UNIVERSITY OF MINNESOTA
Driven to Discover®

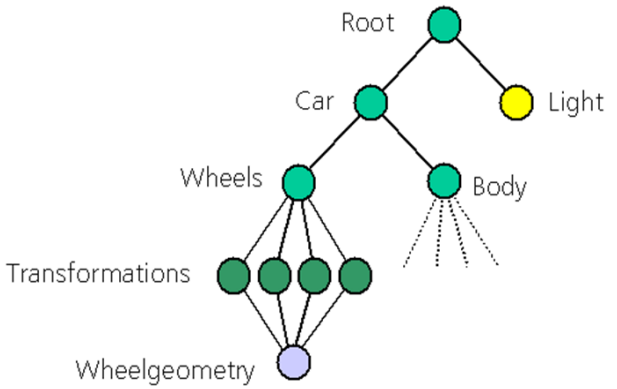
3D Transformations and Physical Simulation

CSCI 4611: Programming Interactive Computer Graphics and Games

Evan Suma Rosenberg | CSCI 4611 | Fall 2022

Geometric Transformations in Graphics

- Object and scene construction
  - Hierarchical representation of spatial relationship between components
  - Leaf nodes of the scene graph contain primitives
- Virtual camera
- 3D tracking for motion capture, animation, virtual reality, etc.
- And more...



Vector and Matrix Notation

Let's go shopping!

- Need 6 apples, 5 cans of soup, 1 box of tissues, and 2 bags of chips
- Stores A, B, and C (Super Target, Trader Joe's and Whole Foods) have following unit prices

	1 apple	1 can of soup	1 box of tissues	1 bag of chips
Super Target	$0.20	$0.93	$0.64	$1.20
Trader Joe's	$0.65	$0.95	$0.75	$1.40
Whole Paycheck	$0.95	$1.10	$0.90	$3.50

Non-Geometric Example

Shorthand representation of the situation
(assuming we can remember order of items and corresponding prices)

Column vector for quantities, q:

$$\begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

Row vectors for prices at stores:

store A (Target)	A = [0.20 0.93 0.64 1.20]
store B (Trader Joe's)	B = [0.65 0.95 0.75 1.40]
store C (Whole Paycheck)	C = [0.95 1.10 0.90 3.50]

What do I pay?

$$q = \begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

A = [0.20 0.93 0.64 1.20]
B = [0.65 0.95 0.75 1.40]
C = [0.95 1.10 0.90 3.50]

$$\text{totalCost}_A = \sum_{i=1}^{4} A_i q_i$$

$$= (0.20 \bullet 6) + (0.93 \bullet 5) + (0.64 \bullet 1) + (1.20 \bullet 2)$$
$$= (1.2 + 4.65 + 0.64 + 2.40)$$
$$= 8.89$$

$$\text{totalCost}_B = \sum_{i=1}^{4} B_i q_i = 3.9 + 4.75 + 0.75 + 2.8 = 12.2$$

$$\text{totalCost}_C = \sum_{i=1}^{4} C_i q_i = 5.7 + 5.5 + 0.9 + 7 = 19.1$$

Using Matrix Notation

We can express these sums more compactly using a **matrix**:

$$P(All) = \begin{bmatrix} totalCost_A \\ totalCost_B \\ totalCost_C \end{bmatrix} = \begin{bmatrix} 0.20 & 0.93 & 0.64 & 1.20 \\ 0.65 & 0.95 & 0.75 & 1.40 \\ 0.95 & 1.10 & 0.90 & 3.50 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

Using Matrix Notation

We can express these sums more compactly using a **matrix**:

$$P(All) = \begin{bmatrix} totalCost_A \\ totalCost_B \\ totalCost_C \end{bmatrix} = \begin{bmatrix} 0.20 & 0.93 & 0.64 & 1.20 \\ 0.65 & 0.95 & 0.75 & 1.40 \\ 0.95 & 1.10 & 0.90 & 3.50 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

Determine totalCost vector by row-column multiplication:

The dot product is the sum of the pairwise multiplications. Apply this operation to rows of prices and column of quantities.

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = ax + by + cz + dw$$

Matrix Multiplication

Each element is the dot product of a row of M with a column of N.

$$\mathbf{M} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{bmatrix} \qquad \mathbf{N} = \begin{bmatrix} n_{xx} & n_{xy} & n_{xz} \\ n_{yx} & n_{yy} & n_{yz} \\ n_{zx} & n_{zy} & n_{zz} \end{bmatrix}$$

$$\mathbf{L} = \mathbf{M}\ \mathbf{N}$$

$$\begin{bmatrix} l_{xx} & l_{xy} & l_{xz} \\ l_{yx} & l_{yy} & l_{yz} \\ l_{zx} & l_{zy} & l_{zz} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{bmatrix} \cdot \begin{bmatrix} n_{xx} & n_{xy} & n_{xz} \\ n_{yx} & n_{yy} & n_{yz} \\ n_{zx} & n_{zy} & n_{zz} \end{bmatrix}$$

$$l_{xy} = m_{xx}n_{xy} + m_{xy}n_{yy} + m_{xz}n_{zy}$$

Matrix Multiplication

Each element is the dot product of a row of M with a column of N.

$$\mathbf{M} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{bmatrix} \qquad \mathbf{N} = \begin{bmatrix} n_{xx} & n_{xy} & n_{xz} \\ n_{yx} & n_{yy} & n_{yz} \\ n_{zx} & n_{zy} & n_{zz} \end{bmatrix}$$

$$\mathbf{L} = \mathbf{M}\ \mathbf{N}$$

$$\begin{bmatrix} l_{xx} & l_{xy} & l_{xz} \\ l_{yx} & l_{yy} & l_{yz} \\ l_{zx} & l_{zy} & l_{zz} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{bmatrix} \cdot \begin{bmatrix} n_{xx} & n_{xy} & n_{xz} \\ n_{yx} & n_{yy} & n_{yz} \\ n_{zx} & n_{zy} & n_{zz} \end{bmatrix}$$

Matrix multiplication is not commutative!

$$l_{xy} = m_{xx}n_{xy} + m_{xy}n_{yy} + m_{xz}n_{zy}$$

Elemental Transformations in Computer Graphics

- Translation
- Rotation
- Scaling
- Shearing

Translation (2D Example)

Translation is the component-wise addition of vectors

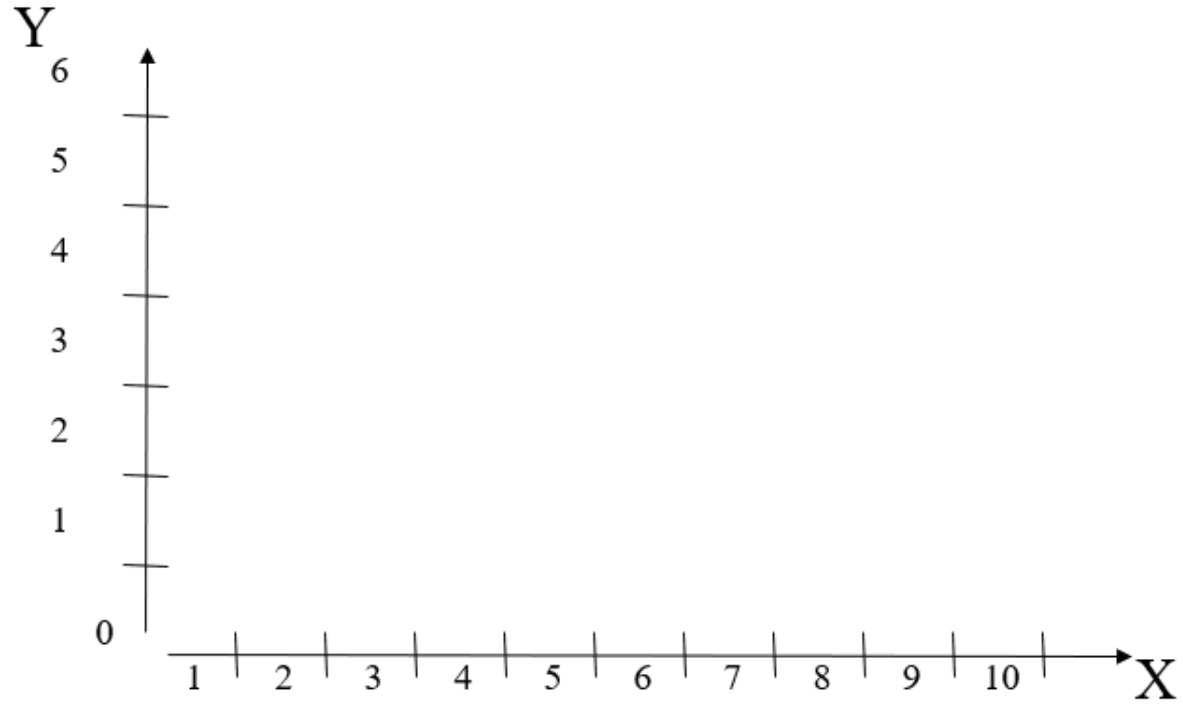
v' = v + t where

$$v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad t = \begin{bmatrix} dx \\ dy \end{bmatrix}$$

and x' = x + dx

y' = y + dy

Operation is isometric (preserves lengths)



