Recap

We have reviewed the relevant linear algebra

Matrices

Vectors

Scalars

Next, we will discuss:

Homogeneous representations of points and vectors

Coordinate systems

Transformations

Points vs Vectors

What is the difference?

Points have location, but no size or direction

Vectors have size and direction, but no location

Problem: We represent both as 3-tuples

Homogeneous Representation

Convention:
Vectors and Points are represented as 4x1 column matrices, as follows:

Switching Representations

Normal to homogeneous:

- Vector: append as fourth coordinate 0

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \rightarrow \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix}$$

- Point: append as fourth coordinate 1

$$P = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \rightarrow \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$

Switching Representations

Homogeneous to normal:

- Vector: remove fourth coordinate (0)

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

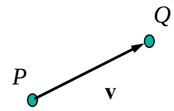
- Point: remove fourth coordinate (1)

$$P = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Relationship Between Points and Vectors

A difference between two points is a vector:

$$Q - P = \mathbf{v}$$

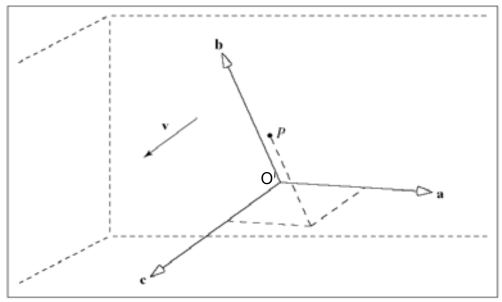


We can consider a point as a base point plus a vector offset:

$$Q = P + \mathbf{v}$$

Coordinate Systems

Defined by: **a**,**b**,**c**,*O*



$$\mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c}$$

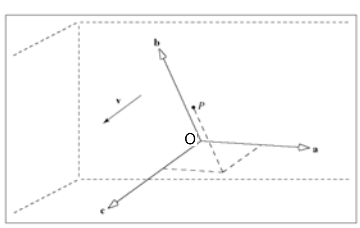
$$P - O = p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$

$$P = O + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$

Homogeneous Representation of Points and Vectors

$$\mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c} \rightarrow \mathbf{v} = [\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}\ \ O]\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix}$$

$$P = O + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c} \rightarrow P = [\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}\ \ O]\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$



Does the Homogeneous Representation Support Operations?

Operations :

- $\mathbf{v} + \mathbf{w} = [v_1,v_2,v_3,0]^T + [w_1,w_2,w_3,0]^T$
  $= [v_1 + w_1, v_2 + w_2, v_3 + w_3, 0]^T$ Vector
- $a\mathbf{v} = a[v_1,v_2,v_3,0]^T = [av_1, av_2, av_3, 0]^T$ Vector
- $a\mathbf{v} + b\mathbf{w} = a[v_1,v_2,v_3,0]^T + b[w_1,w_2,w_3,0]^T$
  $= [av_1 + bw_1, av_2 + bw_2, av_3 + bw_3, 0]^T$ Vector
- $P + \mathbf{v} = [p_1, p_2, p_3, 1]^T + [v_1, v_2, v_3, 0]^T$
  $= [p_1 + v_1, p_2 + v_2, p_3 + v_3, 1]^T$ Point
- $P - Q = [p_1,p_2,p_3,1]^T - [q_1,q_2,q_3,1]^T$
  $= [p_1 - q_1, p_2 - q_2, p_3 - q_3, 0]^T$ Vector

Linear Combination of Points

Points P, Q scalars f, g:

$$fP + gQ = f\,[p_1, p_2, p_3, 1]^T + g[q_1, q_2, q_3, 1]^T$$
$$= [fp_1+gq_1,\ fp_2+gq_2,\ fp_3+gq_3,\ f+g]^T$$

What is this?

Linear Combination of Points

Points *P*, *Q* scalars *f*, *g*:

$$fP + gQ = f\,[p_1, p_2, p_3, 1]^T + g[q_1, q_2, q_3, 1]^T$$
$$= [fp_1 + gq_1,\ fp_2 + gq_2,\ fp_3 + gq_3,\ f + g]^T$$

What is it?

- If $(f + g) = 0$ then vector!
- If $(f + g) = 1$ then point!
- Otherwise, ??

