，还非常有成就感。

# 3 感谢

在本次项目中，通过 Matlab 进行编程，在实践中融合理论知识，加深了我对于滤波反投影知识的了解。同时也由于本次大作业是关于生物医学工程的应用，我更加深入了解了医学图像处理知识在其中的应用和重要性。而通过收集资料，我对 CT 重建知识有了更多了解。在实践中也遇到了很多的困难和挑战，但是当我最终实现了预期功能后还是很有成就感的，不过由于所学知识和能力的受限，仍旧有许多不足，说明我仍需不断学习知识，以期更好的完善设计。

最后感谢赵俊老师和助教在医学图像处理课程上的帮助，让我更好的学到了这门课程的知识，并对医学图像处理知识和 CT 重建产生兴趣，将来有机会我还会继续学习相关知识的。

# 第八章 参考资料

- Shepp-Logan 的参数以及解析法投影公式：

参考 Avinash C. Kak, and Malcolm Slaney, “Principles of Computerized Tomographic Imaging”, IEEE Press, 1999. https://engineering.purdue.edu/~malcolm/pct/ , Chapter 3.Algorithms for Reconstruction with Nondiffracting Sources.

- 一般图像的投影（数字积分法）：

参考 Siddon, Robert L. "Fast calculation of the exact radiological path for a three-dimensional CT array." Medical physics 12.2 (1985): 252-255.

- 扇形束的投影反投影方法：

参考 Avinash C. Kak, and Malcolm Slaney, “Principles of Computerized Tomographic Imaging”, IEEE Press, 1999. https://engineering.purdue.edu/~malcolm/pct/ , Chapter 3. Algorithms for Reconstruction with Nondiffracting Sources.

- 其他资料：

➢ 浅谈 CT 图像重建的数理基础|图像|重建|算法|迭代|噪声|-健康界(cn-healthcare.com)

➢ 滤波反投影图像重建算法_滤波反投影法的优缺点_shuangyue 的博客-CSDN 博客

➢ 对滤波反投影重建算法的研究以 phantom 图进行 matlab 仿真，构建滤波器，重建图像_matlab 反投影_我爱 C 编程的博客-CSDN 博客

➢ CT 图像重建技术_ct 图像重建原理_shuangyue 的博客-CSDN 博客

➢ 滤波反投影重建算法（FBP）实现及应用（matlab）_滤波反投影法_土豆洋芋山药蛋的博客-CSDN 博客

➢ CT 成像——等距扇束正反投影的简单介绍以及在 matlab 上的实现_等间距扇形束 fbp 重建_Messiahx 的博客-CSDN 博客

➢ SART 算法的 MATLAB 实现 - 知乎 (zhihu.com)

