Teng-Jui Lin
Professor Neils
BIOEN 217
12 Dec 2020

3D Grayscale Volumetric Image Slicing

Project Summary

Three-dimensional (3D) grayscale volumetric images are stacks of two-dimensional images that can provide an extra dimension of information that have applications in science and medicine, such as medical and microscopic imaging. For example, computer topography (CT) and magnetic resonance imaging (MRI) scans utilize 3D volumetric data to visualize internal organs such as brain and heart to facilitate diagnosis. The 3D images are stored as slices along the coordinate planes, therefore retrieving slices along the coordinate planes are easy to perform. Slices along an arbitrary oblique plane, however, require more elaborate algorithms to perform. In this project, a MATLAB oblique slicing function is developed for 3D grayscale volumetric images using built-in rotation function.

While 3D visualization and slicing algorithm is complex, 2D oblique slicing is simpler and can provide insights for the 3D case. The idea behind the algorithm is to rotate the image so that the oblique line becomes parallel to the *x*-coordinate, as shown in Figure 1. MATLAB built-in function `imrotate()` provides convenient way to rotate the image about its center. The angle of rotation is calculated by taking the inverse tangent of the slope of the line connecting the user-defined points. The user-defined points are also rotated using the rotation matrix to locate the slice through the y-coordinate. The rotation of image generates black pixels around the image. To extract only the parts of the image, the intersections of the line and the image edge are calculated using the line function and rotated using the same procedure. The pixels between the two points are the target oblique slice.

Similarly, 3D oblique slicing can be done by rotating the volumetric image so that the oblique plane is parallel to the *xy*-plane, as shown in Figure 2. The algorithm first calculates the rotational axis by finding the line that is intersected by the user-defined plane and the *xy*-plane by taking the cross product of the normal vectors of the planes. It then calculates the angle between the two planes that characterizes the angle of rotation. MATLAB built-in function `imrotate3()` rotates the volumetric data along the axis, giving slices that are parallel to the user-defined oblique plane. The algorithm then rotates the user-provided point and uses the *z*-coordinate of the rotated point to extract the target oblique slice. The oblique slicing function for 3D grayscale volumetric data can be applied to microscopy images as in Figure 3.

In conclusion, the project presents two packaged functions for oblique slicing of 2D and 3D images. The main problem solving approaches are to extract algorithmic patterns from lower dimensions and to reduce complex problem of oblique slicing to a simpler problem of slicing along coordinate axis/plane. Since oblique slice inputs characterized by plane parameters can be difficult to visualize and requires trail and error to get the right slice, a graphic-user-interface-guided slicing method can be a direction of future work.

