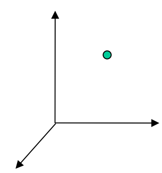
3D Computer Game Programming

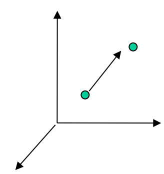
Basic Math for Game Development

VECTOR

Points vs Vectors

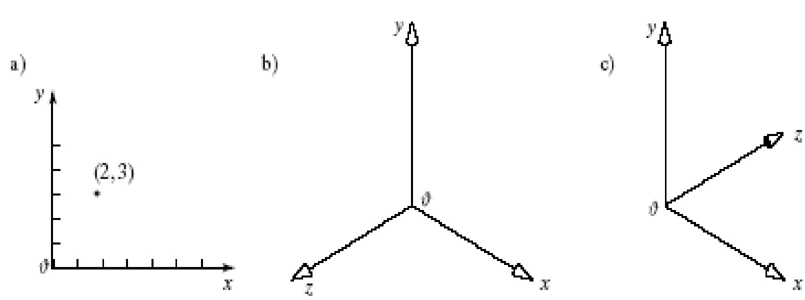
- A **point** has **position** but NOT length and direction (relative to a coordinate system).
- A **vector** represents a displacement from a point (relative to a coordinate system) and it has **length** and **direction**, but not position . It can be moved anywhere.
- A **scalar** has only **size** (a number).





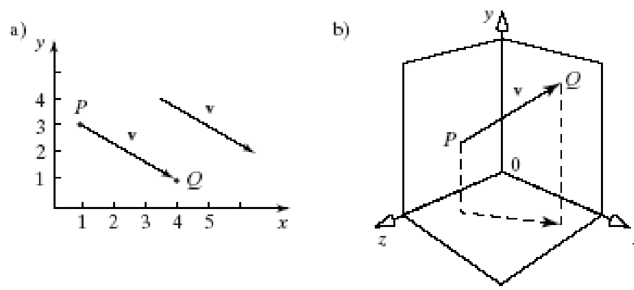
Coordinate Systems

- All points and vectors are defined relative to some coordinate system. Shown below are a 2D coordinate system and a right- and a left-handed 3-D coordinate system.



Vectors and Coordinate Systems

- A vector **v** between points P = (1, 3) and Q = (4, 1),
  - v = (3, -2)
  - calculated by (Q – P) subtracting the coordinates individually
  - To "go" from P to Q, we move down by 2 and right by 3.
  - Since v has no position, the two arrows labeled v are the same vector. The 3D case is also shown.



Vector as a Displacement b/w Two Points

- The difference between 2 points is a vector:

  v = Q – P.
- The sum of a point and a vector is a point:

  P + **v** = Q

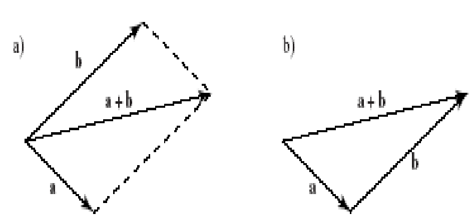
Vector Representations

- A vector **v** = (33, 142.7, 89.1) is a row vector.
- A vector **v** = $(33, 142.7, 89.1)^T$ is a column vector. It is the same as

$$v = \begin{pmatrix} 33 \\ 142.7 \\ 89.1 \end{pmatrix}$$

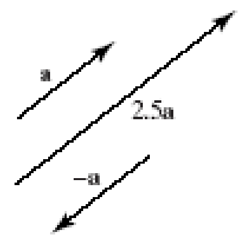
Basic Vector Operation - Addition

- Given two vectors **v** = (x, y, z) and **w** = (a, b, c)
  v + **w** = (x+a, y+b, z+c)
- Properties of Vector addition
  - Commutative: **v** + **w** = **w** + **v**
  - Associative: (**u** + **v**) + **w** = **u** + (**v** + **w**)
  - Additive Identity: **v** + 0 = **v**
  - Additive Inverse: **v** + (-**v**) = 0