Scaling (2D Example)

Scaling is the component-wise multiplication of vectors

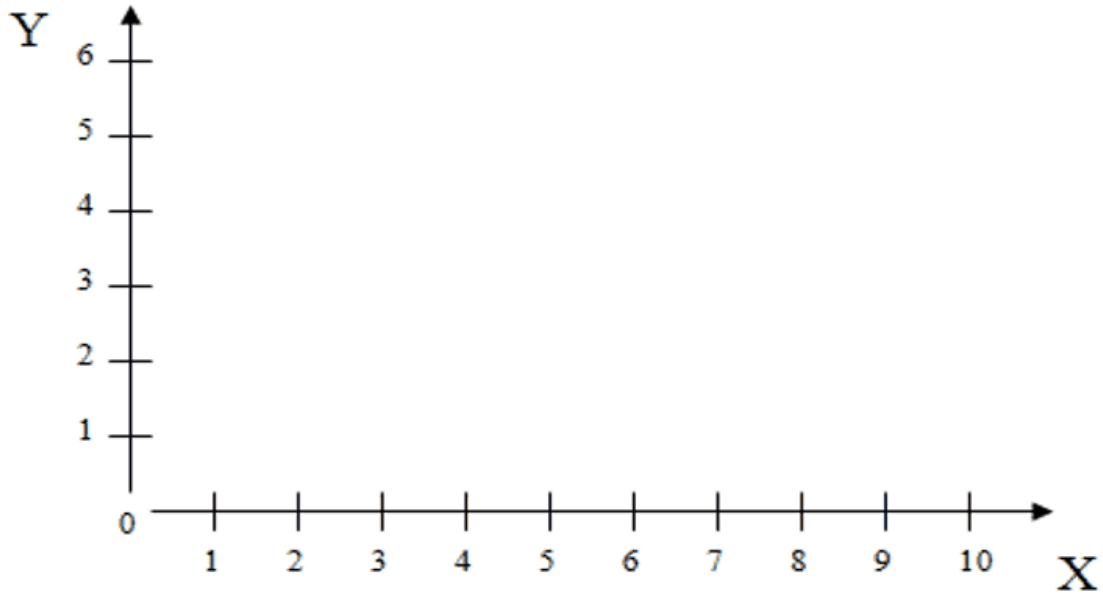
v' = Sv where $v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

and $S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$ $\quad x' = s_x x$ $\quad y' = s_y y$

Does not preserve lengths

Does not preserve angles

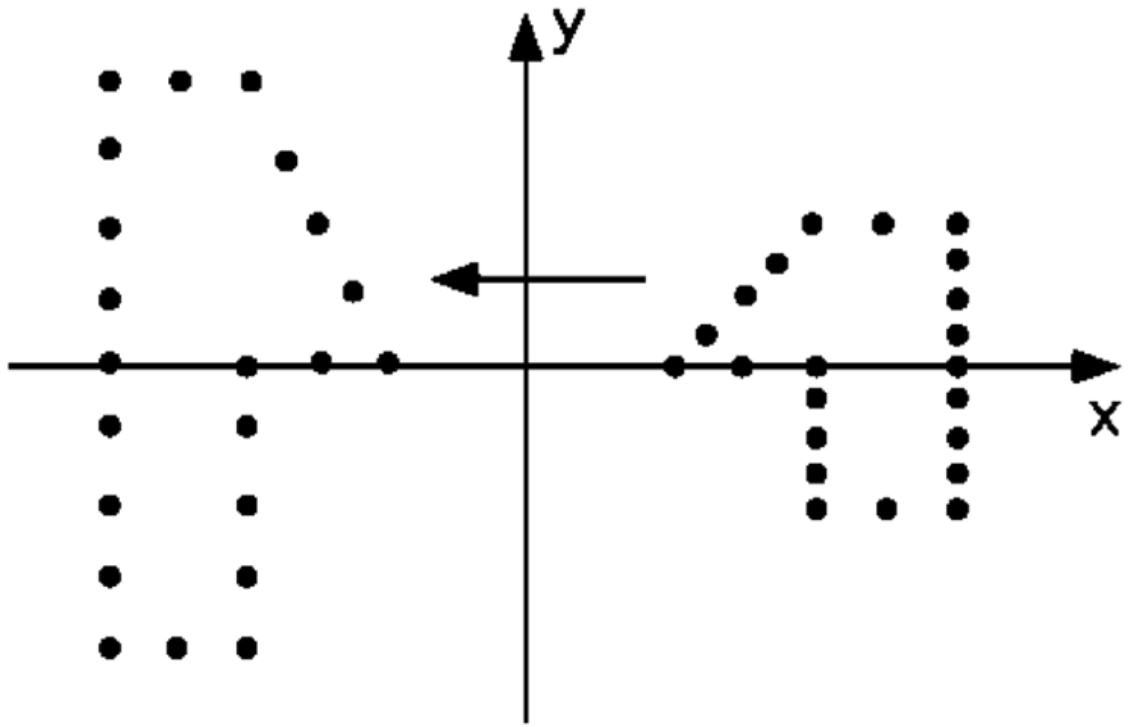
(unless scaling is uniform)



Scaling as a Reflection

The scaling (*S*x, *S*y) = (-1, 2) is applied to a collection of points.

Each point is both reflected about the y-axis and scaled by 2 in the y-direction.



Rotation (2D Example)

Rotation of Θ about the origin

$v' = R_\Theta$ where

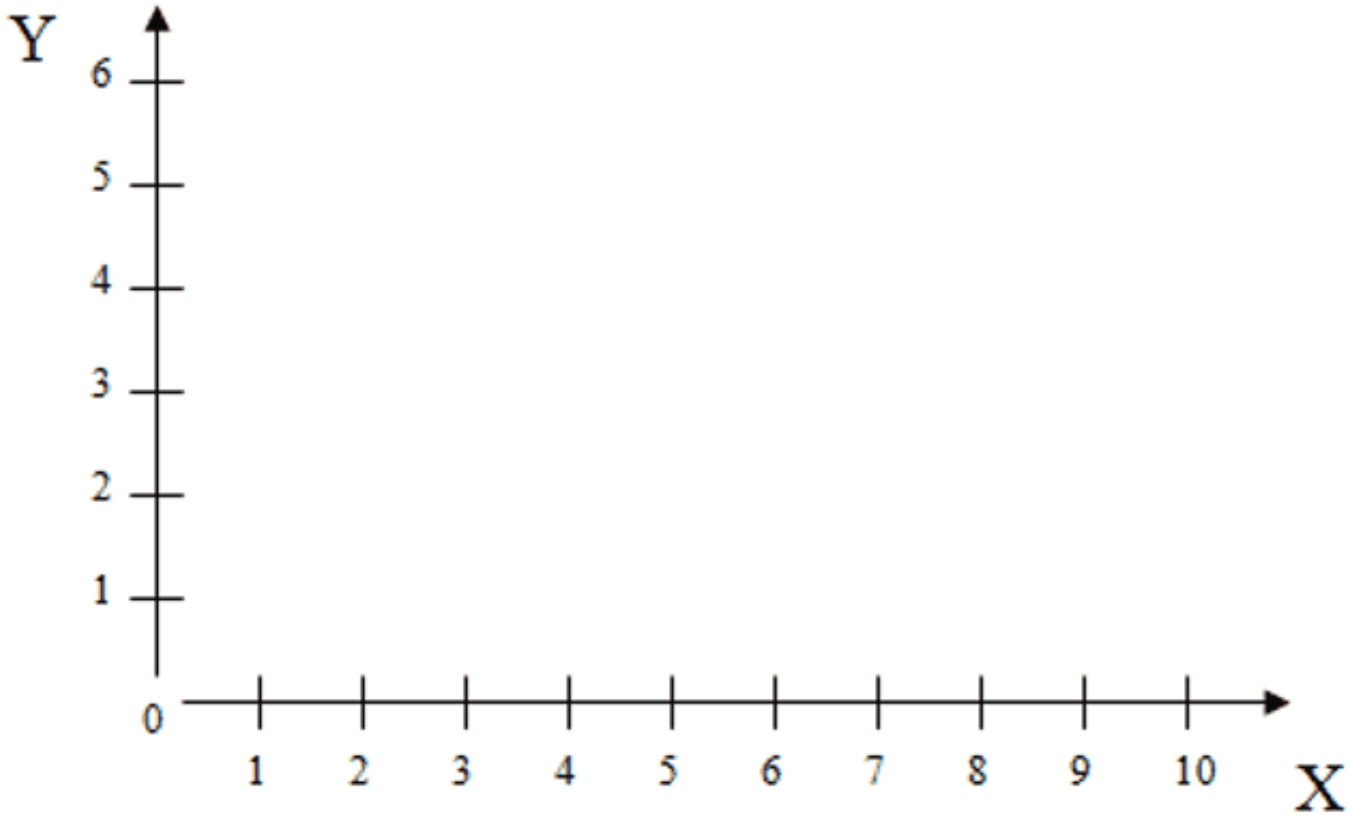
$$v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

