Table of Contents

generateLungPhantom

Version number: 1.0

Arguments: infile -- .txt file containing specifications of the atelectasis phantom to be created

Return: outImages -- structure containing the names of the images created by the function.

Other Effects: Image files are generated for the phantom in the curent MATLAB directory. Images and masks are written as .mhd format.

Description: This function creates a pair of virtual CT images which contain simple geometric objects specified by the input parameter file along with other properties such as intensity, mass preservation, etc. Together, the images form a virtual phantom which can be used for image registration testing.

Example:

```
in = phantom_parameters.txt;

out = generateLungPhantom(in);
```

Revision / Date / Author / Description

1.0 / 04 05 17 / Chris Guy / Initial build

```
function [ outImages ] = generateLungPhantom( infile )

    % Phantom specification file is read.
    input = parsePhantomParameterFile(infile);
    % Binary masks of the ojbects are created.
    [firstMask, secondMask] = generateImageMasks(input);

    % Features are added as specified in the parameter file.
    if input.addFirstFeatures
        firstFeatureImg = generateFeatureImage(1,input);
    else
        firstFeatureImg = 0;
    end

    if input.addSecondFeatures
        secondFeatureImg = generateFeatureImage(2,input);
    else
        secondFeatureImg = 0;
    end

    % Phantom images are created.
    [firstImg, firstHdr] = generateFirstImage(firstMask,
 secondMask, ...
                               firstFeatureImg, input);
    [secondImg, secondHdr] = generateSecondImage(firstImg,
 firstMask, ...
                                 secondMask, secondFeatureImg, input);

    % Names of created files are returned.
    outImages.first = [input.firstName '.mhd'];
    outImages.firstMask = [input.firstName '_mask.mhd'];
    outImages.second = [input.secondName '.mhd'];
    outImages.secondMask = [input.secondName '_mask.mhd'];

    % Phantom images are saved as .mhd files.
    WriteMetaImage(input.firstName, firstImg, firstHdr);
    WriteMetaImage([input.firstName '_mask'], firstMask, firstHdr);
    WriteMetaImage(input.secondName, secondImg, secondHdr);
    WriteMetaImage([input.secondName '_mask'], secondMask, secondHdr);

end % generateLungPhantom
```

generateImageMasks

This subfunction creates masks for the two images of the phantom pair using shape files specified in the phantom parameter file.

```
function [ firstMask, secondMask ] = generateImageMasks( info )

    firstInfo.size = info.size;
    firstInfo.obj = info.firstObj;
    secondInfo.size = info.size;
    secondInfo.obj = info.secondObj;

    if (info.dims == 3)
```

```
        disp('Creating first mask');
        tic
        firstMask = generateMask3d(firstInfo);
        toc

        disp('Creating second mask');
        tic
        secondMask = generateMask3d(secondInfo);
        toc

    else

        firstMask = 0;
        secondMask = 0;

    end

end % generateImageMasks
```

isInSphere

This subfunction tests if the given coordinates are within the sphere described by the given parameters.

```
function [ isIn ] = isInSphere(x, y, z, params)

    radius = str2double(params{3});
    centerX = str2double(params{4});
    centerY = str2double(params{5});
    centerZ = str2double(params{6});

    distance = (x-centerX)^2 + (y-centerY)^2 + (z-centerZ)^2;

    if(distance < radius^2)
        isIn = 1;
    else
        isIn = 0;
    end

end % isInSphere
```

isInZCylinder

This subfunction tests if the given coordinates are within the cylinder described by the given parameters.

```
function [ isIn ] = isInZCylinder(x, y, z, params)

    radius = str2double(params{3});
    centerX = str2double(params{4});
    centerY = str2double(params{5});
    zMin = str2double(params{6});
    zMax = str2double(params{7});

    distance = sqrt(((x-centerX)^2)+((y-centerY)^2));
    if (z >= zMin) && (z <= zMax)
        isIn = 1;
    else
        isIn = 0;
    end

    if(distance < radius) && isIn
        isIn = 1;
    else
        isIn = 0;
    end

end % isInZCylinder
```