Affine Combinations of Points

Definition:

n points P_i: $i = 1, \ldots, n$

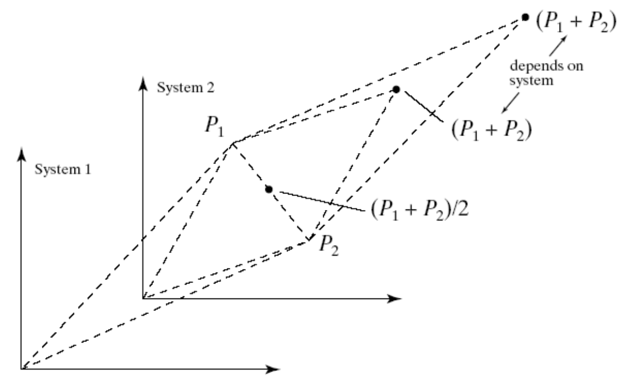
n scalars f_i: $i = 1, \ldots, n$

$f_1 P_1 + \ldots + f_n P_n$ iff $f_1 + \ldots + f_n = 1$

Example (n = 2): $0.5P_1 + 0.5P_2$

Example (n = 2): $(1\text{-}s)P_1 + sP_2$

Geometric Interpretation



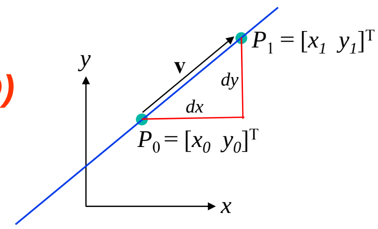
Lines and Planes

In addition to vectors and points, lines and planes are fundamental geometric entities in computer graphics

- Recall (from Analytic Geometry) how we represent them mathematically…

Lines

Representations of a line (in 2D)



- Explicit $y = \alpha x + \beta$

  $y = m(x - x_0) + y_0; \quad m = \frac{dy}{dx} = \frac{y_1 - y_0}{x_1 - x_0}$

- Implicit $f(x, y) = (x - x_0)dy - (y - y_0)dx$

  if $f(x, y) = 0$ then (x, y) is **on** the line

  $f(x, y) > 0$ then (x, y) is **below** the line

  $f(x, y) < 0$ then (x, y) is **above** the line

- Parametric $x(t) = x_0 + t(x_1 - x_0)$

  $y(t) = y_0 + t(y_1 - y_0)$

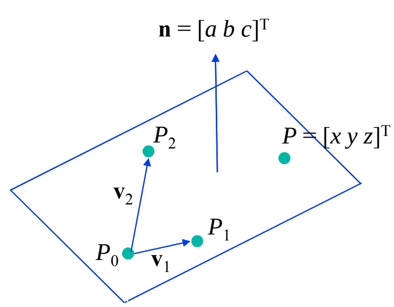
  $t \in [0,1]$ for line segment, or $t \in [-\infty, \infty]$ for infinite line

  $P(t) = P_0 + t(P_1 - P_0)$ or $P(t) = P_0 + t\,\mathbf{v}$

  $P(t) = (1 - t)P_0 + tP_1$

Planes

Plane equations



- Explicit

  $z = \alpha x + \beta y + \gamma$

  $\alpha = -a / c$

  $\beta = -b / c$

  $\gamma = -d / c$

  $c \neq 0$

- Implicit

  $F(x, y, z) = ax + by + cz + d = \mathbf{n} \bullet P + d$

  Points on Plane: $F(x, y, z) = 0$

- Parametric

  $\text{Plane}(s,t) = P_0 + s(P_1 - P_0) + t(P_2 - P_0)$

  P_0, P_1, P_2 are not collinear

  or

  $\text{Plane}(s,t) = P_0 + s\mathbf{v}_1 + t\mathbf{v}_2$, where $\mathbf{v}_1$, $\mathbf{v}_2$ are basis vectors

  Convex combination defines a triangle:

