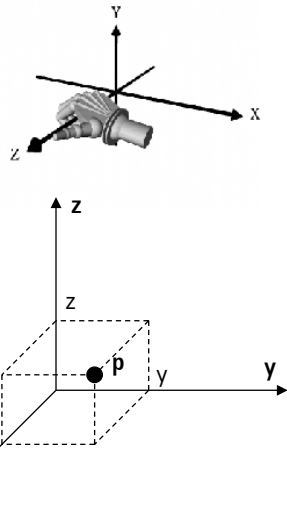


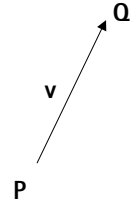
Coordinate systems

- Also known as “frame”
- Usually orthonormal
 - orthographic = axes perpendicular to each other
 - normal = the length of axes is unit
- Right handed coordinates
 - xyz = right thumb, index, middle
 - point right thumb at z, x turns to y along the fingers
- $\mathbf{p} = [x, y, z]^T$
- Modeling:
 - Object coordinates
 - World coordinates
 - Camera coordinates
- Rendering pipeline
 - Normalized device coordinates
 - Window coordinates



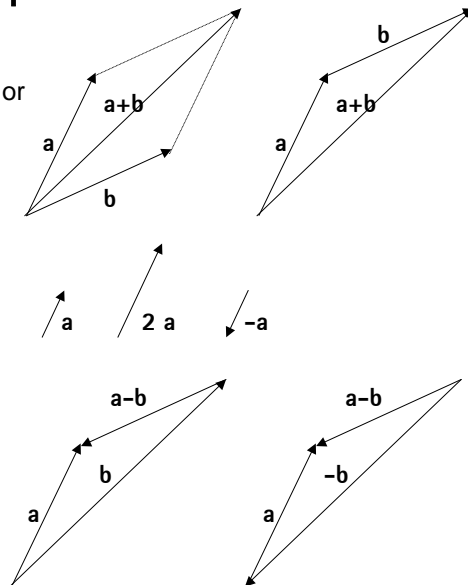
Vectors

- In graphics we deal with vectors of 2, 3, or 4 dimensions
 - we'll represent a vector as an n-tuple or vertical n-matrix
 - most rules don't care about the dimension
- Definition: **vector is a displacement**
 - it has a *direction*
 - and a *length*
 - but *no location*
- The **difference** between two points is a vector
 - $\mathbf{v} = \mathbf{Q} - \mathbf{P}$
- The **sum** of a point and a vector is a point
 - $\mathbf{P} + \mathbf{v} = \mathbf{Q}$



Vector operations

- Vector addition
 - the diagonal of a parallelogram or concatenate the vectors
 - $\mathbf{a} = (1,2,3)$, $\mathbf{b}=(4,5,6)$
 - $\mathbf{a}+\mathbf{b} = (1+4, 2+5, 3+6)$
- Multiplication by scalar
 - change the length, maybe flip direction
 - $\mathbf{a} = (1,2,3)$ $s \mathbf{a} = (s, 2s, 3s)$
 - $5\mathbf{a} = (5, 10, 15)$
- Subtraction
 - addition and negative scaling
 - $\mathbf{a} - \mathbf{b} = \mathbf{a} + -\mathbf{b}$



Linear combinations

- A linear combination of two vectors \mathbf{v} and \mathbf{w}
 - $a\mathbf{v}+b\mathbf{w}$
- More generally
 - $\mathbf{w} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_n\mathbf{v}_n$
- Affine combination
 - linear combination with $a_1 + a_2 + \dots + a_n = 1$
- Convex combination
 - affine combination with $a_i \geq 0$
 - aka partitioning of a unity

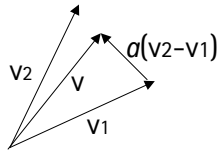
Examples

- Convex combination of two vectors

- interpolates the end points
- a line between the end points, parameterized by $0 \leq a \leq 1$
- what does an affine combination do?

$$\mathbf{v} = (1-a)\mathbf{v}_1 + a\mathbf{v}_2$$

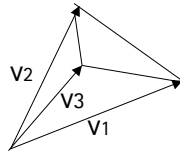
$$= \mathbf{v}_1 + a(\mathbf{v}_2 - \mathbf{v}_1)$$



- of three vectors

- defines the triangle
- affine combination?

$$\mathbf{v} = (1-a-b)\mathbf{v}_1 + a\mathbf{v}_2 + b\mathbf{v}_3$$



- You can use the affine coefficients as coordinates

- called barycentric coordinates

Vector magnitude

- Representation of a vector

- $\mathbf{w} = (w_1, w_2, \dots, w_n)$

- Its magnitude

- use the Pythagorean theorem

- $|\mathbf{w}| = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$

- Unit vector

- normalize a vector by dividing it by its magnitude
- the length will be 1
- represents a direction