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

and $x' = x \cos\Theta - y \sin\Theta$

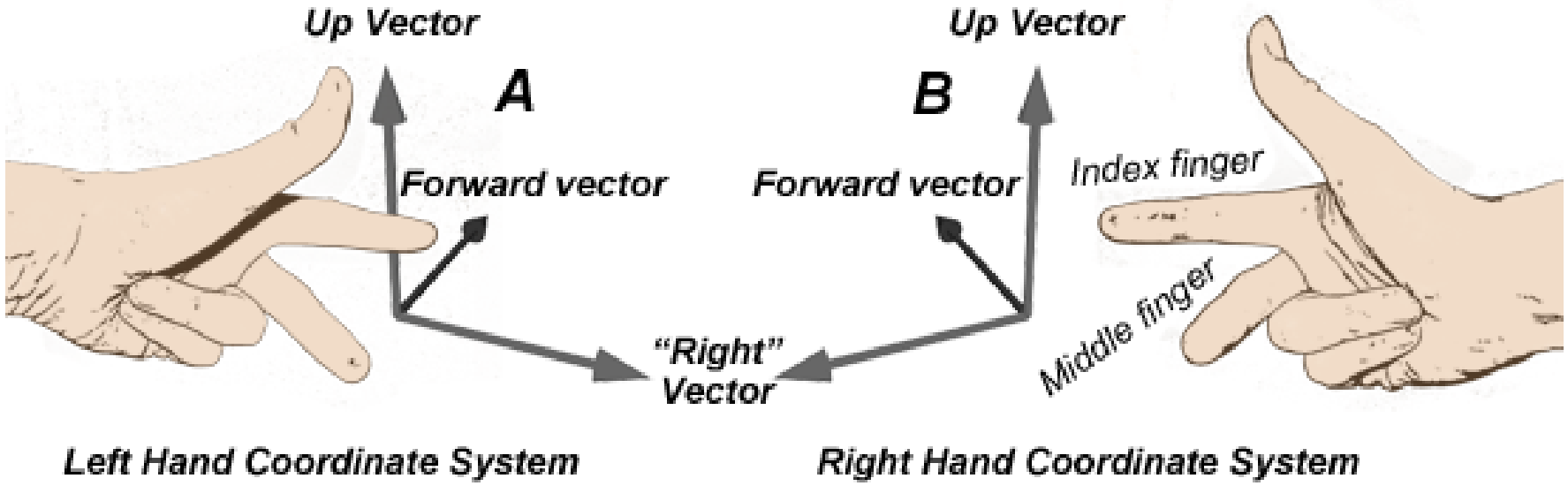
$y' = x \sin\Theta - y \cos\Theta$

A rotation of zero (no rotation) is results in the identity matrix.



3D Transformations

Left Hand vs. Right Hand Coordinate Systems



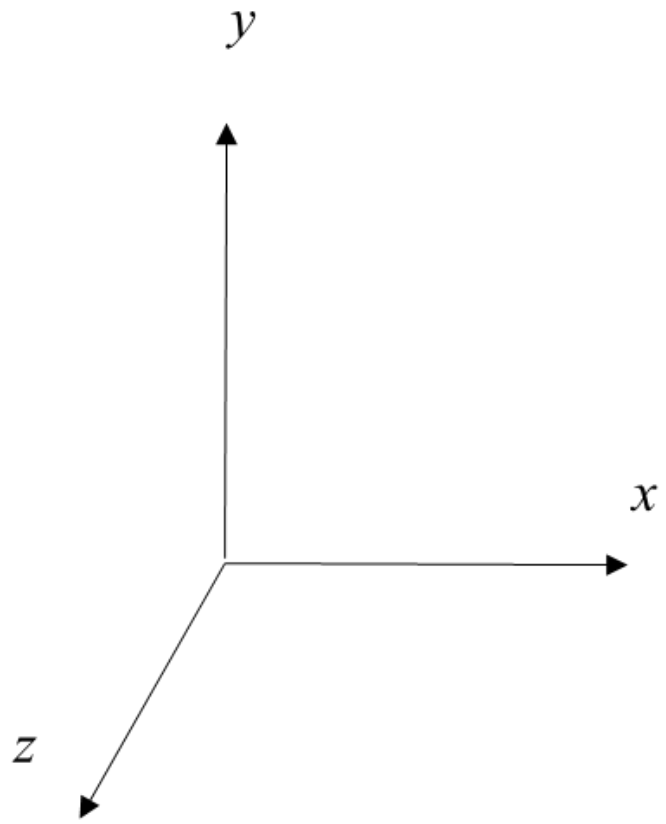
Basic 3D Transformations (Right Handed)

Translation

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Basic 3D Transformations (Right Handed)

Rotation about X-axis

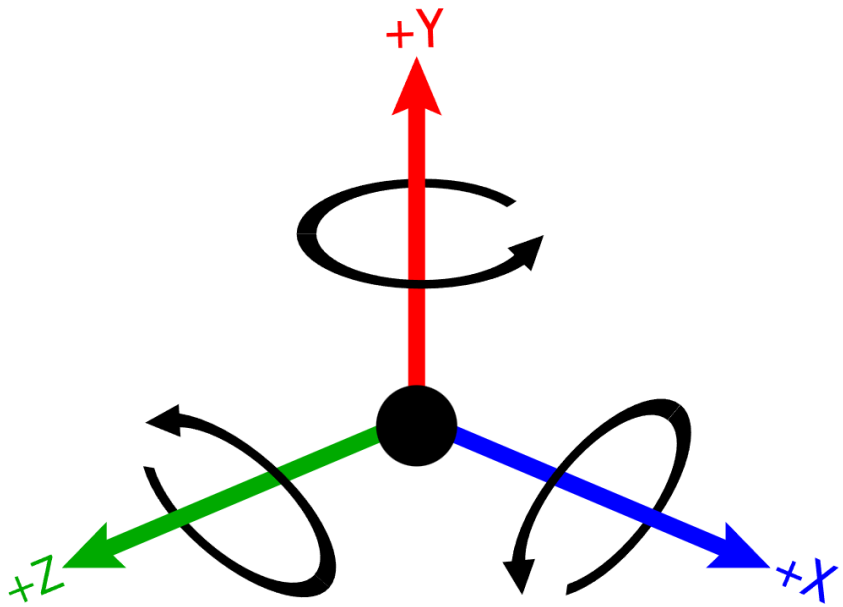
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y-axis

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

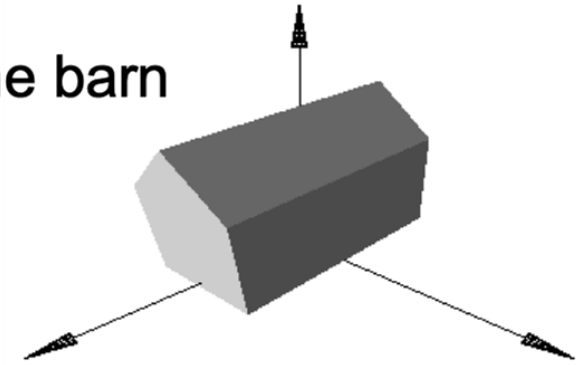
Rotation about Z-axis

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

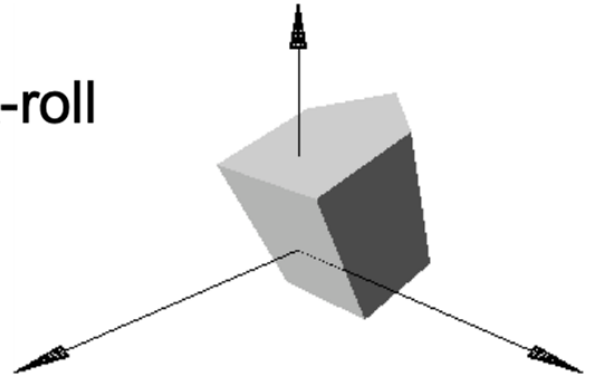


Basic 3D Rotations

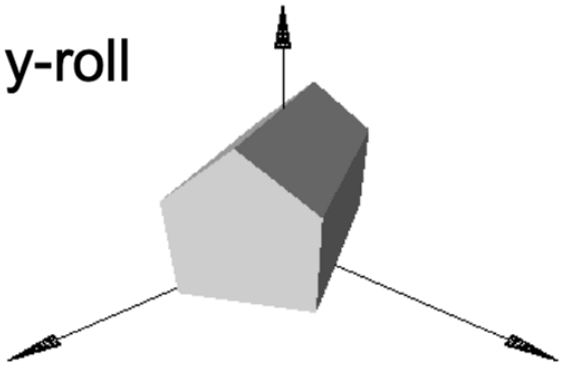
a). the barn



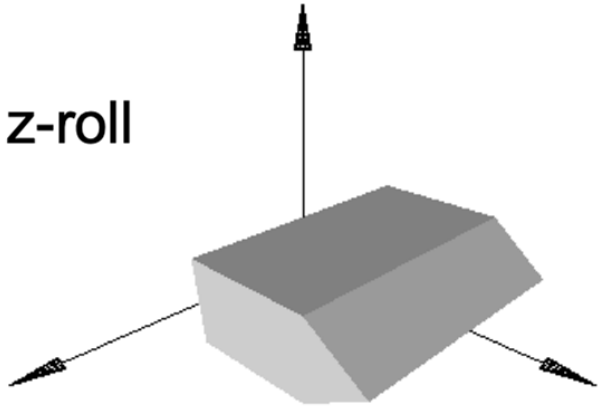
b). -70^0 x-roll



c). 30^0 y-roll



d). -90^0 z-roll



Combining 3D Rotations

One way to build a rotation in 3D is by composing three elementary rotation transformations:

an x-rotation(**pitch**),

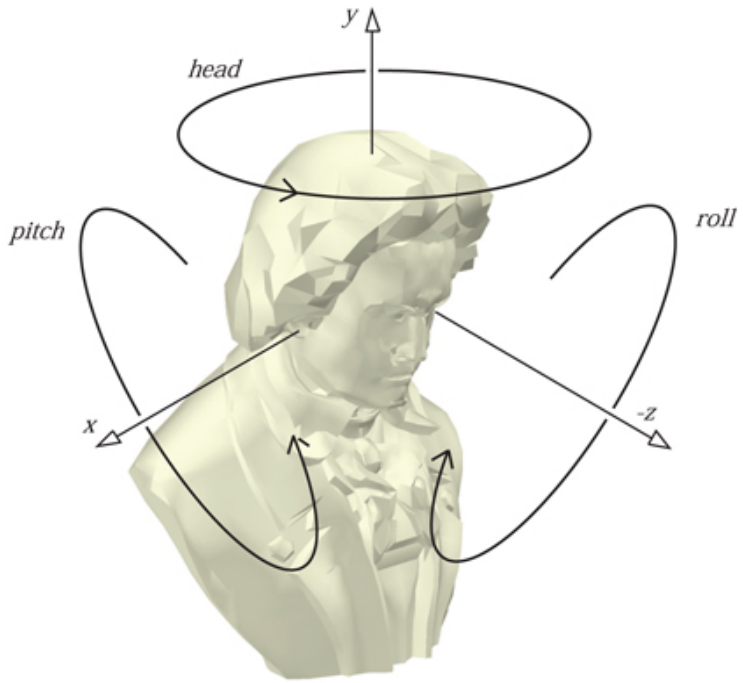
followed by a y-rotation (**yaw** or **head**),

and then a z-rotation (**roll**).

The overall rotation is given by:

$M = Rz(\beta 3)Ry(\beta 2)Rx(\beta 1)$

In this context the angles β1, β2, and β3 are often called **Euler angles.**

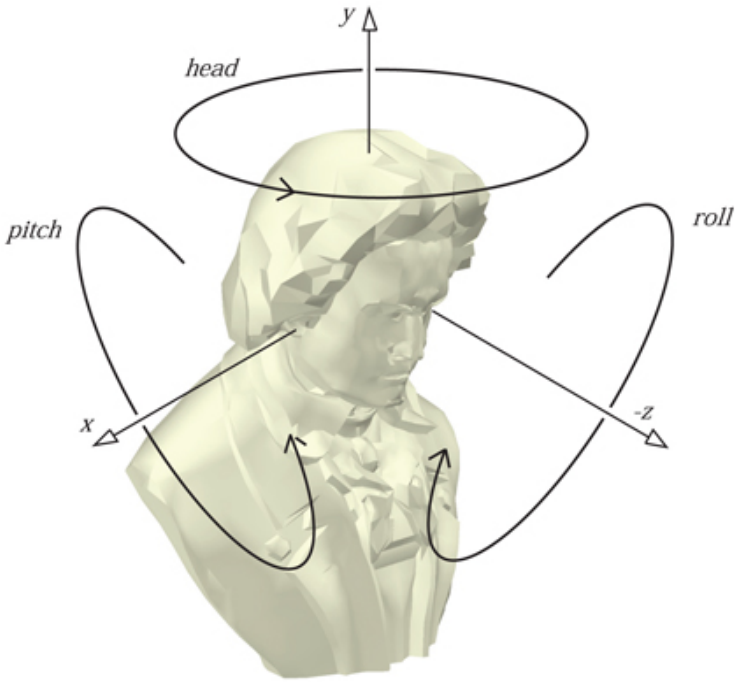


Combining 3D Rotations

One way to build a rotation in 3D is by composing three elementary rotation transformations: an x-roll followed by a y-roll, and then a z-roll. The overall rotation is given by:

$$M = Rz(\beta 3)Ry(\beta 2)Rx(\beta 1)$$

In this context the angles β1, β2, and β3 are often called **Euler angles.**



3D rotation matrices are not commutative!!!!

If you want to describe this rotation to me...

- I need to know B1, B2, B3

  AND, I need to know the order of rotation

Rotation about an Arbitrary Axis

- **Classic approach is to use Euler's theorem**

  Euler's theorem: Any sequence of rotations = one rotation about some axis.

- **So, how to rotate around arbitrary axis *u* by angle B?**

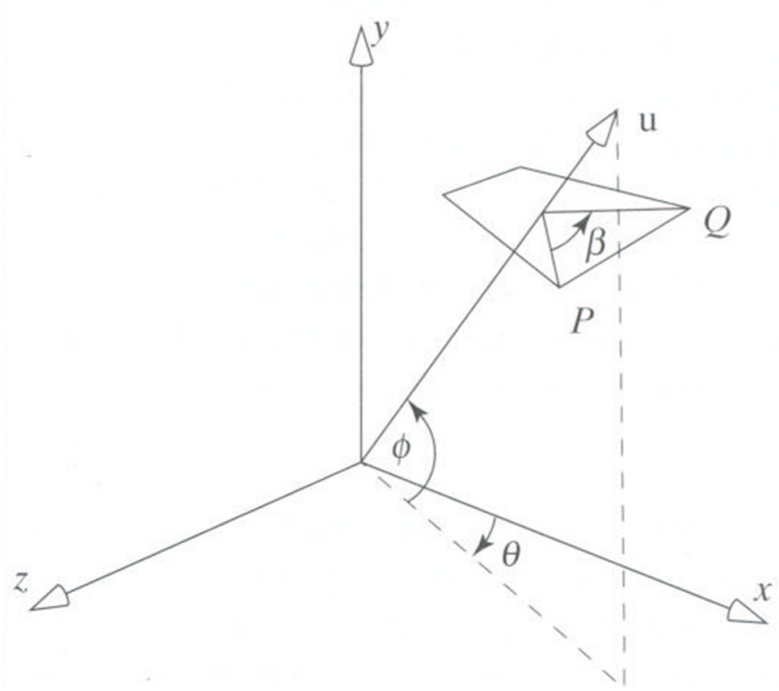
  Use 2 rotations to align *u* with the X-axis.

  Rotate around the X-axis (an X roll) by angle B.

  Undo the original 2 rotations.

Rotation about an Arbitrary Axis

$$R_u(\beta) = R_y(-\theta)R_z(\phi)R_x(\beta)R_z(-\phi)R_y(\theta)$$



That's a lot of work for just one rotation. Surely there must be a better way?

Euler Angles

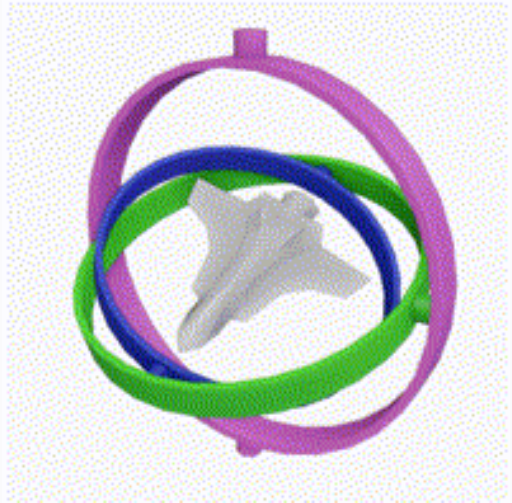
Euler angles may seem convenient because they are human-readable.

But they are not the most efficient way to represent arbitrary 3D rotations!

They suffer from a phenomenon known as **Gimbal lock**.

You cannot compute intermediate rotations or smoothly interpolate (important for animation).

Working with them in computer graphics code is a pain!



Quaternions

- **Invented in 1843 by Sir William Rowan Hamilton as an extension to the complex numbers.**
- **Introduced to the field of computer graphics in 1985 by Ken Shoemake.**
- **Quaternions are awesome!**

  They can represent any 3D rotation using 4 real numbers.

  They do not suffer from Gimbal lock.

  You can smoothly interpolate between them.