  $\text{Triangle}(s,t) = (1 - s - t)P_0 + sP_1 + tP_2$, with $s, t \in [0,1]$

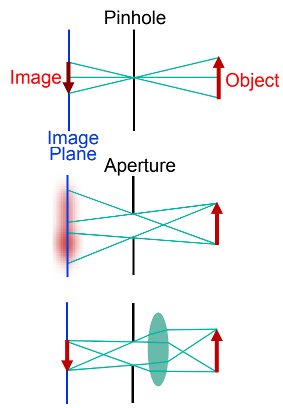
Cameras (and the Eye)

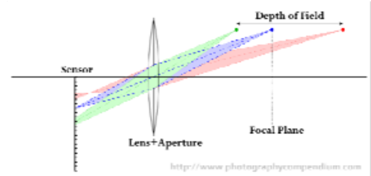
Ideal pinhole

Aperture

Aperture + lens

Depth of field





How Do We Draw Objects?

Z-buffer

- Polygon Based
- Fast

Raytracing

- Ray/Object intersections
- Slow

Preview:
Raytracing Algorithm

for each pixel on screen

determine ray from eye through pixel

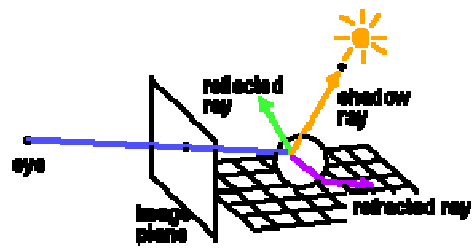
find closest intersection of ray with an object