- $\hat{\mathbf{w}} = \mathbf{w}/|\mathbf{w}|$

Dot product

- Definition $\mathbf{v} = (v_1, v_2, \dots, v_n), \mathbf{w} = (w_1, w_2, \dots, w_n)$

$$d = \mathbf{v} \cdot \mathbf{w} = \mathbf{v}^t \mathbf{w} = \sum_{i=1}^n v_i w_i$$

- Properties

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$$

$$(\mathbf{a} + \mathbf{c}) \cdot \mathbf{b} = \mathbf{a} \cdot \mathbf{b} + \mathbf{c} \cdot \mathbf{b}$$

$$(s\mathbf{a}) \cdot \mathbf{b} = s(\mathbf{a} \cdot \mathbf{b})$$

$$|\mathbf{b}|^2 = \mathbf{b} \cdot \mathbf{b}$$

- Example:

$$|\mathbf{a} - \mathbf{b}|^2 = (\mathbf{a} - \mathbf{b}) \cdot (\mathbf{a} - \mathbf{b}) = \mathbf{a} \cdot (\mathbf{a} - \mathbf{b}) + (-\mathbf{b}) \cdot (\mathbf{a} - \mathbf{b})$$

$$= \mathbf{a} \cdot (\mathbf{a} - \mathbf{b}) - \mathbf{b} \cdot (\mathbf{a} - \mathbf{b}) = (\mathbf{a} - \mathbf{b}) \cdot \mathbf{a} - (\mathbf{a} - \mathbf{b}) \cdot \mathbf{b}$$

$$= \mathbf{a} \cdot \mathbf{a} - \mathbf{b} \cdot \mathbf{a} - \mathbf{a} \cdot \mathbf{b} + \mathbf{b} \cdot \mathbf{b} = |\mathbf{a}|^2 - 2\mathbf{a} \cdot \mathbf{b} + |\mathbf{b}|^2$$

Dot product

- The angle between two vectors

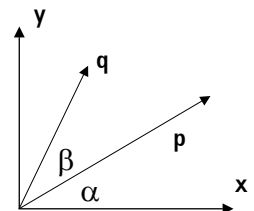
$$\mathbf{p} = (|\mathbf{p}| \cos \alpha, |\mathbf{p}| \sin \alpha)$$

$$\mathbf{q} = (|\mathbf{q}| \cos(\alpha + \beta), |\mathbf{q}| \sin(\alpha + \beta))$$

$$\mathbf{p} \cdot \mathbf{q} = |\mathbf{q}||\mathbf{p}| \cos(\alpha + \beta) \cos \alpha + |\mathbf{q}||\mathbf{p}| \sin(\alpha + \beta) \sin \alpha$$

$$= |\mathbf{q}||\mathbf{p}| \cos(\alpha + \beta - \alpha) = |\mathbf{q}||\mathbf{p}| \cos \beta$$

$$\Rightarrow \cos \beta = \hat{\mathbf{p}} \cdot \hat{\mathbf{q}}$$

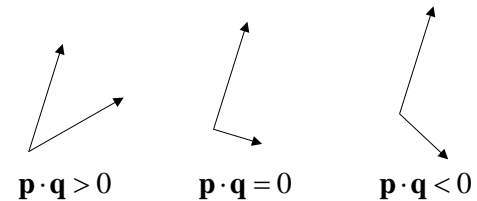


- The sign of the dot product

- follows the sign of cosine

- What is \mathbf{a}^\perp , the perpendicular vector to $\mathbf{a} = (a_x, a_y)$?

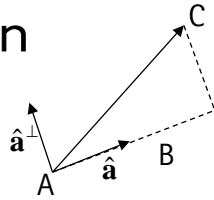
$$\mathbf{a}^\perp = (-a_y, a_x)$$



Dot as a projection

- Project point C onto line AB

- find vectors $\hat{\mathbf{a}} = \frac{B-A}{|B-A|}$ and $\mathbf{c} = C-A$



- project \mathbf{c} onto \mathbf{a} : $|\mathbf{c}| \cos \angle(\mathbf{a}, \mathbf{c}) \hat{\mathbf{a}} = |\mathbf{c}| (\hat{\mathbf{a}} \cdot \hat{\mathbf{c}}) \hat{\mathbf{a}} = (\hat{\mathbf{a}} \cdot \mathbf{c}) \hat{\mathbf{a}} = \left(\frac{\mathbf{a} \cdot \mathbf{c}}{|\mathbf{a}|} \right) \frac{\mathbf{a}}{|\mathbf{a}|} = \frac{\mathbf{a} \cdot \mathbf{c}}{\mathbf{a} \cdot \mathbf{a}} \mathbf{a}$

- add the projection (vector) to A (point):

$$A + \frac{\mathbf{a} \cdot \mathbf{c}}{\mathbf{a} \cdot \mathbf{a}} \mathbf{a}$$

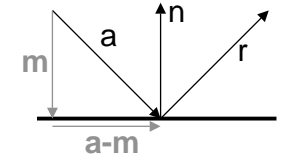
- What is the (perpendicular) distance of C from the line AB?