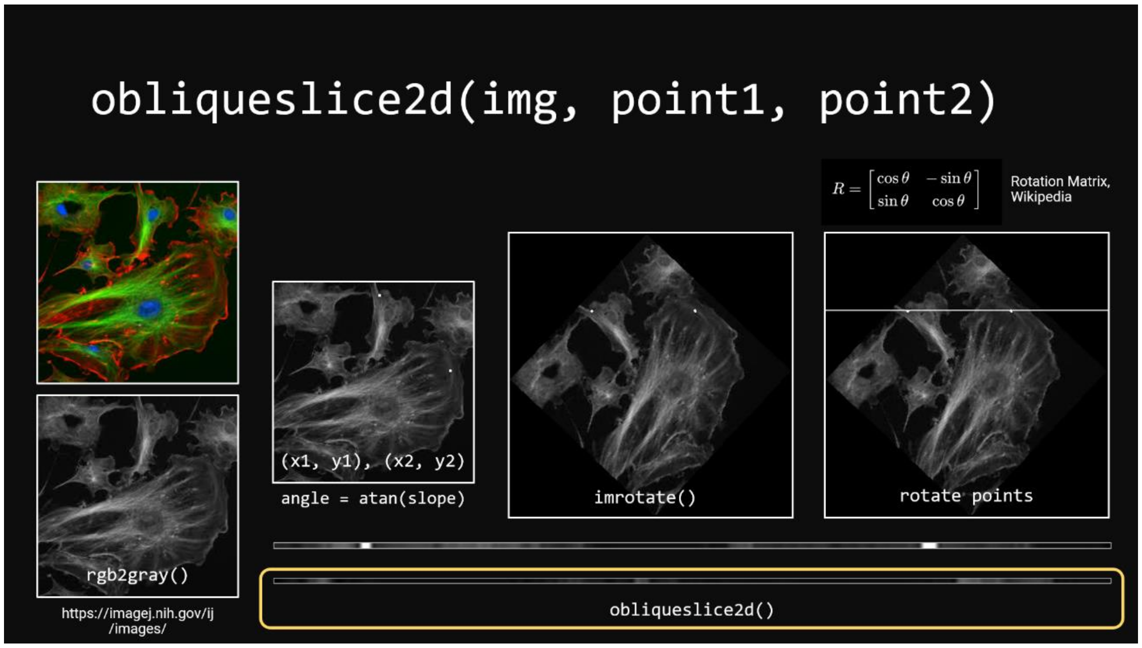
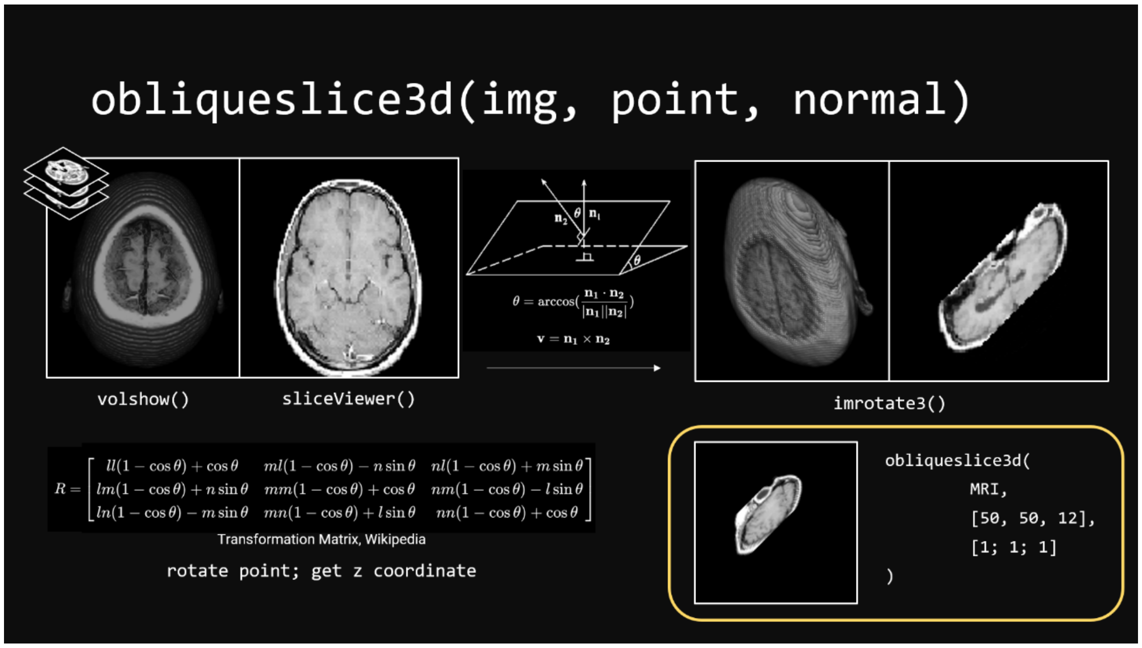

obliqueslice2d(img, point1, point2)
$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
Rotation Matrix, Wikipedia
rgb2gray()
https://imagej.nih.gov/ij/images/
(x1, y1), (x2, y2)
angle = atan(slope)
imrotate()
rotate points
obliqueslice2d()

Figure 1: 2D Oblique Slicing Algorithm*. `obliqueslice2d` takes three arguments, including a 2D grayscale image and two points to establish an oblique line. The two points provided by the user are visualized by the white pixels. The function returns a row vector that represents the oblique slice. The algorithm calculates the angle of rotation about the center of the image by the slope of the line created by connecting the user-defined points and rotates the image and the user defined points. The target row number of pixels is defined by the y-coordinate of the rotated points. The algorithm removes the black edges generated by image rotation by calculating the intersecting points of the line and the image edge.*



```
obliqueslice3d(img, point, normal)
```