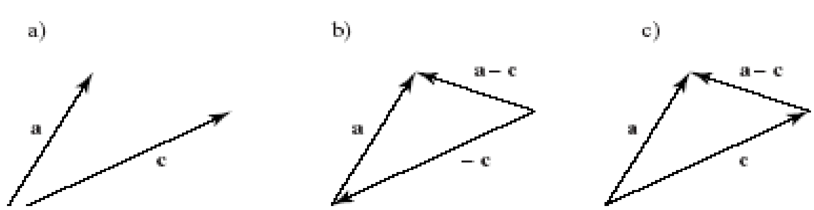
Basic Vector Operation - Multiplication

- Given **v** = (x, y, z) and a Scalar *s* and *t, s***v** = (*s*x, *s*y, *s*z) and *t***v** = (*t*x, *t*y, *t*z)
- Properties of Vector multiplication
  - Associative: (*st*)**v** = *s*(*t***v**)
  - Multiplicative Identity: 1**v** = **v**
  - Scalar Distribution: **v**(*s*+*t*) = *s***v**+*t***v**
  - Vector Distribution: *s* (**v**+**w**) = *s***v**+*s***w**
- If **v** and **w** are vectors,
  - so is **v** + **w**,
  - and so is *s***v**, where *s* is a scalar.



Basic Vector Operation - Subtraction

- Subtracting **c** from **a** is equivalent to adding **a** and (-**c**), where **–c** = (-1)**c**.



Linear Combinations of Vectors

- $\mathbf{v}_1 \pm \mathbf{v}_2 = (v_{1x} \pm v_{2x}, v_{1y} \pm v_{2y}, v_{1z} \pm v_{2z})$
- $s\mathbf{v} = (sv_x, sv_y, sv_z)$
- A linear combination of the m vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_m}$ is a vector $\mathbf{w} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_m\mathbf{v}_m$.
  - Example: 2(3, 4,-1) + 6(-1, 0, 2) forms the vector (0, 8, 10).

Vector Magnitude and Unit Vectors

- **|w|** - the magnitude (length, size) of *n*-vector **w**

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

Example:
the magnitude of **w** = (4, -2) is sqrt(20)
and that of **w** = (1, -3, 2) is sqrt(14).

- A **unit vector** has magnitude **|v|** = 1.
- The unit vector pointing in the same direction as vector **a** is **â= a**/**|a|** (if **|a|** ≠0).
- Converting **a** to **â** is called ***normalizing*** vector **a**.

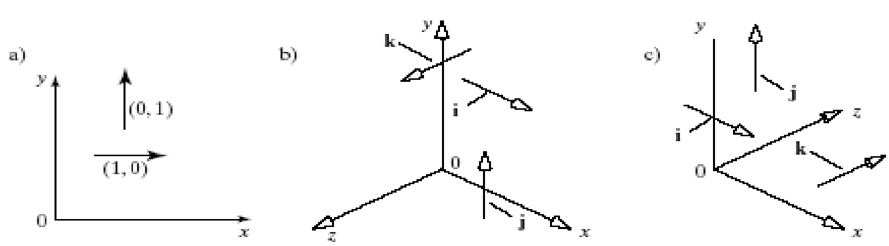
Normalizing a Vector

- v = (2,4,4)
- |v|=sqrt(4+16+16)=6

$$\hat{v} = (2/6, 4/6, 4/6) = (0.33, 0.66, 0.66)$$

Standard Unit Vectors

- The standard unit vectors in 3D are **i** = (1,0,0), **j** = (0, 1, 0), and **k** = (0, 0, 1). **k** always points in the positive z direction
- In 2D, **i** = (1,0) and **j** = (0, 1).
- The standard unit vectors are orthogonal.



