Interactive Computer Graphics

CS 438 002

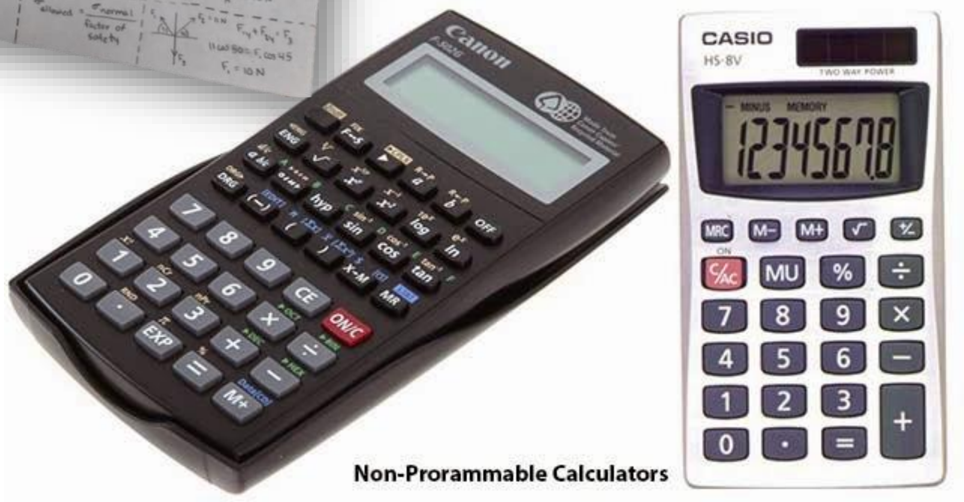
Dr. Przemyslaw Musialski

Department of Computer Science

New Jersey Institute of Technology

Midterm Exam

- Date: 3/12/2020 at 11:30
- Timeframe: 60 minutes
- Allowed material:
  - 1 letter size cheat sheet
    - **MUST BE self-handwritten !!!**
    - No printed, copied, photographed, etc. sheets are allowed !
    - Can be filled on both sides
  - Non-programable calculator
    - aka “old-style”
    - No cellphones, touchpads, notebooks, etc.
  - Writing utensils, ruler, triangle, etc.



Midterm Exam — Material

- Rendering Pipeline
  - Stages
  - Variable Types: Uniforms, Varying, Attributes, Textures
  - Vertex Buffers
- Shading and Lighting
  - Shading Types
    - Flat, Gouraud, Phong
  - Local Illumination
    - Phong, Blinn-Phong
- Linear Algebra and Geometry
  - Points, Vectors, Matrices
  - Dot Product, Cross Product
  - Lines, Planes
  - Coordinate Systems
  - Affine Transformations
  - Interpolation and Barycentric Coordinates
  - Projections
    - Orthographic
    - Perspective
- Viewing
  - Look at

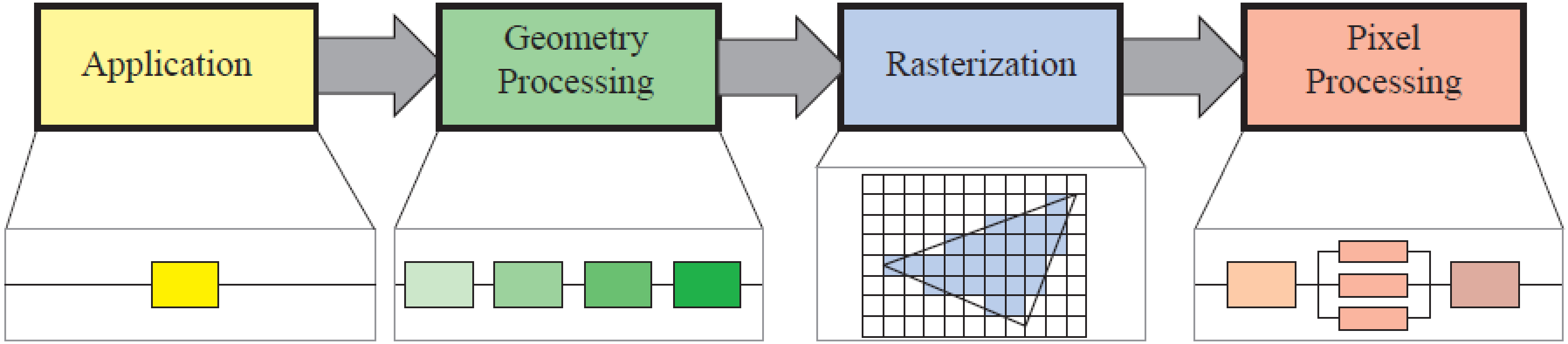
Reading

- **Interactive Computer Graphics: A Top-Down Approach with WebGL , 7th Edition, Publisher: Pearson; ISBN: 978-0133574845**
  by Edward Angel (Author), Dave Shreiner (Author)
  - Chapter 4,
  - Chapter 5,
  - Chapter 6
- **Fundamentals of Computer Graphics, 4th Edition, Publisher: A K Peters/CRC Press; ISBN: 978-1482229394**
  by Steve Marschner (Author), Peter Shirley (Author)
  - Chapter 6,
  - Chapter 7
  - Chapter 17!

Graphics Pipeline

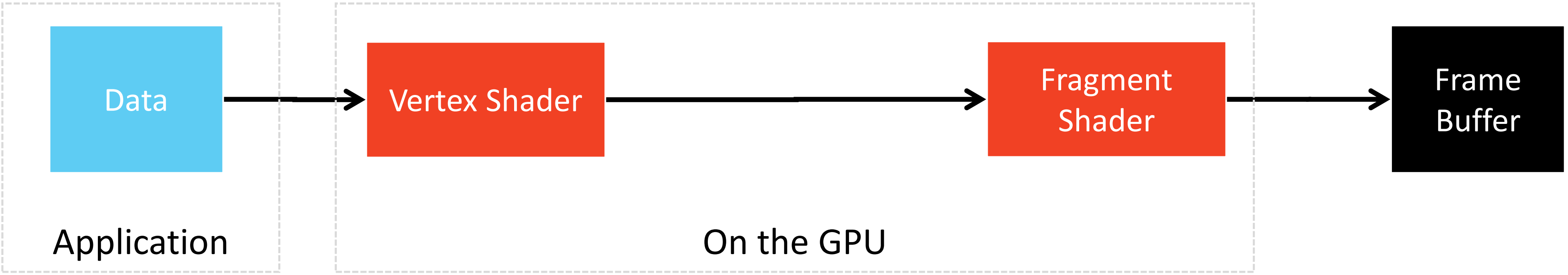
Rendering Pipeline Overview

- The high-level view of the rendering pipeline:



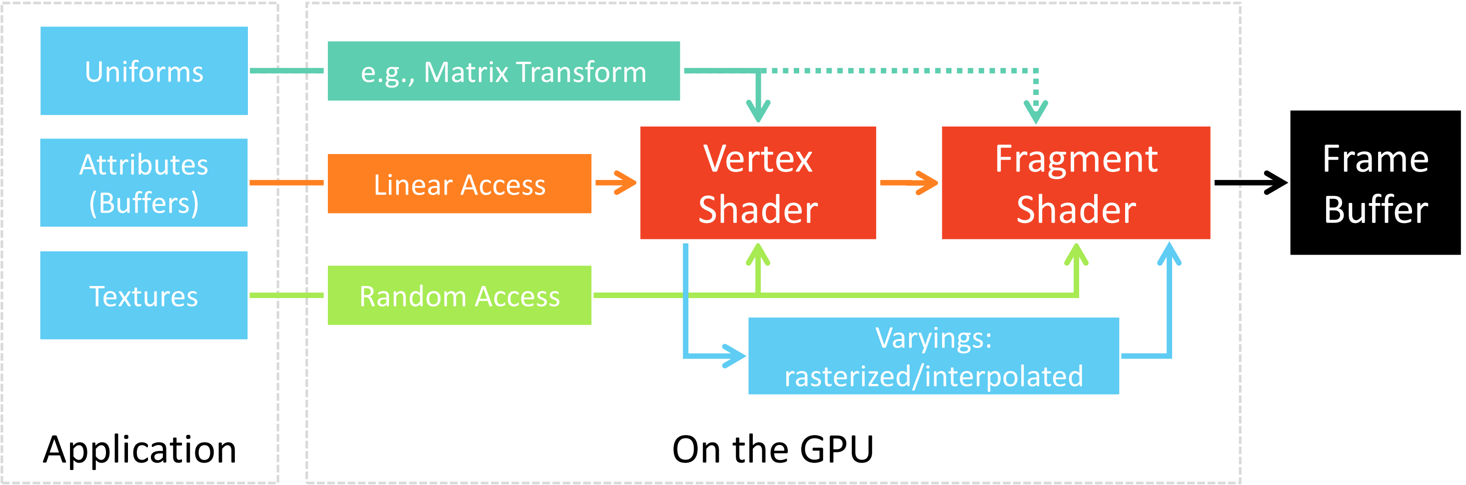
- Each stage can be subdivided into sub-stages
- Some sub-stages can be run in parallel
- Some stages are fixed
- Some are configurable
- And some are programmable

(Simplified) Pipeline



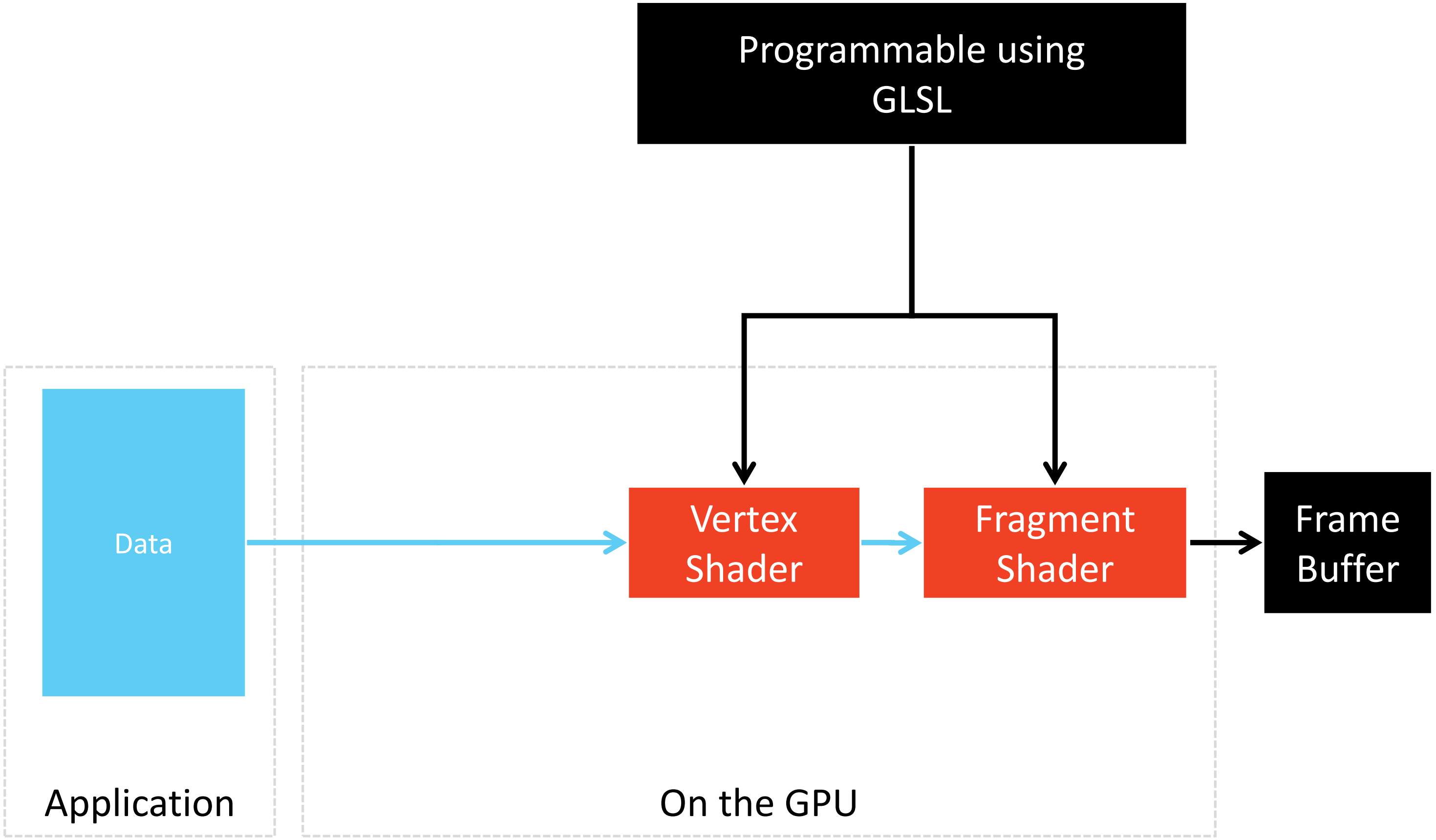
Ways to Pass Data to Shaders

- Attributes and Buffers
- Textures
- Uniforms
- Varyings



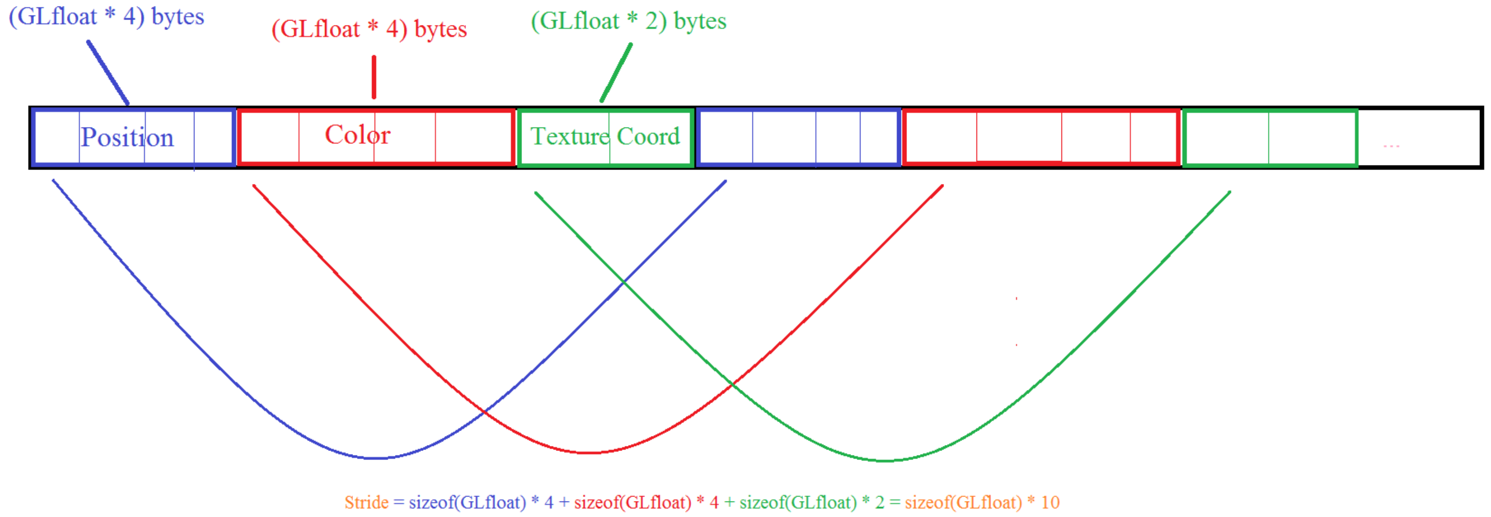
Shaders

- Vertex Shader
- Fragment Shader



Vertex Buffer

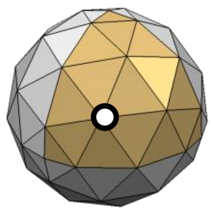
- What is an interleaved vertex buffer?
- What is the offset?
- What is the stride?



Meshes

Triangle Meshes

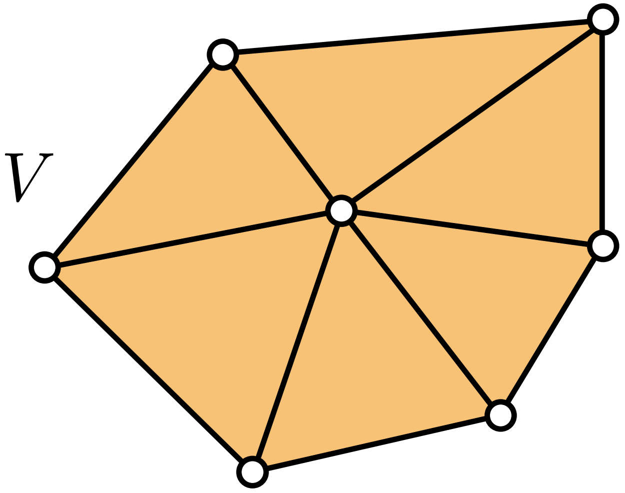
- Connectivity: vertices, edges, triangles
- Geometry: vertex positions



$$V = \{v_1, \ldots, v_n\}$$

$$E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V$$

$$F = \{f_1, \ldots, f_m\}, \quad f_i \in V \times V \times V$$



$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}, \quad \mathbf{p}_i \in \mathbb{R}^3$$

Triangle List

- STL format (used in CAD)
- Storage
  - Face: 3 positions
  - 4 bytes per coordinate
  - 36 bytes per face
    - Euler: $f = 2v$
    - $72*v$ bytes for a mesh with v vertices
- No connectivity information
- This is a “triangle soup”

Triangles			
0	x0	y0	z0
1	x1	x1	z1
2	x2	y2	z2
3	x3	y3	z3
4	x4	y4	z4
5	x5	y5	z5
6	x6	y6	z6
…	…	…	…

Indexed Face Set

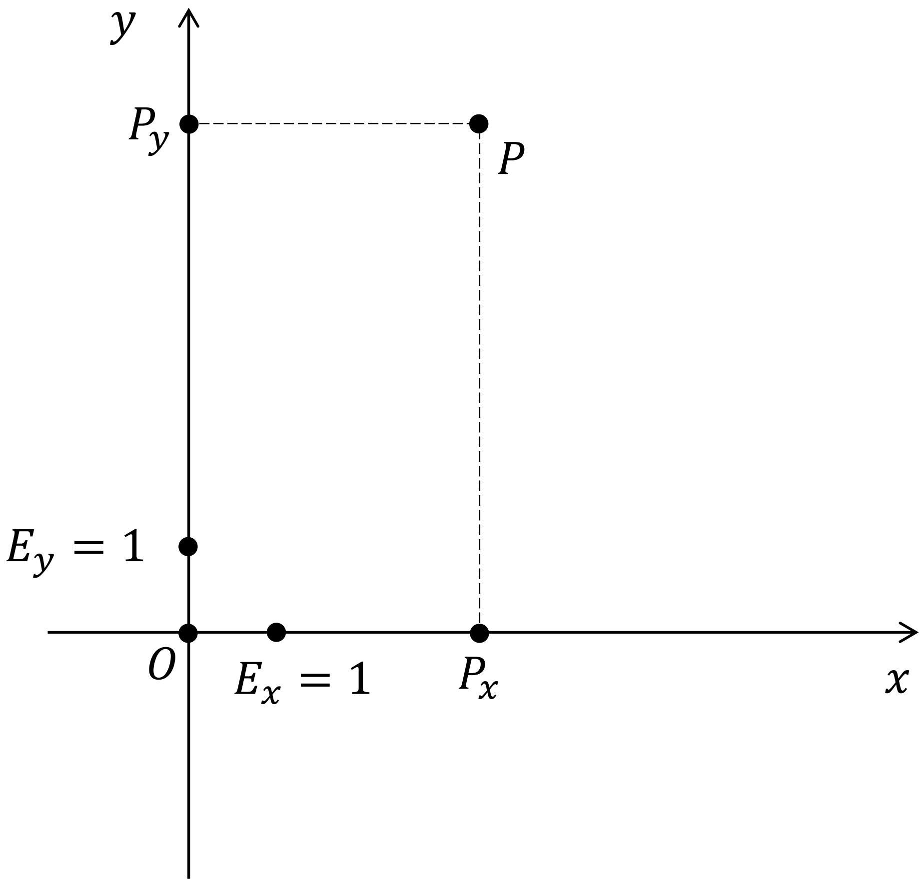
- Used in formats
  - OBJ, OFF, VRML
- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - 36*v bytes for the mesh (~half of triangle list)
- No *explicit* neighborhood info
- Well suitable for rendering!