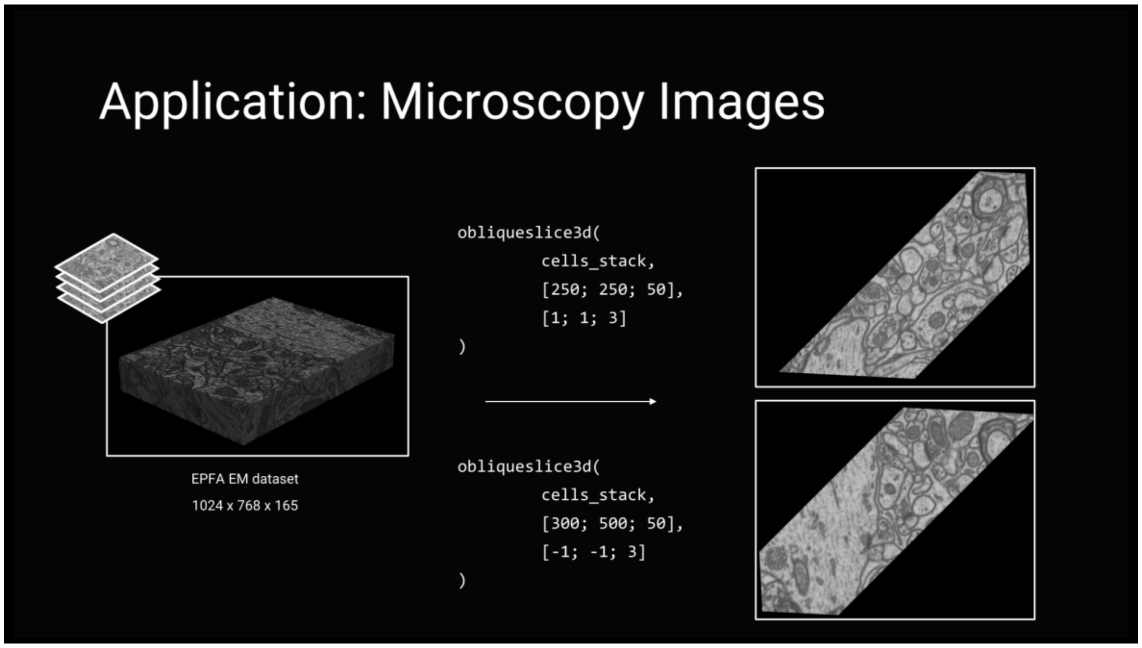
$$\theta = \arccos\left(\frac{\mathbf{n}_1 \cdot \mathbf{n}_2}{|\mathbf{n}_1||\mathbf{n}_2|}\right)$$

$$\mathbf{v} = \mathbf{n}_1 \times \mathbf{n}_2$$

$$R = \begin{bmatrix} ll(1-\cos\theta)+\cos\theta & ml(1-\cos\theta)-n\sin\theta & nl(1-\cos\theta)+m\sin\theta \\ lm(1-\cos\theta)+n\sin\theta & mm(1-\cos\theta)+\cos\theta & nm(1-\cos\theta)-l\sin\theta \\ ln(1-\cos\theta)-m\sin\theta & mn(1-\cos\theta)+l\sin\theta & nn(1-\cos\theta)+\cos\theta \end{bmatrix}$$

```
obliqueslice3d(
        MRI,
        [50, 50, 12],
        [1; 1; 1]
)
```

Figure 2: 3D Oblique Slicing Algorithm. *`obliqueslice3d` takes three arguments, including a 3D grayscale volumetric image, a point in the image, and a normal vector that characterizes an oblique plane. The function returns a matrix that represents the oblique slice. The algorithm calculates the angle theta between the user defined-plane and the xy-plane and rotates the volumetric data about the line that intersects the user-defined plane and the xy-plane by angle theta. The algorithm then rotates the user-provided point, where the layer defined by the z-coordinate of the rotated point is the target oblique slice layer.*



Application: Microscopy Images

```
obliqueslice3d(
        cells_stack,
        [250; 250; 50],
        [1; 1; 3]
)
```

```
obliqueslice3d(
        cells_stack,
        [300; 500; 50],
        [-1; -1; 3]
)
```

Figure 3: Application of `obliqueslice3d` on Microscopy Volumetric Images.

MATLAB Code

The MATLAB code for the project is presented in this section. There are in total four files. `obliqueslice2d.m` is a function file that implements oblique slicing through 2D grayscale image. `obliqueslice3d.m` is a function file that implements oblique slicing through 2D grayscale volumetric image. `Teng_Jui_Lin_Final_Project` gives one demonstration for each function. `demo_images.m` stores the code for drawing images that appeared on the presentation.