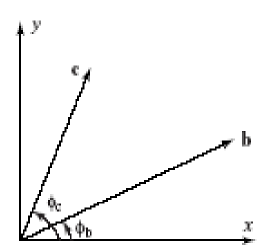
Vector Dot Product

- The dot product of n-vectors **v** and **w** is
  $\mathbf{v}\cdot\mathbf{w} = v_1w_1 + v_2w_2 + \ldots + v_nw_n$
- The dot product properties:
  - The dot product is commutative: $\mathbf{v}\cdot\mathbf{w} = \mathbf{w}\cdot\mathbf{v}$
  - The dot product is distributive: $(\mathbf{a} \pm \mathbf{b})\cdot\mathbf{c} = \mathbf{a}\cdot\mathbf{c} \pm \mathbf{b}\cdot\mathbf{c}$
  - The dot product is associative over multiplication by a scalar: $(s\mathbf{a})\cdot\mathbf{b} = s(\mathbf{a}\cdot\mathbf{b})$
  - The dot product of a vector with itself is its magnitude squared: $\mathbf{b}\cdot\mathbf{b} = |\mathbf{b}|^2$

Vector Dot Product - Angle Between 2 Vectors (1)

- Given two vectors **b** and **c** and the angle θ between **b** and **c**,

$$\cos(\theta) = \hat{\mathbf{b}}\cdot\hat{\mathbf{c}}$$



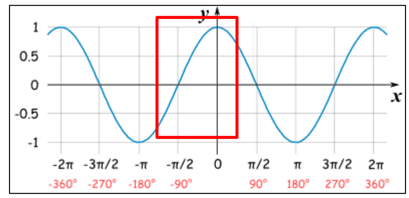
because

$\mathbf{b} = (|\mathbf{b}| \cos \varphi_b, |\mathbf{b}| \sin \varphi_b)$ and
$\mathbf{c} = (|\mathbf{c}| \cos \varphi_c, |\mathbf{c}| \sin \varphi_c)$
$\mathbf{b}\cdot\mathbf{c} = |\mathbf{b}||\mathbf{c}| \cos \varphi_c \cos \varphi_b + |\mathbf{b}||\mathbf{c}| \sin \varphi_b \sin \varphi_c$
$= |\mathbf{b}||\mathbf{c}| \cos (\varphi_c - \varphi_b)$
$= |\mathbf{b}||\mathbf{c}| \cos \theta$

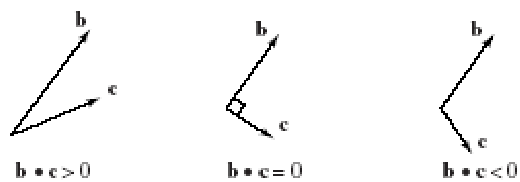
where $\theta = \varphi_c - \varphi_b$ is the smaller angle between **b** and **c**:

Vector Dot Product - Angle Between 2 Vectors (2)

- The cosine is
  - positive if $\theta < 90^o$,
  - 0 if $\theta = 90^o$,
  - and negative if $\theta > 90^o$.



- Unit vectors **b** and **c** are perpendicular (orthogonal, normal) if **b**·**c** = 0.

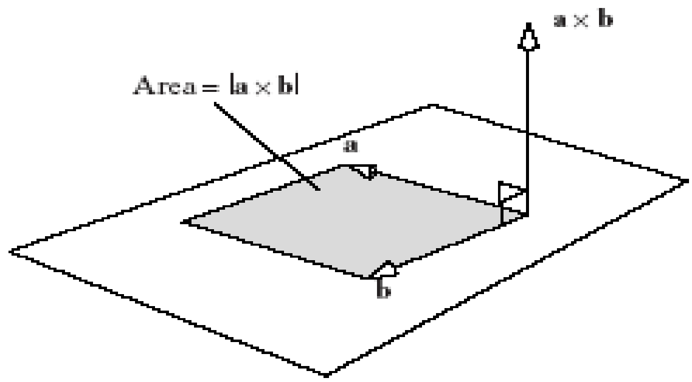


Vector Cross Product (3D Vectors Only)

