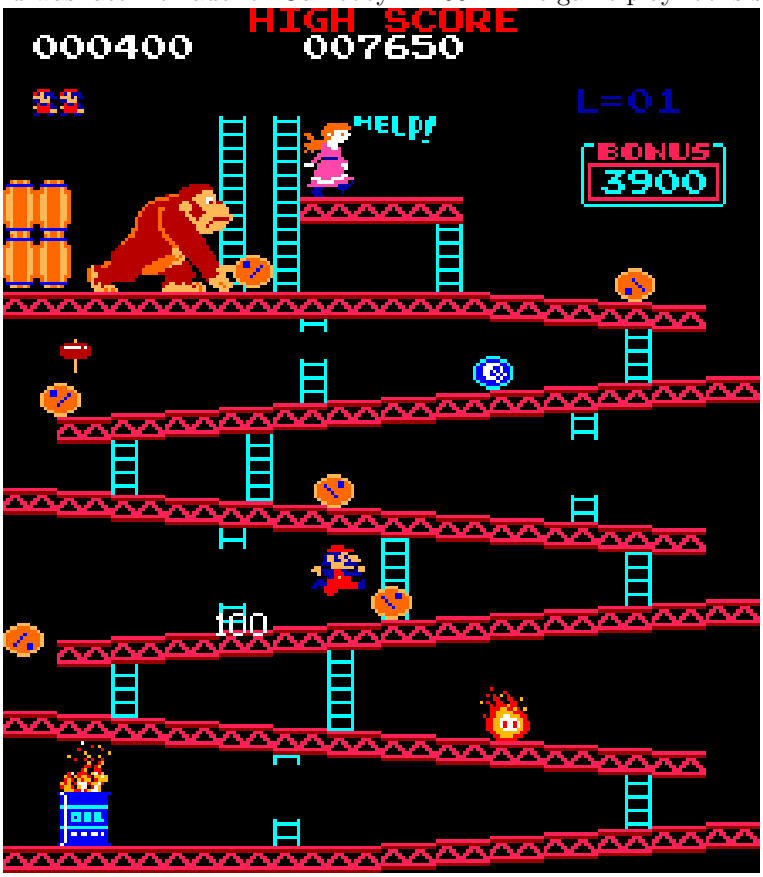
AFFINE TRANFORMATIONS

MTH1035

Semester 1, 2016

Linear algebra is the back-bone to computer graphics. Today, the people and scenery in video games can seem more resolved than real life. Think of a 2D video game you have played- Donkey Kong for example. This game was created for NES in 1981 and was later remade for Gameboy in 1994. The game play looks something like this:



The player controls Mario, who begins the level at the bottom of the screen. He must traverse ladders and avoid obstacles such as rolling barrels, flaming barrels and fireballs in order to reach the top and save Pauline. Throughout the game, Mario moves along a fixed background left, right, up, down. When he climbs up a ladder he can be seen from the back. That aside, every other movement of Mario as he climbs higher to Donkey Kong involves him being either translated in space (Jumping, moving left/right continuously), being reflected (changing horizontal direction), or rotating (for arguments sake, let's pretend that Mario falls down and rotates when he dies). The purpose of this will be to understand simple graphics like those in Donkey Kong given what you have learnt in MTH1030.

Motivation

Graphics are drawn onto co-ordinate space. Let's use the Cartesian co-ordinate system. This is an 'affine co-ordinate space', however in general affine co-ordinate spaces do not require that the axes are perpendicular. Now, we want to

manipulate the images which in essence are made up of individual pixels. Here are some operations we want to be able to do to the images:

1. translate
2. rotate
3. dilate/contract
4. reflect in an arbitrary axis
5. skew

We can use matrices to construct these operations. They then act on each co-ordinate that makes up the image, which are mapped onto a reference grid. Provided a transformation has taken place, the final image will be at least one of the following:

1. In a new location relative to the co-ordinate space (background in Donkey Kong)
2. Rotated
3. Bigger or smaller along one of more axes
4. Flipped (Mirror image)
5. Slanted in some direction

Quaternions

A complex number $a + bi$ can be expressed as a rotation from some point along the x axis, where $a = \cos(\theta)$ and $b = \sin(\theta)$. Quarternions do the equivalent in 3D. The set of quaternions are as follows:

$$\mathbb{H} = a + bi + cj + dk$$

where $a, b, c, d \in \mathbb{R}$. Say we have a number represented by a quaternion, q. The following is true:

1. $Re(q) = \frac{1}{2}(q + \overline{q}) = a$
2. $Im(q) = \frac{1}{2}(q - \overline{q}) = bi + cj + dk$

where

1. $i^2 = j^2 = k^2 = -1$
2. $ij = -ji = k$
3. $\overline{q} = a - bi - cj - dk$

2D rotations

To rotate the point $\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ by some angle θ, we apply matrix R_θ where

$$R_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

R preserves both the length and origin of $\vec{x}$.