File 1: `obliqueslice2d.m`

```
function [oblique_slice] = obliqueslice2d(img, point1, point2)
% Returns a vector that represents the oblique slice of a
% 2D grayscale image.

% aliasing coordinates
x1 = point1(1);
x2 = point2(1);
y1 = point1(2);
y2 = point2(2);

% argument check
if length(size(img)) > 2
    error('Support only gray-scale 2D image.')
end
if x1 == x2 && y1 == y2
   error('Given two identical points.')
end

% slicing along x-/y-axis
if x1 == x2
    oblique_slice = img(:, x1);
    return
end
if y1 == y2
    oblique_slice = img(y1, :);
    return
end

% chracterize the oblique line
slope = (y2-y1) / (x2-x1);
angle = atand(slope);
y = @(x) slope*x - slope*x1 + y1;
```

```
% find the points intersecting the oblique line at img edge
[xlen, ylen] = size(img);
yvals = y(1:xlen);
x_slice_range_vals = find(yvals > 0.5 & yvals < ylen-0.5);
y_slice_range_vals = yvals(x_slice_range_vals);
point1_edge = round([x_slice_range_vals(1); y_slice_range_vals(1)]);
point2_edge = round([x_slice_range_vals(end);
y_slice_range_vals(end)]);

% rotate the original img
rotated_img = imrotate(img, angle);
[rot_xlen, rot_ylen] = size(rotated_img);

% rotate the points at edge
R = @(theta) [cosd(theta), -sind(theta); sind(theta), cosd(theta)];
rotated_point1_edge = round(R(-angle) * (point1_edge - [xlen/2;
ylen/2]) + [rot_xlen/2; rot_ylen/2]);
rotated_point2_edge = round(R(-angle) * (point2_edge - [xlen/2;
ylen/2]) + [rot_xlen/2; rot_ylen/2]);

% find oblique slice on rotated img
oblique_slice = rotated_img(rotated_point1_edge(2),
rotated_point1_edge(1):rotated_point2_edge(1));
end
```

File 2: `obliqueslice3d.m`

```
function [oblique_slice, rot_img] = obliqueslice3d(img, point, normal)
% Returns a 2D matrix (grayscale image) that represents the oblique
slice
% of 3D grayscale volumetric image data.

% argument check
[xlen, ylen, zlen] = size(img);
img_size = [xlen; ylen; zlen];
if ~all(point >= 1) || ~all(point <= img_size)
    error('coordinate of point out of bound.')
end
xy_plane_n = [0, 0, 1];
if isequal(normal, xy_plane_n) || isequal(normal', xy_plane_n)
    oblique_slice = img(:, :, point(3));
    return
end

% plane params
plane_angle = @(n1, n2) acosd(dot(n1, n2)/(norm(n1)*norm(n2)));
angle = plane_angle(normal, xy_plane_n);  % angle between xy-plane and
user-defined plane
v = cross(normal, xy_plane_n);  % direction of the line intersecting
xy-plane and user-defined plane
v = v/norm(v);  % find unit vector (required by rot. matrix R)

% rotating img
rot_img = imrotate3(img, angle, v);
[rot_xlen, rot_ylen, rot_zlen] = size(rot_img);

% rotating point
R3d = @(l, m, n, theta) [
    l*l*(1-cos(theta))+cos(theta), m*l*(1-cos(theta))-n*sin(theta),
n*l*(1-cos(theta))+m*sin(theta)
    l*m*(1-cos(theta))+n*sin(theta), m*m*(1-cos(theta))+cos(theta),
n*m*(1-cos(theta))-l*sin(theta)
    l*n*(1-cos(theta))-m*sin(theta), m*n*(1-cos(theta))+l*sin(theta),
n*n*(1-cos(theta))+cos(theta)
    ];
R = @(v, theta) R3d(v(1), v(2), v(3), theta);
```

```
rot_point = R(v, angle) * (point - [xlen/2; ylen/2; zlen/2]) +
[rot_xlen/2; rot_ylen/2; rot_zlen/2]

% get oblique slice on rotated img
oblique_slice = rot_img(:, :, round(rot_point(3)));
end
```

File 3: `Teng_Jui_Lin_Final_Project`

```
% Teng-Jui Lin
% BIOEN 217
% 8 Dec 2020
% Final Project

%% `obliqueslice2d` demo
clear; clc; close all

img = rgb2gray(imread('cell.jpg'));
point1 = [120, 15];
point2 = [10, 100];

oblique_slice = obliqueslice2d(img, point1, point2);
imshow(oblique_slice, 'InitialMagnification', 2000)
imwrite(oblique_slice, '3d-0.jpg')


%% `obliqueslice3d` demo
clear; clc; close all
load mri
MRI = squeeze(D);
point = [50; 50; 12];
normal = [0; 0; 1];
oblique_slice = obliqueslice3d(MRI, point, normal);
imshow(oblique_slice, map)
imwrite(oblique_slice, 'mri-0.jpg')
```