- The Cross Product of **a** and **b** is denoted by **a**×**b.**
- It is a VECTOR, perpendicular to the plane defined by **a** and **b**.
- **a** x **b** = $(a_y b_z - a_z b_y)$**i** + $(a_z b_x - a_x b_z)$**j** + $(a_x b_y - a_y b_x)$**k** where **i**=(1,0,0), **j**=(0,1,0), **k**=(0,0,1)
- The determinant below also gives the result:

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

Geometric Interpretation of the Cross Product



Cross Product Example

- Ginen two vectors a=(3,-3,1) and b=(4,9,2).

1. Calculate the cross product between them.
   Sol.) axb =
   $\mathbf{i}(-3\cdot2-1\cdot9)-\mathbf{j}(3\cdot2-1\cdot4)+\mathbf{k}(3\cdot9+3\cdot4)=-15\mathbf{i}-2\mathbf{j}+39\mathbf{k}$

2. Calculate the area of the parallelogram spanned by the vectors.
   Sol.) the area is |axb| = sqrt($15^2+2^2+39^2$)

Properties of the Cross-Product

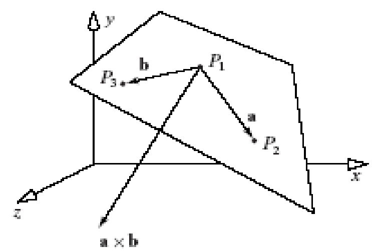
- **i** x **j** = **k**; **j** x **k** = **i**; **k** x **i** = **j**
- **a** x **b** = - **b** x **a**;
- **a** x (**b** ± **c**) = **a** x **b** ± **a** x **c**;
- (s**a**) x **b** = s(**a** x **b**)
- **a** x (**b** x **c**) ≠ (**a** x **b**) x **c**
  - for example, **a** = (a_x, a_y, 0), **b** = (b_x, b_y, 0), **c** = (0, 0, c_z)
  - **c** = **a** x **b** is perpendicular to **a** and to **b**. The direction of **c** is given by a right/left hand rule in a right/left-handed coordinate system.

Properties of the Cross-Product (2)

- **a** · (**a** x **b**) = 0
- **|a** x **b| = sqrt(|a|²|b|²-(a·b) ²)**
- **|a** x **b|** = **|a||b| sin θ**, where θ is the smaller angle between **a** and **b**.
- **|a** x **b|** is also the area of the parallelogram formed by **a** and **b**.
- **a** x **b** = 0 if **a** and **b** point in the same or opposite directions, or if one or both has length 0.

Application: Finding the Normal to a Plane

- Given any 3 non-collinear points p1, p2, and p3 in a plane, we can find a normal to the plane:
  - **a** = p2-p1, **b** = p3-p1, **n** = **a** x **b**. The normal on the other side of the plane is –**n**.



LINEAR INTERPOLATION OF 2 POINTS

Linear Interpolation of 2 Points

- Given two points A and B, a **linear interpolation (lerp)** of 2 points is given by

$$P = (1-t)A + tB$$

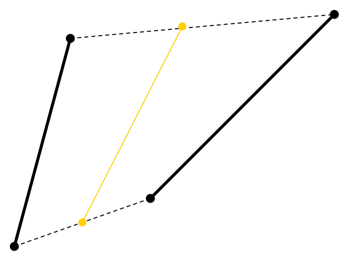
```
float lerp (float a, float b, float t)
{
   return a + (b – a) * t;
}
```


Tweening and lerp

- One often wants to compute the point *P*(*t*) that is fraction *t* of the way along the straight line from point *A* to point *B* [the tween (for in-between) at *t* of points *A* and *B*].
- Tweening takes 2 polylines (shapes) and interpolates between them (using lerp) to make one turn into another (or vice versa).

