"Computer Graphics"
Lay 2.7 (and some of 1.9)

Oct. 3, 2013

1 2D coordinates

Let's say that you want to store a two-dimensional image as data. One way to do this is to treat the image as a set of points ("vertices") in the x-y plane and record the coordinates of each point. Recall from a previous lecture that you can treat a point in the plane as a vector–if the x coordinate of the point is a number r, we make the first entry of the vector r, and so on.

In general, you might not want to store every point of an image, but instead to store some smaller sequence of points $p_1, \ldots, p_n$ and draw lines between subsequent points–a line between p_1 and p_2, another between p_2 and p_3, etc. Think of a "connect-the-dots" puzzle [1] if you need a way to imagine how this works [2]. We will generally keep this method of image construction in mind in what follows.

For an image which consists of a large number n of points, we need n vectors, each with two entries. It is easiest from our perspective to store this information as a $2 \times n$ matrix, each of whose columns is a vector containing the coordinates of a single point.

Example 1.1. Consider the triangle defined by the vertices with coordinates $(0, 1)$, $(0, 0)$, and $(1, 0)$. We store this information as the matrix

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

[1] Do people still do these?

[2] See `https://en.wikipedia.org/wiki/Connect_the_dots` for a nice picture

1.1 Transforming Points

Generally we want to do more than store images; we want to manipulate them in some way. Most of the ways for transforming 2D points that we care about are linear transformations. That is, the image is transformed by performing the same linear transformation on each point of the image. Lay covered a number of ways to do this in Section 1.9.

Example 1.2. Reflections. Imagine we want to "flip" the image about the y-axis–i.e., reflect each point about the y-axis. This is done by multiplying by the matrix

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

That is, for every point of the image, we write the coordinates (x, y) of the point as the vector

$$\begin{bmatrix} x \\ y \end{bmatrix};$$

after reflection, our image will instead have a point at the coordinates

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}.$$

Think about why a reflection has this form: when we reflect about the y axis, what we are really doing is changing replacing each point of the image which has a positive x coordinate with the corresponding point with negative x coordinate, and vice-versa. You can also reflect about the x-axis, or about other lines through the origin, and represent this as multiplication by a matrix[3].

Example 1.3. Rotations. Imagine the transformation which acts on points by rotating them counter-clockwise, through an angle θ, about the origin (that is, as though the axis were sticking through the page at the origin). A little geometric thinking or sketching will convince you that this transformation is linear. Recall that the way to find the matrix is by seeing how it acts on the vectors $\mathbf{e}_1$ and $\mathbf{e}_2$–that is, the points $(1, 0)$ and $(0, 1)$.

You can see with some thought that the point $(1, 0)$ is mapped to $(\cos(\theta), \sin(\theta))$, and the point $(0, 1)$ is mapped to $(-\sin(\theta), \cos(\theta)$. I will draw a picture in

[3]In general, reflections about a line which doesn't pass through the origin are not given by a linear transformation–why is this?

class to convince you of this; there is a very nice picture in your textbook which I will not reproduce here (see Lay 1.9, Example 3, and the figure below). Therefore, the matrix of the transformation is

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

When we want to act on a whole image via some linear transformation, we have to multiply each vector corresponding to each point in the image by the matrix A of the transformation. Fortunately, our matrix notation for points takes care of this. That is, if D is the $2 \times n$ matrix whose columns are the vectors of coordinates of the points of the image, we find the matrix with the coordinates of the transformed image via the product

$$AD.$$

Example 1.4. Say our image consists of the two points $(1, 1)$ and $(-1, 1)$. We write the matrix which stores their coordinates as

$$D = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}.$$

If we rotate counterclockwise by an angle of $45°$, the matrix of our transformation is

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix},$$

so the matrix with the coordinates of our transformed image is

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}.$$

The reason that linear transformations are so nice is because of the way they compose. Say we want to transform our image by rotating, then reflecting, then rotating by a different angle, then reflecting again. After these four linear transformations, the image will look as though it had only undergone one linear transformation, and we find its matrix by multiplying the matrices of the different transformations.

1.2 Translations and homogeneous coordinates

Another natural way to change an image is by translating–that is, shifting the image. As we saw in a past lecture, this type of transformation is not a linear transformation (it corresponds to adding a nonzero vector **a** to our points, and so it does not fix the origin). Since linear transformations have such nice properties and are easy to handle, this is an obstacle for us.

However, there is actually a nice trick which lets us treat translations as linear transformations. The idea is to pretend that our two-dimensional points actually lie in $\mathbb{R}^3$ by using what are called **homogeneous coordinates**. If a point in the plane has coordinates (x, y), it has homogeneous coordinates $(x, y, 1)$

Example 1.5. Consider the function $T : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ defined by

$$T(\mathbf{x}) = \mathbf{x} + \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}.$$

In homogenous coordinates, we can write this transformation as multiplication by the matrix

$$\begin{bmatrix} 1 & 0 & a_1 \\ 0 & 1 & a_2 \\ 0 & 0 & 1 \end{bmatrix}.$$

To see this, notice that

$$[\begin{bmatrix} 1 & 0 & a_1 \\ 0 & 1 & a_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + a_1 \\ y + a_2 \\ 1 \end{bmatrix}.$$

Example 1.6. What do functions T which *are* linear transformations from $\mathbb{R}^2$ to $\mathbb{R}^2$ look like in homogeneous coordinates? They are linear transformations which map $\mathbb{R}^3$ to $\mathbb{R}^3$ by ignoring the third coordinate, and just applying T to the first two coordinates. That is, they are linear transformations with block matrix form

$$\begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix},$$

where the 2×2 matrix A is the standard matrix of the transformation T.

As before, if we want to compose linear transformations in homogeneous coordinates, the matrix of the composed transformation is given by the product of the matrices of each transformation in the composition.

2 3D coordinates

If we want to store the coordinates of points in a 3D space (for instance, if we wanted to build a 3D map of a room), all of the above applies with appropriate modifications. The only real difference is that now points have three coordinates (x, y, z). If we want to make translations linear again, we can introduce the homogeneous coordinates such that the point (x, y, z) is treated as a point of $\mathbb{R}^4$ with coordinates $(x, y, z, 1)$.

We can generalize our notion of homogeneous coordinates by letting the last entry be a value other than one; we say that the point (x, y, z) has homogeneous coordinates (X, Y, Z, H) if $H \neq 0$ and $x = X/H$, $y = Y/H$, etc.

3 Perspective Projection

The idea behind a perspective projection is the following. You want to take a map of a 3d object inside a data matrix (imagine if we had a 3d map of a garden in our computer, for instance), and figure out how to represent it on a 2d screen. The way to do this is to figure out where the viewer is located relative to both the object and the screen, then project onto the screen by drawing rays from the viewer's eye as straight lines from the viewer to the screen. If the ray hits a point of the object, we project it onto the screen at the point where the ray hits the screen (see Figure 6 in Lay). The reason for this is that it preserves the fraction of the total viewing area that objects occupy.

Lay gives a quick derivation of the perspective projection matrix in the case that the viewer is located on the positive z axis at $(0, 0, d)$ and the "screen" is the $x - y$ plane (the plane of all points with coordinates $(x, y, 0)$). The upshot is that an object at a point (x, y, z) is projected to the point $(x^*, y^*, 0)$, and this "ray" recipe gives

$$\frac{x^*}{d} = \frac{x}{d - z}$$

and similarly for y^* (see Figure 6 again to understand why).

Some manipulation gives that this transformation has the matrix (using

homogeneous coordinates)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}. \tag{1}$$