isAboveYPlane

This subfunction tests if the given coordinates are above the plane sloping in the XZ direction, extending unchanged in Y direction.

```
function [ isIn ] = isAboveYPlane( x, z, params )

    height = str2double(params{3});
    slope = str2double(params{4});

    distance = height + (slope * x);

    if(distance > z)
        isIn = 1;
    else
        isIn = 0;
    end

end % isAboveYPlane
```

isBelowYPlane

This subfunction tests if the given coordinates are below the plane sloping in the XZ direction, extending unchanged in Y direction.

```
function [ isIn ] = isBelowYPlane( x, z, params )

    height = str2double(params{3});
    slope = str2double(params{4});

    distance = height + (slope * x);

    if(distance < z)
        isIn = 1;
    else
        isIn = 0;
    end
```

```
end % isBelowYPlane
```

isInZSpheroid

This subfunction tests if the given coordinates are above the plane sloping in the XZ direction, extending unchanged in Y direction.

```
function [ isIn ] = isInZSpheroid( x, y, z, params )

    rada = str2double(params{3});
    radc = str2double(params{4});
    centerX = str2double(params{5});
    centerY = str2double(params{6});
    centerZ = str2double(params{7});

    distance = ( ( ( x - centerX )^2 + ( y - centerY )^2 ) / rada^2 )
 + ...
               ( ( ( z-centerZ )^2 ) / radc^2 );

    if(distance < 1.0)
        isIn = 1;
    else
        isIn = 0;
    end

end % isInZSpheroid
```

isInYCylinder

This subfunction tests if the given coordinates are within the cylinder described by the given parameters.

```
function [ isIn ] = isInYCylinder(x, y, z, params)

    radius = str2double(params{3});
    centerX = str2double(params{4});
    centerZ = str2double(params{5});
    yMin = str2double(params{6});
    yMax = str2double(params{7});

    distance = sqrt(((x-centerX)^2)+((z-centerZ)^2));

    if (y >= yMin) && (y <= yMax)
        isIn = 1;
    else
        isIn = 0;
    end

    if(distance < radius) && isIn
        isIn = 1;
    else
        isIn = 0;
    end
```

```
end % isInYCylinder
```

generateMask3d

This subfunction generates the image masks for the phantom based on the object specifications.

```
function [ mask ] = generateMask3d( info )

    mask = zeros(info.size, info.size, info.size);

    % First, included objects are added.
    for iObj = 1:length(info.obj(:,1))

        params = info.obj(iObj,:);

        if str2double(params{2}) == 1

            objectMask = zeros(info.size, info.size, info.size);

            objectMask = generateObjectMask(objectMask, params);

            mask(objectMask == 1) = 1;

        end % if include

    end % for iObj

    % Then, excluded objects are subtracted.
    for iObj = 1:length(info.obj(:,1))

        params = info.obj(iObj,:);

        if str2double(params{2}) == 0

            objectMask = zeros(info.size, info.size, info.size);

            objectMask = generateObjectMask(objectMask, params);

            mask(objectMask == 1) = 0;

        end % if include

    end % for iObj

end % generateMask3d
```