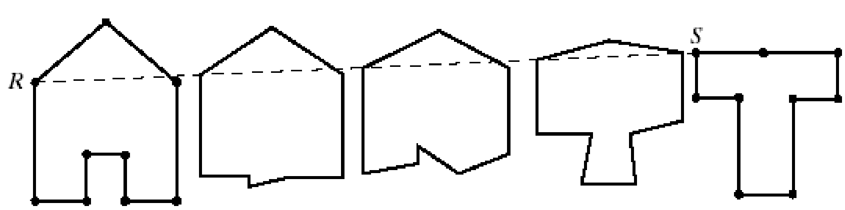
Tweening and Animation

- To start, it is easiest if you use 2 shapes with the same number of lines.



Tweening and Animation (2)

- We use polylines A and B, each with n points numbered 0, 1, …, n-1.
- We form the points $P_i(t) = (1-t)A_i + tB_i$, for t = 0.0, 0.1, …, 1.0 (or any other set of t in [0, 1]), and draw the polyline for P_i.



Tweening Examples

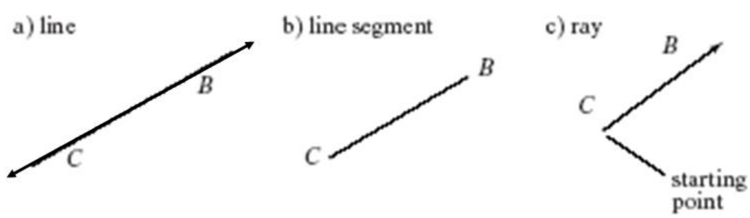
Uses of Tweening

- In films,
  - Artists draw only the key frames of an animation sequence (usually the first and last).
  - Tweening is used to generate the in-between frames.

LINE, RAY, LINE SEGMENT

Line, Line Segment, Ray

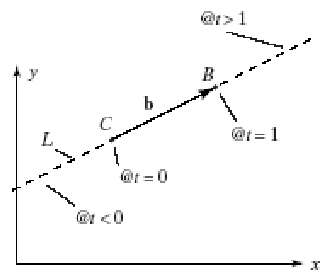
- A line passes through 2 points and is infinitely long.
- A line segment has 2 endpoints.
- A ray has a single endpoint.



Representing Lines – Parametric Form

Parametric form: Given 2 points B and C, on the line, the line equation in parametric form is

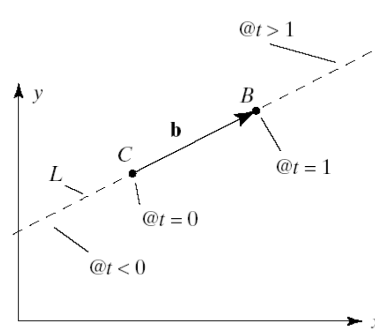
L(t) = C + ***bt***
where ***b*** *= (B-C)*
L(t) is a specific point on the line at *t*.



If $-\infty \le t \le \infty$: it represents **line.**
If $-\infty \le t \le 0$ or $0 \le t \le \infty$: **ray**.
If $0 \le t \le 1$: it represents **line segment.**

Representing Lines – Parametric

- As *t* varies so does the position of *L*(*t*) along the line. (Let *t* be time.)
- If *t*=0, *L*(0) = *C*
- If *t*=1, *L*(1) = *C* + (*B* - *C*) = *B*.
- If *t*>1, *L*(*t*) lies somewhere on the opposite side of *B* from *C*
- When *t*< 0 *L*(*t*) lies on the opposite side of *C* from *B*.
- If 0<*t*<*1, L*(*t*) lies fraction *t* of the way between *C* and *B.*
  - When *t* = 0.5, the point *L*(0.5) is the **midpoint** between *C* and *B*
  - When *t* = 0.3 the point *L*(0.3) is 30% of the way from *C* to *B*
  - The value of |t| is the ratio of the distances |*L*(*t*) - *C*| to |*B* - *C*|.



