2D Transformations

Reading

Required:

- Hearn and Baker, Sections 5.1–5.4, 5.6, 6.1–6.3, 6.5

Optional:

- Foley *et al.*, Chapter 5.1–5.5
- David F. Rogers and J. Alan Adams, *Mathematical Elements for Computer Graphics*, Second edition, McGraw-Hill, New York, 1990, Chapter 2.

2D drawing

Think of a program like PowerPoint, Illustrator, MacDraw...

- Interactively create a number of primitives, e.g., polygons and circles.
- Indicate a front-to-back ordering.
- Scale, translate, and rotate objects, as well as group them together.
- Scroll or zoom the "canvas" to look at different parts of the drawing.
- Generate an image and displays it on the screen.

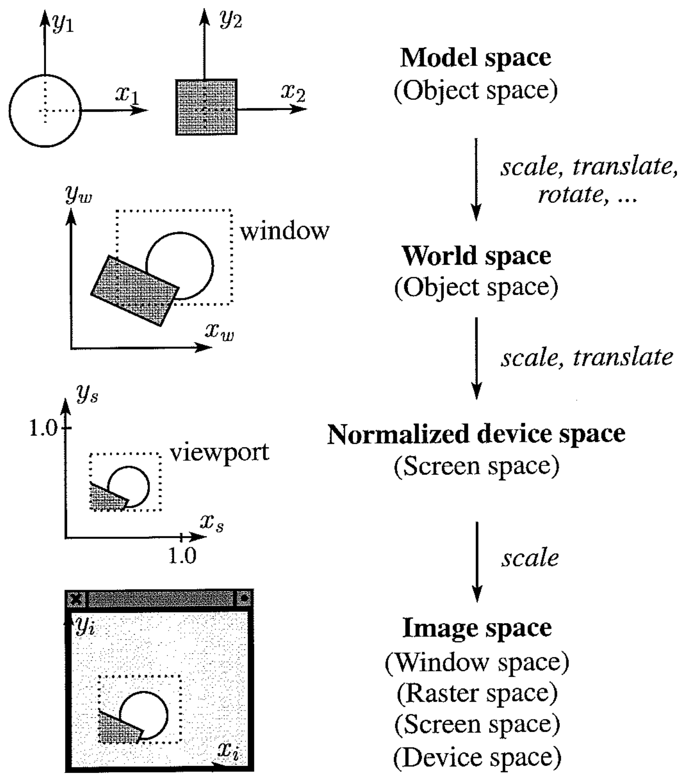
2D drawing, cont'd

What are some of the key ingredients needed to make this work?

- Specification of the front-to-back ordering.
- A sequence of geometric transformations, some of them stored in hierarchies corresponding to groups of primitives.
- Definition of the "visible" portion of the canvas.
- A mapping from the visible portions of the canvas to pixels on the screen.
- Software or hardware that is able to "rasterize" the primitives, i.e., draw the pixels corresponding to the primitives.

2D geometry pipeline

Let's think about this in terms of a set of coordinate systems:



Clipping

To avoid drawing primitives or parts of primitives that do not appear in the viewport, we perform "clipping".

Clipping includes:

- Removal of primitives wholly outside of the viewport (a.k.a., "culling")
- Intersection of the viewport with primitives that straddle the viewport boundary.

Clipping can happen:

- In world space
- In normalized device space
- In image space

A simple OpenGL example

Here's an example of an OpenGL program that will draw a black square over a white background:

```
makeADrawingWindow();
glOrtho(xw_min, xw_max, yw_min, yw_max, -1.0, 1.0);
glViewport(xi_min, yi_min, width_i, height_i);
glClearColor(1.0, 1.0, 1.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_POLYGON);
  glVertex2f(0.0, 0.0);
  glVertex2f(1.0, 0.0);
  glVertex2f(1.0, 1.0);
  glVertex2f(0.0, 1.0);
glEnd();
glFlush();
```

For the remainder of this lecture, we will focus on 2D geometric transformations...

Representation

We can represent a **point** $p = (x, y)$ in the plane

- as a column vector $\begin{bmatrix} x \\ y \end{bmatrix}$
- as a row vector $\begin{bmatrix} x & y \end{bmatrix}$

Representation, cont.

We can represent a **2-D transformation** M by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

If p is a column vector, M goes on the left:

$$\begin{aligned} p' &= M\,p \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

If p is a row vector, M^{T} goes on the right:

$$\begin{aligned} p' &= p\,T \\ \begin{bmatrix} x' & y' \end{bmatrix} &= \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & c \\ b & d \end{bmatrix} \end{aligned}$$

We will use **column vectors**.

Two-dimensional transformations

Here's all you get with a 2×2 transformation matrix M:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

So

$$\begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

We will develop some intimacy with the elements a, b, c, d. . . .