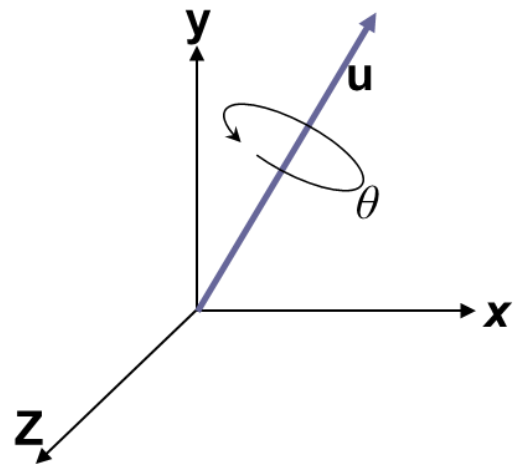
  Composing them requires fewer operations than rotation matrices.

Quaternions

Rotation representation:

note: some notations use *w* instead of *s*.

$$\mathbf{q} = (s, \mathbf{v}) = (s, (v_x, v_y, v_z))$$



Relation to axis-angle representation (u,θ):

$$s = \cos\frac{\theta}{2}, \qquad \mathbf{v} = \sin\frac{\theta}{2}\,\mathbf{u}$$

Quaternion Rotation

1. Represent point **p** as a quaternion:

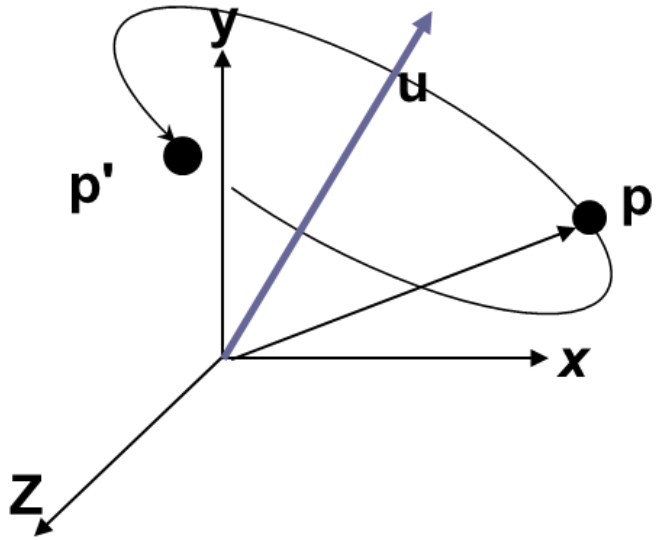
$$\mathbf{q}_\mathbf{p} = (0, \mathbf{p})$$

2. Compute rotated quaternion point representation $\mathbf{q_p}'$

$$\mathbf{q}'_\mathbf{p} = \mathbf{q}\mathbf{q}_\mathbf{p}\mathbf{q}^{-1}$$

3. Extract standard coordinate representation $\mathbf{p}'$ from $\mathbf{q_p}'$

$$\mathbf{q}'_\mathbf{p} = (0, \mathbf{p}')$$



Quaternion Math

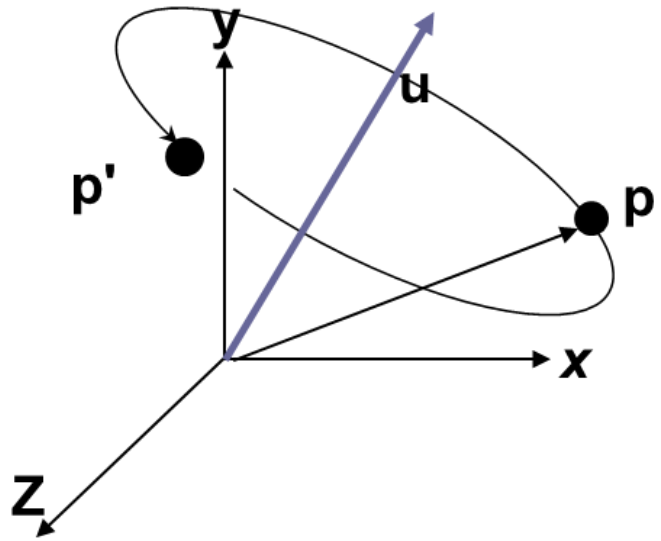
Quaternions can be composed through multiplication:

$$\mathbf{q}_1\mathbf{q}_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$

Quaternion multiplication is not commutative!

Magnitude: $|\mathbf{q}| = \sqrt{s^2 + \mathbf{v} \cdot \mathbf{v}}$

Inverse: $\mathbf{q}^{-1} = \frac{1}{|\mathbf{q}|^2}(s, -\mathbf{v})$



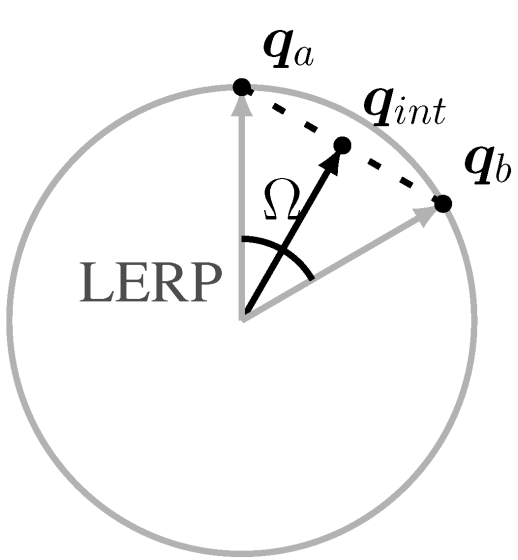
For rotations we assume unit quaternions, hence:

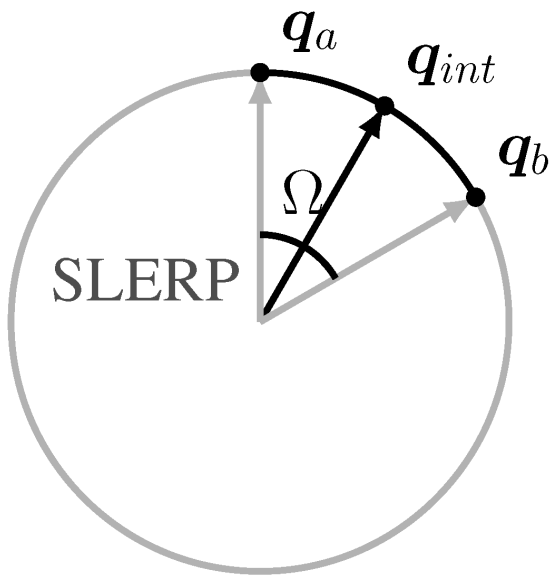
$$\mathbf{q}^{-1} = (s, -\mathbf{v}) \quad \text{when} \quad |\mathbf{q}| = 1$$

Quaternion Interpolation

Intermediate rotations can be computed using **spherical linear interpolation**.

This can be used to generate smooth transitions between two rotations (e.g., animation).





Inside the GopherGfx Transform3 Class

```
class Transform3
{
    public children: Array<Transform3>;

    public position: Vector3;
    public rotation: Quaternion;
    public scale: Vector3;
}
```

Overview of Assignment 2

https://github.com/CSCI-4611-Fall-2022/Assignment-2

Physical Simulation

Physics Simulation in a Nutshell

Everything has a position **p** and a velocity **v**.

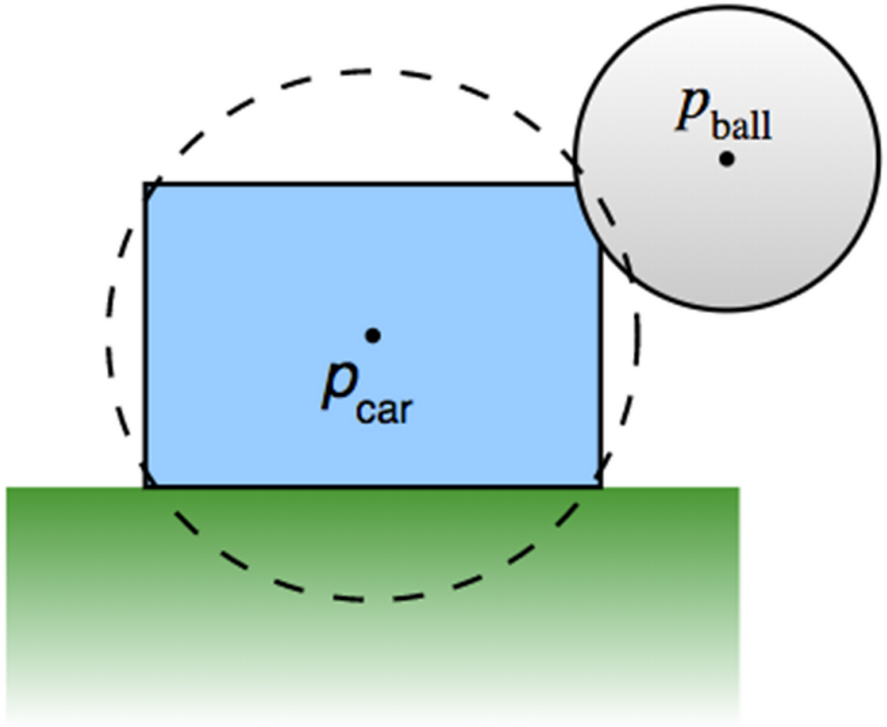
After some time Δt has passed since last frame:

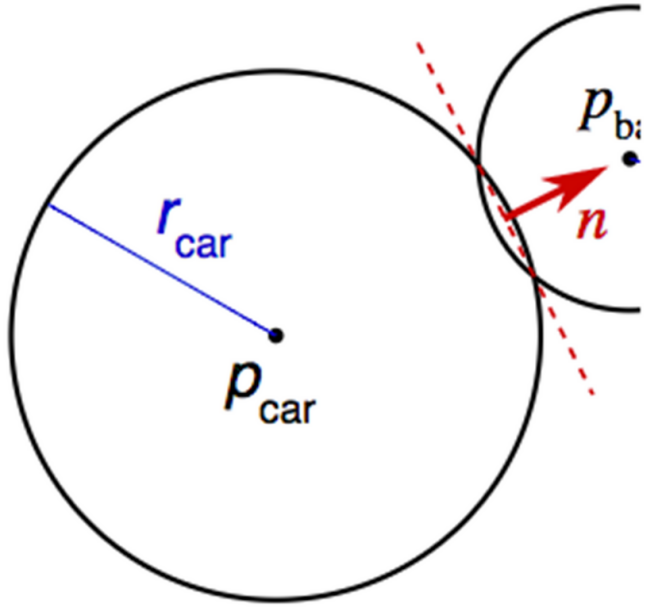
- Find all the forces, calculate acceleration $\mathbf{a} = \mathbf{F}/m$
- $\mathbf{v}_{new} = \mathbf{v}_{old} + \mathbf{a}\,\Delta t$
- $\mathbf{p}_{new} = \mathbf{p}_{old} + \mathbf{v}_{new}\,\Delta t$

  Handle collisions

In games, we often just approximate acceleration using a constant instead of precisely modeling F / *m*.

Review: Sphere Collisions

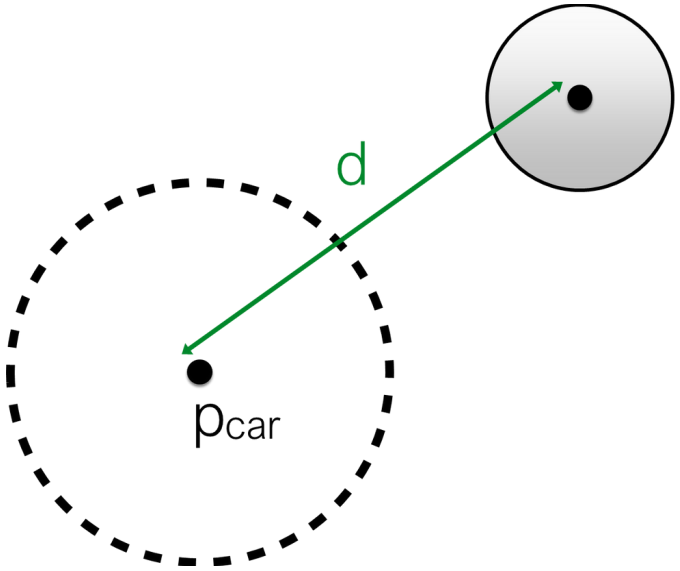




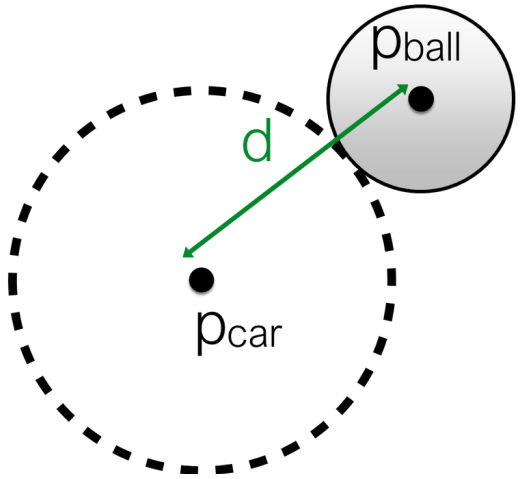
Review: Sphere Collisions

What can you tell me about the distance (d) in these three cases?

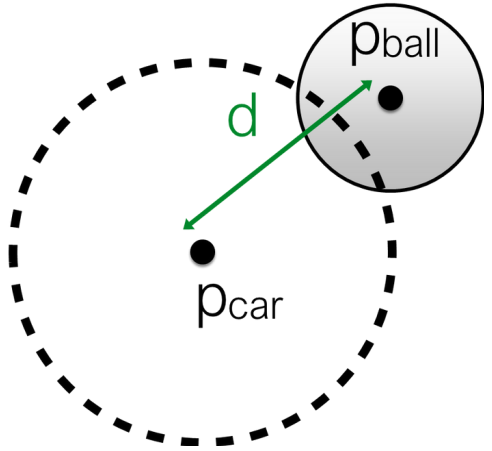
Case 1: Far Away



Case 2: Touching



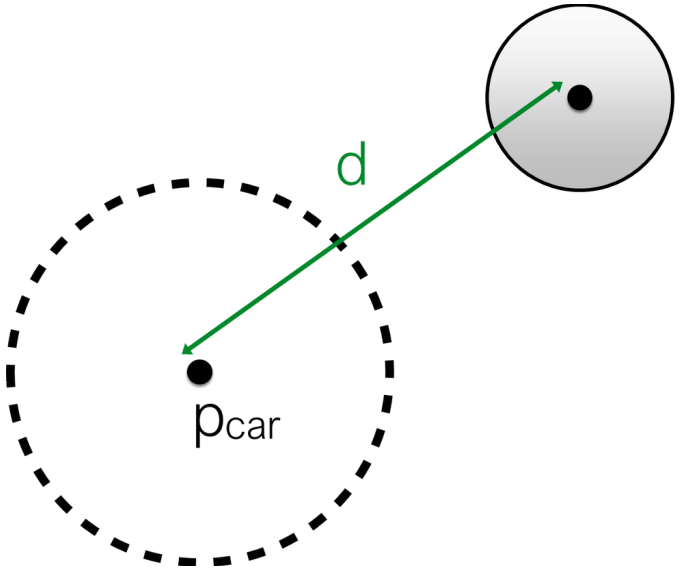
Case 3: Intersecting



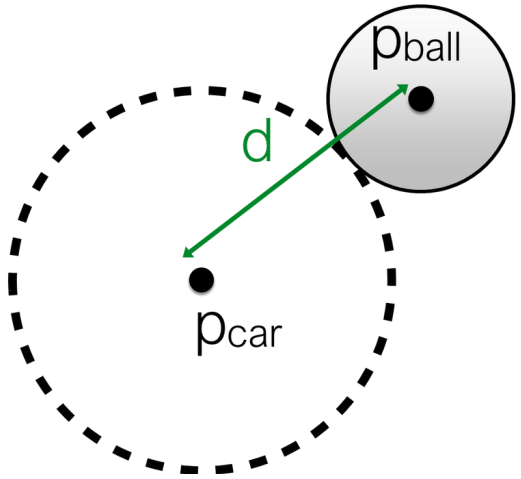
Review: Sphere Collisions

What can you tell me about the distance (d) in these three cases?

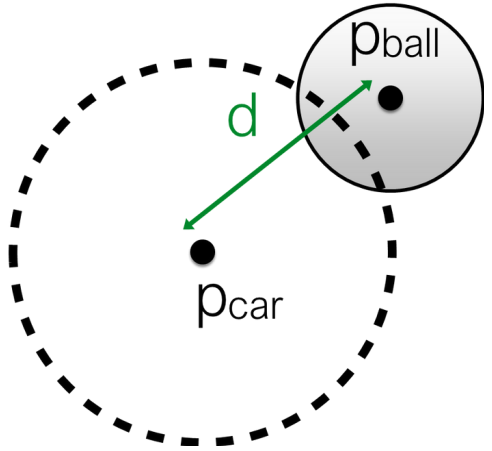
Case 1: Far Away



Case 2: Touching



Case 3: Intersecting

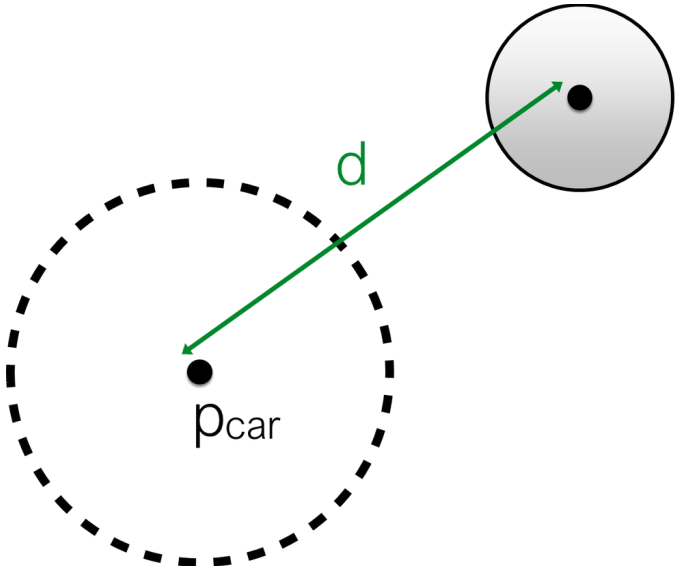


$d > (r_{car} + r_{ball})$

Review: Sphere Collisions

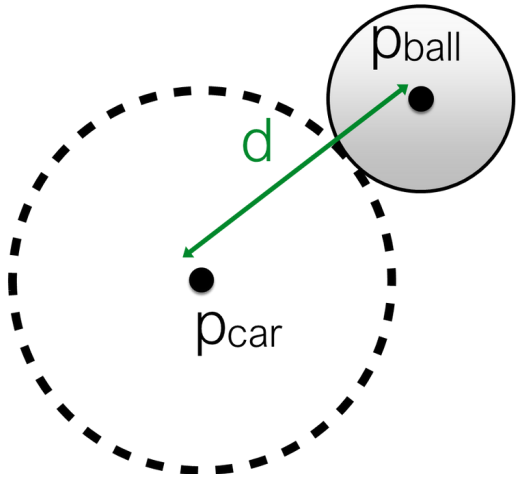
What can you tell me about the distance (d) in these three cases?