L(t) = C + ***bt*** where ***b*** *= (B-C)*

Representing Lines – Parametric Form(3)

- Find a parametric form for the line that passes through C=(3,5) and B=(2,7)

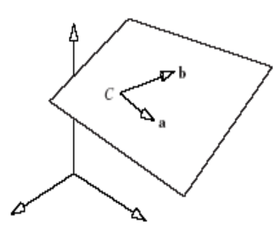
  Sol)

  $L(t) = C + \boldsymbol{b}t$ where $\boldsymbol{b} = (B\text{-}C)$

  Therefore,
  $L(t) = (3,5) + (-1,2)t = (3-t, 5+2t)$ where $-\infty \leq t \leq \infty$

Planes: Parametric Form (1)

- A plane can be infinite in 2 directions, semi-infinite, or finite.
- Parametric form:
  - requires 3 non-collinear points on the plane, A, B, and C.
  - Given A, B, C on the same plane, **the parametric form of the plane** is
    $P(s, t) = C + s\mathbf{a} + t\mathbf{b}$,
    where $\mathbf{a} = A - C$ and $\mathbf{b} = B - C$.
  - $-\infty \leq s \leq \infty$ and $-\infty \leq t \leq \infty$: infinite plane.
  - $0 \leq s \leq 1$ and $0 \leq t \leq 1$: a finite plane, or patch.



Planes: Parametric Form (2)

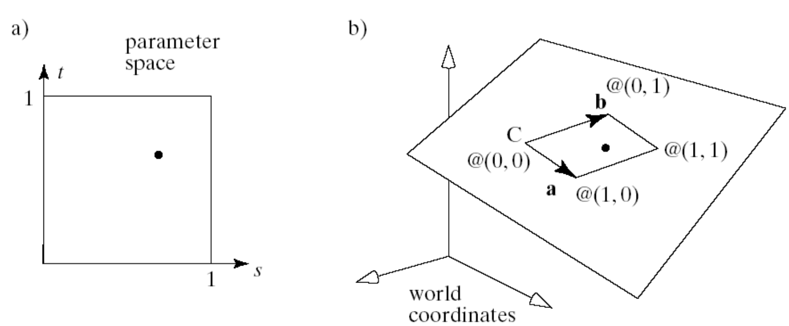
- We can rewrite

  $P(s,t) = C + s\mathbf{a} + t\mathbf{b}$,
  where $\mathbf{a} = A - C$ and $\mathbf{b} = B - C$

  as an affine combination of points:
  $P(s, t) = s\,A + t\,B + (1 - s - t)\,C$

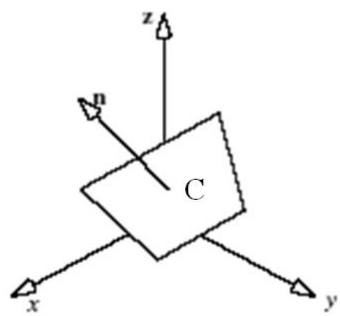
Planes: Parametric Form (3)

- The figure shows the available range of *s* and *t* as a square in **parameter space**, and the patch that results from this restriction in object space.



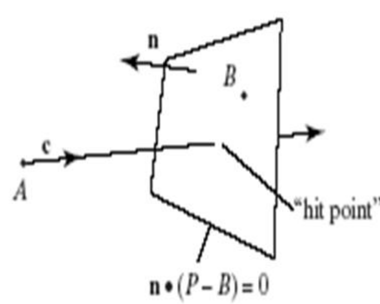
Representing Planes: Point-Normal Form

- $ax + by + cz = 1$
- Point-normal form: $\mathbf{n} \cdot (P - C) = 0$ where C is a given point on the plane and P(x,y,z) is any point on the plane.



Intersections of a Line and a Plane

- Intersections of a line and a line or plane are used in ray-tracing : we want to find the "hit point".