- project \mathbf{c} to $\hat{\mathbf{a}}^\perp$: $|\hat{\mathbf{a}}^\perp \cdot \mathbf{c}|$

- or $|\mathbf{c} - (\hat{\mathbf{a}} \cdot \mathbf{c}) \hat{\mathbf{a}}|$

Application: reflection vector

- Reflect the ray of light \mathbf{a} from a plane with normal vector \mathbf{n}
 - angle of incidence = angle of exit



- Break \mathbf{a} into two components

- vertical $\mathbf{m} = \frac{\mathbf{a} \cdot \mathbf{n}}{\mathbf{n} \cdot \mathbf{n}} \mathbf{n}$

- horizontal $\mathbf{a} - \mathbf{m}$

- Take the horizontal component and add the reflected vertical component

- $\mathbf{r} = \mathbf{a} - \mathbf{m} - \mathbf{m} = \mathbf{a} - 2\mathbf{m} = \mathbf{a} - 2 \frac{\mathbf{a} \cdot \mathbf{n}}{\mathbf{n} \cdot \mathbf{n}} \mathbf{n}$

Cross product

- Definition in 3D: $\mathbf{a} \times \mathbf{b} = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$

- $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are unit x,y,z vectors
- $\mathbf{i} \times \mathbf{j} = \mathbf{k}$
- $\mathbf{j} \times \mathbf{k} = \mathbf{i}$
- $\mathbf{k} \times \mathbf{i} = \mathbf{j}$

$$= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

- Like dot, cross is linear $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$
 $(s\mathbf{a}) \times \mathbf{b} = s(\mathbf{a} \times \mathbf{b})$

- but it's antisymmetrical $\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$

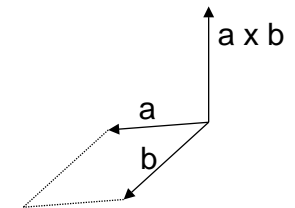
- what is $\mathbf{a} \times \mathbf{a}$?

must be zero: $\mathbf{a} \times \mathbf{a} = -\mathbf{a} \times \mathbf{a} \Rightarrow 0 = -0$

also the cross of any two parallel vectors must be zero

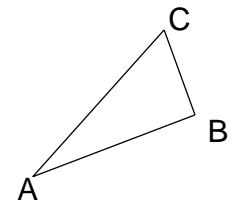
Cross product

- Geometric interpretation
 - $\mathbf{a} \times \mathbf{b}$ is perpendicular to both \mathbf{a} and \mathbf{b}
 - $|\mathbf{a} \times \mathbf{b}|$ = the area of the parallelogram determined by \mathbf{a} and \mathbf{b} : $|\mathbf{a}||\mathbf{b}|\sin \theta$, where θ is the angle between the vectors
 - the direction of $\mathbf{a} \times \mathbf{b}$ follows from the right hand rule



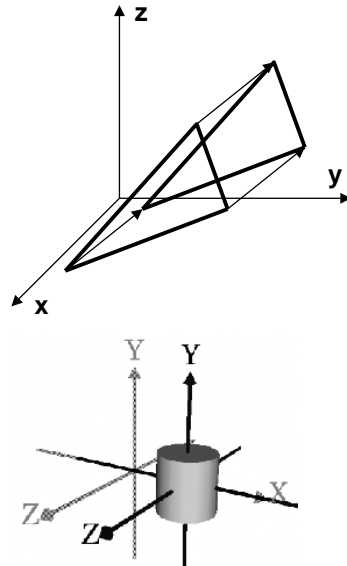
- Example:
 - Find a normal vector to the triangle ABC

$$(\mathbf{B} - \mathbf{A}) \times (\mathbf{C} - \mathbf{A})$$



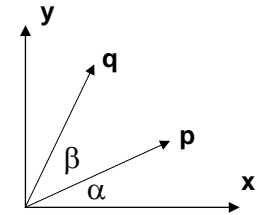
Transformations: Translation

- Translation of a geometric primitive is done by adding a vector to every point on it
- $\mathbf{Q} = \mathbf{P} + \mathbf{t}$
- $\mathbf{t} = [t_x, t_y, t_z]^t$
- Inverse: translate back
 $-\mathbf{t} = [-t_x, -t_y, -t_z]^t$
- Another interpretation
 - move the local coordinate system of an object
 - all the points therefore move as well, though they remain the same in the local coordinates