Identity

Suppose we choose $a = d = 1$, $b = c = 0$:

- Gives the "identity" matrix

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

- Doesn't move the points at all

Scaling

Suppose we set $b = c = 0$, but let a and d take on any positive value:

- Gives a "scaling" matrix

$$\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$$

- Provides differential scaling in x and y:

$$\begin{aligned} x' &= ax \\ y' &= dy \end{aligned}$$

Suppose we keep $b = c = 0$, but let a and d go negative.

Examples:

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

Mirror reflection about y-axis

Mirror reflection about the 45° line

Now let's leave $a = d = 1$ and experiment with c. . . .

The matrix

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

gives:

$$x' = x$$
$$y' = cx + y$$

Effect is called a "shearing."

Effect on unit square

Let's see how a general 2×2 transformation M affects the unit square:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & a & a+b & b \\ 0 & c & c+d & d \end{bmatrix}$$

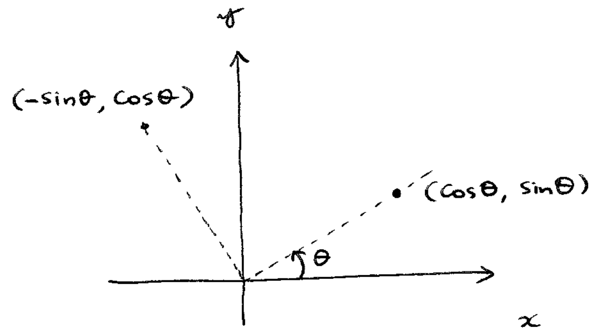
Effect on unit square, cont.

Observe:

- Origin invariant under M
- M can be determined just by knowing how the corners $(1,0)$ and $(0,1)$ are mapped
- a and d give x- and y-scaling
- b and c give x- and y-shearing

Rotation

From our observations of the effect on the unit square, it should be easy to write down a matrix for "rotation about the origin":



- $\begin{bmatrix} 1 \\ 0 \end{bmatrix} \to \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$
- $\begin{bmatrix} 0 \\ 1 \end{bmatrix} \to \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$

Limitations of the 2×2 matrix

A 2×2 matrix allows

- Scaling
- Rotation
- Reflection
- Shearing

Q: What important operation does that leave out?

Translation

Homogeneous coordinates

Idea is to loft the problem up into 3-space, adding a third component to every point:

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

And then transform with a 3×3 matrix:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = x + m$$
$$y' = y + n$$
$$w' = 1$$

. . . Gives **translation**!

Rotation about arbitrary points

Until now, we have only considered rotation about the origin.

With homogeneous coordinates, you can specify rotations about any point q with a matrix:

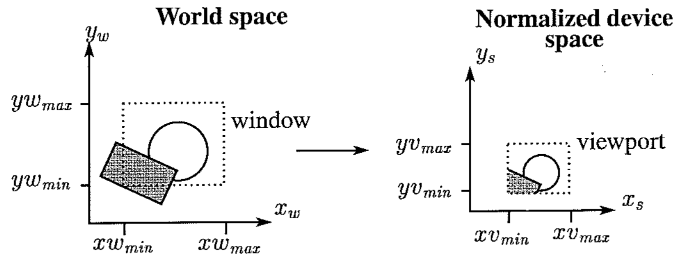
$$\begin{bmatrix} \cancel{0}\,1 & 0 & q_x \\ 0 & 1 & q_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & -s & 0 \\ s\theta & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -q_x \\ 0 & 1 & -q_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_x \\ q_y \\ 1 \end{bmatrix}$$

1. Translate q to origin
2. Rotate
3. Translate back to q

<u>Note</u>: Transformation order is important!

Window-to-viewport transformation

How do we transform from the window in world coordinates to the viewport in screen space?



1) translate to the origin
2) scaling
3) translate to ~~the~~ (xv_{min}, yv_{min})

$$M_1 = \begin{bmatrix} 1 & 0 & -xw_{min} \\ 0 & 1 & -yw_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} \qquad s_y = \ldots$$

$$M_3 = \begin{bmatrix} 1 & 0 & xv_{min} \\ 0 & 1 & yv_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$M = M_3 M_2 M_1$$

Mathematics of affine transformations

All of the transformations we've looked at so far are examples of "affine transformations."

Here are some useful properties of affine transformations:

- Lines map to lines
- Parallel lines remain parallel
- Midpoints map to midpoints
  (in fact, ratios are always preserved)

Summary

What to take away from this lecture:

- All the underlined names and names in quotations.
- How points and transformations are represented.
- What all the elements of a 2×2 transformation matrix do.
- What homogeneous coordinates are and how they work for affine transformations.
- How to concatenate transformations.
- The mathematical properties of affine trasnformations.