File 4: `demo_images.m`

```
%% 2D slicing figures
% original img
img = rgb2gray(imread('cell.jpg'));
[xlen, ylen] = size(img);
imshow(img)
imwrite(img, '2d-1.jpg')

%%% demo point pairs
% point1 = [120; 15];
% point2 = [10; 100];
point1 = [120; 15];
point2 = [200; 100];
% point1 = [120; 200];
% point2 = [10; 100];
% point1 = [100; 200];
% point2 = [200; 100];
% point1 = [100; 15];
% point2 = [120; 200];
% point1 = [10; 100];
% point2 = [200; 120];
% point1 = [10; 100];
% point2 = [10; 120];
% point1 = [10; 100];
% point2 = [200; 100];
% point1 = [10; 10];
% point2 = [10; 10];

% aliasing coordinates
x1 = point1(1);
x2 = point2(1);
y1 = point1(2);
y2 = point2(2);

% argument check
if length(size(img)) > 2
    error('Support only gray-scale 2D image.')
end
if x1 == x2 && y1 == y2
   error('Given two identical points. ')
end
```

```
% slicing along x-/y-axis
if x1 == x2
    oblique_slice = img(:, x1);
end
if y1 == y2
    oblique_slice = img(y1, :);
end

% chracterize the oblique line
slope = (y2-y1) / (x2-x1);
angle = atand(slope);
y = @(x) slope*x - slope*x1 + y1;

% find the points intersecting the oblique line at img edge
yvals = y(1:xlen);
x_slice_range_vals = find(yvals > 0.5 & yvals < ylen-0.5);
y_slice_range_vals = yvals(x_slice_range_vals);
point1_edge = round([x_slice_range_vals(1); y_slice_range_vals(1)])
point2_edge = round([x_slice_range_vals(end);
y_slice_range_vals(end)])

% magnify the designated points as white dots
box = 1;
img(y1-box:y1+box, x1-box:x1+box) = 255;
img(y2-box:y2+box, x2-box:x2+box) = 255;
imshow(img)
imwrite(img, '2d-2.jpg')

% rotate original img
rotated_img = imrotate(img, angle);
[rot_xlen, rot_ylen] = size(rotated_img);
figure
imshow(rotated_img)
imwrite(rotated_img, '2d-3.jpg')

% rotate the points at edge
R = @(theta) [cosd(theta), -sind(theta); sind(theta), cosd(theta)];
rotated_point1_edge = round(R(-angle) * (point1_edge - [xlen/2;
ylen/2]) + [rot_xlen/2; rot_ylen/2])
```

```
rotated_point2_edge = round(R(-angle) * (point2_edge - [xlen/2;
ylen/2]) + [rot_xlen/2; rot_ylen/2])

% find oblique slice on rotated img
oblique_slice = rotated_img(rotated_point1_edge(2),
rotated_point1_edge(1):rotated_point2_edge(1));
rotated_img(rotated_point1_edge(2), :) = 255;
imshow(rotated_img)
imwrite(rotated_img, '2d-4.jpg')

% oblique slice with reference point
figure
imshow(oblique_slice, 'InitialMagnification', 2000)
imwrite(oblique_slice, '2d-5.jpg')


%% 3D slicing figures
clear; clc; close all
% sample data
load mri
img = squeeze(D);
[xlen, ylen, zlen] = size(img);
sliceViewer(img)
title('MRI Slices')

% define coord planes
xy_plane_n = [0, 0, 1];
plane_angle = @(n1, n2) acosd(dot(n1, n2)/(norm(n1)*norm(n2)));

% plane params
point = [50, 50, 12];
normal = [1; 1; 1];
angle = plane_angle(normal, xy_plane_n);
v = cross(normal, xy_plane_n);

% rotating img
rot_img = imrotate3(img, angle, v);
[rot_xlen, rot_ylen, rot_zlen] = size(rot_img);

figure, sliceViewer(rot_img)
title('Rot. MRI Slices')
```

```
figure
vol_MRI = volshow(img, 'CameraPosition', [0 1 5], 'BackgroundColor',
[0 0 0]);
figure
vol_rot_MRI = volshow(rot_img, 'CameraPosition', [0 1 5],
'BackgroundColor', [0 0 0]);


%% Extract cell data
clear; clc; close all
t = Tiff('cells.tif');

% get tiff data
first_img= read(t);
[height, width] = size(first_img);
num_img = 165;

% setup empty cells
cells = zeros(height, width, num_img);

for k = 1:num_img
    setDirectory(t, k);
    cells(:, :, k) = read(t);
end
cells = uint8((cells));


%% obliqueslice3d on cell volume
point = [250; 250; 50];
normal = [1; 1; 3];
[one_slice, stack] = obliqueslice3d(cells, point, normal);
imshow(one_slice)
figure, sliceViewer(stack)
imwrite(one_slice, 'cells-1.jpg')

% show as volume
volshow(cells, 'BackgroundColor', [0, 0, 0])
```