generateObjectMask

This subfunction creates an object mask according to the given parameters.

```
function [ objectMask ] = generateObjectMask(objectMask, params)

    dimSize = size(objectMask);
```

```
   % Z Cylinder
   if str2double(params{1}) == 1

       for iZ = 1:dimSize(3)
           for iY = 1:dimSize(1)
               for iX = 1:dimSize(2)

                   objectMask(iY,iX,iZ) = 
isInZCylinder(iX,iY,iZ,params);

               end % iX
           end %iY
       end % iZ

   % Sphere
   elseif str2double(params{1}) == 2

       for iZ = 1:dimSize(3)
           for iY = 1:dimSize(1)
               for iX = 1:dimSize(2)

                   objectMask(iY,iX,iZ) = 
isInSphere(iX,iY,iZ,params);

               end % iX
           end %iY
       end % iZ

   % Above Y Plane
   elseif str2double(params{1}) == 3

       for iZ = 1:dimSize(3)
           for iY = 1:dimSize(1)
               for iX = 1:dimSize(2)

                   objectMask(iY,iX,iZ) = 
isAboveYPlane(iX,iZ,params);

               end % iX
           end %iY
       end % iZ

   % Z Spheroid
   elseif str2double(params{1}) == 4

       for iZ = 1:dimSize(3)
           for iY = 1:dimSize(1)
               for iX = 1:dimSize(2)

                   objectMask(iY,iX,iZ) = 
isInZSpheroid(iX,iY,iZ,params);

               end % iX
```

```
            end %iY
        end % iZ

    % Below Y Plane
    elseif str2double(params{1}) == 5

        for iZ = 1:dimSize(3)
            for iY = 1:dimSize(1)
                for iX = 1:dimSize(2)

                    objectMask(iY,iX,iZ) =
 isBelowYPlane(iX,iZ,params);

                end % iX
            end %iY
        end % iZ

    % Y Cylinder
    elseif str2double(params{1}) == 6

        for iZ = 1:dimSize(3)
            for iY = 1:dimSize(1)
                for iX = 1:dimSize(2)

                    objectMask(iY,iX,iZ) =
 isInYCylinder(iX,iY,iZ,params);

                end % iX
            end %iY
        end % iZ

    end % if object type

end % generateObjectMask
```

generateFirstImage

This function generates the first image of the pair and returns both the image and header.

```
function [ image, hdr ] = generateFirstImage( firstMask,
 secondMask, ...
                                              features, info )

    firstInfo.spacing = info.spacing;
    firstInfo.origin = info.origin;
    firstInfo.size = info.size;
    firstInfo.obj = info.firstObj;
    firstInfo.density = info.firstDensity;
    firstInfo.bgDensity = info.bgDensity;
    firstInfo.noise = info.noise;

    if (info.dims == 3)
```

```
        image = generate3dImage(firstMask, firstInfo);

        if info.addFirstZGradient
            image = addZGradient(image, firstMask, secondMask, ...
                                 info.firstZGradient);
        end

        if info.addFirstFeatures
            image = addFeatures(image, features);
        end

        hdr = generate3dHeader(firstInfo);

    else

        image = 0;
        hdr = 0;

    end

end % generateFirstImage
```

generateSecondImage

This function generates the first image of the pair and returns both the image and header.

```
function [ image, hdr ] = generateSecondImage( firstImg,
 firstMask, ...
                              secondMask, secondFeatures, info )

    secondInfo.spacing = info.spacing;
    secondInfo.origin = info.origin;
    secondInfo.size = info.size;
    secondInfo.obj = info.secondObj;
    secondInfo.bgDensity = info.bgDensity;
    secondInfo.noise = info.noise;

    if info.addSecondFeatures
        secondInfo.density = ...
            calculateSecondDensityWithFeatures(firstImg,
 firstMask, ...
                             secondMask, secondFeatures,
 info.massRatio);
    else
        secondInfo.density = ...
            calculateSecondDensity(firstImg, firstMask, ...
                                   secondMask, info.massRatio);
    end

    if (info.dims == 3)

        image = generate3dImage(secondMask, secondInfo);
        if info.addSecondZGradient
            image = addZGradient(image, secondMask, firstMask, ...
                                 info.secondZGradient);
        end

        if info.addSecondFeatures
            image = addFeatures(image, secondFeatures);
        end

        hdr = generate3dHeader(secondInfo);

    else

        image = 0;
        hdr = 0;

    end

end % generateSecondImage
```

generate3dImage

This subfunction generates a 3D image based on the given mask.

```
function [ image ] = generate3dImage( mask, info )

    image = mask;

    if (info.noise)
        image = imnoise(image, 'gaussian', 0, 0.005);
    end

    imageForeground = image;
    imageBackground = image;

    imageForeground(mask == 0) = 0;
    imageBackground(mask == 1) = 0;

    imageForeground = imageForeground .* info.density;
    imageBackground = imageBackground .* info.bgDensity;

    image = imageForeground + imageBackground;

    % Image is converted to units of HU.
    image = image - 1000;

end % generate3dImage
```

addFeatures

This subfunction generates a 3D image based on the given mask. NOTE: Assumes feature intensity greater than object intensity.

```
function [ image ] = addFeatures( image, features )

    image = max(image,features);

end % generate3dImage
```