2D rotation

- ccw = counterclockwise*
- Rotation in a plane (ccw) by α
 - express \mathbf{p} with length and angle: $\mathbf{p} = [x, y]^t = [\rho \cos \alpha, \rho \sin \alpha]^t$
 - \mathbf{q} is \mathbf{p} rotated further by angle β $\mathbf{q} = [\rho \cos \alpha + \beta, \rho \sin \alpha + \beta]^t$
- $$\mathbf{q} = [\rho \cos \alpha \cos \beta - \rho \sin \alpha \sin \beta, \rho \cos \alpha \sin \beta + \rho \sin \alpha \cos \beta]^t$$
- $$= [x \cos \beta - y \sin \beta, x \sin \beta + y \cos \beta]^t$$
- $$= \begin{bmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
- Inverse
 - rotate \mathbf{q} back by $-\beta$ to get \mathbf{p} again $\begin{bmatrix} \cos -\beta & -\sin -\beta \\ \sin -\beta & \cos -\beta \end{bmatrix} = \begin{bmatrix} \cos \beta & \sin \beta \\ -\sin \beta & \cos \beta \end{bmatrix}$
 - or **transpose** the rotation matrix



Aside: Matrix-vector multiplication

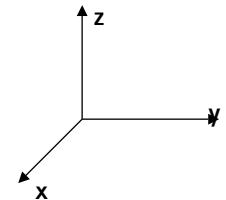
- Two ways of thinking and calculating the outcome $\begin{bmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
- Component at a time
 - dot each row at a time with the vector
 - "the x component will be $(\cos \beta, -\sin \beta) \cdot (x, y) = x \cos \beta - y \sin \beta$ and the y component ..."
- Linear combination of vectors (columns)
 - the new "x vector" is $(\cos \beta, \sin \beta)$
 - and the new "y" is $(-\sin \beta, \cos \beta)$
 - and the outcome is $x \begin{bmatrix} \cos \beta \\ \sin \beta \end{bmatrix} + y \begin{bmatrix} -\sin \beta \\ \cos \beta \end{bmatrix} = \begin{bmatrix} x \cos \beta - y \sin \beta \\ x \sin \beta + y \cos \beta \end{bmatrix}$

often a much better geometrical intuition!

Rotation around a coordinate axis

- $\mathbf{q} = \mathbf{R} \mathbf{p}$
- What's the 3D rotation matrix around z?

$$\mathbf{R}_z = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



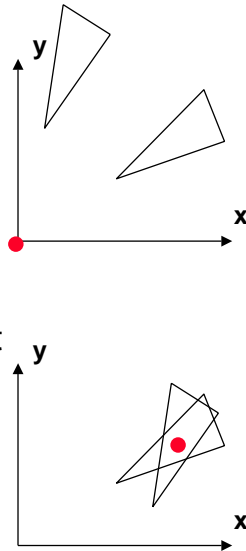
- Around x? y?

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

$$\mathbf{R}_y = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{bmatrix}$$

Rotation around a point

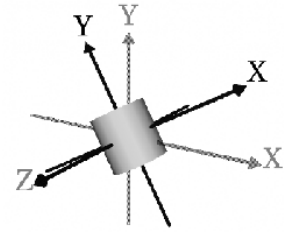
- Rotation takes normally place around the origin
 - origin is the only stationary point
 - so object away from it also moves
- If you want to rotate around a certain point, e.g., center of object
 - move the center to origin
 - rotate
 - move the center back
 - $\mathbf{q} = \mathbf{R}(\mathbf{p}-\mathbf{c})+\mathbf{c}$
 - $\mathbf{q} = \mathbf{T}_c \mathbf{R} \mathbf{T}_{-c} \mathbf{p}$



Quaternions:

Rotation about an arbitrary axis

- Axis $\mathbf{n} = [x,y,z]'$, by an angle α
- Use unit quaternions $\mathbf{q} = [q_0, q_1, q_2, q_3]'$
 - unit: $\mathbf{q}' \mathbf{q} = 1$
 - a generalized imaginary number
 - q_0 the real part
 - q_1 the i component, q_2 j, q_3 k
 - $q_0 = \cos(\alpha/2)$, $[q_1, q_2, q_3]' = \sin(\alpha/2) \mathbf{n} / |\mathbf{n}|$
- from quaternion to rotation matrix



$$\mathbf{R} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Example: $\alpha = 90^\circ, \mathbf{n} = [1,1,1]$
 $\sin(45) = \cos(45) = 1/\sqrt{2}$
 $\mathbf{n}/|\mathbf{n}| = [1,1,1]/\sqrt{3}$
 $\mathbf{q} = [1/\sqrt{2}, 1/\sqrt{6}, 1/\sqrt{6}, 1/\sqrt{6}]$

Inverse of rotation

- You can read the new coordinate axes after the rotation from the columns
 - expressed in coordinates of the old frame
- Assume old coordinate axes are unit vectors, perpendicular to each other
 - they remain so after rotation
- What happens when we multiply \mathbf{R}' with \mathbf{R} ?

$$\mathbf{R} = \begin{bmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{bmatrix}$$

$$\mathbf{R}' = \begin{bmatrix} x_x & x_y & x_z \\ y_x & y_y & y_z \\ z_x & z_y & z_z \end{bmatrix}$$