Vertices			
v0	x0	y0	z0
v1	x1	x1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
…	…	…	…

Triangles			
t0	v0	v1	v2
t1	v0	v1	v3
t2	v2	v4	v3
t3	v5	v2	v6
…	…	…	…

Points, Vectors, Coordinate Systems

Points and Coordinate Systems

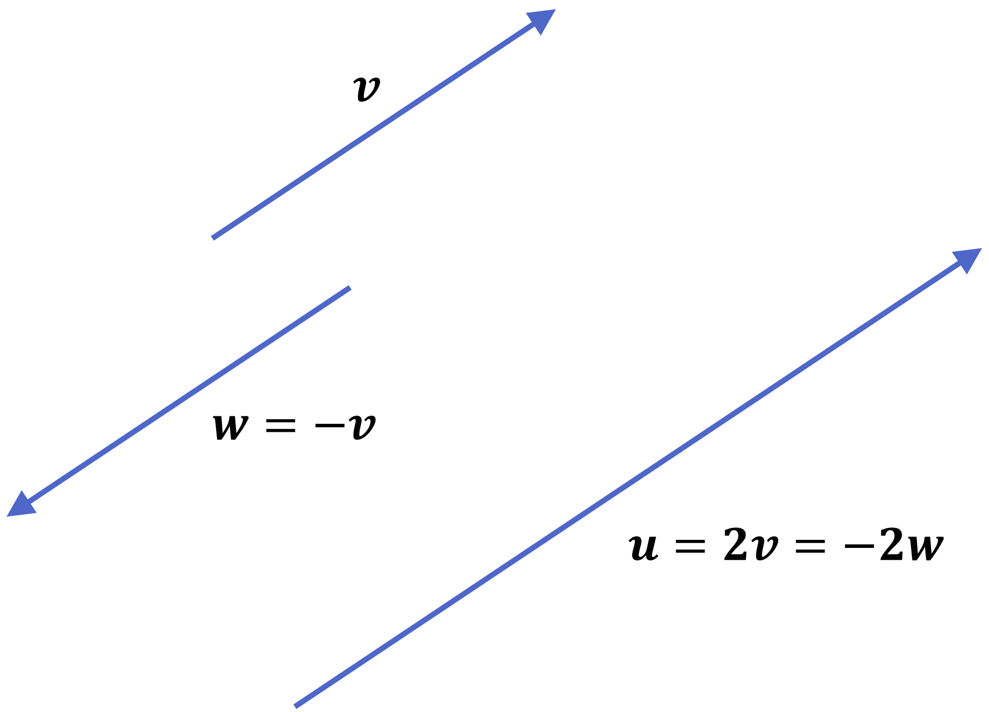


- We can quantify the position with respect to the origin O, the axes x, y, and reference points on each axis

$$P = \begin{bmatrix} P_x = 3E_x \\ P_y = 5E_y \end{bmatrix}$$

- Cartesian Coordinate System
  - Axes are mutually orthogonal
  - The reference points E have the same distance to the origin O

Vectors



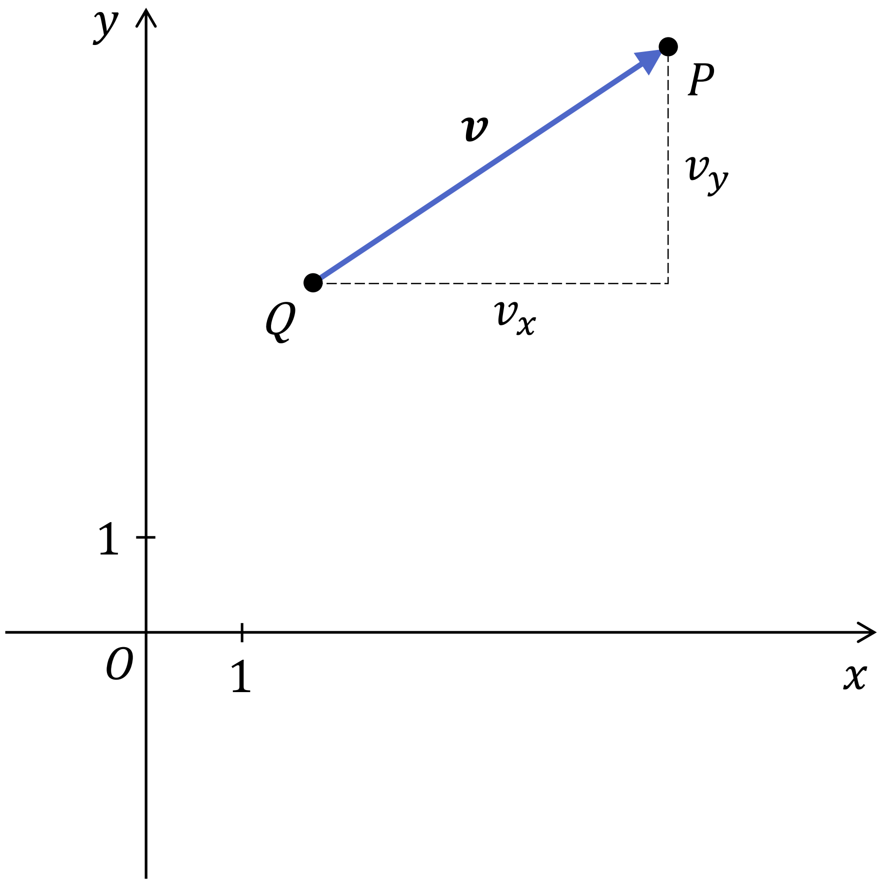
- Vectors multiplied by a scalar
  - Direction $\boldsymbol{v}$
  - Negated vector:

$$\boldsymbol{w} = -1\boldsymbol{v} = -\boldsymbol{v}$$

  - Vector scaled by a scalar $\lambda \in \mathbb{R}$

$$\boldsymbol{u} = \lambda\boldsymbol{v} = -\lambda\boldsymbol{w}$$

Vectors



- Vector coordinates

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} P_x - Q_x \\ P_y - Q_y \end{bmatrix}$$

- Vector length

$$\|\boldsymbol{v}\| = \sqrt{v_x^2 + v_y^2}$$

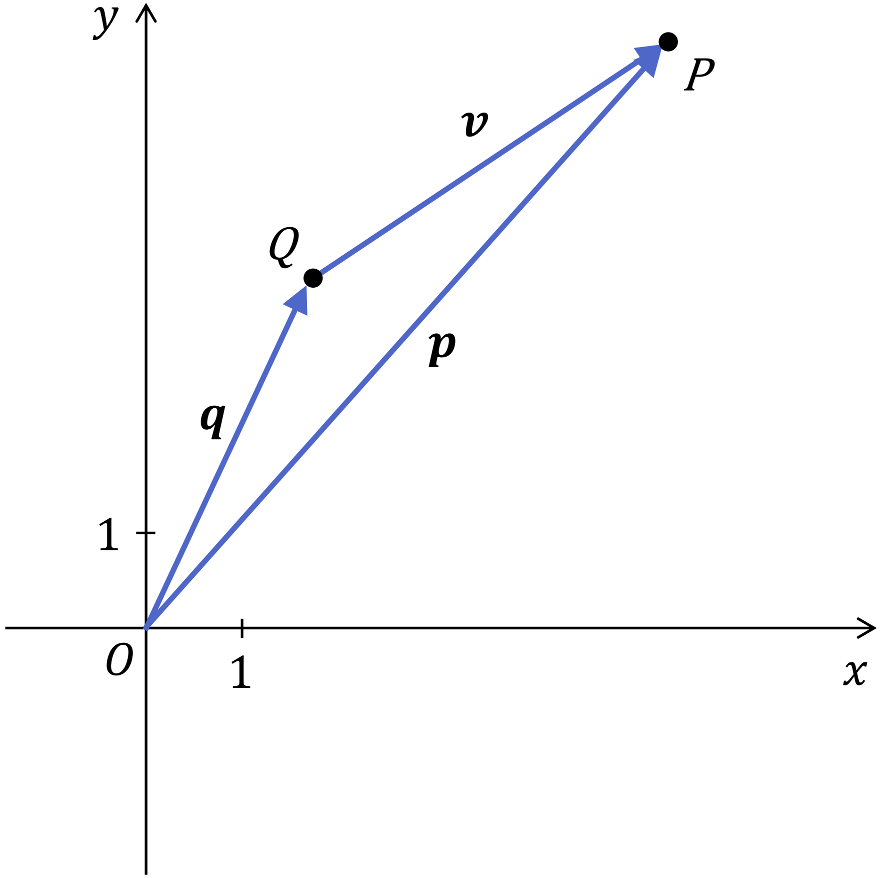
- Unit vector

$$\boldsymbol{e} = \frac{1}{\|\boldsymbol{v}\|}\boldsymbol{v}$$

- Zero vector

$$\boldsymbol{o} \text{ with } \|\boldsymbol{o}\| = 0$$

Vectors vs Points



- Point coordinates give a vector with the origin

$$\boldsymbol{p} = P - O = \begin{bmatrix} P_x - 0 \\ P_y - 0 \end{bmatrix}$$

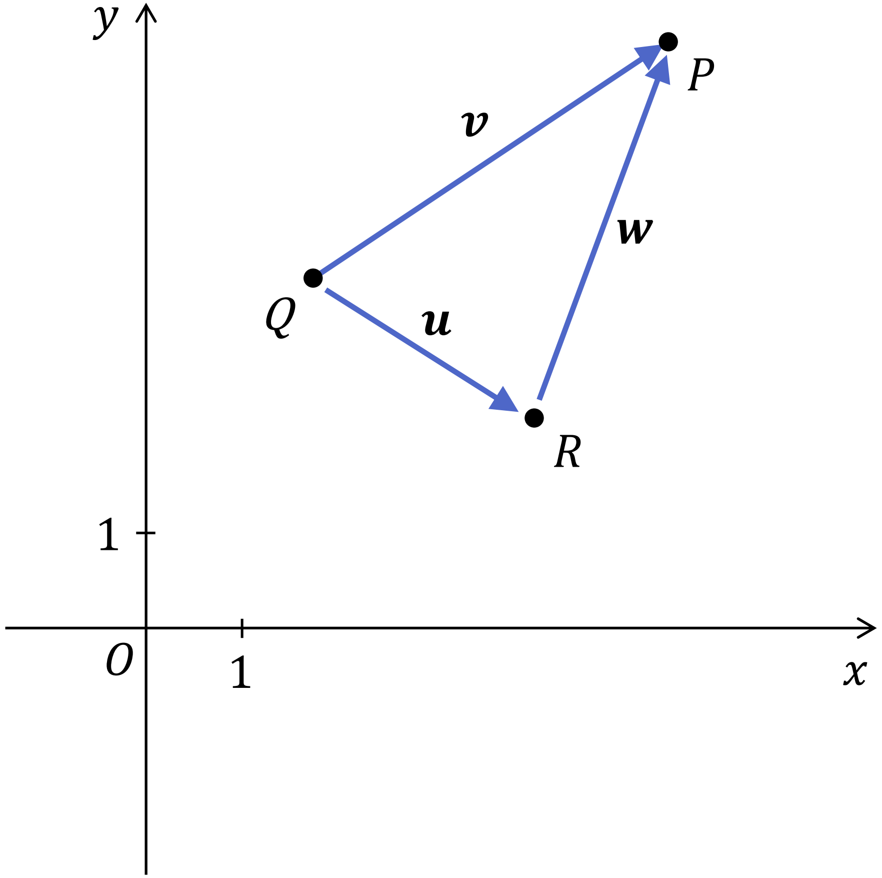
called **position vector**

- Vector

$$\boldsymbol{v} = P - Q = \begin{bmatrix} P_x - Q_x \\ P_y - Q_y \end{bmatrix}$$

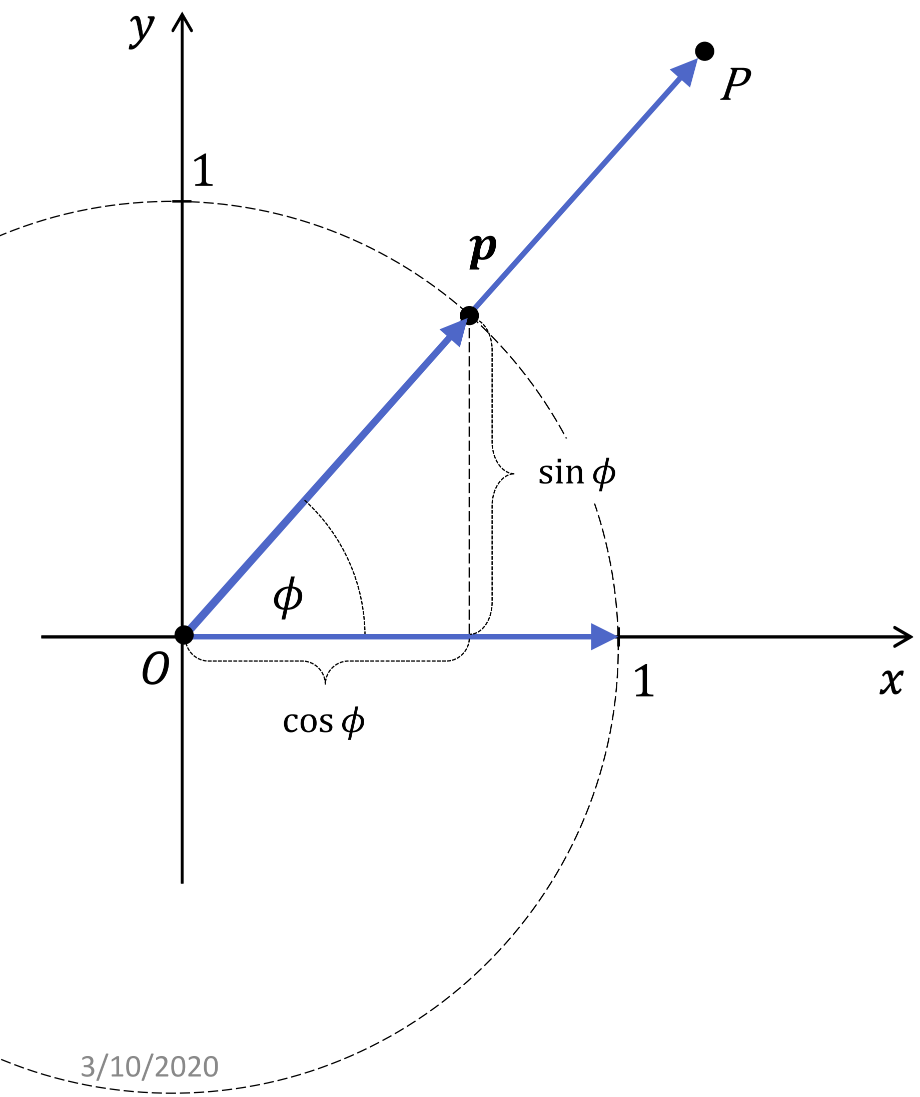
is a **free vector**

Vector Operations Summary



- Vector addition forms an Abelian group V
  1. Associativity
     $$\boldsymbol{v} + (\boldsymbol{u} + \boldsymbol{w}) = (\boldsymbol{v} + \boldsymbol{u}) + \boldsymbol{w}$$
  2. Commutativity
     $$\boldsymbol{v} + \boldsymbol{u} = \boldsymbol{u} + \boldsymbol{v}$$
  3. Identity element (neutral element)
     $$\boldsymbol{v} + \boldsymbol{o} = \boldsymbol{o} + \boldsymbol{v} = \boldsymbol{v}$$
  4. Inverse element
     $$\boldsymbol{v} + (-\boldsymbol{v}) = \boldsymbol{o}$$
  5. Closure
     $$\boldsymbol{v} + \boldsymbol{u} = \boldsymbol{w} \text{ with } \boldsymbol{w} \in V$$

Polar Coordinates



- We can express a position vector using polar coordinates:

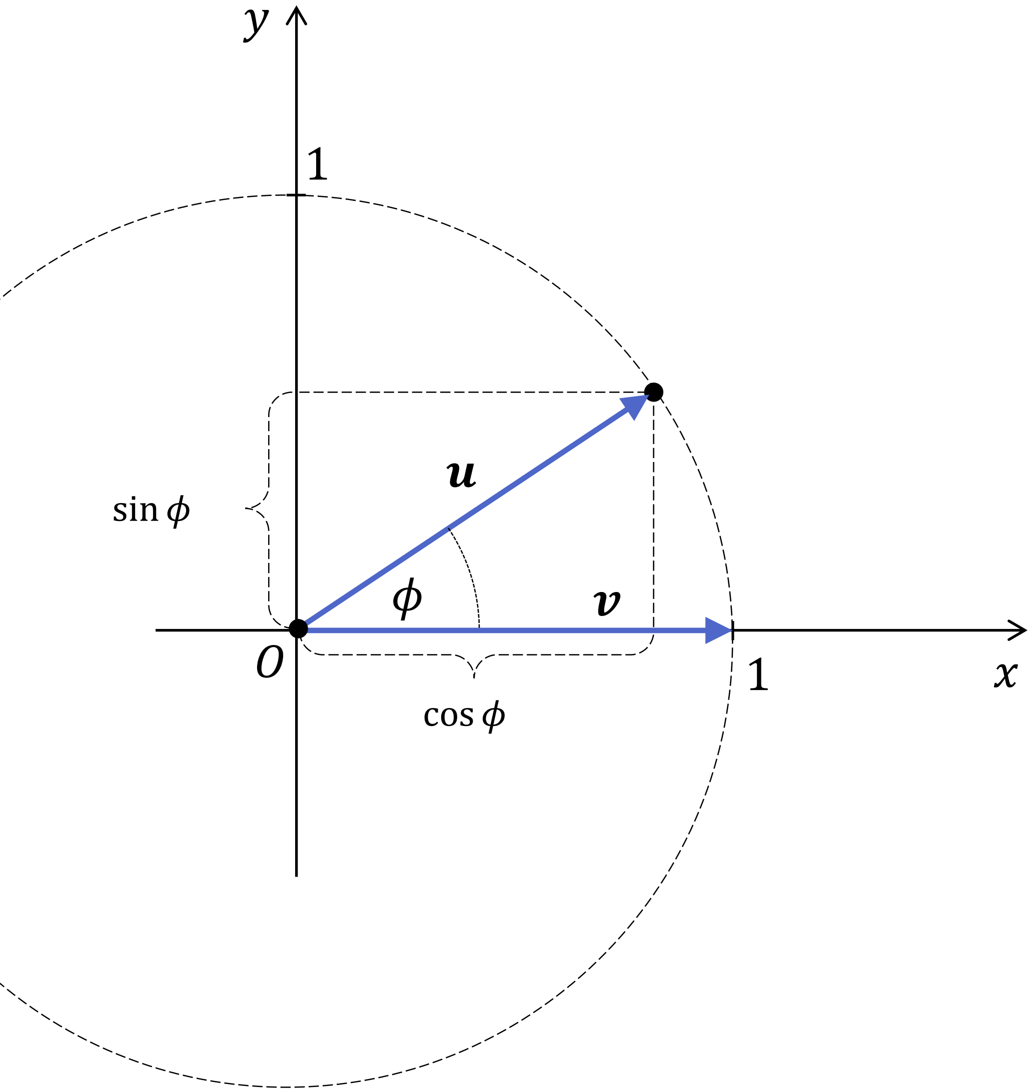
$$\boldsymbol{p} = \begin{bmatrix} \phi \\ r \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(y, x) \\ \sqrt{x^2 + y^2} \end{bmatrix}$$