2D scaling

Stretching in x

To scale a point $\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$, in the x direction, we apply matrix S_x where

$$S_x = \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix}$$

Noting that:

1. if $a > 0$, x' does not cross the y axis.
2. if $a < 0$, x' is reflected in the y axis.
3. if $a = 0$, $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}$. In other words S_x yields the orthogonal projection in the x direction.

Stretching in y

To scale a point $\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ in the y direction, we apply matrix S_y where

$$S_y = \begin{bmatrix} 1 & 0 \\ 0 & b \end{bmatrix}$$

Noting that:

1. if $b > 0$, x' does not cross the x axis.
2. if $b < 0$, x' is reflected in the x axis.
3. if $b = 0$, $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix}$. In other words S_y yields the orthogonal projection in the y direction.

2D skewing

Skewing involves a combination of stretching and rotations. A matrix K will look something like:

$$K = \begin{bmatrix} a & k_x \\ k_y & b \end{bmatrix}$$

where

$$\begin{bmatrix} a & k_x \\ k_y & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + k_x y \\ by + k_y x \end{bmatrix}$$

2D Translations

You should be familiar with adding M to $\vec{x}$ where $M = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$ to translate a point in space. In other words,

$$\vec{x'} = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

So if we want to apply both a translation *and* a rotation for example, we would first need to apply R_θ to $\vec{x}$, then add M (two separate operations). However it would be much quicker, especially in the context of computers transforming millions of points per second, if this information could be condensed to one operator matrix.

Affine transformations

For such operations we use *Affine Transformations*, which are any transformation that preserves straight lines. This means that points conserve their co-linearity, and the ratio of distance between points stays the same. One consequence of this is that the midpoint in the old co-ordinates is mapped to the midpoint in the new co-ordinates. Linear transformation matrices act on a 2D vector $[x, y]$ to get $[x', y']$. They involve the scaling and addition/subtraction of vectors $\vec{v_1}$ and $\vec{v_2}$. The origin of the vector $[x', y']$ is preserved. However linear transformations are just special cases of Affine transformations, namely where $M = \vec{0}$. Affine transformations include translations, and for linear transformations the translation is 0. For non-zero translations (non-linear transformations), the origin of the vector co-ordinates changes. So for Affine transformations we represent a vector $[x, y]$ as a 3-vector: $[x, y, 1]$. To translate a vector $\vec{v}$ in space, we let matrix T operate on it such that

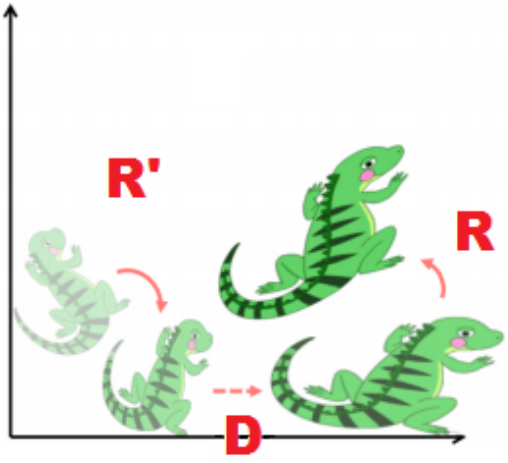
$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

In other words,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = A = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z \end{bmatrix}$$

Computer graphics

Affine transformations are therefore used to create 2D graphics. Because of all the properties outlined above, we can use a 3×3 matrix to scale, rotate or translate a point $[x, y]$ in 2-space with one operation. Think about playing a simple game- we want our character to move around the screen, or perhaps shrink/grow in size... but how might we instruct the image to move/distort mathematically? Affine transformations conserve co-linearity are relative distance, so in the context of an image, all the points which make up the image stay in the same positions relative to each other. This means that the image in its totality looks the same, despite being rotated, skewed or translated if A acts on all points that constitute the image.



General rotations about an arbitrary axis

In MTH1030 you have seen how points in 3-space can be rotated around the x, y and z axes.

Rotation about the x-axis

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_x \underline{\mathbf{x}}$$

Rotation about the y-axis

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_y \underline{\mathbf{x}}$$

Rotation about the z-axis

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_z \underline{\mathbf{x}}$$

Note that the column vectors of the rotation matrices R are unit vectors representing the image of our basis vectors in $\mathbb{R}^3$, $\underline{\mathbf{e}}_1$, $\underline{\mathbf{e}}_2$ and $\underline{\mathbf{e}}_3$. So the 3×3 matrix R is an orthogonal matrix, and the point $\underline{\mathbf{x}}$ could equivalently be operated on by the 3 column vectors. Contructing an R using orthogal unit vectors will motivate general rotations in $\mathbb{R}^3$.