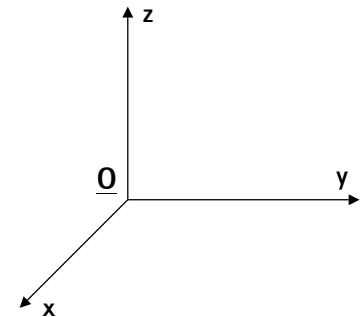
$$\mathbf{R}'\mathbf{R} = \begin{bmatrix} \mathbf{x} \cdot \mathbf{x} & \mathbf{x} \cdot \mathbf{y} & \mathbf{x} \cdot \mathbf{z} \\ \mathbf{y} \cdot \mathbf{x} & \mathbf{y} \cdot \mathbf{y} & \mathbf{y} \cdot \mathbf{z} \\ \mathbf{z} \cdot \mathbf{x} & \mathbf{z} \cdot \mathbf{y} & \mathbf{z} \cdot \mathbf{z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- so even in general the transpose of the rotation matrix is the inverse! $\mathbf{R}' = \mathbf{R}^{-1}$

Coordinate systems again

- A 3D coordinate system consists of
 - origin \mathbf{O}
 - vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$
- Form a matrix $[\mathbf{x} \ \mathbf{y} \ \mathbf{z} \ \mathbf{O}]$

$$\begin{bmatrix} x_x & y_x & z_x & \mathbf{O}_x \\ x_y & y_y & z_y & \mathbf{O}_y \\ x_z & y_z & z_z & \mathbf{O}_z \end{bmatrix}$$
- Form a point
 - start from the origin, "move" it by adding some multiples of the vectors
 - $[\mathbf{x} \ \mathbf{y} \ \mathbf{z} \ \mathbf{O}] [a \ b \ c \ 1]'$
- Form a vector
 - just add multiples of vectors, ignore the origin
 - $[\mathbf{x} \ \mathbf{y} \ \mathbf{z} \ \mathbf{O}] [a \ b \ c \ 0]'$



Homogeneous coordinates

- Move input, output, and the transformation matrix into 4D :
 - $[x, y, z]^t \Rightarrow [x, y, z, w]^t$
 - $w = 1$ for a point
 - $w = 0$ for a vector
- Generalize even more: allow any w
 - also, any scalar multiple is the same point
 - $[x, y, z, w]^t = [ax, ay, az, aw]^t$
- Back to 3D by a central projection, through the origin, to the plane $w = 1$
 - $[x, y, z, w]^t = [x/w, y/w, z/w, w/w]^t \Rightarrow [x/w, y/w, z/w]^t$
 - a vector becomes a point at infinity
- These are called homogeneous coordinates

$$\begin{bmatrix} x_x & y_x & z_x & \mathbf{O}_x \\ x_y & y_y & z_y & \mathbf{O}_y \\ x_z & y_z & z_z & \mathbf{O}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous coordinates

- The convention of homogenous coordinates allows both rotations and translations to be done using the same operation
 - matrix multiplication
- Simplifies
 - analysis
 - hardware implementation

$$\mathbf{R} = \begin{bmatrix} x_x & y_x & z_x & 0 \\ x_y & y_y & z_y & 0 \\ x_z & y_z & z_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

An example exam question

- Question:
 - Give the 2D homogeneous matrix that performs a 45 degree counterclockwise rotation around point (1,2).
- Answer 1:
 - Hmm... rotation, I recall that... it's that matrix thing
 - The angle? I'll just use calculator...
 - $\sin(45) = 0.707$
 - Hey, $\cos(45)$ is also 0.707!
 - Hmm... the point was [1 2]'
 - I guess I'll just multiply that out
 - and so on...

$$\begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix}$$

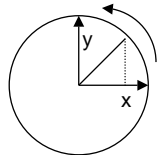
$$\begin{bmatrix} 0.707 & 0.707 \\ -0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Better answer

- Rotation
 - but where's the negative sign?
 - 90 deg ccw rotation moves (1,0) to (0,1)
 - $\cos(90) = 0, \sin(90) = 1$
 - check: minus goes to upper right
 - angle?
 - sin and cos were x and y on the unit circle
 - at 45 deg $x = y$ so $\cos(45) = \sin(45)$
 - $\cos^2(x) + \sin^2(x) = 1$
 - $\Rightarrow 2 \cos^2(45) = 1 \Rightarrow \cos(45) = \sqrt{1/2}$
 - homogeneous?
 - right, add that extra row and column so translations and rotations can be multiplied together

$$\begin{bmatrix} \cos \alpha & + - \sin \alpha \\ + - \sin \alpha & \cos \alpha \end{bmatrix}$$

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



$$\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Translation
 - first center to origin $T(-1,-2)$
 - then rotate $R(45)$
 - move center back $T(1,2)$
- Solution
 - a matrix (not point)

$$\mathbf{T}(1,2) * \mathbf{R}(45) * \mathbf{T}(-1,-2) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 1 \\ 1/\sqrt{2} & 1/\sqrt{2} & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 1+1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} & 2-3/\sqrt{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Compounded transformation