- with $r = \|\boldsymbol{p}\|$

- or vice versa

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix}$$

Dot Product



- Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, the dot product is defined as

$$\boldsymbol{u} \cdot \boldsymbol{v} = u_x v_x + u_y v_y$$

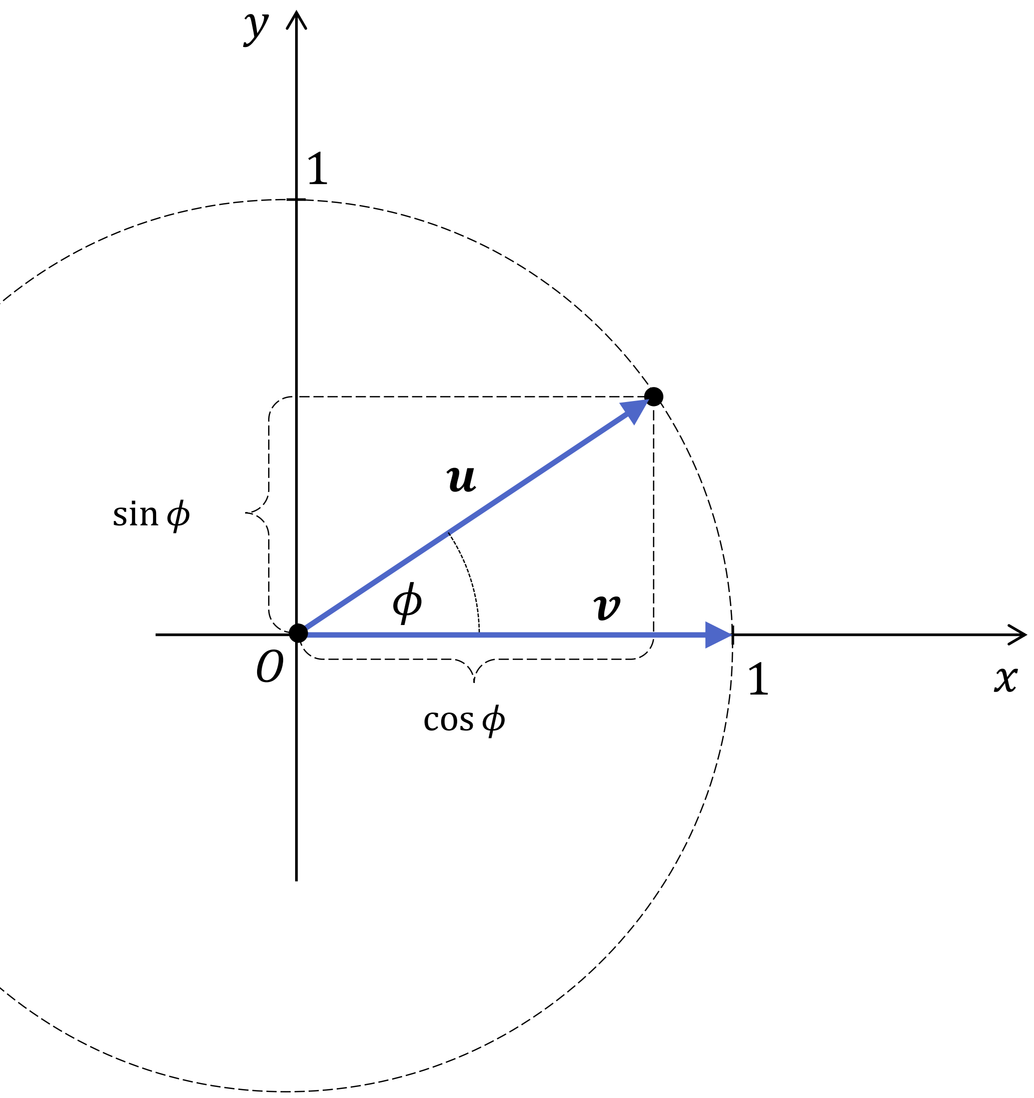
  - The dot product is the projection of one vector on the another
  - It gives also the cosine of the angle between them

$$\cos\phi = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \|\boldsymbol{v}\|}$$

  - If the vectors are unit length, it applies

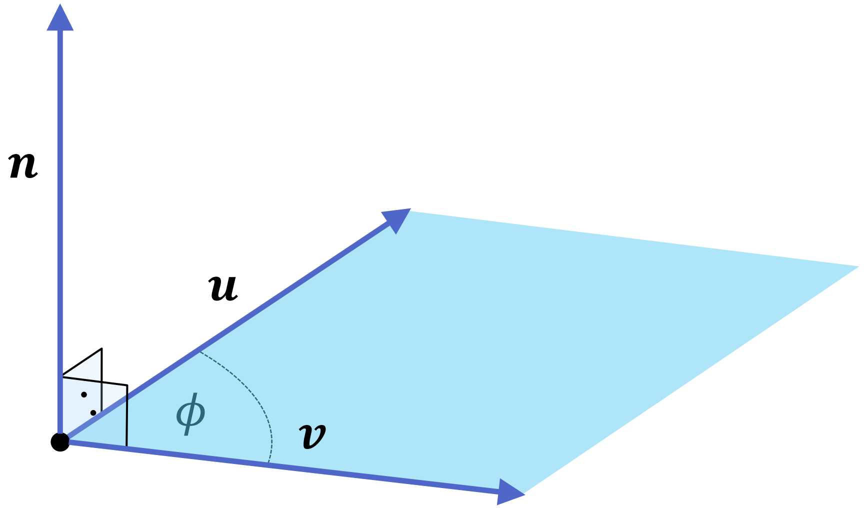
$$\cos\phi = \boldsymbol{u} \cdot \boldsymbol{v}$$

Dot Product



- Properties of the dot product
  - Denoted also as
    $$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v}$$
  - Length of
    $$\|\boldsymbol{v}\| = \sqrt{\boldsymbol{v}^T \boldsymbol{v}}$$
- Rules
  - Commutative
    $$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{v} \cdot \boldsymbol{u}$$
  - Distributive
    $$(\boldsymbol{u} + \boldsymbol{v}) \cdot \boldsymbol{w} = (\boldsymbol{u} \cdot \boldsymbol{w}) + (\boldsymbol{v} \cdot \boldsymbol{w})$$
  - NOT associative
    $$(\boldsymbol{u} \cdot \boldsymbol{v}) \cdot \boldsymbol{w} \neq \boldsymbol{u} \cdot (\boldsymbol{v} \cdot \boldsymbol{w})$$

Cross Product



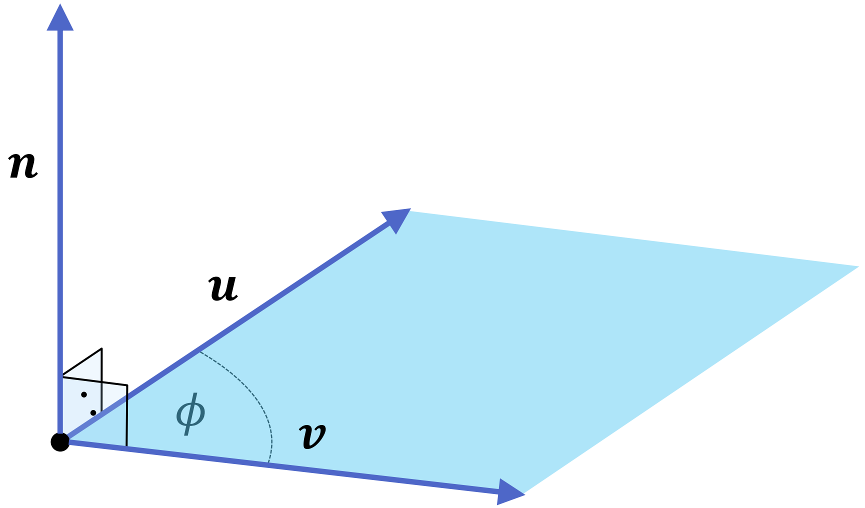
- Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in 3D
  - The cross product (vector product) is defined as

$$\boldsymbol{n} = \boldsymbol{u} \times \boldsymbol{v} = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{bmatrix}$$

  - The cross product delivers a vector perpendicular to $\boldsymbol{u}$ and $\boldsymbol{v}$
  - The length $\|\boldsymbol{n}\| = Area$ of the parallelogram given by $\boldsymbol{u}$ and $\boldsymbol{v}$
  - The angle between $\boldsymbol{u}$ and $\boldsymbol{v}$ is given by

$$\sin\phi = \frac{\|\boldsymbol{u} \times \boldsymbol{v}\|}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|}$$

Cross Product



- Properties of the cross product
  - Alternating

$$\boldsymbol{u} \times \boldsymbol{v} = -(\boldsymbol{v} \times \boldsymbol{u})$$

  - Distributive

$$(\boldsymbol{u} + \boldsymbol{v}) \times \boldsymbol{w} = (\boldsymbol{u} \times \boldsymbol{w}) + (\boldsymbol{v} \times \boldsymbol{w})$$

  - Scalar Multiplication

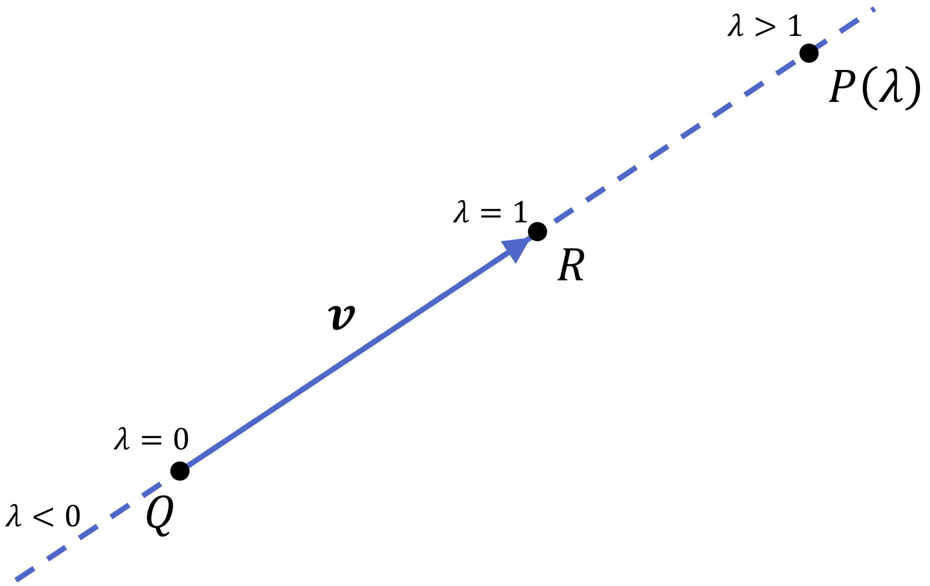
$$\lambda(\boldsymbol{v} \times \boldsymbol{u}) = (\lambda \boldsymbol{v}) \times \boldsymbol{u}$$

  - NOT associative

$$(\boldsymbol{u} \times \boldsymbol{v}) \times \boldsymbol{w} \neq \boldsymbol{u} \times (\boldsymbol{v} \times \boldsymbol{w})$$

Lines and Planes

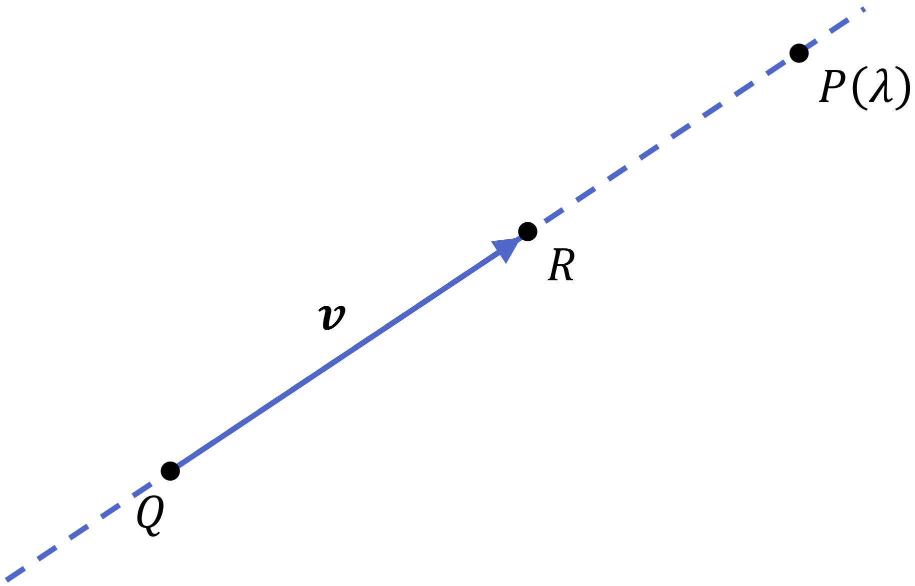
Lines: Parametric Form in 2D and 3D



- A line in 2D and 3D is given in parametric form as

$$P(\lambda) = Q + \lambda \boldsymbol{v}$$

Affine Combination



- We can form an affine combination

$$P(\lambda) = Q + \lambda \boldsymbol{v}$$

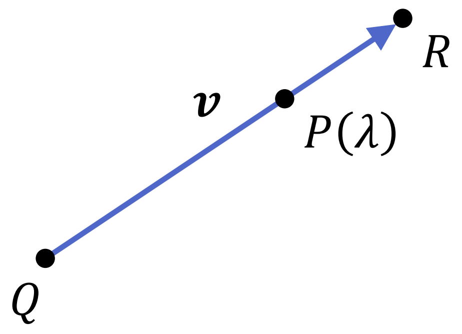
- Using

$$\boldsymbol{v} = R - Q$$

$$\begin{aligned} P(\lambda) &= Q + \lambda(R - Q) \\ &= Q + \lambda R - \lambda Q \\ &= \lambda R + (1 - \lambda) Q \\ &= \lambda_1 R + \lambda_2 Q \end{aligned}$$

with $\lambda_1 + \lambda_2 = 1$

Convex Combination



- We can form an affine combination

$$P(\lambda) = Q + \lambda\boldsymbol{v}$$

- Using

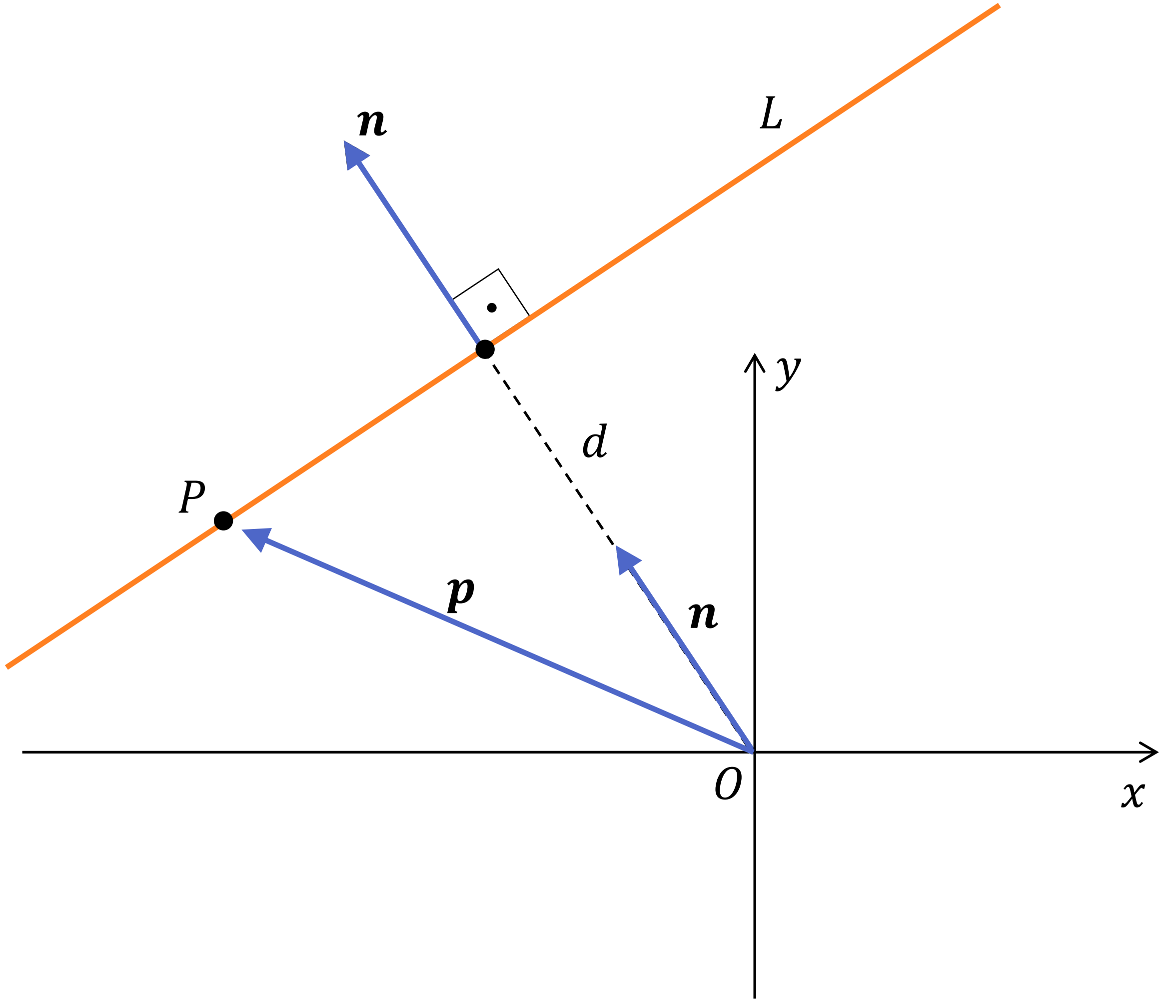
$$\boldsymbol{v} = R - Q$$

$$\begin{aligned} P(\lambda) &= Q + \lambda(R - Q) \\ &= Q + \lambda R - \lambda Q \\ &= \lambda R + (1 - \lambda)Q \\ &= \lambda_1 R + \lambda_2 Q \end{aligned}$$

with $\lambda_1 + \lambda_2 = 1$

and $\lambda_1, \lambda_2 \geq 0$

Lines: Normal Form in 2D

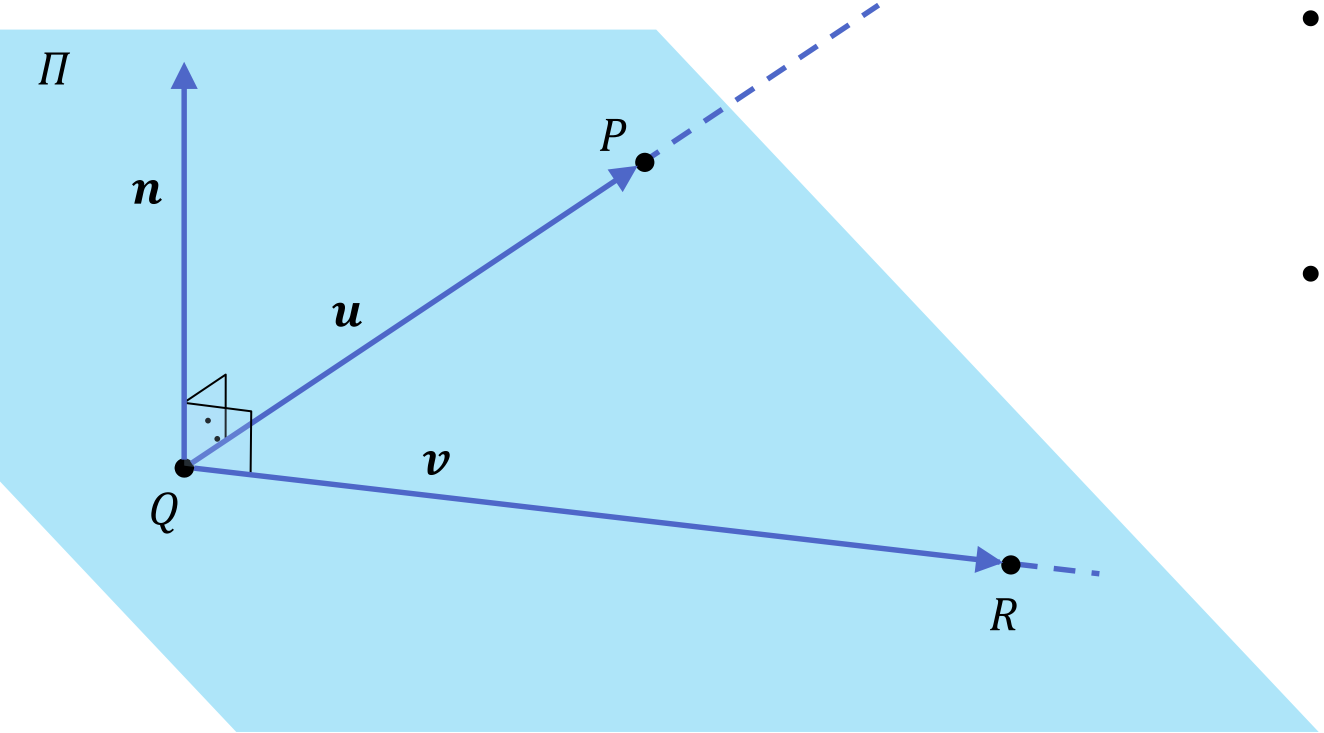


- In 2D we can express the line in its normal form

$$L\colon\ \left(\boldsymbol{p}^T\boldsymbol{n}\right) - d = 0$$

- If $||\boldsymbol{n}|| = 1$, it is denoted as Hesse Normal Form, and d gives the signed distance to the origin

