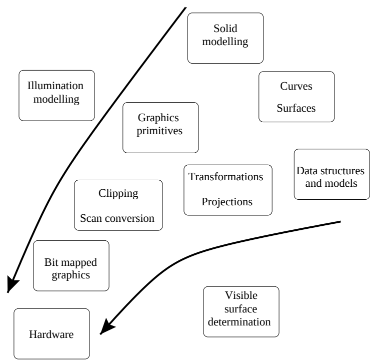
Overview

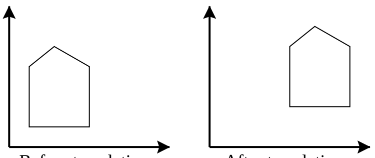
- Where are we in the Graphics Module?
- Why consider 2D transformations?
- Definition of common 2D affine transformations.
- Homogeneous coordinates.
- Combining transformations.

Where are we in the Graphics Module?



2D transformations

- Use the vector and matrix algebra from last lecture.
- Translations are offsets from the existing position of the object. Consider a point at $\boldsymbol{r}$.
- Translate it by an amount $\boldsymbol{t} = (t_x, t_y)'$: new location will be $\boldsymbol{r}^* = \boldsymbol{r} + \boldsymbol{t}$.



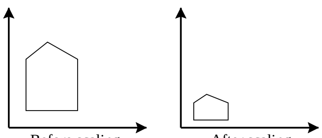
Before translation After translation

2D transformations

- Scalings are stretchings of the object, about the origin. The scaling matrix S is:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} ,$$

- $\boldsymbol{r}^* = S\boldsymbol{r}$: s_x is the x-axis scaling and s_y is the y-axis scaling.
- If $s_x = s_y = s$ the scaling is said to be uniform. If not the scaling is called differential.



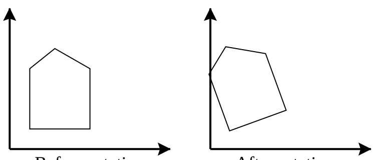
Before scaling After scaling

2D transformations

- Rotations about the origin by an angle θ are defined by the rotation matrix R which is given by:

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

- The rotated point, $\boldsymbol{r}^* = R\boldsymbol{r}$.
- A positive θ implies an anti-clockwise rotation.



Before rotation After rotation

Homogeneous coordinates

- Homogeneous coordinates allow us to treat all transformations in the same way, as matrix multiplications.
- The consequence is that our 2-vectors become extended to 3-vectors, with a resulting increase in storage and processing.
- We represent a point (x, y) by the extended triple (x, y, w).
- The normalised homogeneous coordinates are $(x/w, y/w, 1)$.
- Points with $w = 0$ are called points at infinity, and are not frequently used.
- If you like then you can think of 2D space corresponding to plane $w = 1$.

Homogeneous coordinates

- In homogeneous coordinates the transformations are:
  - translation:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T\boldsymbol{r} \; ;$$

  - scaling:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = S\boldsymbol{r} \; ;$$

  - rotation:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = R\boldsymbol{r} \; .$$

Homogeneous coordinates

- Apply each of these transformations to a vector $[x, y, 1]'$] and compute the resulting vector:
  - translation:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix} \; ;$$

  - scaling:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix} \; ;$$

  - rotation:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix} \; .$$

Homogeneous coordinates

- Rigid body transformations preserve length and angles (e.g. translation or rotation).
- Affine transformations preserve parallelism in lines (e.g. translation, rotation, scaling and shearing).
- A shear transformation is given by:

$$r^* = \begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & h_x & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H r \,,$$

- h_x and h_y represent the amount of shear along the x and y axes respectively.

Composition of transformations

- Big advantage of homogeneous coordinates is that transformations can be very easily combined.
- All that is required is multiplication of the transformation matrices.
- This makes otherwise complex transformations very easy to compute.

Composition of transformations

- For instance if we wanted to rotate an object about some point, $\boldsymbol{p}$.
- Achieved by:
  1. translate object by $-\boldsymbol{p}$,
  2. rotate object by angle θ,
  3. translate object by $\boldsymbol{p}$.

Composition of transformations

- This can be written as:

$$T(\boldsymbol{p})R(\theta)T(-\boldsymbol{p}) = \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & p_x(1-\cos\theta) + p_y \sin\theta \\ \sin\theta & \cos\theta & p_y(1-\cos\theta) - p_x \sin\theta \\ 0 & 0 & 1 \end{bmatrix} .$$

- Note the ordering of the transformation matrices.
- For those interested, Matlab provides an excellent platform for investigating these sort of transformations, since its natural matrix format makes things very easy to code.

Transformations and OpenGL

- Basic commands are:
  - `glTranslate#(dx, dy, dz)`
  - `glScale#(sx, sy, sz)`
  - `glRotate#(angle, x, y, z)`
- The matrices are applied to the vertices in the opposite order they are specified (pre-multiplied by existing transformation matrix).
- Can define our own matrices: `glLoadMatrix` and `glMultMatrix`.

Transformations and OpenGL– stacks

- There are two important matrices – `GL_PROJECTION` and `GL_MODELVIEW`.
- OpenGL stores these as composite transformation matrices.
- We use `glPushMatrix()` and `glPopMatrix()` to 'save' the matrix stack.
- OpenGL maintains a matrix stack which is used to store the composite transformation matrices (of all transformations so far specified).

Summary

- Having finished this lecture you should:
  - be able to write down the transformation matrices in both Cartesian (normal) and homogeneous coordinates;
  - understand the role of homogeneous coordinates in *computer graphics*;
  - be able to compute composite transformation matrices;
  - understand the way OpenGL implements transformations.
- Doing the lab classes is key to understanding much of this material.