- Now as both transformations are multiplications, we can combine them:

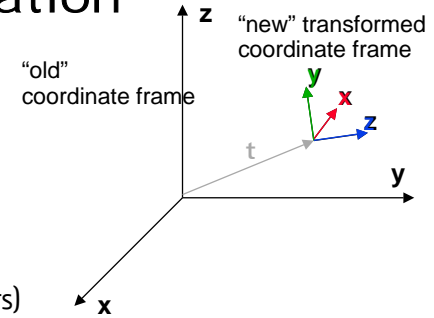
$$\mathbf{q} = (\mathbf{R} (\mathbf{T} (\mathbf{T} (\mathbf{R} \mathbf{p})))) = (\mathbf{R} \mathbf{T} \mathbf{T} \mathbf{R}) \mathbf{p} = \mathbf{M} \mathbf{p}$$

- Which takes more operations? **multiplying matrices first: 3*64+16 vs. 4*16**
matrix * matrix = 64 muls
matrix * vector = 16 muls
- Why is this useful?

typically you have tens, even thousands of points with the same transformations better to precompute them then just 16 muls / vertex

A rigid homogeneous transformation

$$\mathbf{M} = \begin{bmatrix} x_x & y_x & z_x & t_x \\ x_y & y_y & z_y & t_y \\ x_z & y_z & z_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}$$



Columns are new axes (homogeneous vectors) and the location of the new origin (homogeneous point) in the original coordinate system

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{t} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} = \mathbf{TR}$$

Rigid transformation is a rotation followed by a translation

Inverse of rigid transformation

Method 1: Inverse is also a rigid transformation, and applied after the transformation yields identity. Analyze what it must be.

$$\mathbf{M}\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}\mathbf{A} & \mathbf{R}\mathbf{b} + \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} = \mathbf{I}_{4 \times 4}$$

$$\mathbf{R}\mathbf{A} = \mathbf{I} \Rightarrow \mathbf{A} = \mathbf{R}' \quad \mathbf{R}\mathbf{b} + \mathbf{t} = \mathbf{0} \Rightarrow \mathbf{b} = -\mathbf{R}'\mathbf{t}$$

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{R}' & -\mathbf{R}'\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

Method 2: Transformation is rotation followed by translation, their inverses trivial, use rules of matrix algebra.

$$\mathbf{M} = \mathbf{TR}$$

$$\mathbf{M}^{-1} = (\mathbf{TR})^{-1} = \mathbf{R}^{-1}\mathbf{T}^{-1}$$

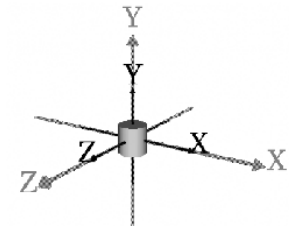
$$= \begin{bmatrix} \mathbf{R}' & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} & -\mathbf{t} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}' & -\mathbf{R}'\mathbf{t} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}$$

Affine transformations

- Any matrix with the last row [0 0 0 1] produces an affine transformation
- Two named transformations in addition to **R** and **T**:

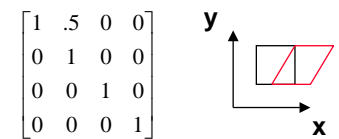
- Scaling

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



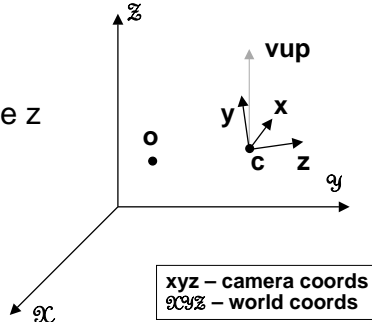
- Shearing

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



Application: Placing a camera

- Input: **c, o, vup**
 - vup = View UP vector
- Convention: camera looks into negative z
- Form a matrix for the camera
 - translation part is **c**
 - **z-axis** is the **negative view direction**, in this case the vector from **o** to **c**
 - y-axis is in the plane of z and vup, so **x-axis** is perpendicular: **vup x z**
 - **y-axis** is perp to z and x: **z x x**
 - note the order of cross products, follow the right hand rule
 - don't forget to normalize all the axes!
- Does this matrix map world to camera or the other way around?
 - maps camera coordinates to world
 - try multiplying with [0,0,0,1]', get camera origin in world coordinates!
- How do we get world-camera xform?
 - invert camera-world