Planes



- A plane in 3D is given in parametric form as

$$\Pi(\lambda, \mu) = Q + \lambda \boldsymbol{u} + \mu \boldsymbol{v}$$

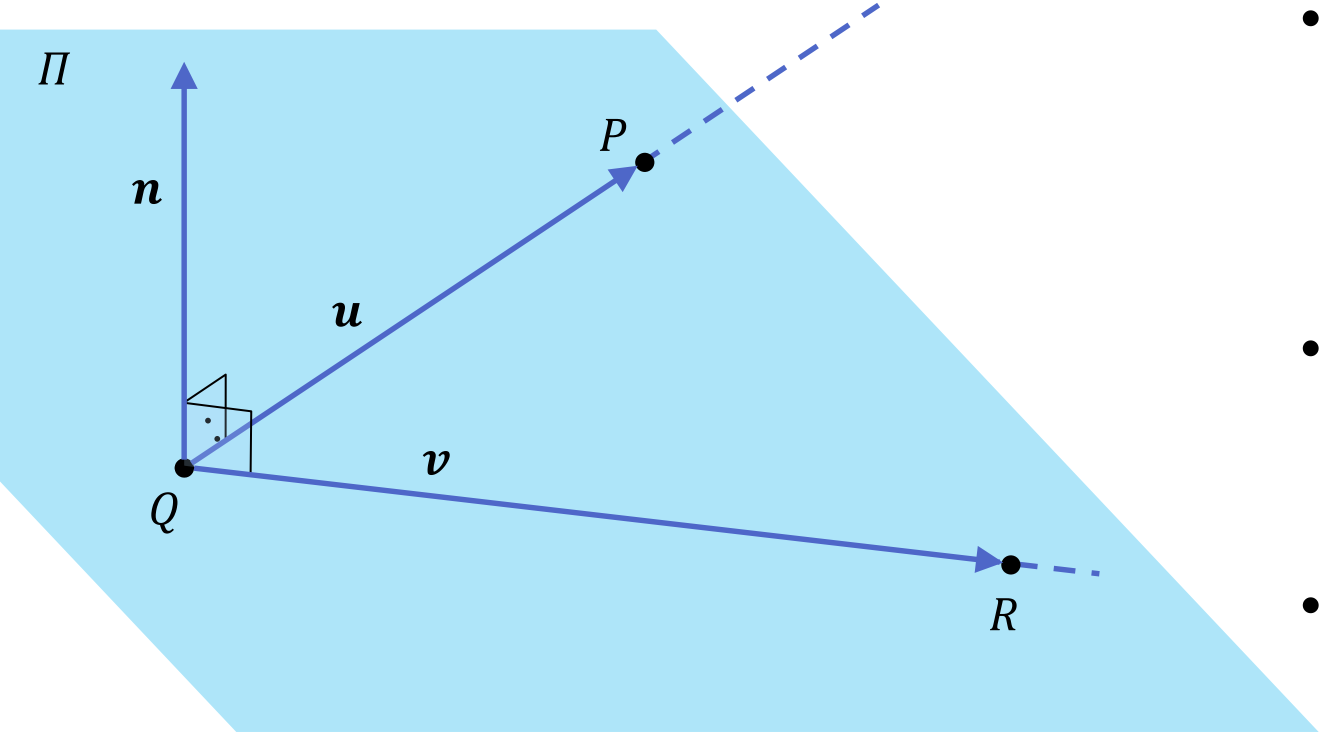
- or

$$\begin{aligned}\Pi(\lambda, \mu) &= Q + \lambda \boldsymbol{u} + \mu \boldsymbol{v} \\ &= Q + \lambda(P - Q) + \mu(R - Q) \\ &= (1 - \lambda - \mu)Q + \lambda P + \mu R\end{aligned}$$

- Normal vector is given by

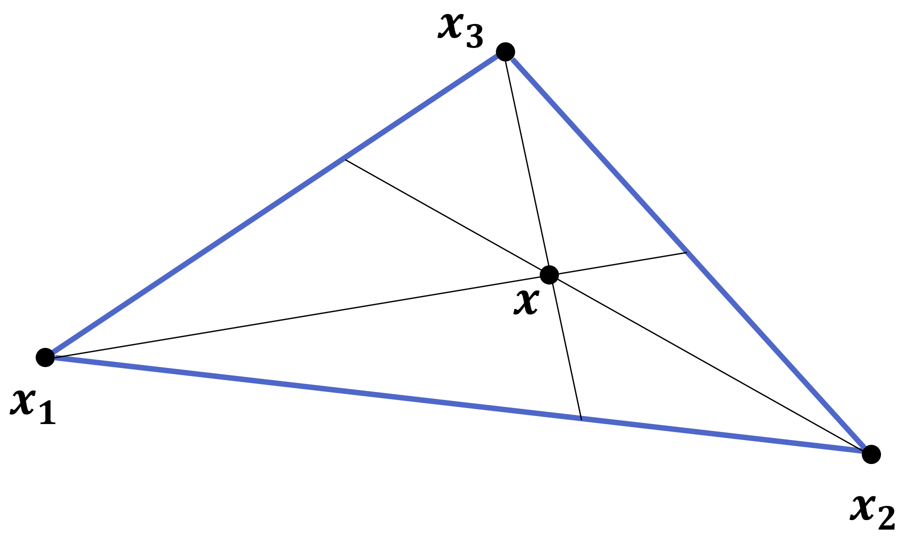
$$\boldsymbol{n} = \boldsymbol{u} \times \boldsymbol{v} = (P - Q) \times (R - Q)$$

Planes



- The **normal form** of the plane can be obtained by solving
$$n_x p_x + n_y p_y + n_z p_z + n_0 = 0$$
- With dot-product, we obtain:
$$\Pi\colon\ \boldsymbol{n}^{\boldsymbol{T}}\boldsymbol{p} + n_0 = 0$$
- If $||\boldsymbol{n}|| = 1$ this form is called Hesse-Normal Form (HNF) of the plane.
- n_0 gives the signed distance of the plane to the origin

Barycentric Coordinates



- A point on a plane can be expressed using barycentric coordinates:

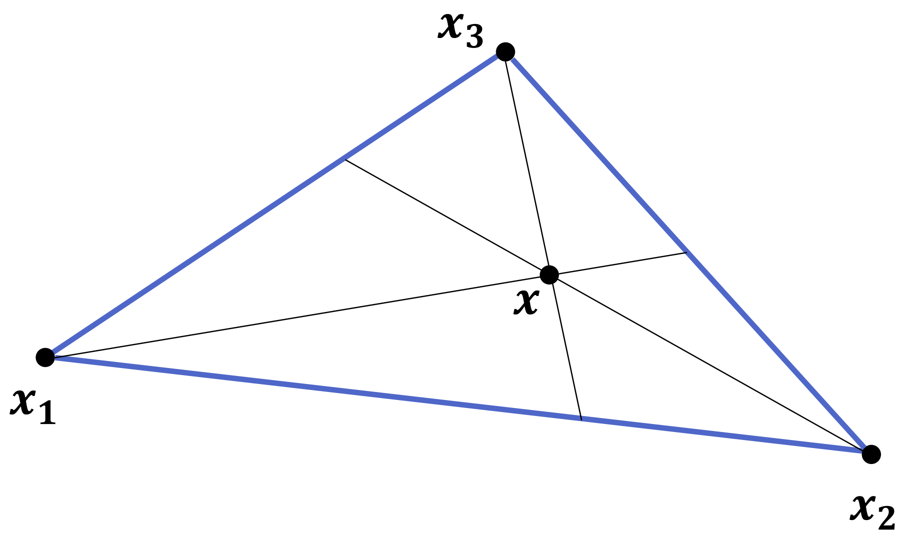
$$\boldsymbol{x}(\lambda_1, \lambda_2, \lambda_3) = \lambda_1 \boldsymbol{x_1} + \lambda_2 \boldsymbol{x_2} + \lambda_3 \boldsymbol{x_3}$$

- with $\lambda_1 + \lambda_2 + \lambda_2 = 1$ we obtain

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

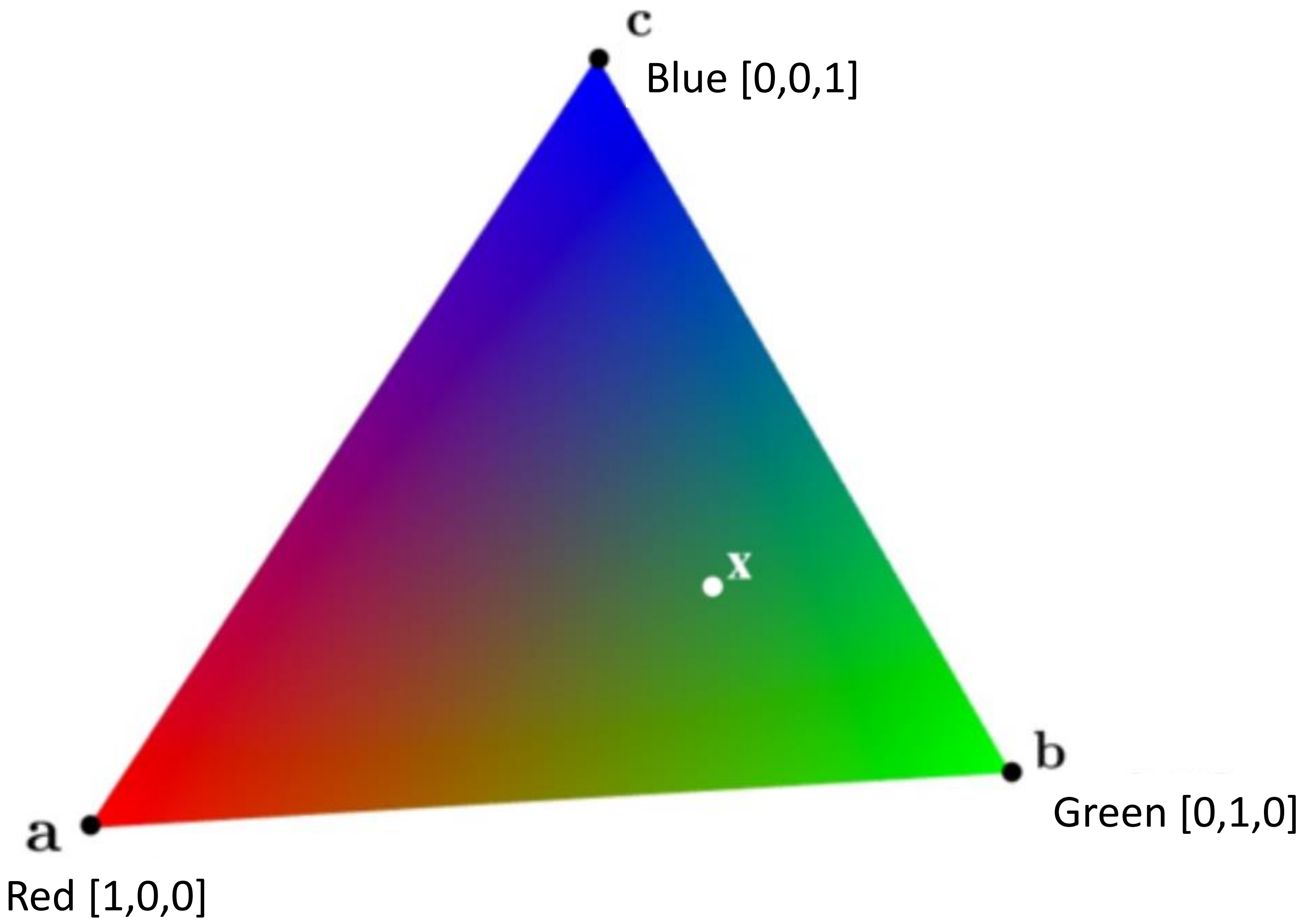
- This leaves us 2 unknowns: λ_1, λ_2

Barycentric Coordinates



- We can obtain the barycentric coordinates as ratios of the areas of the triangles:
  - $\lambda_1 = \frac{A(\boldsymbol{x}, \boldsymbol{x}_2, \boldsymbol{x}_3)}{A(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)}$
  - $\lambda_2 = \frac{A(\boldsymbol{x}, \boldsymbol{x}_3, \boldsymbol{x}_1)}{A(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)}$
  - $\lambda_3 = \frac{A(\boldsymbol{x}, \boldsymbol{x}_1, \boldsymbol{x}_2)}{A(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)}$
- That's why also denoted as *areal coordinates*
- We can also obtain them by solving a system of linear equations (next lecture…)

Interpolation



- What is the color at x?

Matrices

Matrix

- $\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

- $\boldsymbol{A} = \left[a_{ij}\right]$

- $\lambda\boldsymbol{A} = \left[\lambda a_{ij}\right]$

- $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B} = \left[a_{ij} + b_{ij}\right]$

- $\boldsymbol{A}^{\boldsymbol{T}} = \left[a_{ji}\right]$

Matrix Operations

- Matrix Operations
    - $\alpha(\beta\boldsymbol{A}) = (\alpha\beta)\boldsymbol{A}$
    - $\alpha\beta\boldsymbol{A} = \beta\alpha\boldsymbol{A}$
    - $\boldsymbol{A} + \boldsymbol{B} = \boldsymbol{B} + \boldsymbol{A}$
    - $\boldsymbol{A} + (\boldsymbol{B} + \boldsymbol{C}) = (\boldsymbol{A} + \boldsymbol{B}) + \boldsymbol{C}$
    - $\boldsymbol{A}(\boldsymbol{B}\boldsymbol{C}) = (\boldsymbol{A}\boldsymbol{B})\boldsymbol{C}$
- Not Commutative!
    - $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$

Identity Matrix

- The identity matrix $\boldsymbol{I_n}$ is a $n \times n$ square matrix with the diagonal of 1's and all other elements are 0.
  - $\boldsymbol{I_2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\quad \boldsymbol{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\quad \boldsymbol{I_4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- If $\boldsymbol{A}$ is a $n \times n$ matrix, then
  - $\boldsymbol{AI_n} = \boldsymbol{A}$
  - $\boldsymbol{I_nB} =$**B**

- If $\boldsymbol{A}$ is a $m \times n$ matrix, then
  - $\boldsymbol{AI_n} = \boldsymbol{A}$
  - $\boldsymbol{I_mB} =$**B**

Matrix Multiplication

- Dot product of each row with each column
  - $a_1 b_1 + a_2 b_4 + a_3 b_7 = c_1$

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

Matrix Vector Multiplication

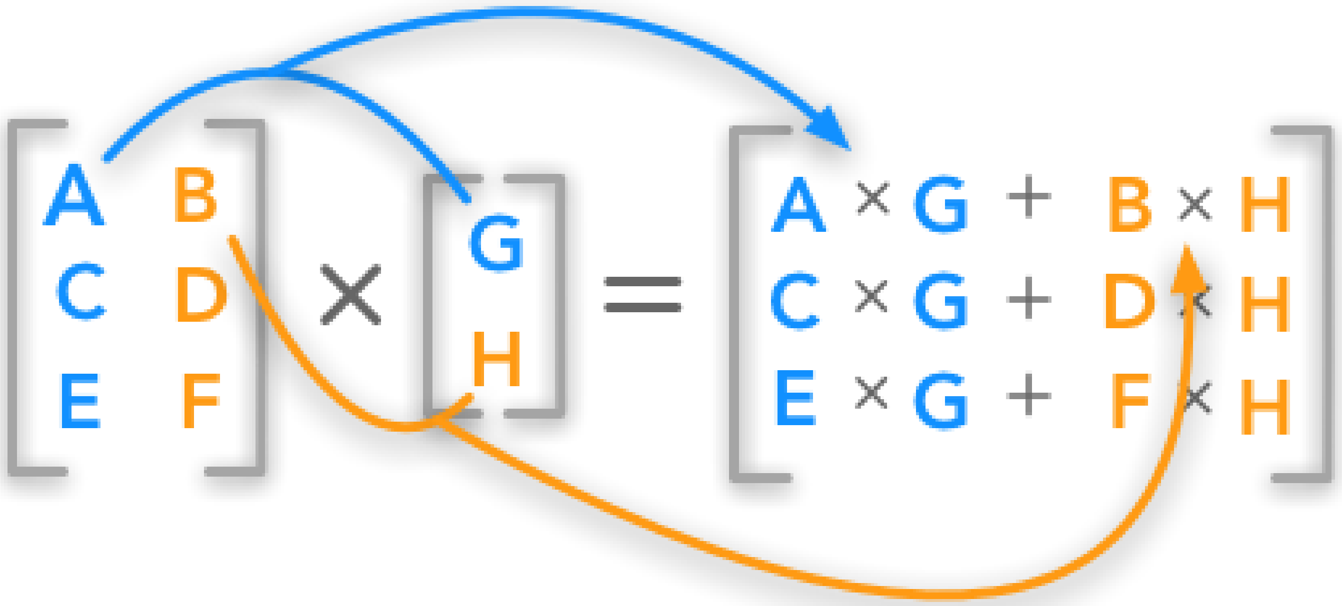
$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \mathbf{p}^T = [\, x \quad y \quad z \,]$$

$$\mathbf{p}' = \mathbf{A}\mathbf{p}$$

$$\mathbf{p}' = \mathbf{A}\mathbf{B}\mathbf{C}\mathbf{p}$$

$$(\mathbf{A}\mathbf{B})^T = \mathbf{B}^T\mathbf{A}^T$$

$$\mathbf{p}'^T = \mathbf{p}^T\mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$$



Affine Transformations

Homogeneous Coordinates

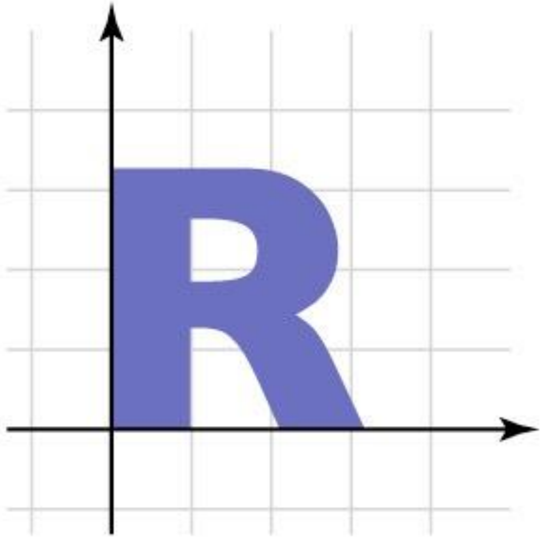
- A trick for representing the foregoing more elegantly
- Extra component *w* for vectors, extra row/column for matrices
  - for affine, can always keep *w* = 1
- Represent linear transformations with dummy extra row and column

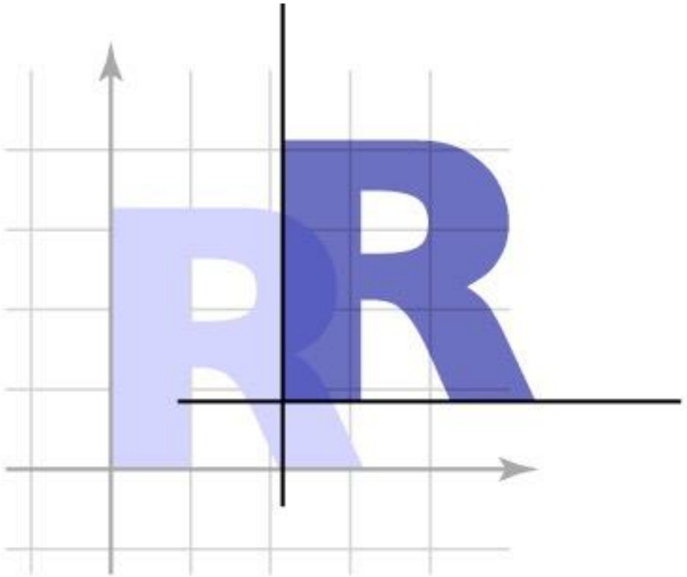
$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \\ 1 \end{bmatrix}$$

Affine transformation gallery

- Translation

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 2.15 \\ 0 & 1 & 0.85 \\ 0 & 0 & 1 \end{bmatrix}$$