generate3dHeader

This function generates header infomation for the 3D image.

```
function [ hdr ] = generate3dHeader ( info )

    hdr.x_dim = info.size;
    hdr.y_dim = info.size;
    hdr.z_dim = info.size;
    hdr.t_dim = 1;

    hdr.x_pixdim = info.spacing / 10;
    hdr.y_pixdim = info.spacing / 10;
    hdr.z_pixdim = info.spacing / 10;

    hdr.x_start = info.origin / 10;
    hdr.y_start = info.origin / 10;
    hdr.z_start = info.origin / 10;

    hdr.byte_order = 'l';

end % generate3dHeader
```

calculateSecondDensity

This function calculates the density of the second image object.

```
function [ secondDensity ] = ...
    calculateSecondDensity( firstImg, firstMask, secondMask,
 massRatio )

    firstImg = firstImg + 1000;

    firstImg(firstMask == 0) = 0;

    firstMass = sum(firstImg(:));

    secondMass = firstMass * massRatio;

    nVoxels = sum(secondMask(:));

    secondDensity = secondMass / nVoxels;

end % calculateSecondDensity
```

calculateSecondDensityWithFeatures

This function calculates the density of the second image object when features are also added. NOTE: For complete correctness, volume of features should be subtracted from nVoxels prior to calculation of secondDensity.

```
function [ secondDensity ] = ...
    calculateSecondDensityWithFeatures( firstImg, firstMask, ...
        secondMask, secondFeatures, massRatio )

    firstImg = firstImg + 1000;
    secondFeatures = secondFeatures + 1000;

    firstImg(firstMask == 0) = 0;
    secondFeatures(secondMask == 0) = 0;

    firstMass = sum(firstImg(:));
    secondFeaturesMass = sum(secondFeatures(:));

    newMass = firstMass - secondFeaturesMass;

    secondMass = newMass * massRatio;

    nVoxels = sum(secondMask(:));

    secondDensity = secondMass / nVoxels;

end % calculateSecondDensityWithFeatures
```

generateFeatureImage

This subfunction creates an image using the given feature file.

```
function [ featureImg ] = generateFeatureImage( image, info )

    % Feature file to be used to generate image is determined.
    if image == 1
        featureInfo.size = info.size;
        featureInfo.obj = info.firstFeatures;
    elseif image == 2
        featureInfo.size = info.size;
        featureInfo.obj = info.secondFeatures;
    else
        errorMsg=['Error - Image not recognized: ' splitLine{1} ...
                  '. Must be 1 or 2'];
        disp(errorMsg);

    end

    % Binary feature image is created based on the feature file.
    if (info.dims == 3)

        disp(['Creating features for image ' num2str(image)]);
        tic
        featureImg = generateMask3d(featureInfo);
        toc

    else

        featureImg = 0;

    end

    % Gaussian noise is added.
    %if (info.noise)
    %    featureImg = imnoise(featureImg, 'gaussian', 0, 0.005);
    %end

    % Intensity is scaled and converted to HU.
    featureImg = featureImg * info.featureDensity;
    featureImg = featureImg - 1000;

end % generateFeatureImage
```

addZGradient

This subfunction adds a gradient along the Z direction based on the difference in mask shapes.

```
function [ image ] = addZGradient( image, mask1, mask2, gradient )

    image = image + 1000;

    maskDiff = mask1 - mask2;

    % First Z slice where masks differ is found.
    firstIndex = find(maskDiff, 1, 'first');
    dims = size(maskDiff);
    xyRes = dims(1) * dims(2);
    zStart = ceil(firstIndex / xyRes);

    % Intensity is increasingly scaled by the gradient as Z increases.
    scaleValue = 1.0;
    for iZ = zStart:dims(3)

        thisSliceMask = mask1(:,:,iZ);
        thisSliceScale = thisSliceMask .* scaleValue;
        image(:,:,iZ) = image(:,:,iZ) .* thisSliceScale;
        scaleValue = scaleValue - gradient;

    end

    image = image - 1000;

end % addZGradient
```

Published with MATLAB® R2016a