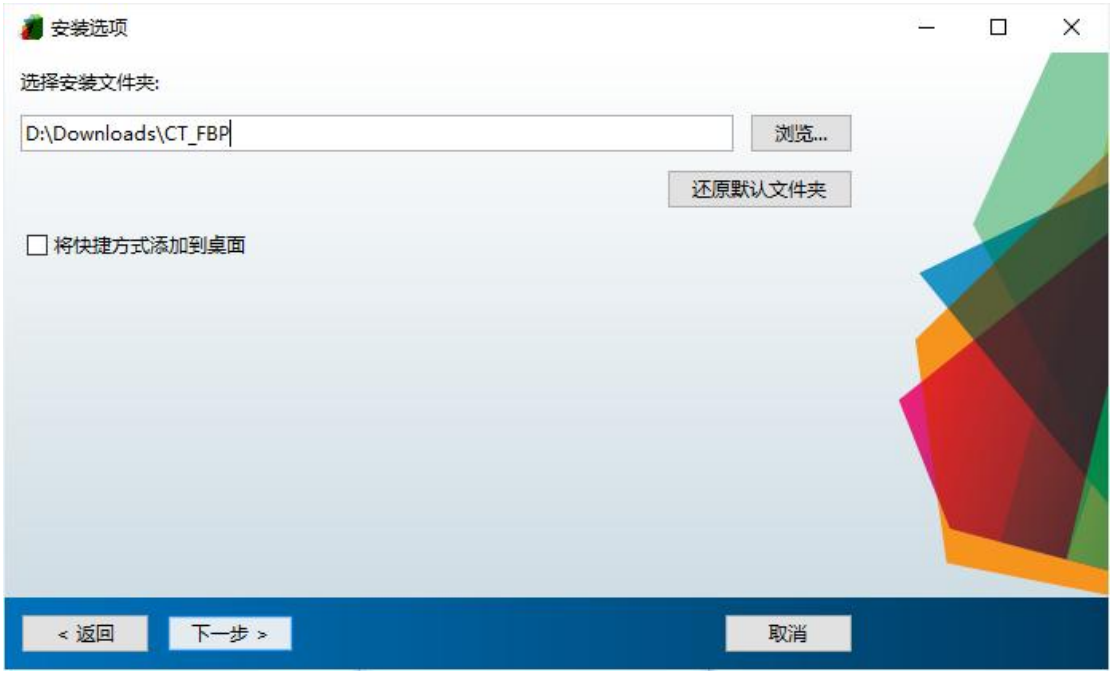
# 附录

## 1、安装软件

（1）进入 for_redistribution 文件夹，点击 MyAppInstaller_mcr.exe，开始对 app 进行安装，进入安装界面，点击“下一步”。在安装过程中建议用户将 app 以及附带的 Runtime

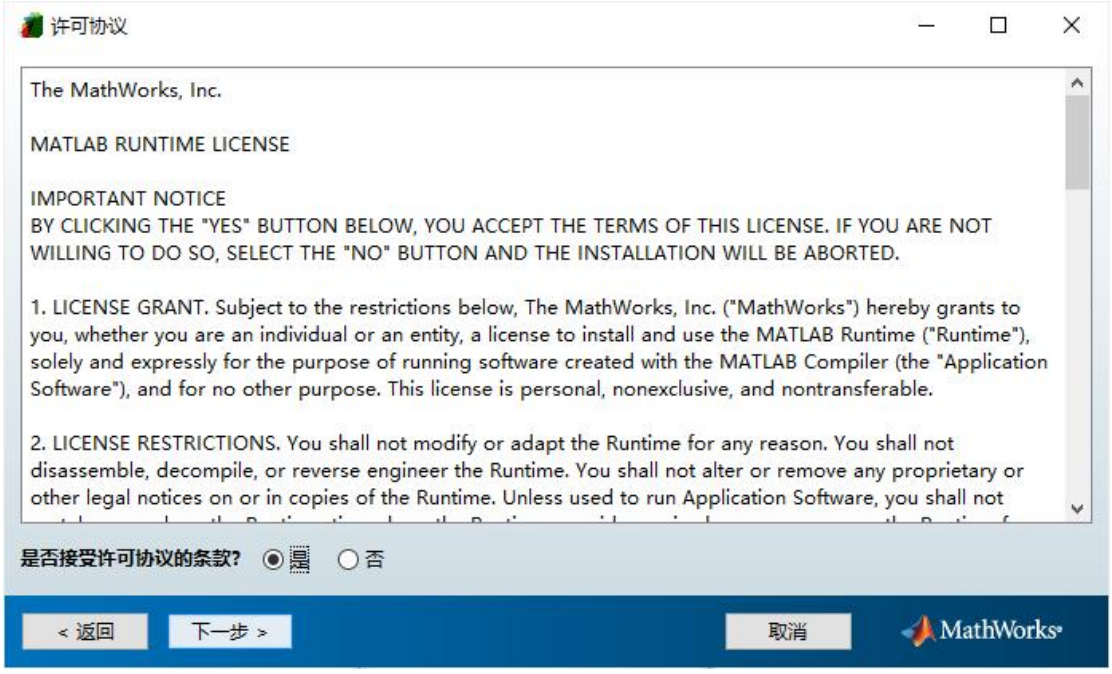
文件安装到非系统盘新建的空文件夹里


（2）选择软件安装文件夹，点击“下一步”。（注意：建议安装路径里面**不要包含中文**。同时建议用户将 app 以及附带的 Runtime 文件安装到**非系统盘新建的空文件夹**里，因为在卸载软件的时候会清空这个文件夹里的所有文件。）