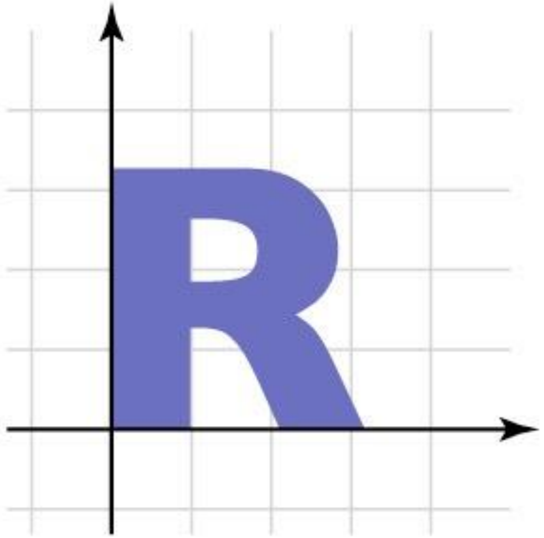


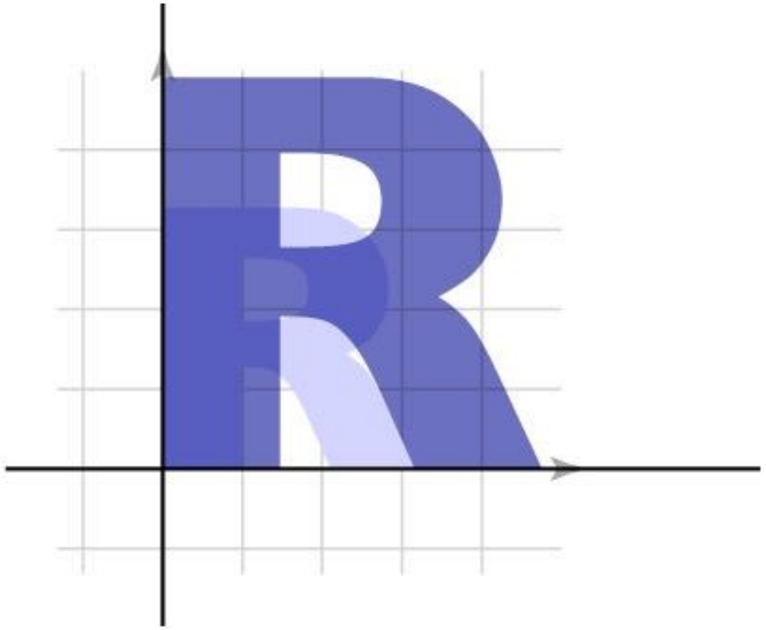


Affine transformation gallery

- Uniform scale

$$\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

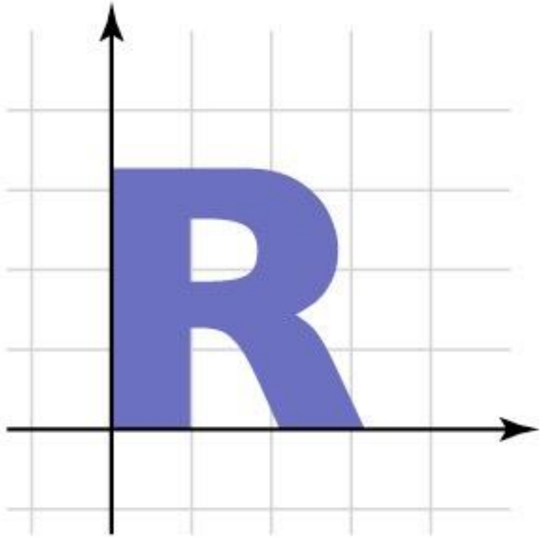


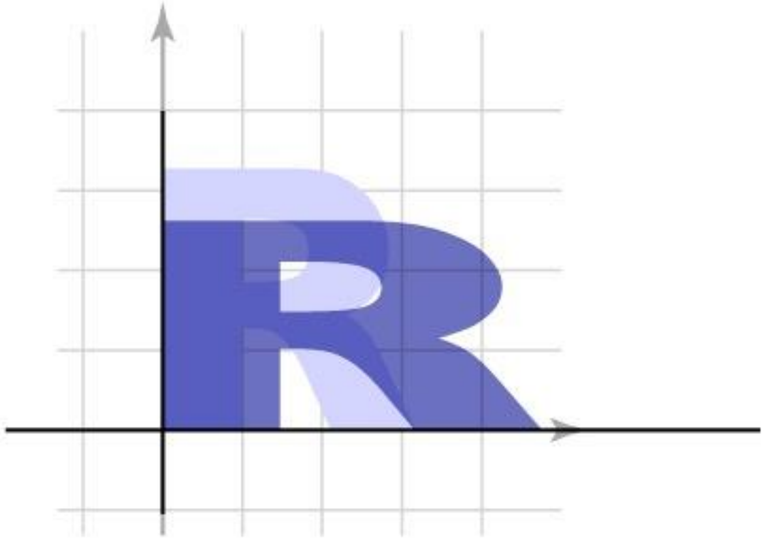


Affine transformation gallery

- Nonuniform scale

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

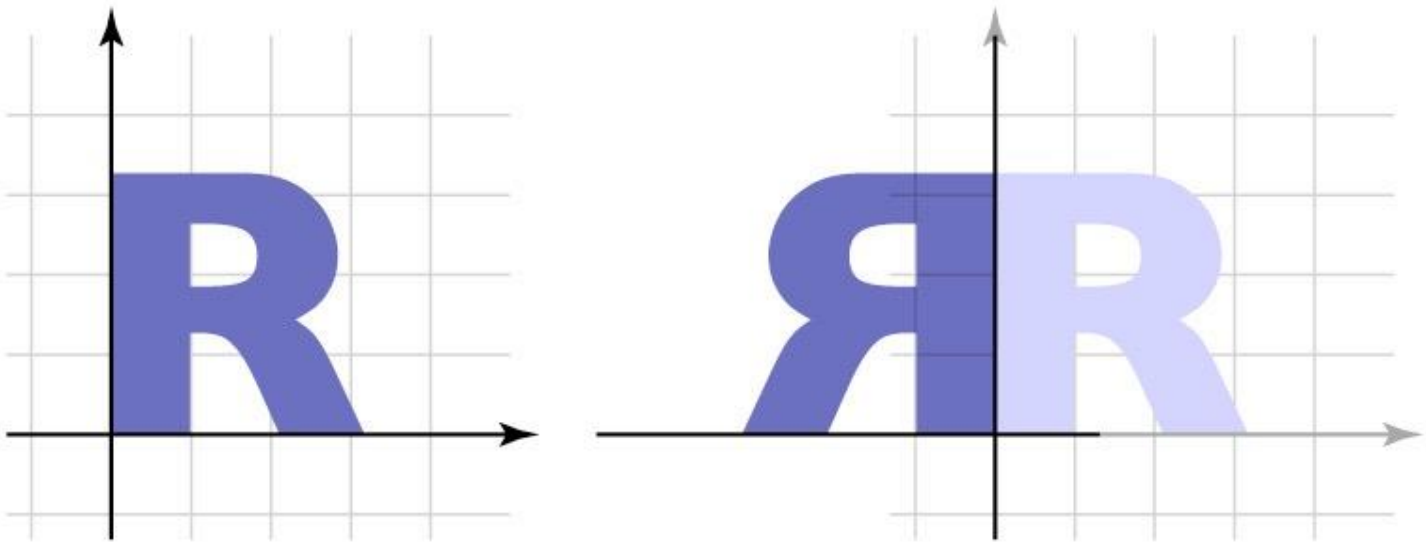




Affine transformation gallery

- Reflection
  - can consider it a special case of nonuniform scale

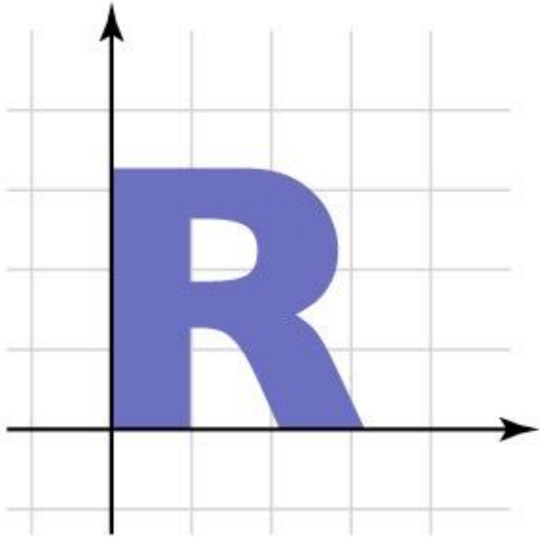
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

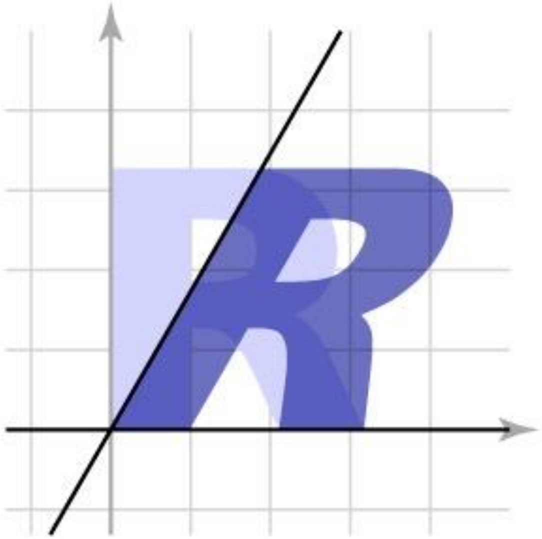


Affine transformation gallery

- Shear

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

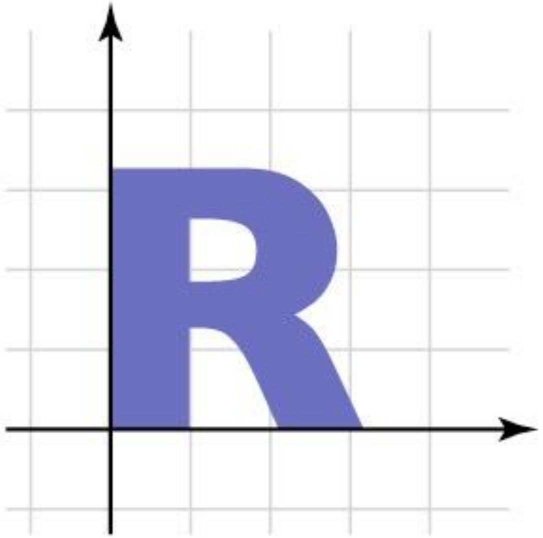


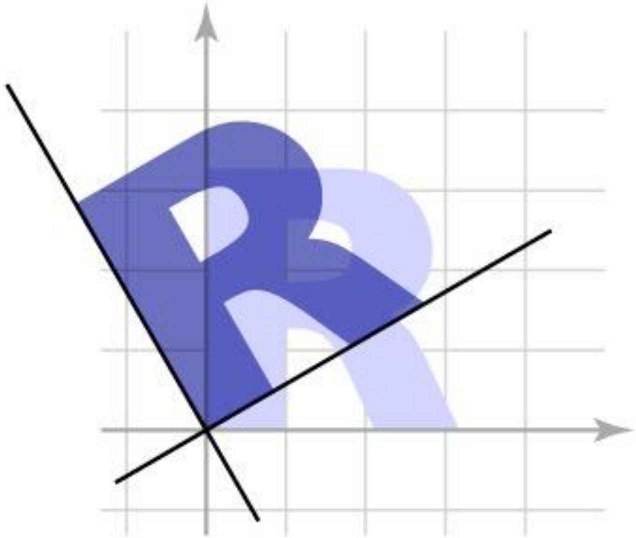


Affine transformation gallery

- Rotation

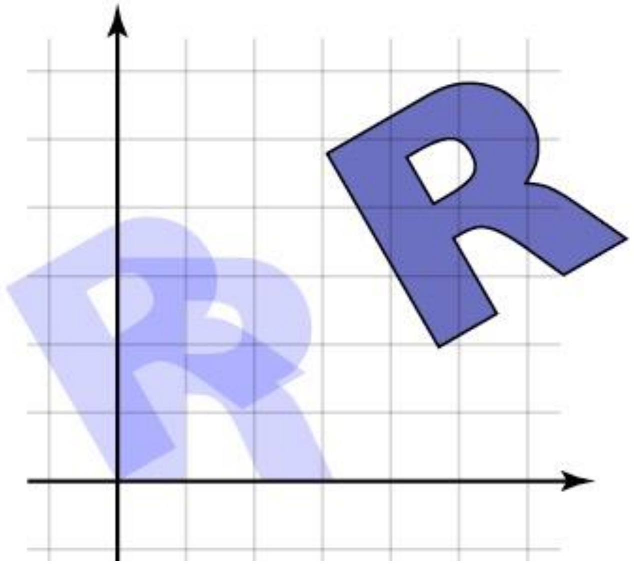
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



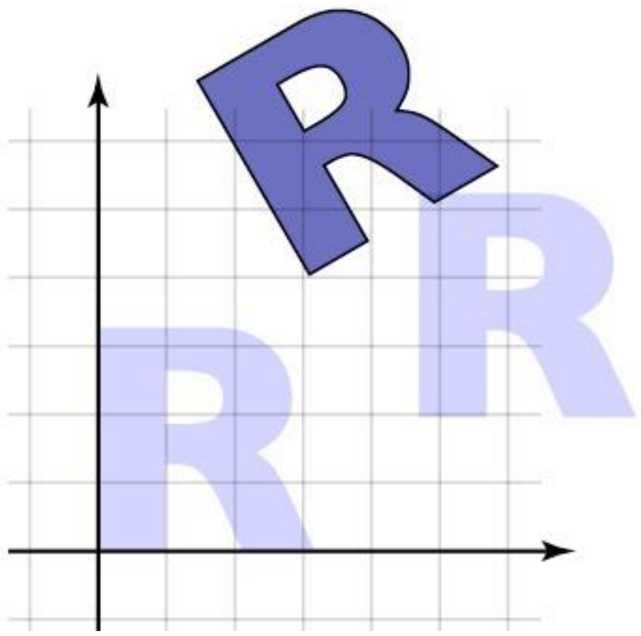


Composite affine transformations

- In general **not commutative**: order matters!



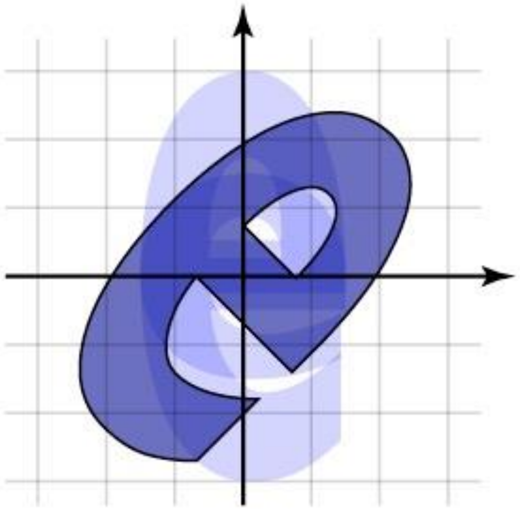
rotate, then translate



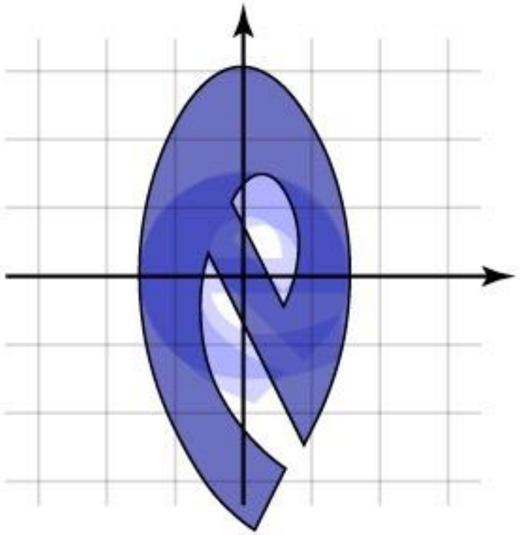
translate, then rotate

Composite affine transformations

- Another example



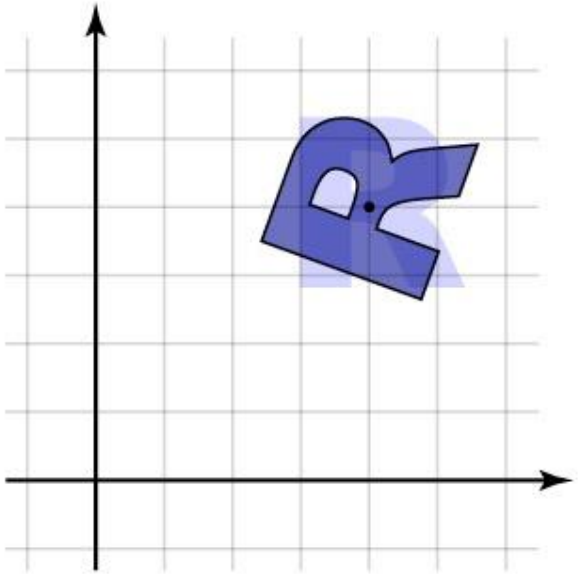
scale, then rotate



rotate, then scale

Composing to change axes

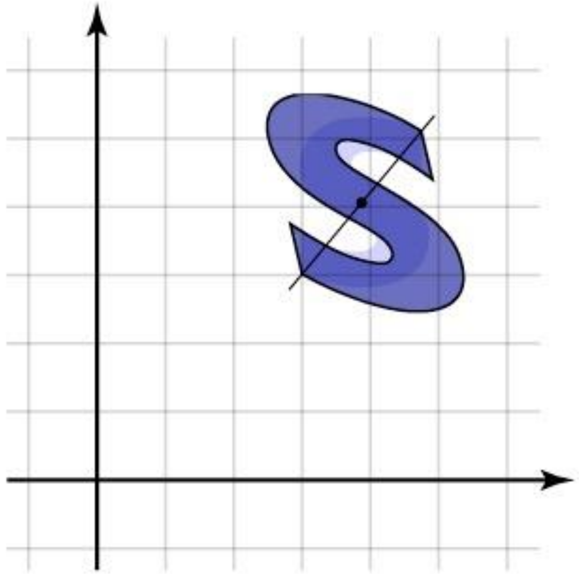
- Want to rotate about a particular point
  - could work out formulas directly…
- Know how to rotate about the origin
  - so translate that point to the origin



$$M = T^{-1}RT$$

Composing to change axes

- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  - so translate to the origin and rotate to align axes



$$M = T^{-1}R^{-1}SRT$$

Rigid motions

- A transform made up of only translation and rotation is a *rigid motion* or a *rigid body transformation*
- The linear part is an orthogonal matrix

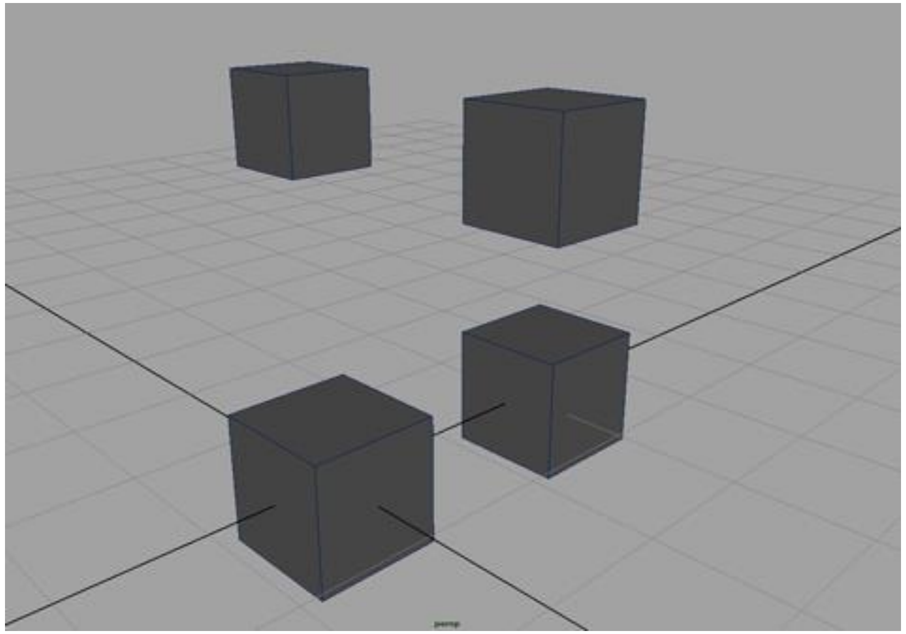
$$R = \begin{bmatrix} Q & \mathbf{u} \\ 0 & 1 \end{bmatrix}$$

- Inverse of orthogonal matrix is transpose
  - so inverse of rigid motion is easy:

$$R^{-1}R = \begin{bmatrix} Q^T & -Q^T\mathbf{u} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Q & \mathbf{u} \\ 0 & 1 \end{bmatrix}$$

