Problem 1 [5 points]: Chapter 2, Exercise 12 page 37

12. The previous exercise used rotation matrices in two dimensions. Now, we explore the speed of matrix operations in a computer graphics model in three dimensions. In Figure 1, we see a model of Yoda. The tessellation contains 33,862 vertices. Let $V$ be a matrix with 3 columns and 33,862 rows, where row $i$ contains the $x$, $y$, and $z$ coordinates of the $i$th vertex in the model. The image can be translated by $t$ units in the $y$ direction by using a translation matrix $T$ where

$$
T = \begin{pmatrix}
0 & t & 0 \\
0 & t & 0 \\
0 & t & 0 \\
\end{pmatrix}.
$$

If $V_t = V + T$, then $V_t$ contains the vertex information for the model after a translation of $t$ units in the $y$-direction.

Figure 1: A model of Yoda created with 33,862 vertices. Model created by Kecskemeti B. Zoltan.

Download the files yoda.m and yodaposelong.mat from the web page. Run the file yoda.m in MATLAB. You will see an animation of the model being translated in space using matrix addition.
(a) The image can be rotated by $\theta$ radians about the $y$-axis by multiplying $V$ on the right by $R_y$ where

$$R_y = \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix}.$$ 

Edit the code to continuously rotate the image by $\pi/24$ radians until the image has made one full rotation about the $y$ axis.

(b) How many multiplications are performed when you use matrix multiplication (with $R_y$) to rotate the image once by $\pi/24$ radians? (Remember that $V$ is a $33862 \times 3$ matrix.) Keep this in mind as you watch how fast MATLAB performs the calculations and displays the results.

**Solution:**

a) The following code rotates Yoda through one full turn about the $y$ axis:

```matlab
% YODA Load low or high resolution model of Yoda character.
% and rotate model about x, y, or z axis.

% Code created by Tim Chartier - June 2006
% Models created by Kecskemeti B. Zoltan.
% Images courtesy of Lucasfilm LTD.

%% Load the model/tessellation information
load yodapose_low
% load yodapose % uncomment to use higher resolution model of Yoda

%% Create initial plot
Vt = V;
clf
patch('Vertices',Vt,'Faces',F3,'FaceColor',[.76 .87 .78]);
patch('Vertices',Vt,'Faces',F4,'FaceColor',[.76 .87 .78]);
axis tight equal vis3d
drawnow

%% Create rotation matrix
slides = 48;

% Create the rotation matrix
theta = pi/24;
Q = [cos(theta) 0 -sin(theta); 0 1 0; sin(theta) 0 cos(theta)];
axisValues = axis; % Get the min and max values on each axis

%% Animate rotation
for i=1:slides
    Vt = Vt*Q;
end
```
cla
patch('Vertices',Vt,'Faces',F3,'FaceColor',[.76 .87 .78]);
patch('Vertices',Vt,'Faces',F4,'FaceColor',[.76 .87 .78]);
axis(axisValues)
drawnow
end

b) To multiply one row of the 33862 x 3 matrix V by one column of the 3 x 3 matrix
R_y requires 3 multiplications. Each of the 33862 rows of V must be multiplied by
each of the 3 columns of R_y, so the total number of multiplications is

\[3 \times 3 \times 33862 = 304,758.\]

Problem 2 [5 points]: Chapter 2, Exercise 14 page 38

14. In this exercise, we will create a fractal coastline. Fractals have many uses, and here we see
how they can be used to create qualitatively realistic-looking pictures.

We will use the following iterative algorithm to create 2D fractal landscapes:

0. Begin with one straight line segment.
1. For each line segment in the current figure, find the midpoint, denoted by a solid diamond
   in the picture below.

   ────

2. Create a new point by moving a random amount in the x and y directions from that
   midpoint as seen below. The size of the random displacement will be adjusted at each
   iteration.

   ──

3. Connect the endpoints of the original line with the new point.

4. If the picture looks good then stop, else adjust the random displacement size and go to
   Step 1.

You will need to determine a suitable range for the random displacement at each iteration to obtain
a realistic-looking picture. One such choice resulted in the figures below.
The following code will create a fractal coastline:

% Define the 0th (starting) iterate
xValues = [0 0];
yValues = [0 1];

% Plot the 0th iterate.
plot(xValues,yValues)
axis equal

scale = .5;

For the exercise, you need not color the land and water masses (although you may do so by using the fill command), but simply generate a realistic looking coastline. If your implementation stores the \( x \) values and \( y \) values for points on the fractal coastline in the vectors \( xValues \) and \( yValues \), respectively, then the MATLAB commands:

\[
\text{plot}(xValues,yValues) \\
\text{axis equal}
\]

will plot the fractal coastline with a 1:1 aspect ratio for the axes.

(a) Write a program to create fractal coastlines using the algorithm above.

(b) Describe how your implementation adjusts the range of the random numbers used for displacements in Step 2 at each iteration.

(c) Create at least two fractal coastlines with your code.

**Solution:**

a) The following code will create a fractal coastline:

% Define the 0th (starting) iterate
xValues = [0 0];
yValues = [0 1];

% Plot the 0th iterate.
plot(xValues,yValues)
axis equal

scale = .5;
% Subdivide until user is satisfied.
flag = input('Continue? (1=yes, 0=no): ');

while flag==1,
    scale = scale*.5; % Adjust scale.
    % If too large, lines may cross.
    % If too small, will not see any change.
    n = length(xValues); % number of points
    xValuesTmp = xValues(1); % Store new points in temp arrays
    yValuesTmp = yValues(1);
    for i=2:n,
        xmid = .5*(xValues(i) + xValues(i-1));
        ymid = .5*(yValues(i) + yValues(i-1));
        xnew = xmid + (2*rand-1)*scale; % Add random amount between
        ynew = ymid + (2*rand-1)*scale; % -scale and +scale
        xValuesTmp = [xValuesTmp xnew];
        yValuesTmp = [yValuesTmp ynew];
        xValuesTmp = [xValuesTmp xValues(i)];
        yValuesTmp = [yValuesTmp yValues(i)];
    end;
    xValues = xValuesTmp;
    yValues = yValuesTmp;

    plot(xValues,yValues); % Plot new coastline
    axis equal

    flag = input('Continue? (1=yes, 0=no): '); % Ask user whether to continue.
end;

b) The above code uses random displacements in x and y of size $0.5^{1+l}$ at iteration $l = 1, 2, ...$

c) The above code created the following (rather jagged) coastline:

![Coastline Diagram]
By reducing the size of the random displacement, we can make a smoother looking coastline. The following coastline was generated by replacing the line

\[ \text{scale} = \text{scale}\times0.5 \text{ by } \text{scale} = \text{scale}\times0.3. \]