Intersections of a Line and a Plane (2)

- Suppose the ray A + **c** t hits at $t = t_{\text{hit}}$, the **hit time**.
  - At this value of $t = t_{\text{hit}}$, the ray and line or plane must have the same coordinates.
  - so A + **c** t_{hit} must satisfy the equation of the point normal form for the line or plane, **n**·(P – B)= 0.
- When the ray intersects (hits) the line or plane, A + **c**t_{hit} = P, giving **n**·(A + **c**t_{hit} – B) = 0.

Intersections of a Line and a Plane(3)

- Expanding and solving for t_{hit} gives

  $$t_{hit} = \mathbf{n}\cdot(B - A)/\mathbf{n}\cdot\mathbf{c}, \text{ if } \mathbf{n}\cdot\mathbf{c} \neq 0.$$
  - If **n**·**c** = 0, the line is parallel to the plane and there is no intersection.
- To find the hit/intersection point P_{hit}, substitute t_{hit} into the representation of the ray: P_{hit} = A + **c**t_{hit} = A + **c**(**n**·(B - A)/**n**·**c**).

Intersections of a Line and a Plane - Example

- Find where the ray A+**c**t hits the object **n**·(P-B)=0 given A=(2,3), **c**=(4,-4), **n**=(6,8), B=(7,7).

Sol)

n·(A+**c** t_{hit} -B)=0

t_{hit} = -7.75

Intersection = (-29,34)

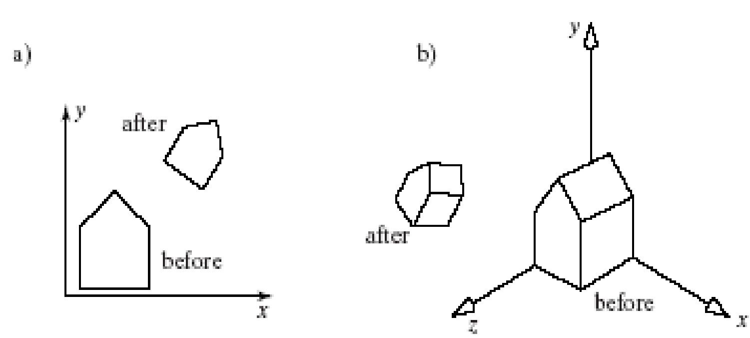
TRANSFORM

Transformation

- What is transformation?
  - Maps points (*x, y*) to another points (*x′, y′*)
- Why do we need transformations in Computer Graphics and game development?
  - To position objects in a scene
  - To change the shape of objects
  - To create multiple copies of objects
  - To do projection for virtual cameras
  - To make animations

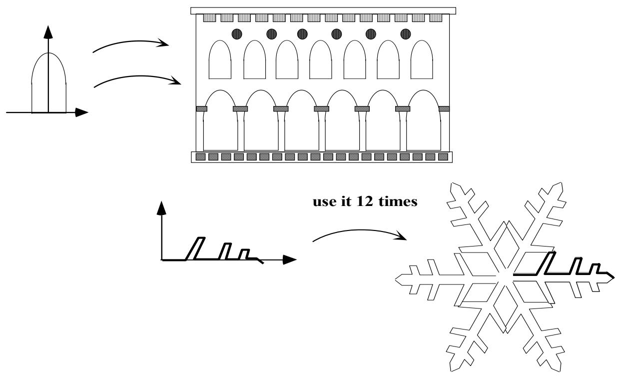
Example Transforms

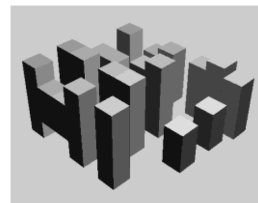
- The house has been *scaled*, *rotated* and *translated*, in both 2D and 3D.