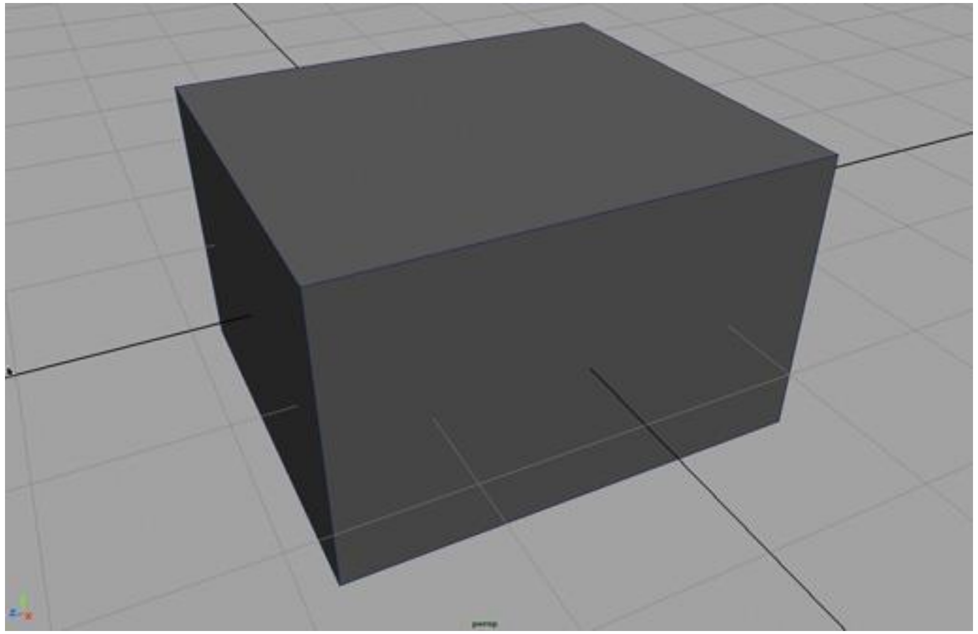
Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



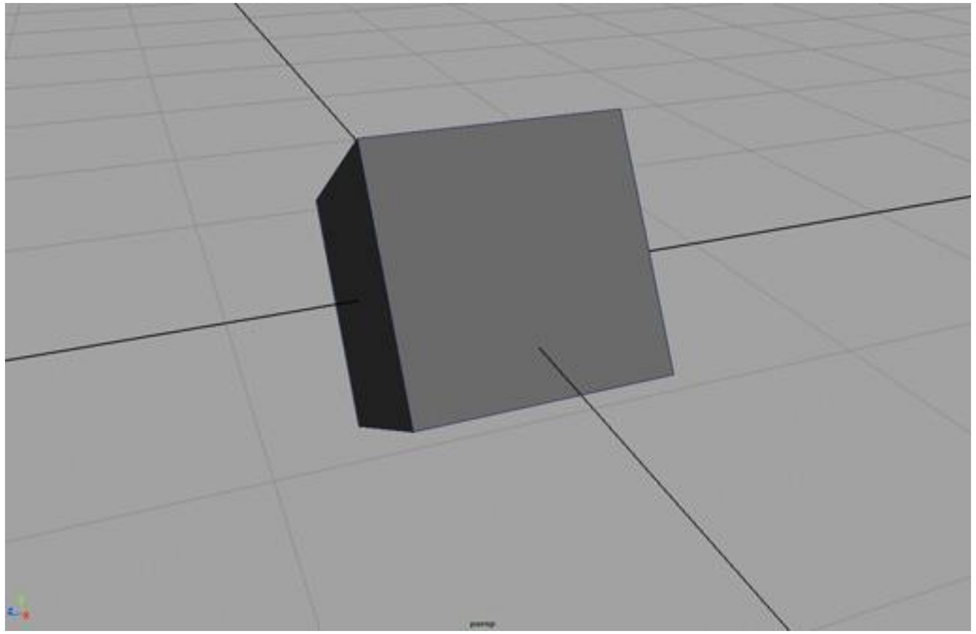
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



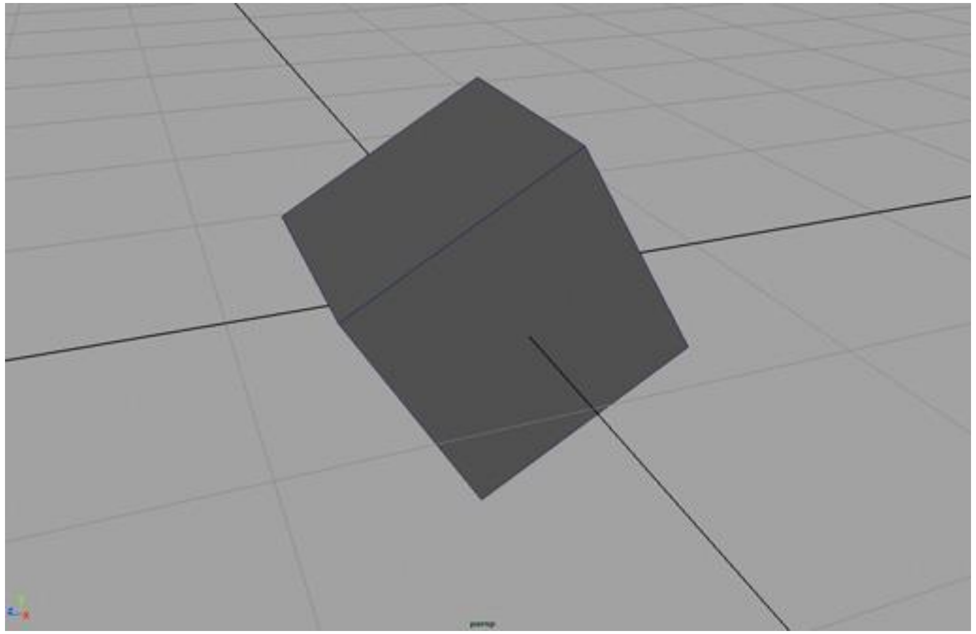
Rotation about *z* axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



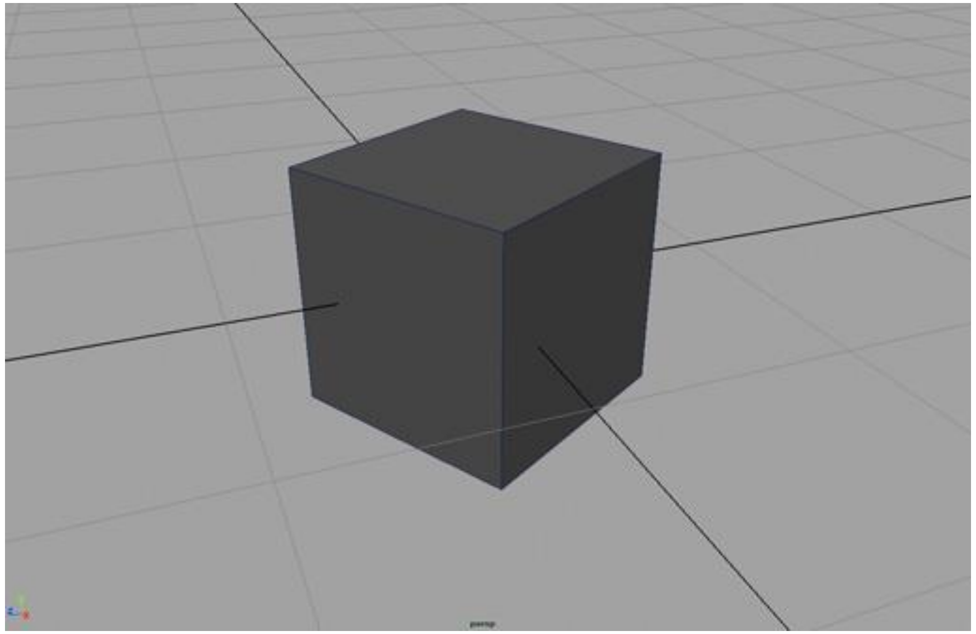
Rotation about *x* axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Rotation about *y* axis

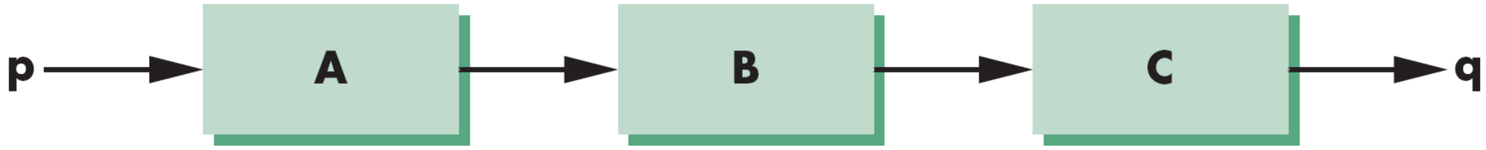
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

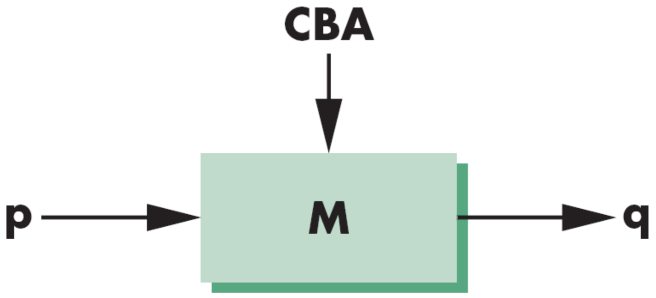


Properties of Matrices

- Translations: linear part is the identity
- Scales: linear part is diagonal
- Rotations: linear part is orthogonal
  - Columns of R are mutually orthonormal: $RR^T = R^TR = I$
  - Also, determinant of R is 1.0 [$\det(R) = 1$]

Concatenation of Transforms





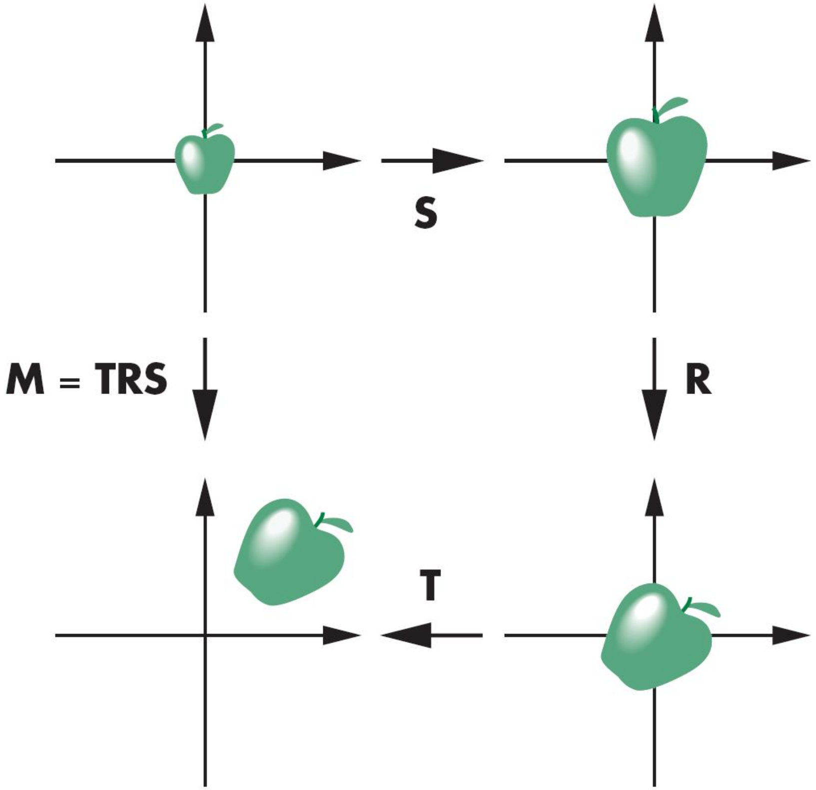
$$\mathbf{q} = \mathbf{CBAp}$$

$$\mathbf{q} = (\mathbf{C}(\mathbf{B}(\mathbf{Ap})))$$

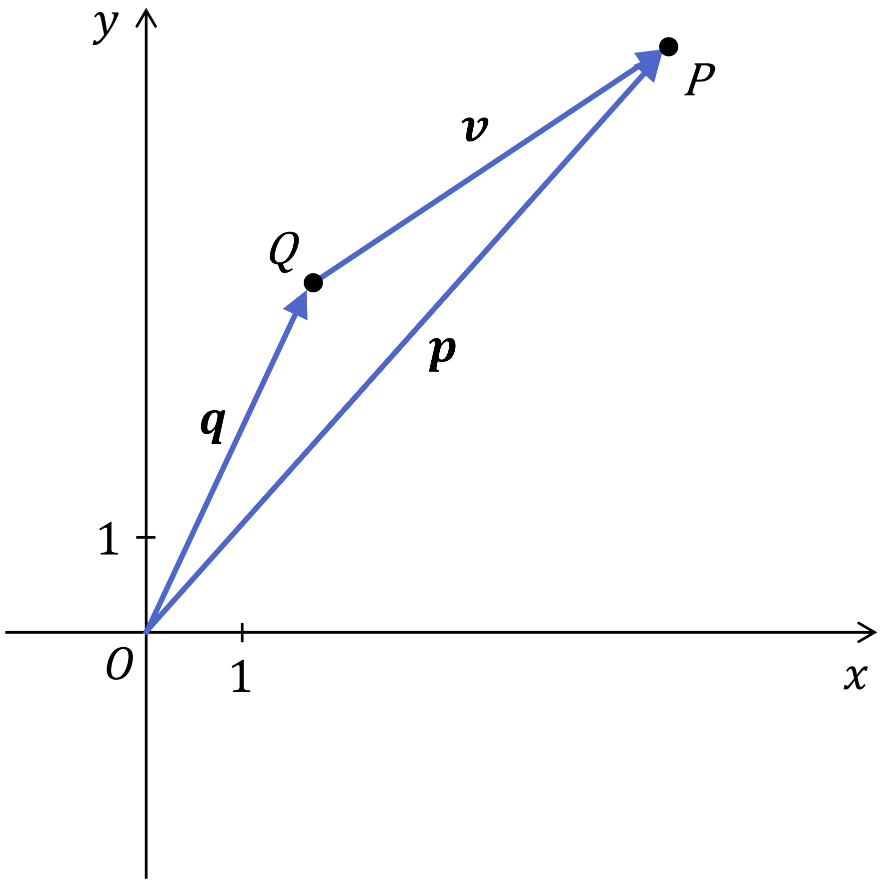
$$\mathbf{M} = \mathbf{CBA}$$

$$\mathbf{q} = \mathbf{Mp}$$

The Instance Transformation



Transforming Points and Vectors



- Recall distinction points vs. vectors
  - vectors are just offsets (differences between points)
  - points have a location
    - represented by vector offset from a fixed origin
- Points and vectors transform differently:
  - points respond to translation;
  - vectors do not

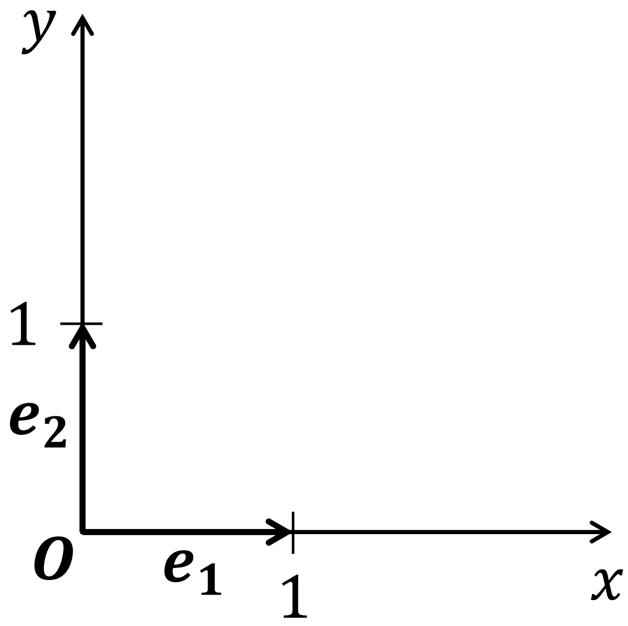
Transforming Points and Vectors

$$\begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} M\mathbf{p} + \mathbf{t} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ 0 \end{bmatrix} = \begin{bmatrix} M\mathbf{v} \\ 0 \end{bmatrix}$$

- Homogeneous coordinates let us exclude translation
  - just put 0 rather than 1 in the last place
  - and note that subtracting two points cancels the extra coordinate, resulting in a vector!

Recall: Basis



- Basis vectors in homogenous coordinates

$$\boldsymbol{e_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \boldsymbol{e_2} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- Origin in homogeneous coordinates

$$\boldsymbol{O} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

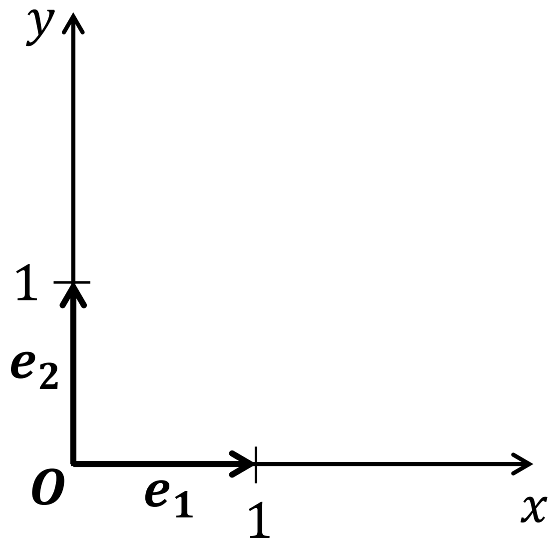
- Together, basis and point (origin): canonical frame

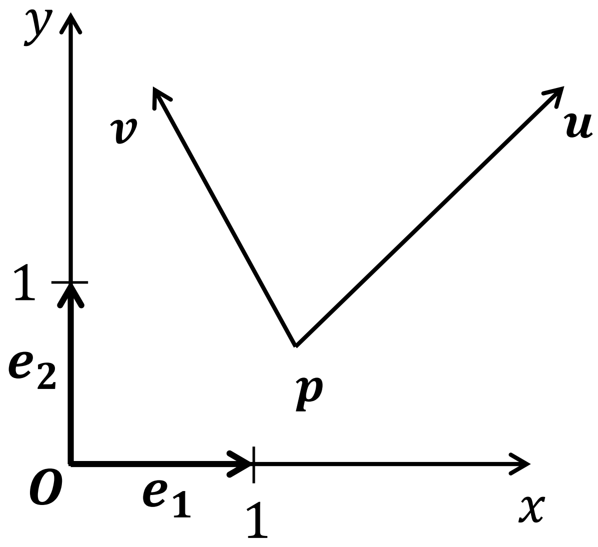
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Affine Change of Coordinates

- transformation matrix from "local frame" to "canonical frame"

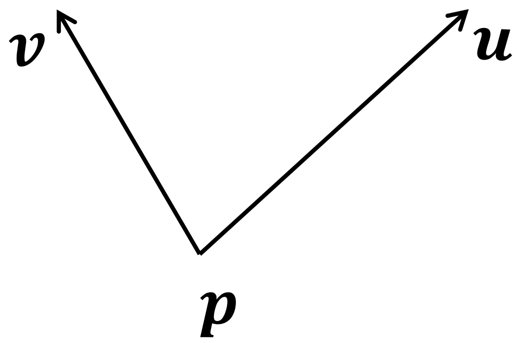
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} u_x & v_x & p_x \\ u_y & v_y & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{u} & \boldsymbol{v} & \boldsymbol{p} \\ 0 & 0 & 1 \end{bmatrix}$$





Affine Change of Coordinates

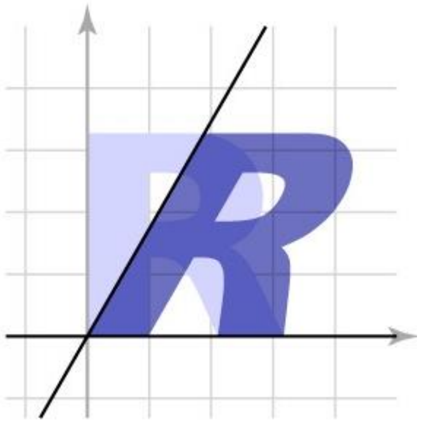
- Coordinate frame: point plus basis
- Interpretation: transformation changes representation of point from one basis to another
- "Frame to canonical" matrix has local frame in columns
    - takes points represented in frame
    - represents them in canonical basis
    - e.g. [0 0], [1 0], [0 1]
- Seems backward but bears thinking about



$$\begin{bmatrix} u_x & v_x & p_x \\ u_y & v_y & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{u} & \boldsymbol{v} & \boldsymbol{p} \\ 0 & 0 & 1 \end{bmatrix}$$