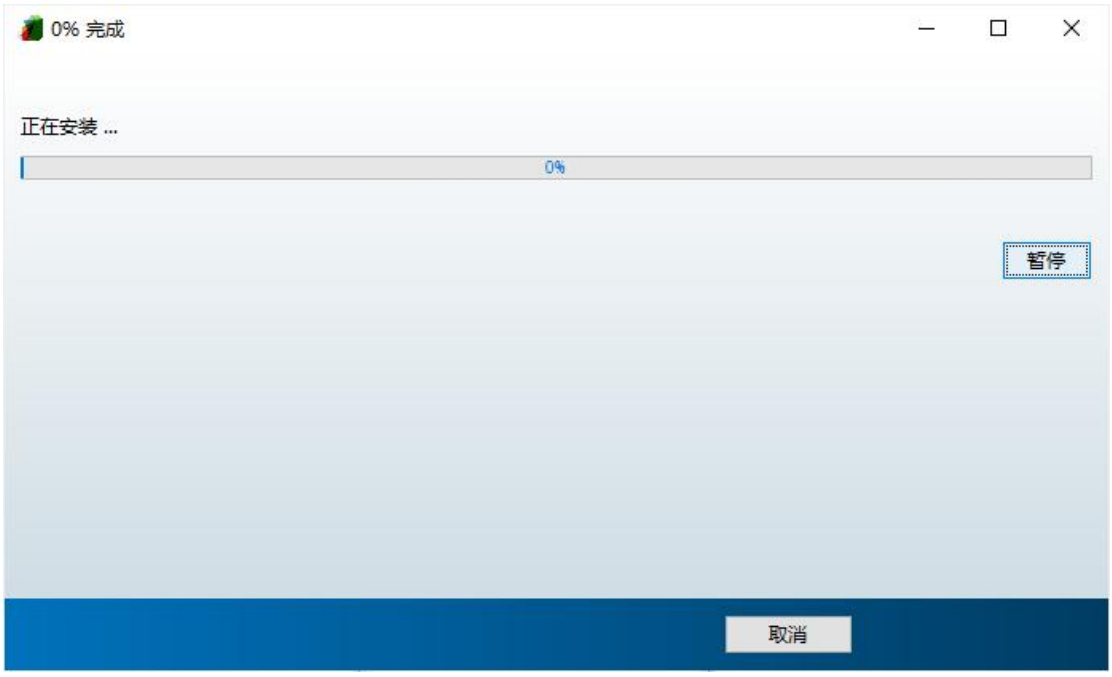

（3）选择 Matlab Runtime 安装文件夹，点击“下一步”。（注意：建议安装路径里面不要包含中文。同时建议用户将 app 以及附带的 Runtime 文件安装到非系统盘新建的空文件夹里，因为在卸载软件的时候会清空这个文件夹里的所有文件。）