cast off reflected and refracted ray, recursively

calculate pixel color

draw pixel

end



Preview:
Z-Buffer Algorithm

set pixels to background color and z-buffer to maximum z-values

for each polygon in model

project vertices of polygon onto viewing plane

for each pixel inside the projected polygon

calculate pixel <u>color</u>

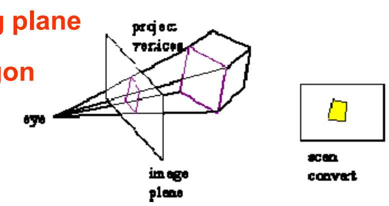
calculate pixel <u>z-value</u>

if <u>z-value</u> is less than z-value stored for pixel in z-buffer

set pixel to <u>color</u> and store <u>z-value</u> into z-buffer

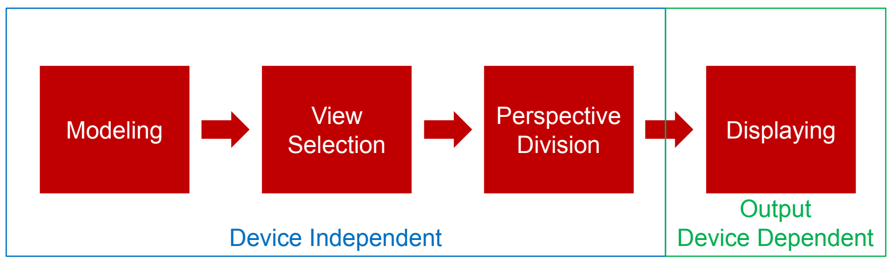
end

end

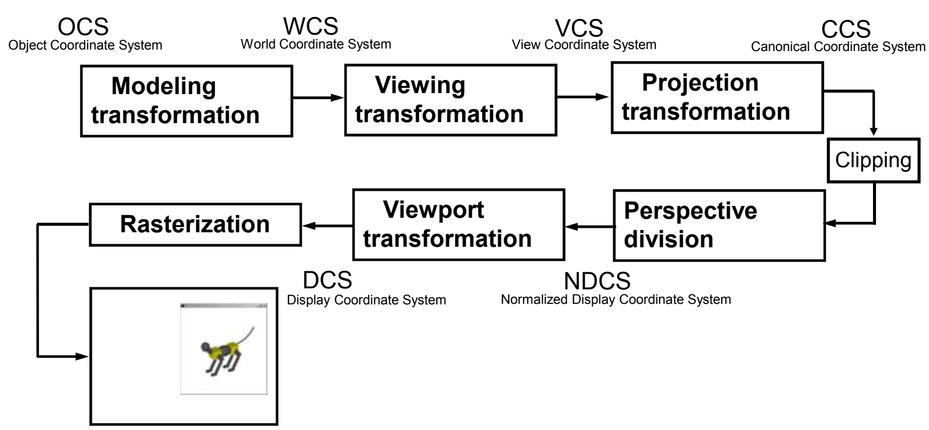


Z-Buffer Graphics Pipeline

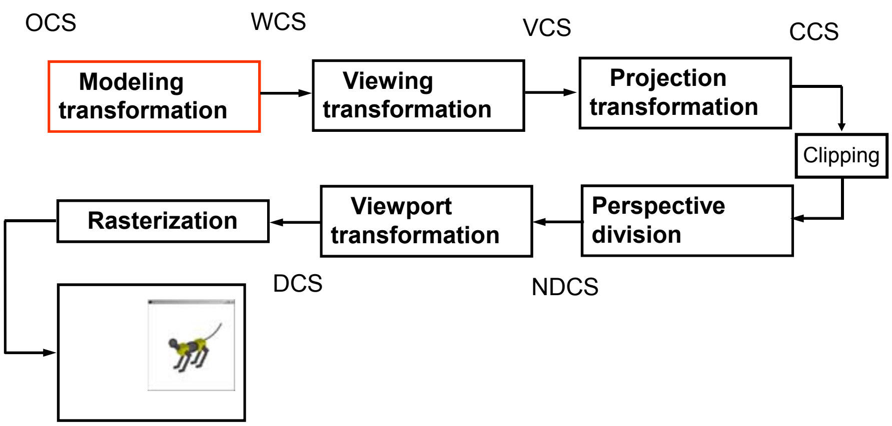
4 stages



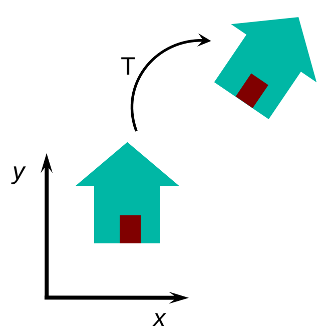
Z-Buffer Graphics Pipeline

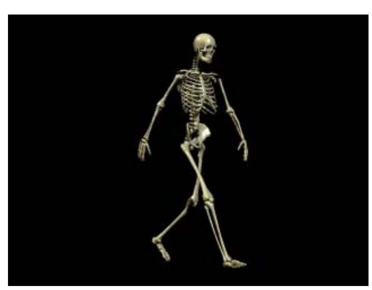


Z-Buffer Graphics Pipeline



Why Transformations?

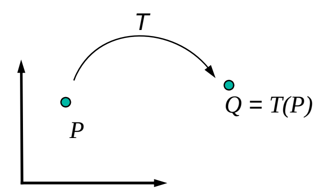




Transformations

General Form: $Q = \mathcal{T}(P), \quad P \in \Re^n, \quad Q \in \Re^m$

If $n > m$, $\mathcal{T}$ is known as a Projection



A 2D example: $\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} \cos P_y e^{-P_y} \\ \ln P_x \end{bmatrix}$

Linear Transformations in 2D

Linear in the coordinates of P

$$Q = \mathcal{T}(P)$$

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} m_{11}P_x + m_{12}P_y \\ m_{21}P_x + m_{22}P_y \end{bmatrix}; \quad m_{11}, m_{12}, m_{21}, m_{22} \in \Re$$

$$= \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

They are written compactly as matrix multiplications:

$$Q = \mathbf{M}P$$

Is translation ($Q = P + \mathbf{t}$) *a linear transformation?*

Affine Transformations in 2D

Linear in the coordinates of P

$$Q = \mathcal{T}(P)$$

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} m_{11}P_x + m_{12}P_y + m_{13} \\ m_{21}P_x + m_{22}P_y + m_{23} \end{bmatrix}; \quad m_{11}, \ldots, m_{23} \in \Re$$

Additional constants m_{13} and m_{23} handle translations

But we cannot write the above as $Q = \mathbf{M}P$

Matrix Form of the Affine Transformations

The trick is to use homogeneous coordinates

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}P_x + m_{12}P_y + m_{13} \\ m_{21}P_x + m_{22}P_y + m_{23} \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Then, affine transformation is a matrix multiplication

$$Q = \mathbf{M}P$$

Elementary Affine Transformations

Any affine transformation is equivalent to a combination of four elementary affine transformations

- Translation
- Scaling
- Rotation
- Shearing

Transforming Points and Vectors

Points:

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} \overbrace{\boxed{\begin{matrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{matrix}}}^{\text{Rotation/Scaling/Shearing}} & \overbrace{\boxed{\begin{matrix} m_{13} \\ m_{23} \end{matrix}}}^{\text{Translation}} \\ \begin{matrix} 0 & \quad 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

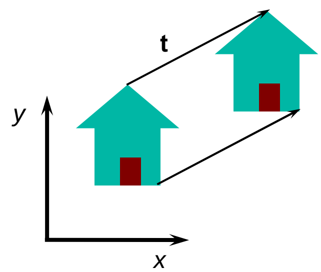
Vectors:

$$\begin{bmatrix} W_x \\ W_y \\ 0 \end{bmatrix} = \begin{bmatrix} \overbrace{\boxed{\begin{matrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{matrix}}}^{\text{Rotation/Scaling/Shearing}} & \overbrace{\boxed{\begin{matrix} m_{13} \\ m_{23} \end{matrix}}}^{\text{Translation}} \\ \begin{matrix} 0 & \quad 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ 0 \end{bmatrix}$$

Note: Translation modifies points, but not vectors

Translation

$$Q = P + \mathbf{t}, \quad \mathbf{t} = (t_x \;\; t_y)^T$$

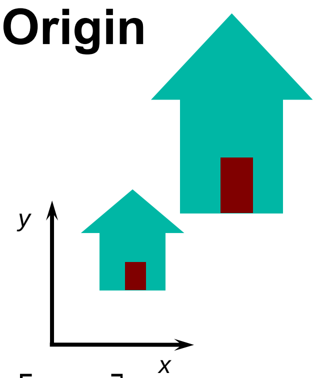


$$Q_x = P_x + t_x$$

$$Q_y = P_y + t_y$$

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Scaling Around the Origin



$$Q_x = s_x P_x$$

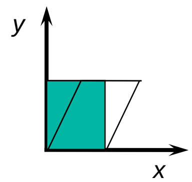
$$Q_y = s_y P_y$$

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Uniform scaling: $s_x = s_y$

Shear Around the Origin

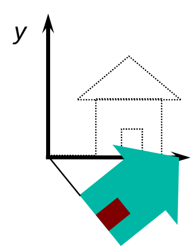
In the x-direction



$$Q_x = P_x + aP_y$$
$$Q_y = P_y$$

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Rotation Around the Origin

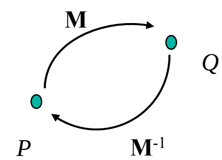


$$Q_x = \cos\theta\, P_x - \sin\theta\, P_y$$
$$Q_y = \sin\theta\, P_x + \cos\theta\, P_y$$

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Inverse of a Transformation

Inverse transformation: $Q = \mathbf{M}P, \quad P = \mathbf{M}^{-1}Q$



We can use Cramer's rule to invert $\mathbf{M}$, or we can be smarter about it

Inverse of Translation

$$Q = \mathcal{T}(\mathbf{t})\mathbf{P} \rightarrow \mathbf{P} = \mathcal{T}(-\mathbf{t})\mathbf{Q}$$

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

Inverse of Scaling

$$Q = \mathcal{S}(s_x, s_y)P \rightarrow P = \mathcal{S}(\frac{1}{s_x}, \frac{1}{s_y})Q$$

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse of a Shear in *x*

$$Q = \mathcal{S}h_x(a)P \rightarrow P = \mathcal{S}h_x(-a)Q$$

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

Inverse of Rotation

$$Q = \mathcal{R}(\theta)P \rightarrow P = \mathcal{R}(-\theta)Q$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Composing 2D Affine Transformations

Composing two affine transformations produces an affine transformation

$$Q = \mathcal{T}_2(\mathcal{T}_1(P))$$

In matrix form:

$$Q = \mathrm{M}_2(\mathrm{M}_1 P) = (\mathrm{M}_2\mathrm{M}_1)P = \mathrm{M}P$$

Which transformation happens first?