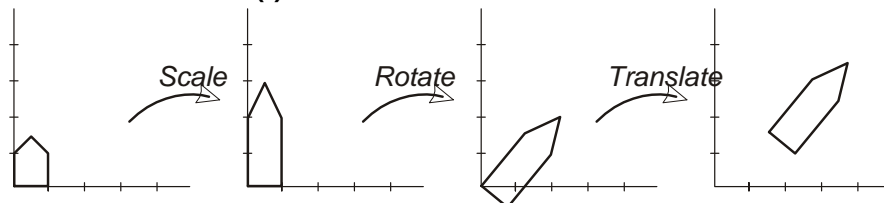
Modeling transformations in OpenGL

- `glMatrixMode(GL_MODELVIEW)`
- `glLoadIdentity()`
- `glTranslate(tx, ty, tz)`
- `glScale(sx, sy, sz)`
- `glRotate(ang_deg, nx, ny, nz)`
- **get / set:**
 - `glLoadMatrixf(mat)`
 - `mat = glGetDoublev(GL_MODELVIEW_MATRIX)`
 - **mat is a float[16], in column-major order (like Fortran)!**
- `glMultMatrixf()`
- `glPushMatrix(), glPopMatrix()`
- `gluLookAt(cx,cy,cz, ox,oy,oz, vx,vy,vz)`

Instances and transformations

- **An instance of a house is transformed by an instance transformation**

```
glMatrixMode( GL_MODELVIEW )
glLoadIdentity()
glTranslatef( ... )
glRotatef( ... )
glScalef( ... )
house()
```

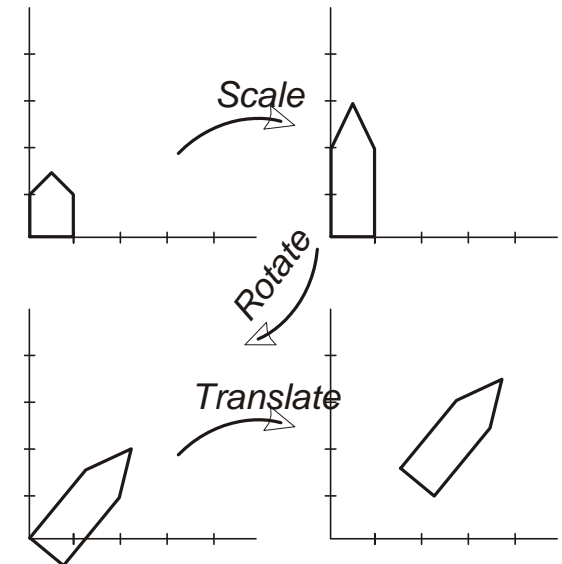


Global, fixed coordinate system

- OpenGL's transforms, logical as they may be, still seem backwards

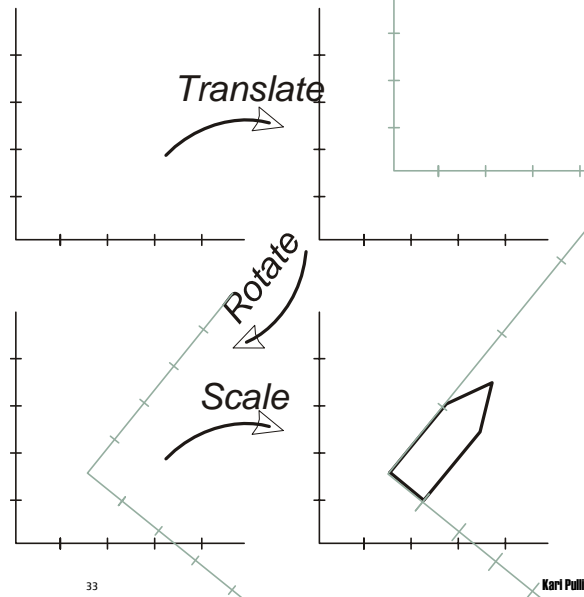
```
glLoadIdentity()
glTranslatef( ... )
glRotatef( ... )
glScalef( ... )
house()
```

- They are, if you think of them as transforming the object in a **fixed** coordinate system



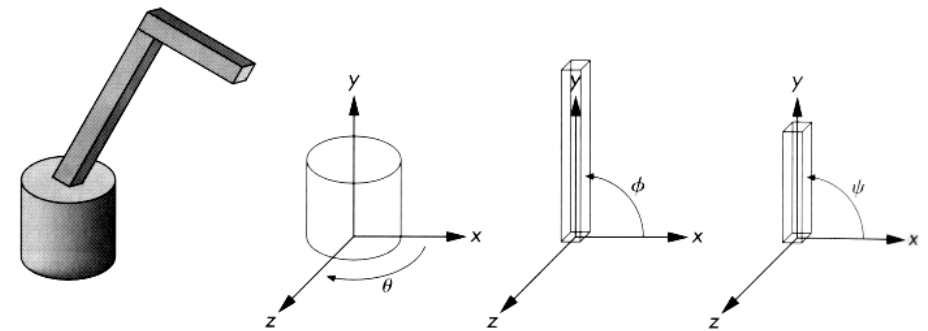
Local, changing coordinate system

- Another way to view transformations is as affecting a *local coordinate system* that the primitive is drawn in
- Now the transforms appear in the “right” order