General rotation

1. Transform $\underline{\mathbf{x}}$ to align rotation axis with one of the co-ordinate axes.
2. Perform rotation about this axis
3. Do the inverse of step 1 to return to the original co-ordinate system.

1. Contructing S_w to use S_w^{-1}

We want to transform the z axis, $\underline{\mathbf{e}_3}$ to $\underline{\mathbf{w}}$. To do this, we need to find $\underline{\mathbf{u}}$ and $\underline{\mathbf{v}}$ such that $\underline{\mathbf{u}} \times \underline{\mathbf{v}} = \underline{\mathbf{w}}$. We then contruct a matrix S_w such that

$$S_w = (\underline{\mathbf{u}}, \underline{\mathbf{v}}, \underline{\mathbf{w}})$$

This corresponds to transforming the point to align with one co-ordinate axis. In this case it is the z axis ($\underline{\mathbf{e}_3}$). It is important to note here that the axis we are rotating about, $\underline{\hat{\mathbf{w}}}$ appears in the third column of S_w when we transform the point to align with z.
By applying S_w^{-1} to $\underline{\mathbf{x}}$, $\underline{\mathbf{w}}$ is transformed to $\underline{\mathbf{e_3}}$

2. Rotating around z axis with angle θ

Since we have transformed $\underline{\mathbf{e}_3}$ to $\underline{\mathbf{w}}$, we will use R_z to rotate about $\underline{\mathbf{e}_3}$

$$R_{z\theta} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

By applying $R_{z\theta}$ to $S_w \underline{\mathbf{x}}$, we rotate our (transformed) point about the z-axis.

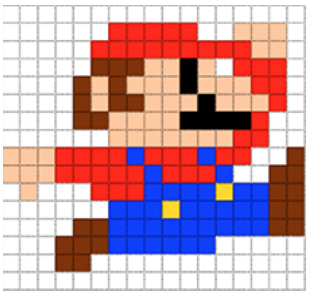
3. Applying S_w

Finally S_w is applied to $R_{z\theta} S_w^{-1} \underline{\mathbf{x}}$. This transforms $\underline{\mathbf{e_3}}$ back to $\underline{\mathbf{w}}$, which gives the result of rotating a point $\underline{\mathbf{x}}$ some angle θ about a vector $\underline{\mathbf{w}}$.

$$R_{\theta w} \underline{\mathbf{x}} = S_w R_{z\theta} S_w^{-1} \underline{\mathbf{x}} \tag{1}$$

EXERCISES

1. Consider the following image of Mario.



Say that he is running right, until you control him to jump left diagonally in the air. In general a jump moves his image +40 pixels higher and ±30 pixels horizontally at max height. Because he changes direction for the jump, his image will be reflected in the y axis.

   (a) Give a quick sketch of his new location and form

   (b) Represent the translation and reflection as two separate operations.

   (c) Find the affine transformation that represents this change to a pixel constituting the image.

2. However it is 2016 and Mario has since been promoted to a 3-dimensional creature. In Super Mario Galaxy, the levels are played on planets or dwarf planets. He is often seen to 'blast-off' into space when moving from planet to planet, or when he completes a level. When blasting off, he twirls about an arbitrary axis $\underline{\mathbf{w}}$ perpendicular to the planets surface. Assume that Mario's blast off axis is slightly misaligned with $\underline{\mathbf{w}}$ and gravitational effects from the surface are negligible.

   (a) Let $\underline{\mathbf{w}} = (-2, 1, 2)$. Say that an arbitrary particle of Mario lies at $\underline{\mathbf{x}}$. Find the general rotation matrix $R_{\theta w}$ that acts on $\underline{\mathbf{x}}$. Make sure that S_w is a unitary matrix.

   (b) 1 second before lift-off, Mario has moved $\theta = \pi/6$ around $\underline{\mathbf{w}}$ through a practise twirl on the surface about $\underline{\mathbf{w}}$. His centre of mass is located at (3,0,4) before the twirl. Find the new location of his centre of mass.
   CHALLENGE

   (c) Mario blasts off with velocity $\underline{\mathbf{v}} = (-1, 2, 1)$ and immediately rotates around $\underline{\mathbf{w}}$. Express this general 3D motion in time with an affine transformation by adding an extra column to $R_{w\theta}$.

REFERENCES

1. Buss, S.R. (2003) 3-D computer graphics: A mathematical introduction with OpenGL, Cambridge University Press New York, NY . 27, 44pp

2. Ken Anjyo, K., Ochiai, H. (2014) Mathematical Basics of Motion and Deformation in Computer Graphics, Morgan and Claypool, San Rafael CA, 21-25pp