（4）选择“是”，接受许可协议，点击“下一步”。




（5）安装确认，若想更改点击“返回”进行更改，若无误点击“安装”，即可开始安装。

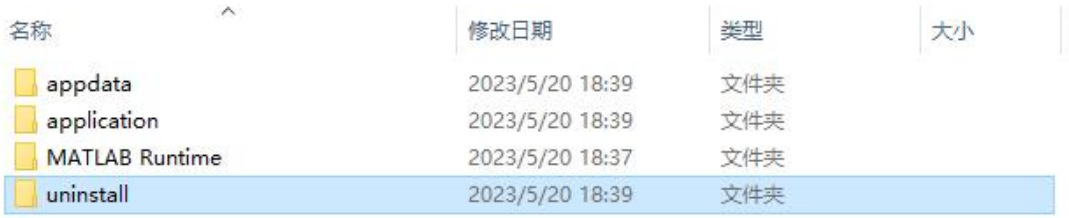

（6）进入安装界面，等待大约 1.5 分钟即可完成安装。



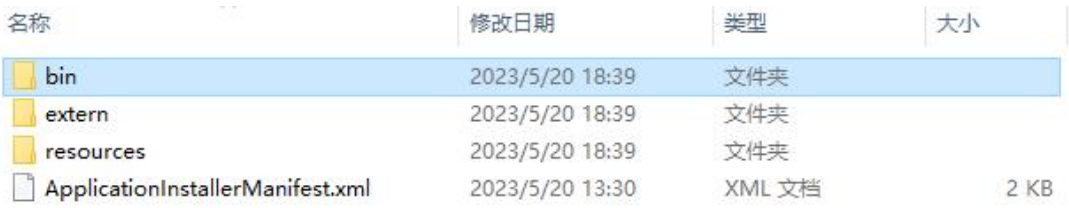

（7）显示安装完成。


## 2、卸载软件：

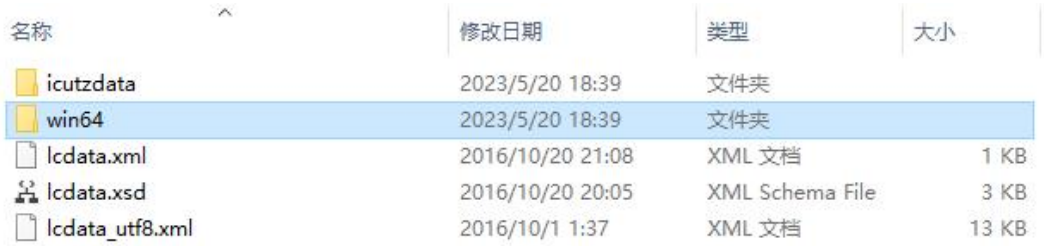
（1）找到安装软件的文件夹并打开“uninstall”文件夹。




（2）打开“bin”文件夹。




（3）打开“win64”文件夹。




（4）找到“Uninstall_Application.exe”，双击运行。运行完成后，软件即可从我们的电脑中移除。

| 名称 | 修改日期 | 类型 | 大小 |
| --- | --- | --- | --- |
| mwboost_iostreams-vc141-mt-x64-1_... | 2020/7/29 18:37 | 应用程序扩展 | 79 KB |
| mwboost_log-vc141-mt-x64-1_70.dll | 2020/7/29 18:40 | 应用程序扩展 | 691 KB |
| mwboost_regex-vc141-mt-x64-1_70.dll | 2020/7/29 18:37 | 应用程序扩展 | 984 KB |
| mwboost_serialization-vc141-mt-x64-... | 2020/7/29 18:37 | 应用程序扩展 | 240 KB |
| mwboost_system-vc141-mt-x64-1_70.... | 2020/7/29 18:25 | 应用程序扩展 | 10 KB |
| mwboost_thread-vc141-mt-x64-1_70.... | 2020/7/29 18:39 | 应用程序扩展 | 94 KB |
| mwboost_timer-vc141-mt-x64-1_70.dll | 2020/7/29 18:39 | 应用程序扩展 | 29 KB |
| registrykeyapi.dll | 2020/7/29 19:52 | 应用程序扩展 | 13 KB |
| SCRCodeGen3.exe | 2020/7/29 18:25 | 应用程序 | 302 KB |
| shortcutsapi.dll | 2020/7/29 19:52 | 应用程序扩展 | 12 KB |
| tbb.dll | 2020/7/29 18:25 | 应用程序扩展 | 382 KB |
| tbbmalloc.dll | 2020/7/29 18:25 | 应用程序扩展 | 234 KB |
| ucrtbase.dll | 2020/7/29 18:25 | 应用程序扩展 | 979 KB |
| Uninstall_Application.exe | 2020/7/29 20:12 | 应用程序 | 91 KB |
| uninstall_helper.exe | 2021/1/6 18:10 | 应用程序 | 795 KB |
| usLogService.dll | 2020/7/29 18:25 | 应用程序扩展 | 13 KB |
| usResourceCompiler3.exe | 2020/7/29 18:25 | 应用程序 | 303 KB |
| usServiceComponent.dll | 2020/7/29 18:25 | 应用程序扩展 | 58 KB |
| vccorlib140.dll | 2020/7/29 18:36 | 应用程序扩展 | 378 KB |
| vcruntime140.dll | 2020/7/29 18:36 | 应用程序扩展 | 86 KB |
| zlib.rights | 2020/7/29 18:25 | RIGHTS 文件 | 1 KB |
| zlib1.dll | 2020/7/29 18:25 | 应用程序扩展 | 84 KB |

（5）进入卸载界面，大致等待 1 分钟即可完成卸载。