Affine Change of Coordinates

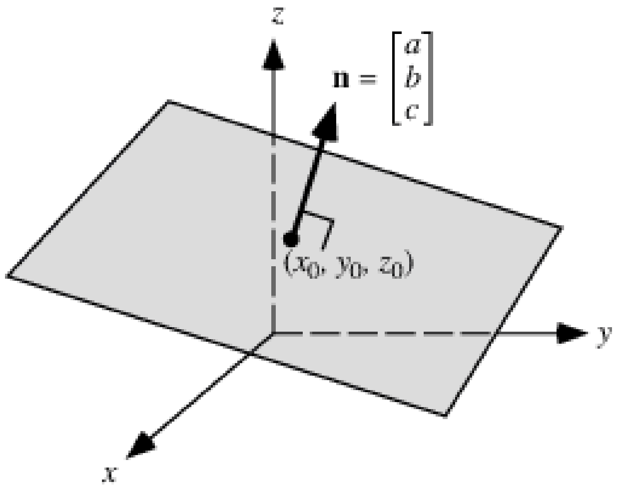
- A new way to “read off” the matrix
  - e.g. shear from earlier
  - can look at picture, see effect on basis vectors, write down matrix
- Also an easy way to construct transforms
  - e. g. scale by 2 across direction (1,2)

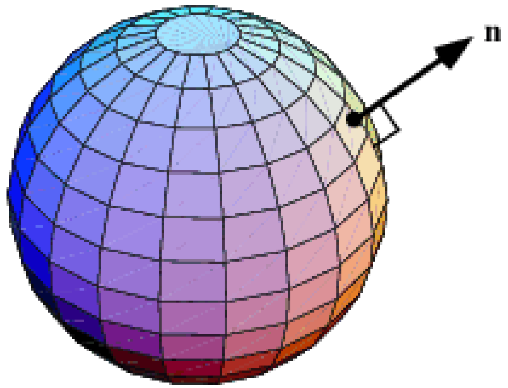


$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Normal Vectors

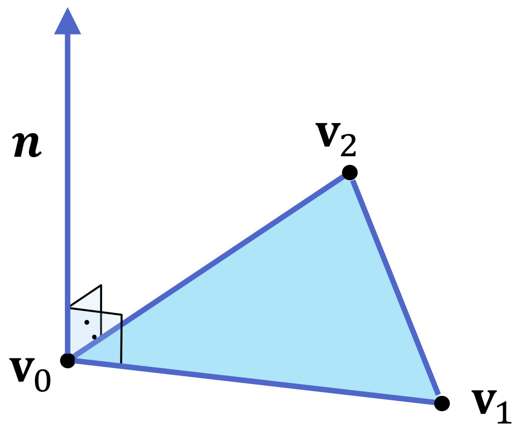
- Normal Vectors:
  - Vectors perpendicular to the surface
- Local linear approximation of the surface
  - First order Taylor approximation
  - Cross product of first partial derivatives of the surface function



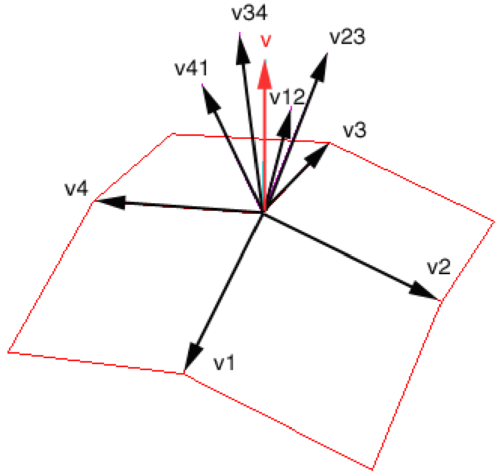


Normal Vectors

- Normal Vectors:
  - Vectors perpendicular to the surface
- Local linear approximation of the surface
  - First order Taylor approximation
  - Cross product of first partial derivatives of the surface function
- On triangle meshes
  - Face normals: cross product of triangle edge-vectors
  - Vertex normals: average of incident face normals



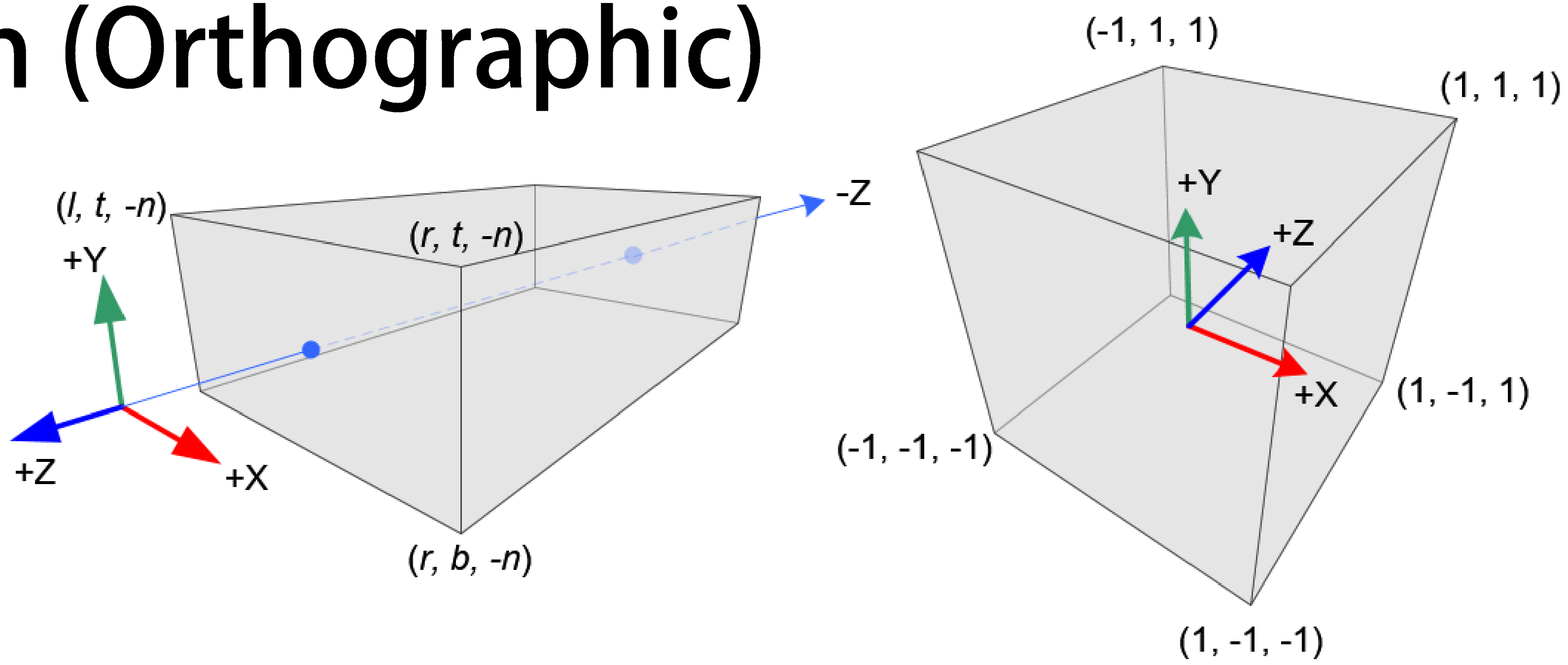
$$\boldsymbol{n} = (\mathbf{v}_2 - \mathbf{v}_0) \times (\mathbf{v}_1 - \mathbf{v}_0)$$



$$\boldsymbol{n}_{\mathrm{v}} = \frac{1}{4}\sum_{i=1}^{4} \boldsymbol{n}_i$$

Projective Transformations

Parallel Projection (Orthographic)



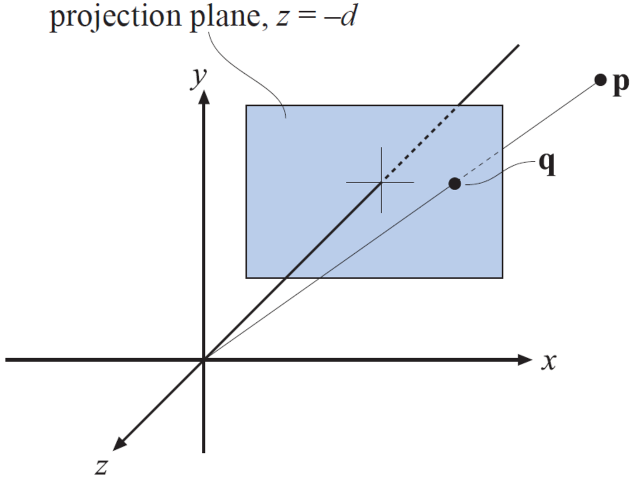
$$\mathbf{T} = \mathbf{T}(-(right + left)/2, -(top + bottom)/2, (far + near)/2)$$

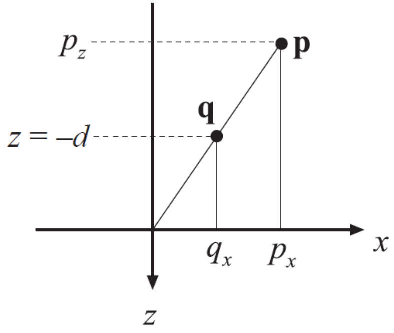
and

$$\mathbf{S} = \mathbf{S}(2/(right - left), 2/(top - bottom), 2/(near - far)),$$

$$\mathbf{N} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{left+right}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection



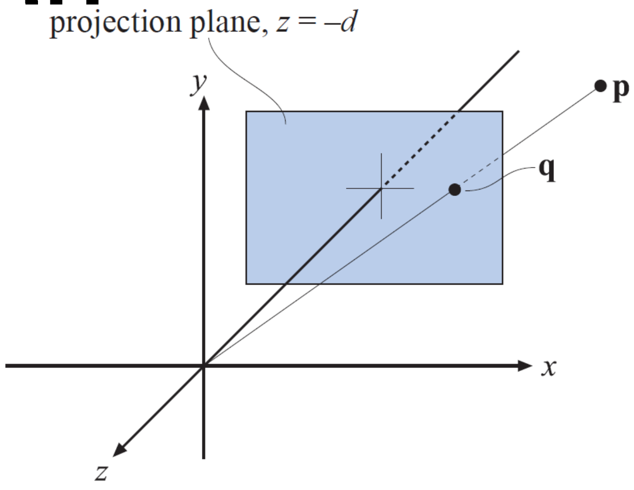


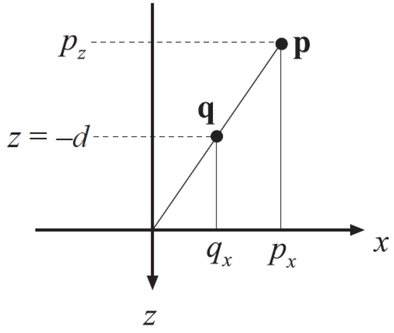
$$\frac{q_x}{p_x} = \frac{-d}{p_z} \iff q_x = -d\frac{p_x}{p_z}$$

$$\mathbf{q} = \mathbf{P}_p\mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \Rightarrow \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{pmatrix}$$

Perspective Projection

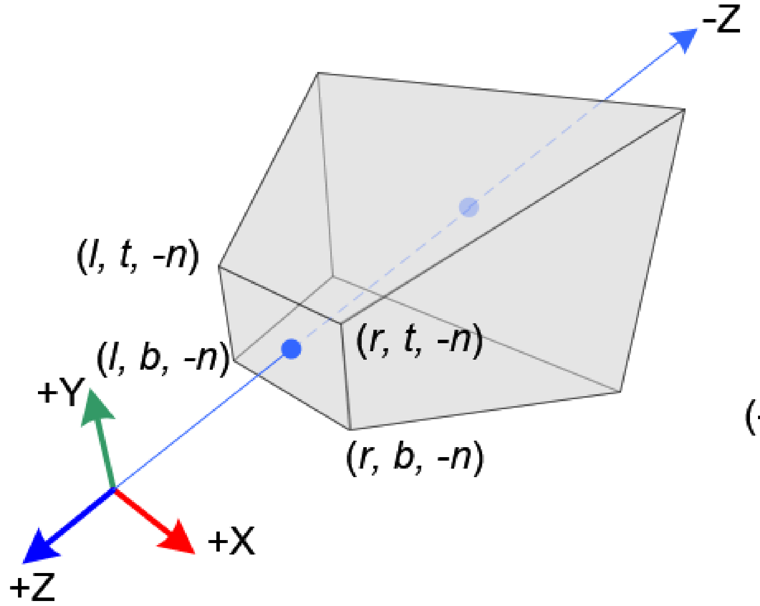
$$\mathbf{p} = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

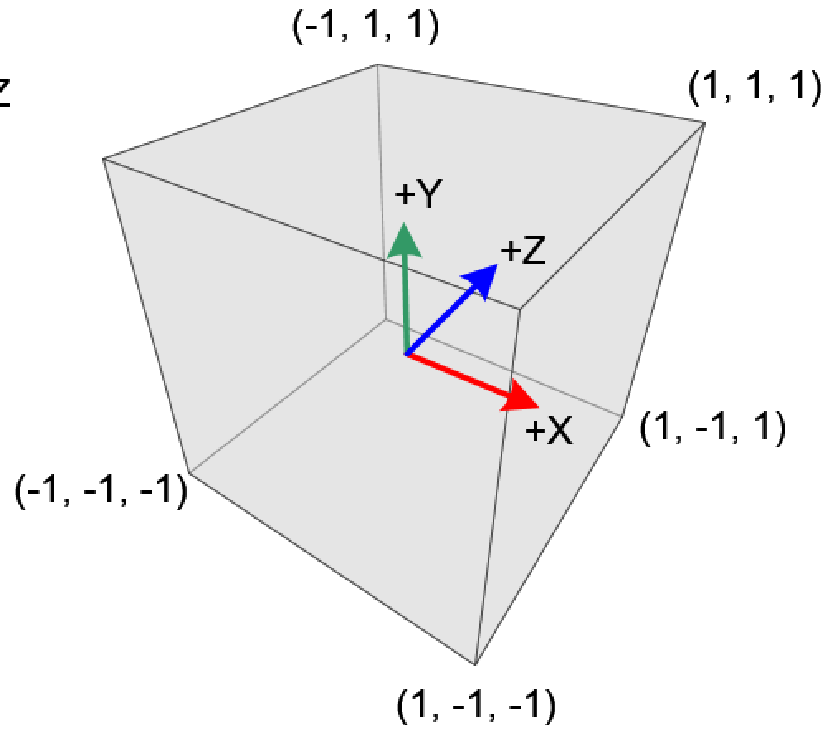




$$\mathbf{q} = \mathbf{P}_p\mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \Rightarrow \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{pmatrix}$$

Perspective Projection



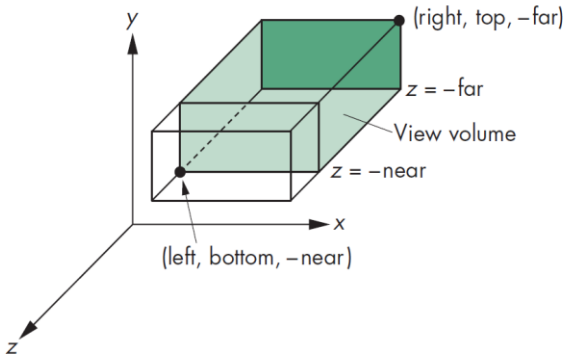


$$P = NSH = \begin{bmatrix} \frac{2*near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2*near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

```
frustum = function(left, right, bottom, top, near, far)
```

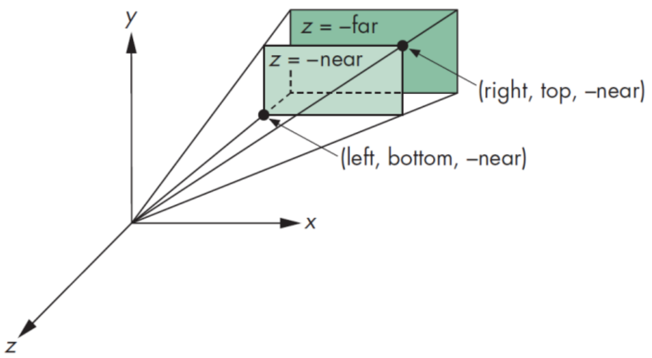
Specifying Projection Matrix

Interactive demo of perspective: https://webglfundamentals.org/webgl/frustum-diagram.html



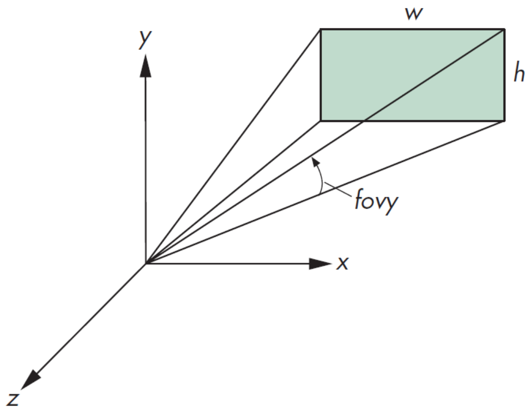

```
let projMat = ortho(left, right, bottom, top, near, far);
```

$$N = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{left+right}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$




```
let projMat = frustum(left, right, bottom, top, near, far);
```

$$P = NSH = \begin{bmatrix} \frac{2*near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2*near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$




```
let projMat = perspective(cameraFovy, aspect, near, far);
```

$$P = NSH = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & \frac{-(far+near)}{far-near} & \frac{-2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$left = -right$

$bottom = -top$,

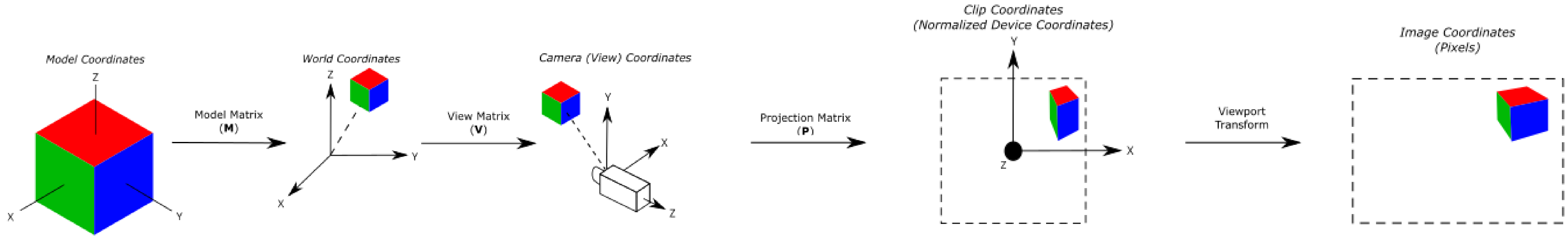
and simple trigonometry to determine

$top = near * \tan(fovy)$

$right = top * aspect$,

Viewing

Model-View-Projection-Viewport

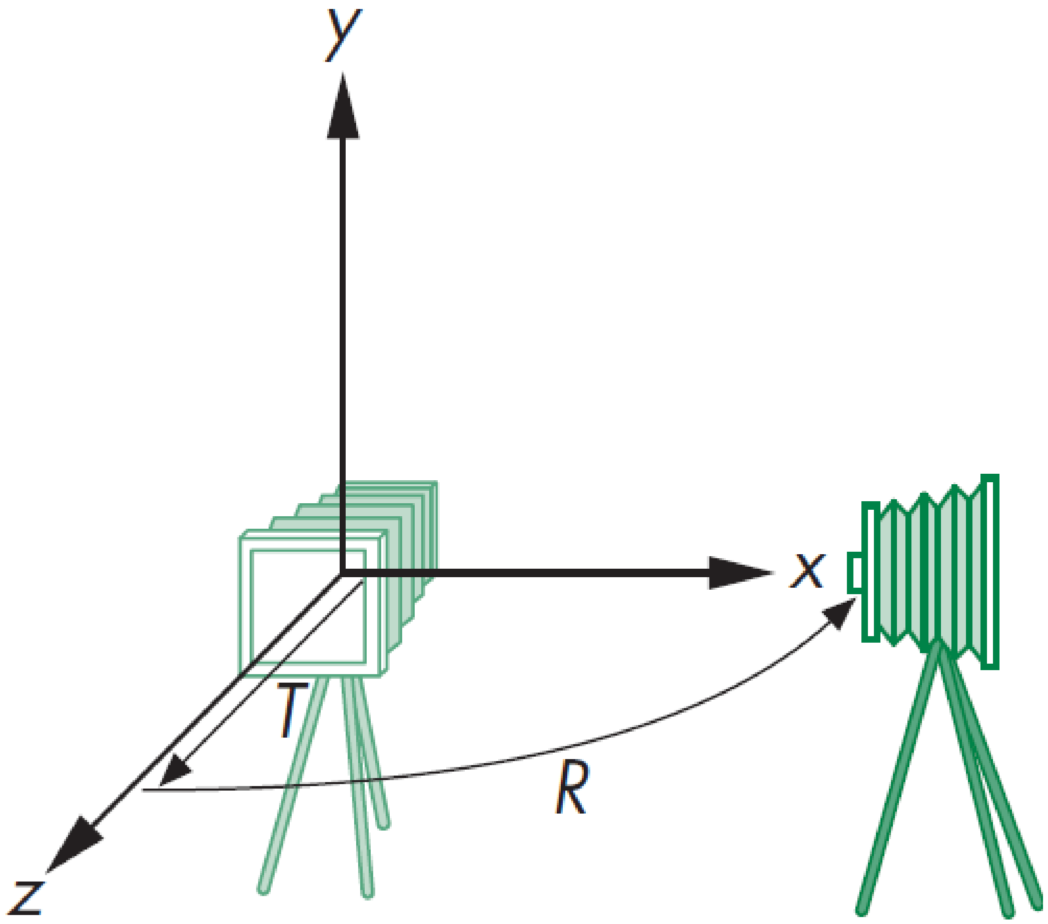


Model View Matrix

```
modelViewMatrix = mult(translate(0, 0, -d), rotateY(-90));
```