Case 1: Far Away



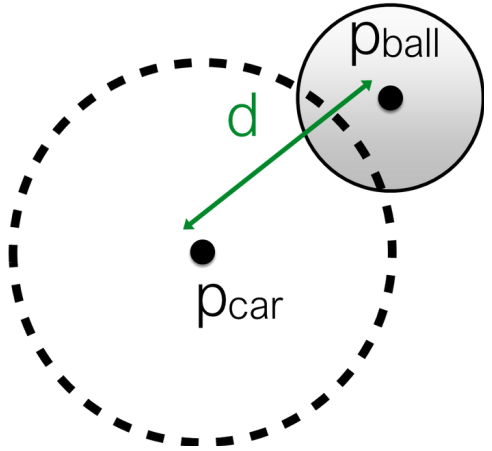
$d > (r_{car} + r_{ball})$

Case 2: Touching



$d == (r_{car} + r_{ball})$

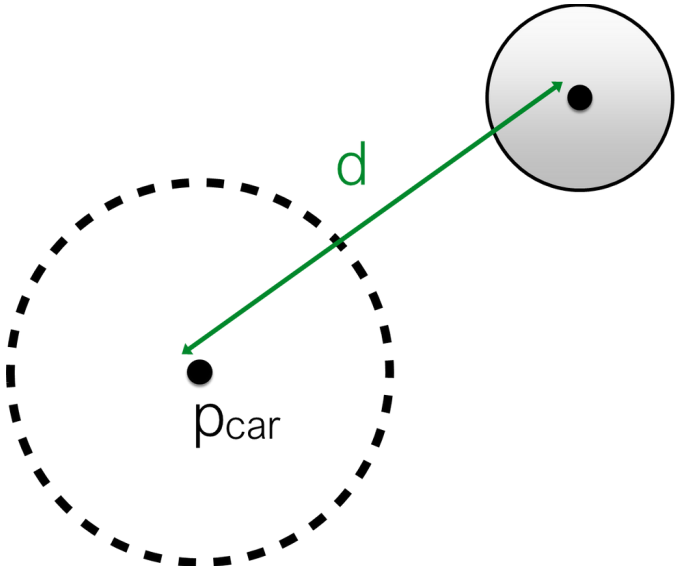
Case 3: Intersecting



Review: Sphere Collisions

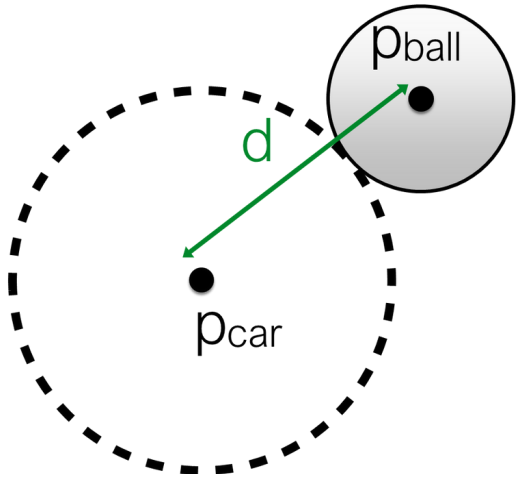
What can you tell me about the distance (d) in these three cases?

Case 1: Far Away



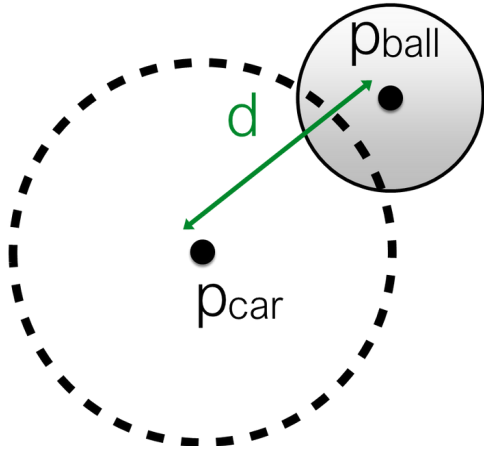
$d > (r_{car} + r_{ball})$

Case 2: Touching



$d == (r_{car} + r_{ball})$

Case 3: Intersecting



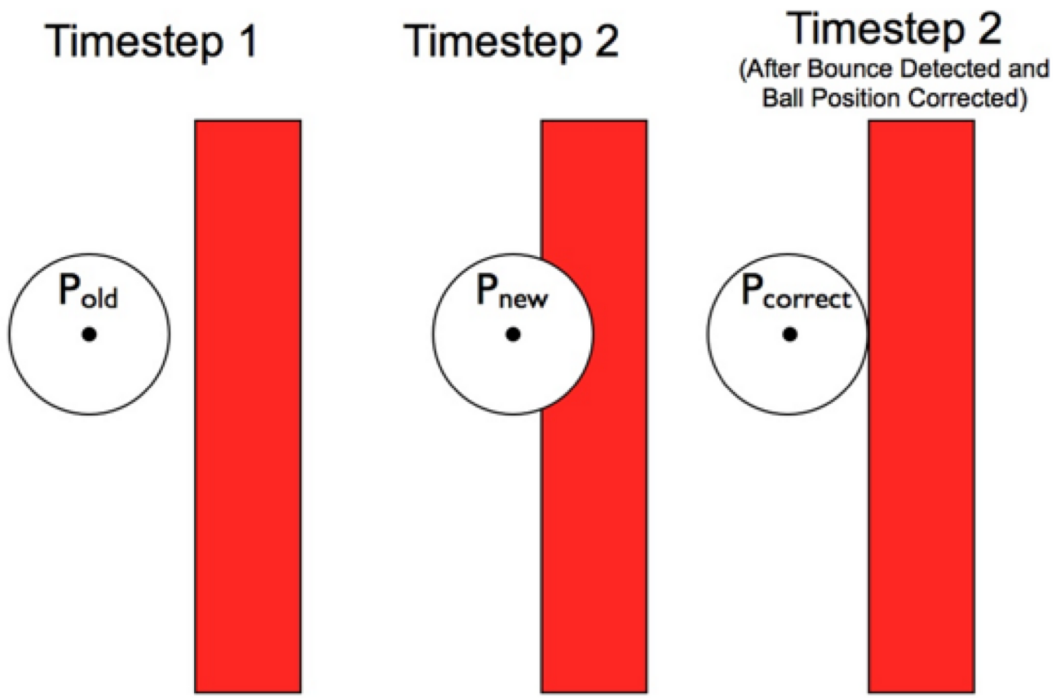
$d < (r_{car} + r_{ball})$

Collision Handling

Computer simulations use discrete timesteps, so a "collision" really means the ball has penetrated the other object.

When we detect this, we simply move the ball backwards along its velocity vector until it no longer penetrates the object.

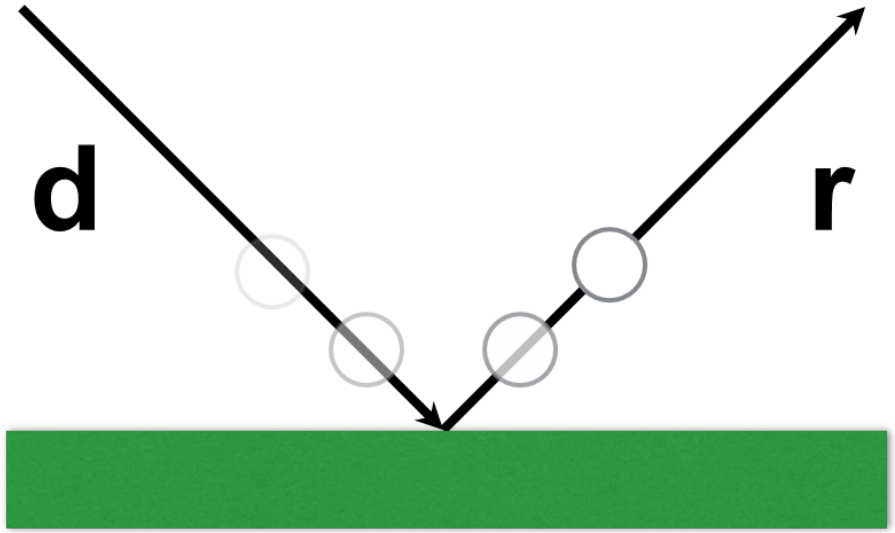
Then, we can figure out how it would bounce off the object.



