Matlab Project: Computer Graphics

Goal: Explore the use of linear transformations in computer graphics

Introduction

In class we have seen that a linear transformation is a function which maps vectors into vectors, while preserving the operations of vector addition and scalar multiplication. Sections 1.8–9 introduced the basic concepts, and section 2.7 introduced their use in computer graphics. In this project we will explore these ideas using matrix multiplication and basic graphics commands from MATLAB, illustrating how these transformations might be used in developing higher-level graphics software. Note that we're just scratching the surface here—for more detail, consider taking the Computer Graphics course CS452/EE465.

Work through the steps below, typing the commands listed in **this font**, and write all answers on the answer sheet as requested.

Step 1: Exploration

A figure in the x-y plane can be specified by a sequence of points to be connected by straight lines. For example, the corners (vertices) of the unit square are the vectors

$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{v}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{v}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

It is convenient to store these as the *columns* of a matrix $V = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 \end{bmatrix}$. In MATLAB, an easy way to define this figure is to enter the points as the rows (for easy typing) and then transpose the result using the single quote **'** as follows:

```
V = [ 0 0; 1 0; 1 1; 0 1; 0 0]'
```

(note that we've added a copy of $\mathbf{v}_1$ at the end to close off the square). Then plot the square using:

```
line( V(1,:), V(2,:) );
axis([-1,3,-1,3]); axis equal
```

Here, **V(1,:)** means the vector of x coordinates in the first row (i.e., **1** for row 1 and **:** for all columns), and **V(2,:)** is the vector of y coordinates in the second row. The **axis** commands set the axis scales so the figure looks right.

Storing the points as *columns* of a matrix (rather than as rows) is significant as follows. As detailed in section 1.9, any linear transformation can be represented by its standard matrix A: the image (output) of the transformation applied to a given vector $\mathbf{v}$ is just the matrix-vector product $A\mathbf{v}$. Therefore, if the points defining a figure are stored as the *columns* of a matrix $V = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_k \end{bmatrix}$, their images under the transformation are given by the matrix-matrix product $AV = \begin{bmatrix} A\mathbf{v}_1 & A\mathbf{v}_2 & \dots & A\mathbf{v}_k \end{bmatrix}$. To illustrate this, find matrices for the following three linear transformations:

- matrix D: scaling by the factor 2 in both directions
- matrix R: rotation (counterclockwise) by the angle 15° about the origin
- matrix S: vertical shear by the factor $k = -0.5$ (see Table 3, page 86).

Write these matrices on the answer sheet, and then define them in MATLAB. Note that the MATLAB functions **sin** and **cos** use angles in *radians* (like most mathematical software)—either convert the angle to degrees (to do it accurately, use the MATLAB constant **pi**) or use the MATLAB functions **sind** and **cosd** for angles in degrees. Then plot the images of the unit square under these transformations by the following commands:

```
square = V; hold on;
V = D*square; line( V(1,:), V(2,:), 'Color', 'Red' );
V = R*square; line( V(1,:), V(2,:), 'Color', 'Green' );
V = S*square; line( V(1,:), V(2,:), 'Color', 'Cyan' );
```

Check the figure to verify that multiplying by each of the matrices **D**, **R**, and **S** does what each should. Note also that the *order* of transformations matters: for example,

```
V = R*S*square; line( V(1,:), V(2,:), 'Color', 'Magenta' ); % shear, then rotate
V = S*R*square; line( V(1,:), V(2,:), 'Color', 'Black' ); % rotate, then shear
```

should produce different results (and by now a very cluttered figure). *DO NOT hand in this figure or your Matlab code up to this point.*

Step 2: Two-Dimensional Animation

To produce a moving or changing figure, we can simply redraw it repeatedly after small moves or changes. MATLAB has several ways to do this, which you can read about in the MATLAB **help** browser (search for "animate"). To simplify producing animations for this project, the basics have been coded into a simple function **anifig** which animates a figure (defined by a matrix you supply) by repeatedly applying a linear transformation (given by another matrix you supply) and redrawing the result each time. Download this function from Moodle into a file **anifig.m** and put it in the directory in which you're running MATLAB for this project. Once you've done that, you can learn how to use it by typing **help anifig**. For example, the commands

```
V = square; clf;
V = anifig( V, R, 24 );
```

apply the rotation matrix R defined above to the unit square 24 times to rotate it a full circle around the origin. Try it now—to slow the rotation down you could use more applications of a smaller angle. Experiment with the command **anifig** until you are sure you know how it works.