Main Points

- *Affine transformations are the main modeling tool in graphics*
- *They are applied as matrix multiplications*
- *Any affine transformation can be performed as a series of elementary affine transformations*
- *Make sure you understand the order of applied transformations*
  (i.e., ordering of matrix multiplications)

Other Examples

Rotation about an arbitrary point

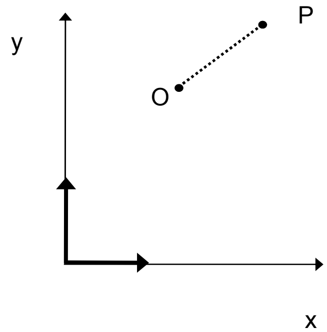
Scaling around an arbitrary point

Reflection

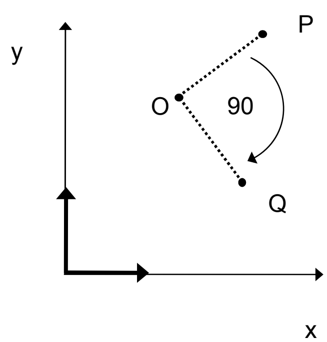
Reflection about an arbitrary line

Example: Another 2D Transformation

Rotate -90 deg around an arbitrary point O:

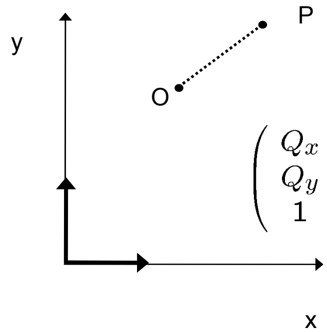


Rotate Around an Arbitrary Point



Rotate Around an Arbitrary Point

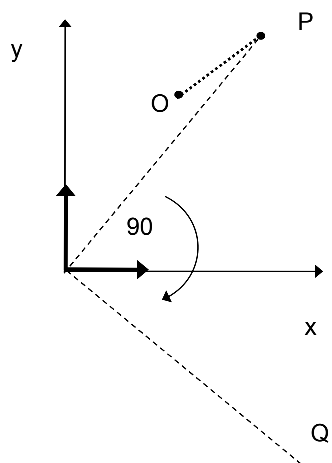
We know how to rotate around the origin



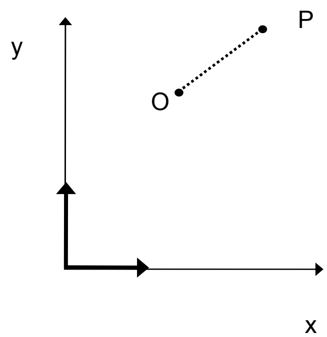
$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

Rotate Around an Arbitrary Point

…but that is not what we want to do!

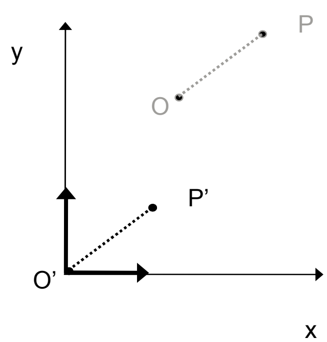


So What Do We Do?



Transform it to the Known Case
Step 1: Translation

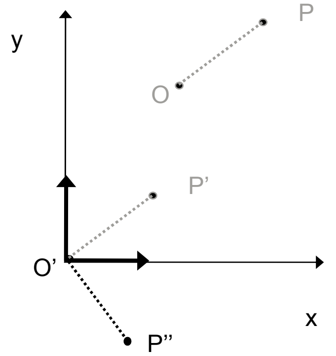
Translate$(-O_x,-O_y)$



Step 2: Rotation

Translate(-O_x,-O_y)

Rotate(-90)

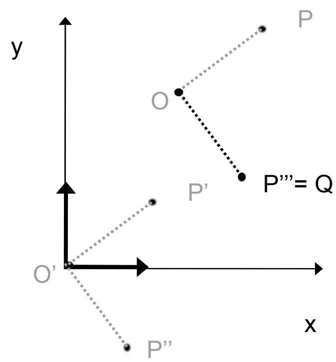


Finally, Put Everything Back

Translate(-O_x,-O_y)

Rotate(-90)

Translate(O_x,O_y)



Rotation About an Arbitrary Point

M = T(O_x,O_y) R(-90) T(-O_x,-O_y)

Order is IMPORTANT!