Reflections and Collisions

A ball hits the ground and bounces. What happens to its velocity?

(Assuming it bounces perfectly with no loss of speed.)

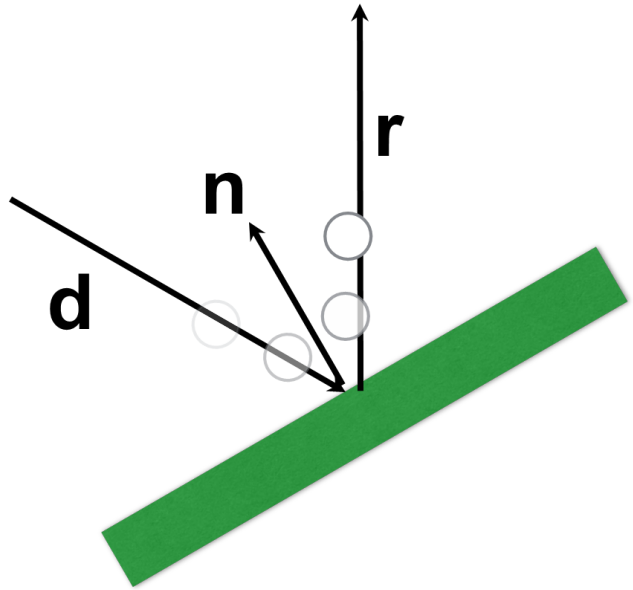


Given **d**, find **r**.

Reflections and Collisions

A ball hits an *inclined plane* and bounces. What happens to its velocity?

(Assuming it bounces perfectly with no loss of speed.)

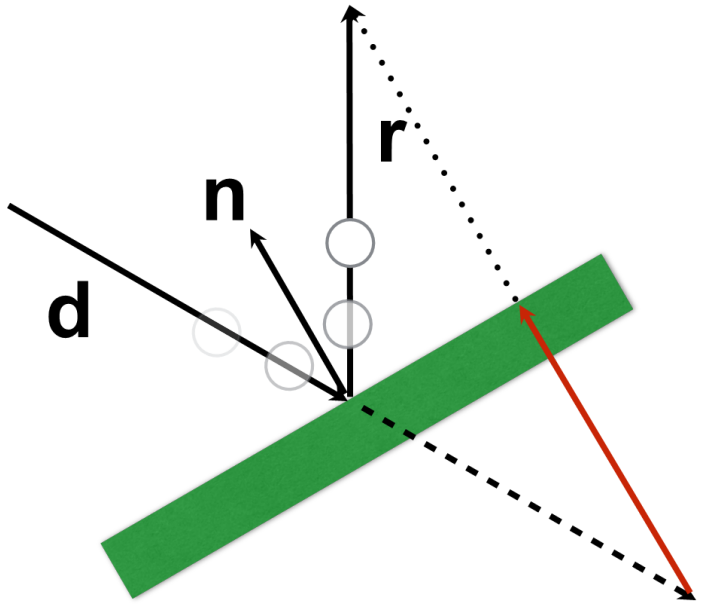


Given **d** and **n**, find **r**.

Reflections and Collisions

A ball hits an *inclined plane* and bounces. What happens to its velocity?

(Assuming it bounces perfectly with no loss of speed.)



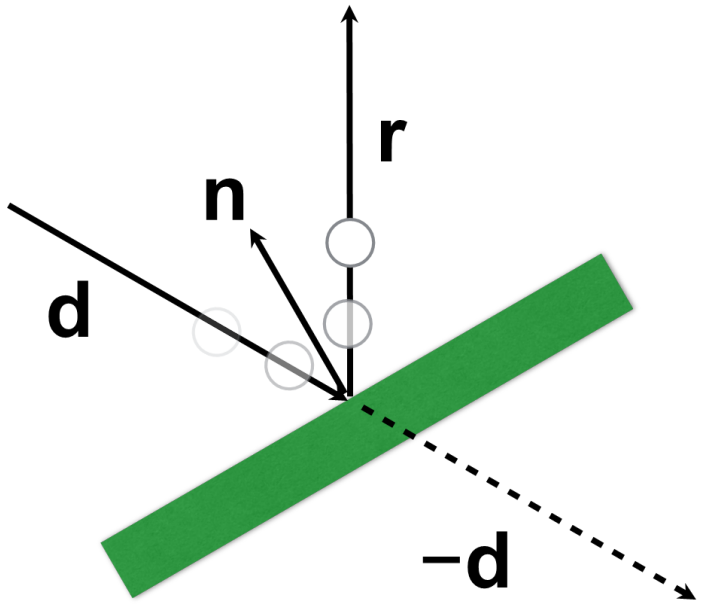
Hint: What is the red vector?

Given **d** and **n**, find **r**.

Reflections and Collisions

A ball hits an *inclined plane* and bounces. What happens to its velocity?

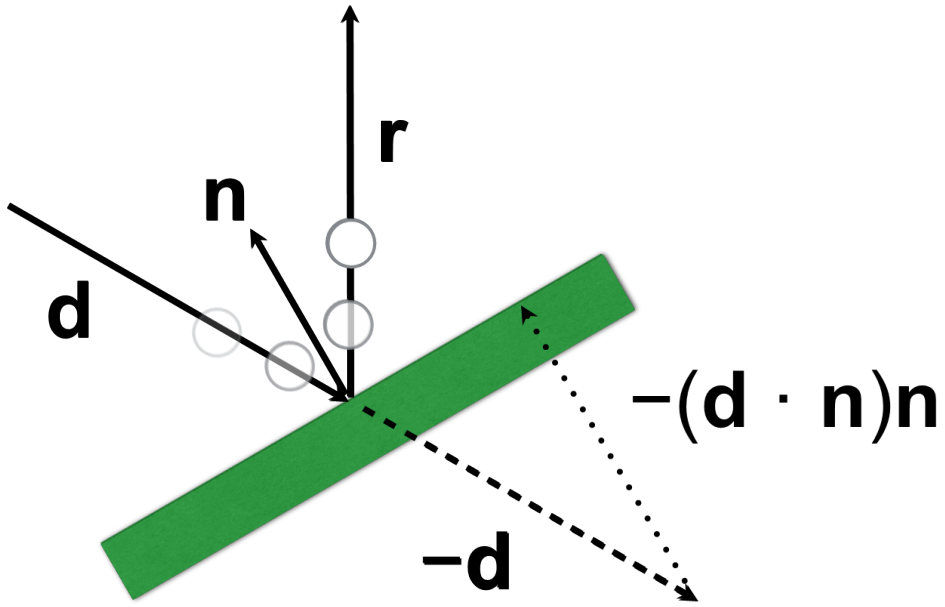
(Assuming it bounces perfectly with no loss of speed.)



Reflections and Collisions

A ball hits an *inclined plane* and bounces. What happens to its velocity?

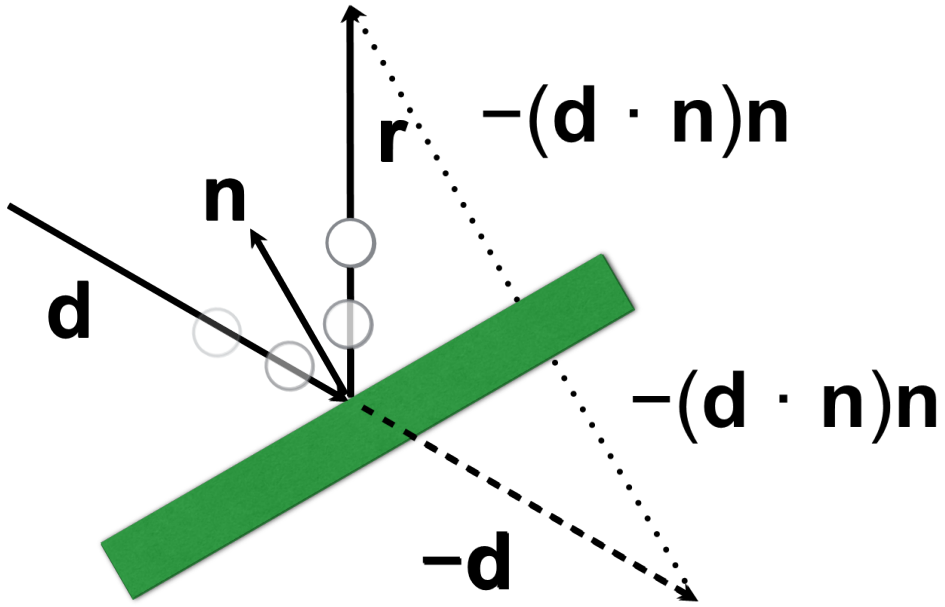
(Assuming it bounces perfectly with no loss of speed.)



Reflections and Collisions

A ball hits an *inclined plane* and bounces. What happens to its velocity?

(Assuming it bounces perfectly with no loss of speed.)



$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$$