Now construct a MATLAB *script file* (a file containing MATLAB commands to run, just as you might type them at the MATLAB prompt) to do a simple animation as follows. First, use the MATLAB editor (choose **file—new**) to create a new M-file. At the top of the file put *your* name in comments (lines starting with **%**) in the following format:

```
% MA339 Project: Computer Graphics
% David R. Bowman ⟵ replace with your name
```

Save your script using your Clarkson login name as the filename. For example, David would save his script as the file **bowmandr.m**. Then add MATLAB commands to the file to do the animation as described below. Note that you can save the script in steps as you write it, and test it by typing the filename into MATLAB to run it (for example, **bowmandr**).

Put commands into your script to do the following:

(a) Define the figure. Either write your own code to set up an array **V** with columns which specify a simple figure of your choice, or download the script **joe.m** from Moodle and use **V = joe;**.

(b) Convert the figure to *homogeneous coordinates* so we can apply translations (see section 2.7). An easy way to do this is to add a third row consisting of **1**s using the commands

```
[m,n] = size(V); V = [V; ones(1,n)];
```

(c) Rotate the figure 360° about the origin in N steps (choose N large enough so the motion is smooth and not too fast—something like $N = 300$ might be reasonable). This will require constructing the rotation matrix R_0 for the angle $\theta = 2\pi/N$. Be sure to use MATLAB to compute the matrix entries directly—otherwise rounding the entries will seriously degrade it. Note that the matrix must be 3×3 (see example 5 on page 160). Write the matrix R_0 on the answer sheet *in terms of the angle* θ.

(d) Translate the figure (smoothly) from the origin to the point $\mathbf{p} = (X, Y)$ where $X = 20$ and $Y = 12$. To do this smoothly, translate N times by $(h, k) = (X/N, Y/N)$. Write the corresponding translation matrix T on the answer sheet *in terms of* h *and* k.

(e) Rotate the figure by 180° about the point $\mathbf{p} = (X, Y)$. Do this in N steps, each with angle $\phi = \pi/N$. To construct the corresponding rotation matrix $R_{\mathbf{p}}$ see the practice problem (page 165) or exercise 7 (page 166) of section 2.7. Write the matrix $R_{\mathbf{p}}$ on the answer sheet *in terms of the angle* ϕ *and the coordinates* (X, Y) *of the point* $\mathbf{p}$.

(f) Include other transformations in your animation if you wish.

When your script is completed properly, running it should show your figure first rotating 360° about the origin, then translating to the point $\mathbf{p}$, and finally rotating 180° about that (plus whatever additional transformations you add—if any). Print a copy of the file and staple it to the answer sheet, and upload the file to Moodle by the due date.

Step 3: Higher-Dimensional Animation [extra credit]

You may add commands to your script to animate transformations of a higher-dimensional figure. The script **monolith.m** (available on Moodle) gives a simple three-dimensional figure; rotating it about various axes gives an interesting animation. Or you might design your own figure, and transform it in various ways. Note that **anifig** works with 3D figures by plotting only the first two coordinates (which corresponds to projecting the figure onto the x-y plane); if you want to do translations, you'll need to convert to *homogeneous 3D coordinates* (see page 162).

For another interesting challenge, note that the unit square in two dimensions has vertices

$$(0,0), (1,0), (0,1), (1,1)$$

and the corresponding square figure in three dimensions (the unit cube) has vertices

$$(0,0,0), (1,0,0), (0,1,0), (1,1,0), (0,0,1), (1,0,1), (0,1,1), (1,1,1).$$

The analogous unit square figure in *four* dimensions is known as the *hypercube*. You can draw it in MATLAB (projected onto two dimensions) and use **anifig** to animate various transformations of it (for example, rotations about various axes).

If you do a higher-dimensional animation, include that in your script and briefly describe it on the answer sheet.

Hand in the following:

1. The completed answer sheet. *DO NOT hand in the instructions (I already have a copy).*
2. A printed copy of your MATLAB script (M-file) containing the commands you used for Step 2 (and Step 3—if you did it). *Staple this to the answer sheet. DO NOT include* **anifig.m**.
3. Upload a copy of your MATLAB script to Moodle by the due date.

Matlab Project: Computer Graphics **Name:**_____

ANSWER SHEET

Step 1: Exploration

Matrices D, R, and S:

Step 2: Two-Dimensional Animation

Matrices R_0, T, and $R_{\mathbf{p}}$:

Step 3: Higher-Dimensional Animation [extra credit]

Description of your figure and animation (if you did this):

M-file: Staple a copy of your script for Step 2 (and Step 3—if you did it) to this answer sheet. Also upload a copy of your script to Moodle by the due date.