3D Example: A robot arm

- Consider this robot arm with 3 degrees of freedom:
 - Base (size: h_{base}) rotates about its vertical axis by θ
 - Lower arm (size: h_1) rotates in its xy -plane by ϕ
 - Upper arm (size: h_2) rotates in its xy -plane by ψ



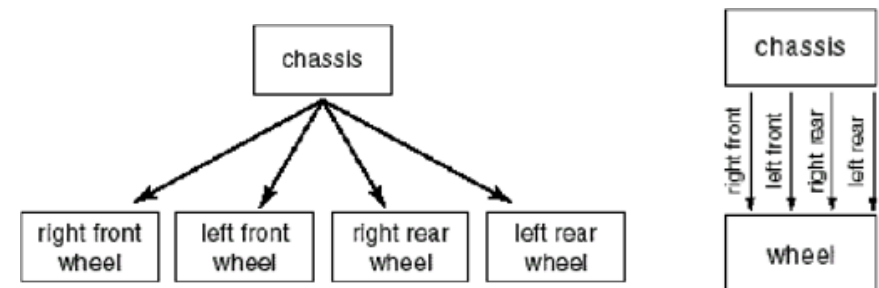
Robot arm implementation

- The robot arm can be displayed by altering the model-view matrix incrementally:

```
def robot_arm(theta, phi, psi):
    glRotatef( theta, 0.0, 1.0, 0.0 ) # rotate around y
    base()                             # draw base
    glTranslatef( 0.0, h_base, 0.0 ) # translate along y
    glRotatef( phi, 0.0, 0.0, 1.0 ) # rotate around z
    lower_arm()                         # draw lower arm
    glTranslatef( 0.0, h1, 0.0 )       # translate along y
    glRotatef( psi, 0.0, 0.0, 1.0 ) # rotate around z
    upper_arm()
```

Hierarchical modeling

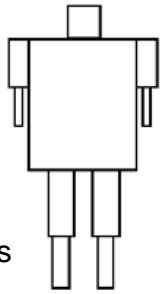
- Hierarchical models can be composed of instances using trees or DAGs:



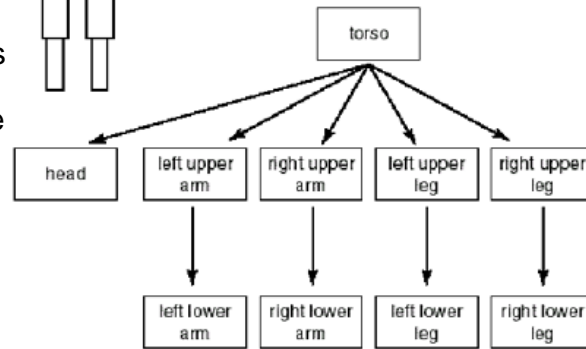
- edges contain geometric transformations
- nodes contain geometry

A complex example: human figure

- **Q:** What's the most sensible way to traverse this tree?



A: Depth-first.
Then the transformations are inherited, when you go down push (store) the current matrix, when up pop (restore) it.
That's why breadth-first doesn't make sense.



Matrix stacks

- `glMatrixMode(mode)`
 - `GL_MODELVIEW`
 - `GL_PROJECTION`
 - `GL_TEXTURE`
- Matrix commands are post-multiplied to the current matrix
 - what's issued last, is applied to the geometry first
- Save/restore current matrix
`glPushMatrix()`
`glPopMatrix()`

Human figure implementation

- The traversal can be implemented by saving the model-view matrix on a stack:

```
def figure():
    torso()

    glPushMatrix()
    glTranslate( ... )
    glRotate( ... )
    head()
    glPopMatrix()

    glPushMatrix()
    glTranslate( ... )
    glRotate( ... )
    left_upper_leg()
    glTranslate( ... )
    glRotate( ... )
    left_lower_leg()
    glPopMatrix()

    . . .
```

Human figure implementation

- We can also push attributes, such as color, into a stack
- For example we could have the "default" color, line width, light setup, etc.
 - let's assume `head()` changes the current color
 - push colors and other current attributes (normal, texture coord, ...) with `GL_CURRENT_BIT`
 - you can save all attributes at once with `GL_ALL_ATTRIB_BITS`
 - much slower

```
def figure():
    torso()

    glPushMatrix()
    glTranslate( ... )
    glRotate( ... )
    glPushAttrib(GL_CURRENT_BITS)
    head()
    glPopAttrib()
    glPopMatrix()

    glPushMatrix()
    glTranslate( ... )
    glRotate( ... )
    left_upper_leg()
    glTranslate( ... )
    glRotate( ... )
    left_lower_leg()
    glPopMatrix()

    . . .
```

Scene graphs

- The idea of hierarchical modeling can be extended to an entire scene, encompassing:
 - many different objects
 - lights
 - camera position
- This is called a **scene tree** or **scene graph**
- More about scene graphs with VRML / X3D / M3G