Example Transforms

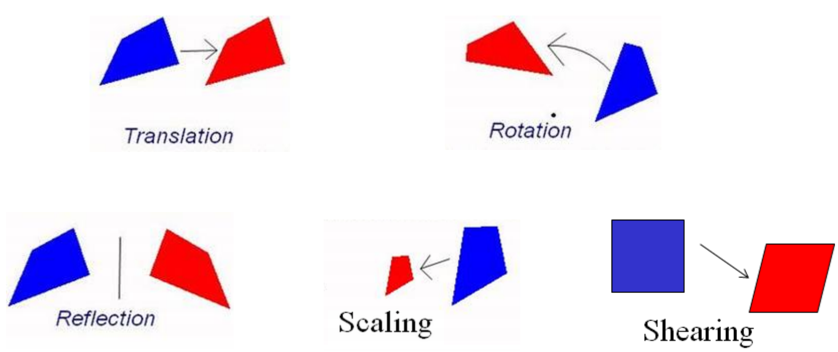
- The scene is drawn by transforming and copying a number of instances of the shape.



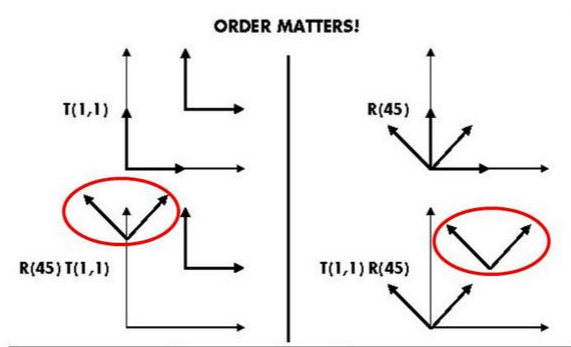


Affine Transforms

- Preserves parallel lines



Transform Order

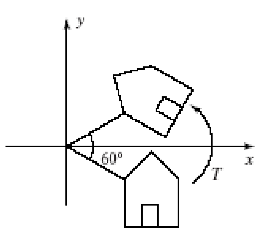


$T_a\ T_b = T_b\ T_a$, $\boxed{R_a\ R_b\ != R_b\ R_a}$ and $T_a\ R_b\ != R_b\ T_a$

rotations around different axes do not commute

Transform – 2D Rotation

- *Counterclockwise* around a point (e.g origin) by angle θ:

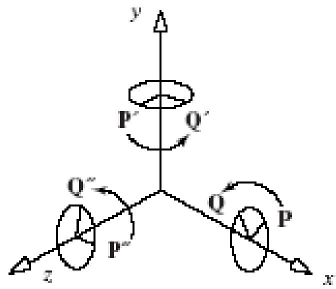


$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) P_x - \sin(\theta) P_y \\ \sin(\theta) P_x + \cos(\theta) P_y \\ 1 \end{pmatrix}$$

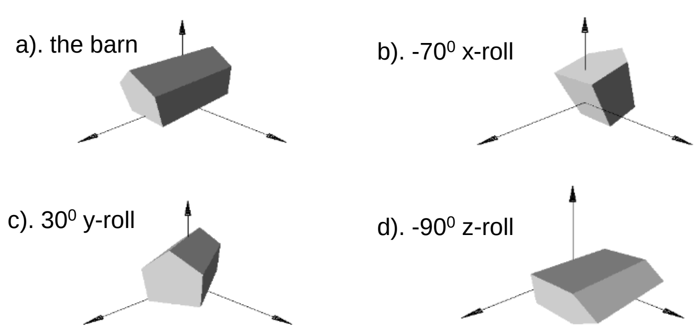
Transform – 3D Rotation

- Rotations are more complicated. We start by defining a **roll** (rotation **counter-clockwise** around an axis looking **toward** the origin):



Example

- A barn in its original orientation, and after a -70° *x*-roll, a 30° *y*-roll, and a -90° *z*-roll.



3D Rotations

- 2D rotations
  - All 2D rotations are R_z.
  - 2D rotation matrices do commute.
    - Two 2D rotations combine to make a rotation given by the sum of the rotation angles.
- In 3D the situation is much more complicated, because rotations can be about different axes.
  - The order in which two rotations about different axes are performed *does* matter.
  - 3D rotations do **not** commute**.**

Rotations about an arbitrary axis (passing through the origin)

Rotate by θ around a unit axis *r*