- Model-View Matrix is a combination of the
  - Model Matrix (transforms model in world space)
  - View Matrix (transforms the camera in world space)

- The View Matrix can be creates using the Look At function

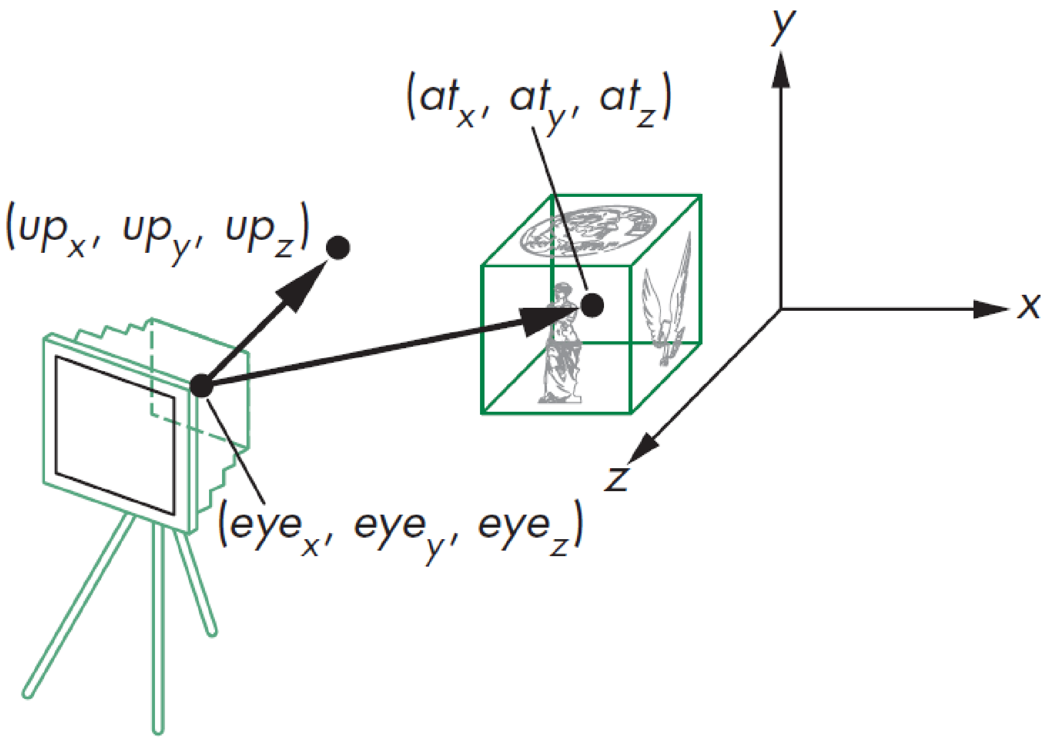


Look At

- The View Matrix can be created using the

LookAt function

```
let eye = vec3(d,0,0);
let up = vec3(0,1,0);
let at = vec3(0,0,0);
let viewMat = lookAt(eye, at, up);
```



Model-View-Projection-Viewport

object/model coordinates — input to the vertex shader, i.e. position in attributes

↓ **modeling transformation**: model matrix $\mathrm{M}_{\mathrm{object}\to\mathrm{world}}$

world coordinates

↓ **viewing transformation**: view matrix $\mathrm{M}_{\mathrm{world}\to\mathrm{view}}$

view/eye coordinates

↓ **projection transformation**: projection matrix $\mathrm{M}_{\mathrm{projection}}$

clip coordinates — output of the vertex shader, i.e. `gl_Position`

↓ **perspective division** (by `gl_Position.w`)

normalized device coordinates

↓ **viewport transformation**

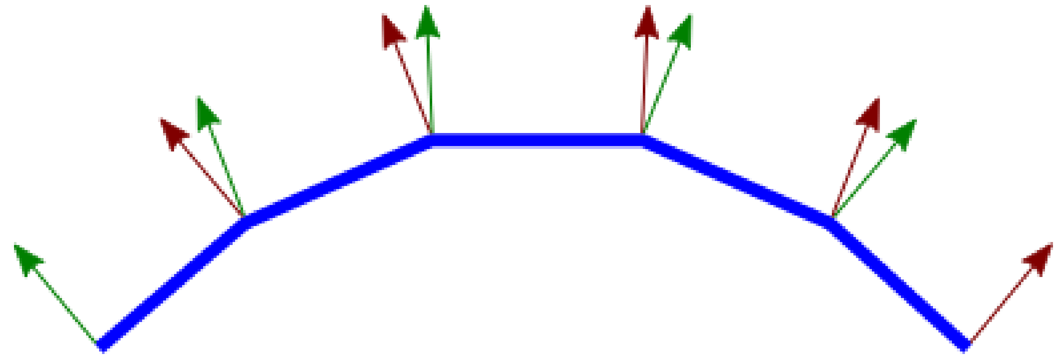
screen/window coordinates — `gl_FragCoord` in the fragment shader

- Other depiction of the MVP and Viewport pipeline

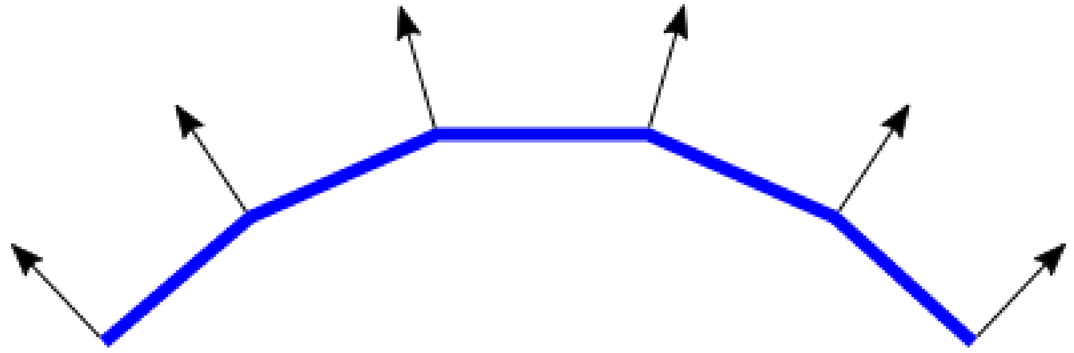
Shading and Lighting

Flat Shading vs Smooth Shading

- Normals can be
  - Per face (per polygon, per triangle)
  - Per Vertex
- Per Face Normals:

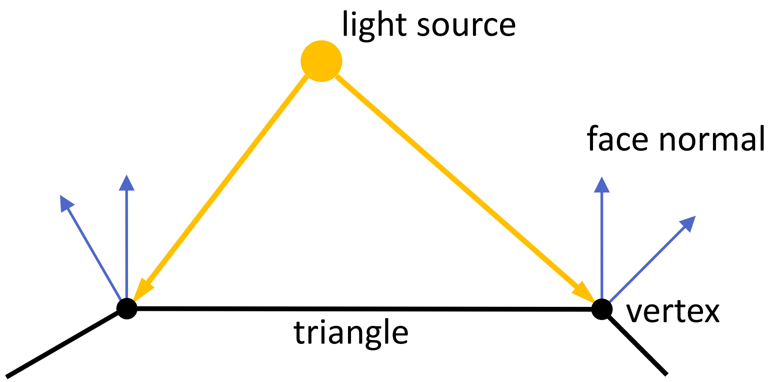


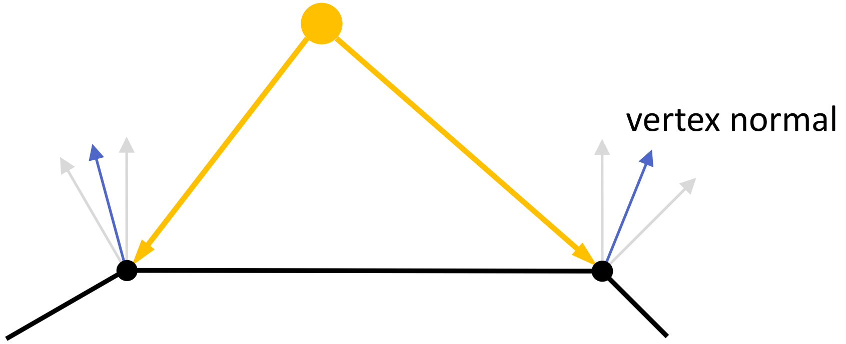
- Per Vertex Normals

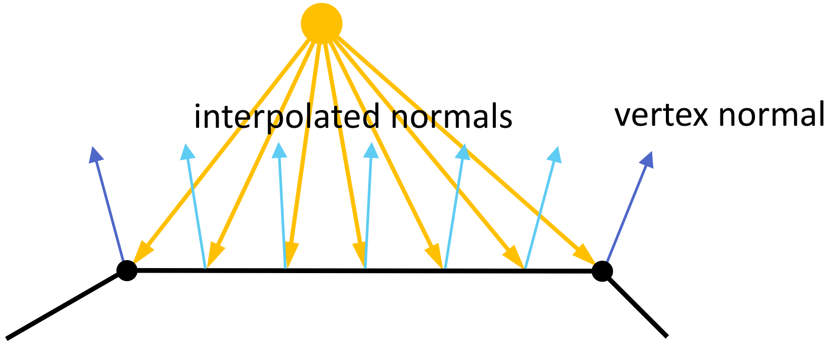


Shading Overview

- Flat
  - Shading computed per vertex
  - Normal per triangle, the same at all vertices (face normal)
  - Color values interpolated per fragment
- Gouraud
  - Shading computed per vertex
  - Vertex normals as average of face normals
  - Color values interpolated per fragment
- Phong
  - Vertex normals are interpolated
  - Shading computed using interpolated normal per fragment



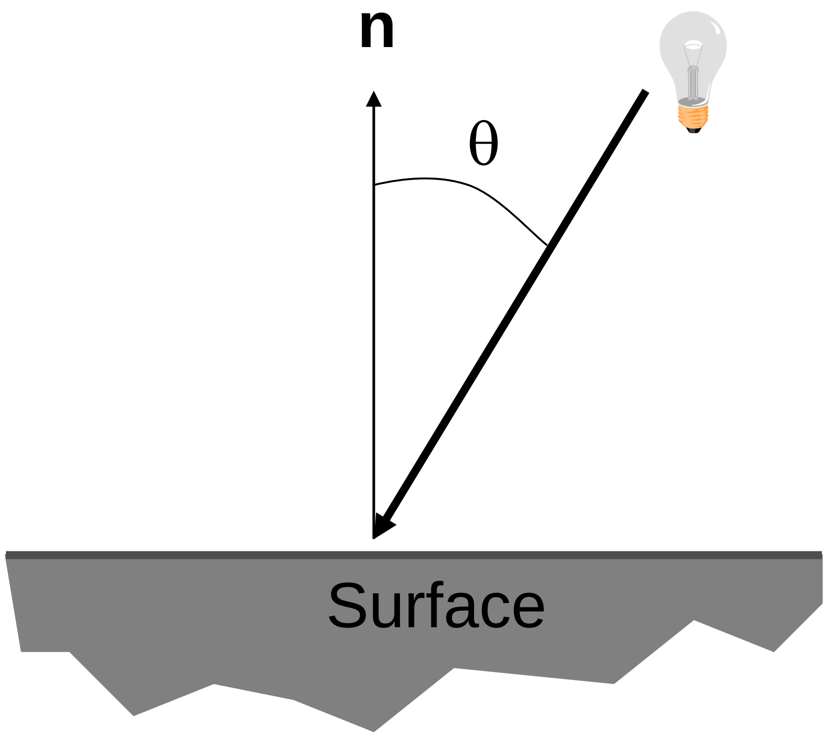




Lambert Cosine Law

- The amount of light received by a surface depends on incoming angle
  - Bigger at normal incidence
    - Similar to Winter/Summer difference

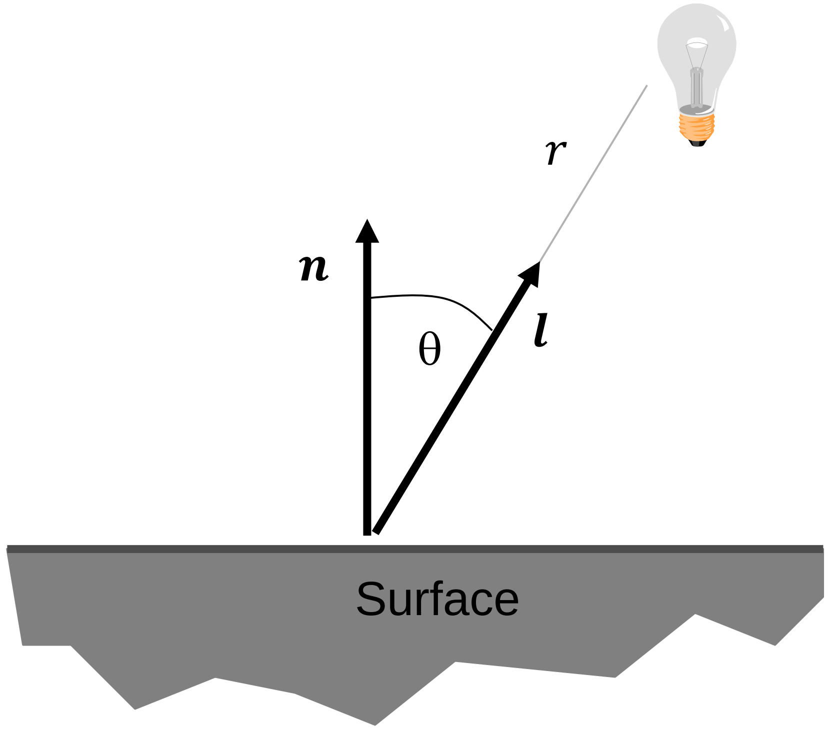
- By how much?
  - **Cos(θ)** law
  - Dot product with normal



Lambert Cosine Law

- Single Point Light Source
  - k_d: diffuse coefficient.
  - $\boldsymbol{n}$: Surface normal.
  - $\boldsymbol{l}$: Light direction.
  - L_i: Light intensity
  - r : Distance to source

$$L_0 = k_d(\boldsymbol{n} \cdot \boldsymbol{l}) \frac{L_i}{r^2}$$



Ideal Diffuse Reflectance

- If $\boldsymbol{n}$ and $\boldsymbol{l}$ are facing away from each other, $(\boldsymbol{n} \cdot \boldsymbol{l})$ becomes negative.

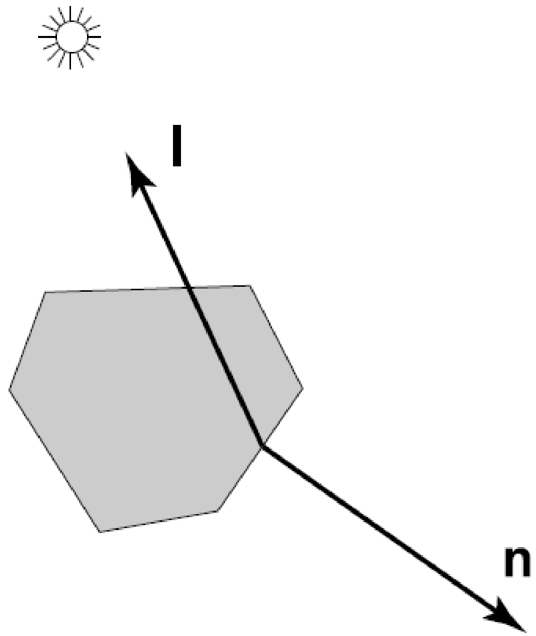
- Using

$$\max((\boldsymbol{n} \cdot \boldsymbol{l}), 0)$$

makes sure that the result is zero.

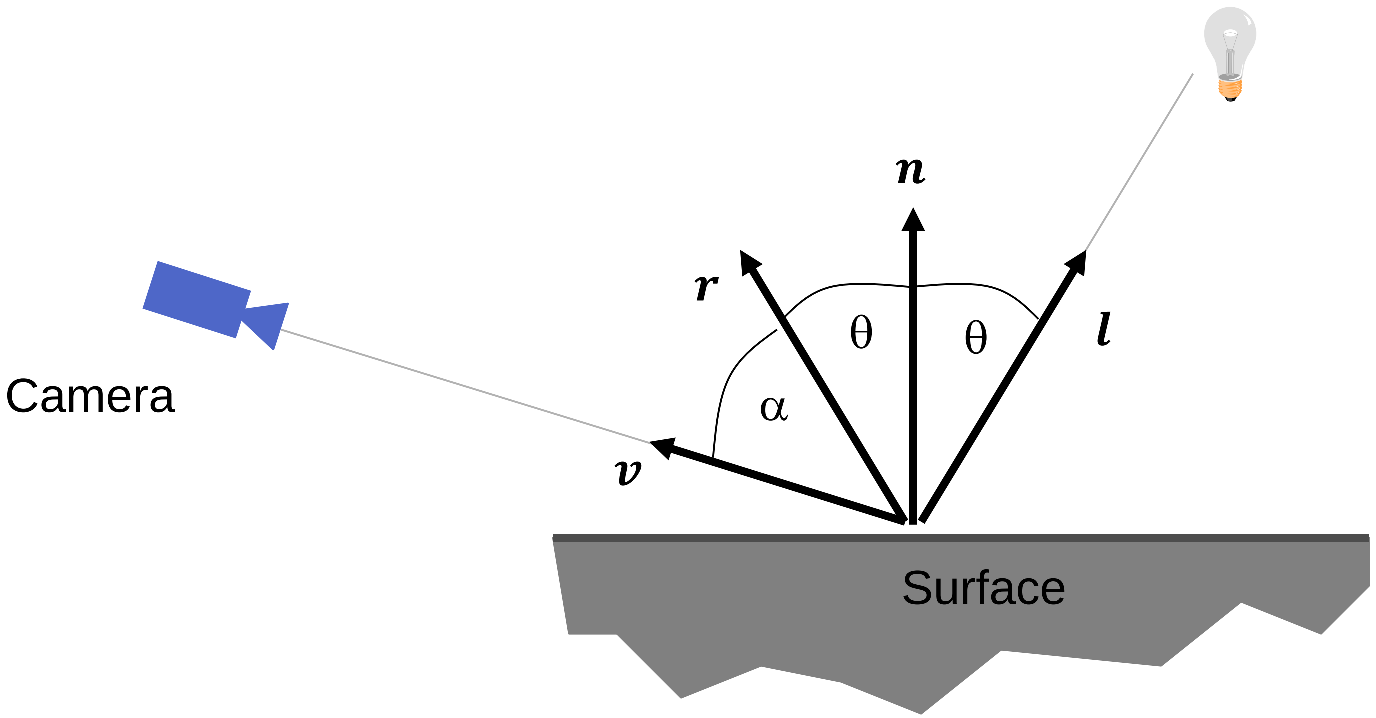
- From now on, we mean max() when we write •.

- Do not forget to normalize your vectors for the dot product!



Phong Lighting Model

- How much light is reflected?
  - Depends on the angle between the ideal reflection direction and the viewer direction α.

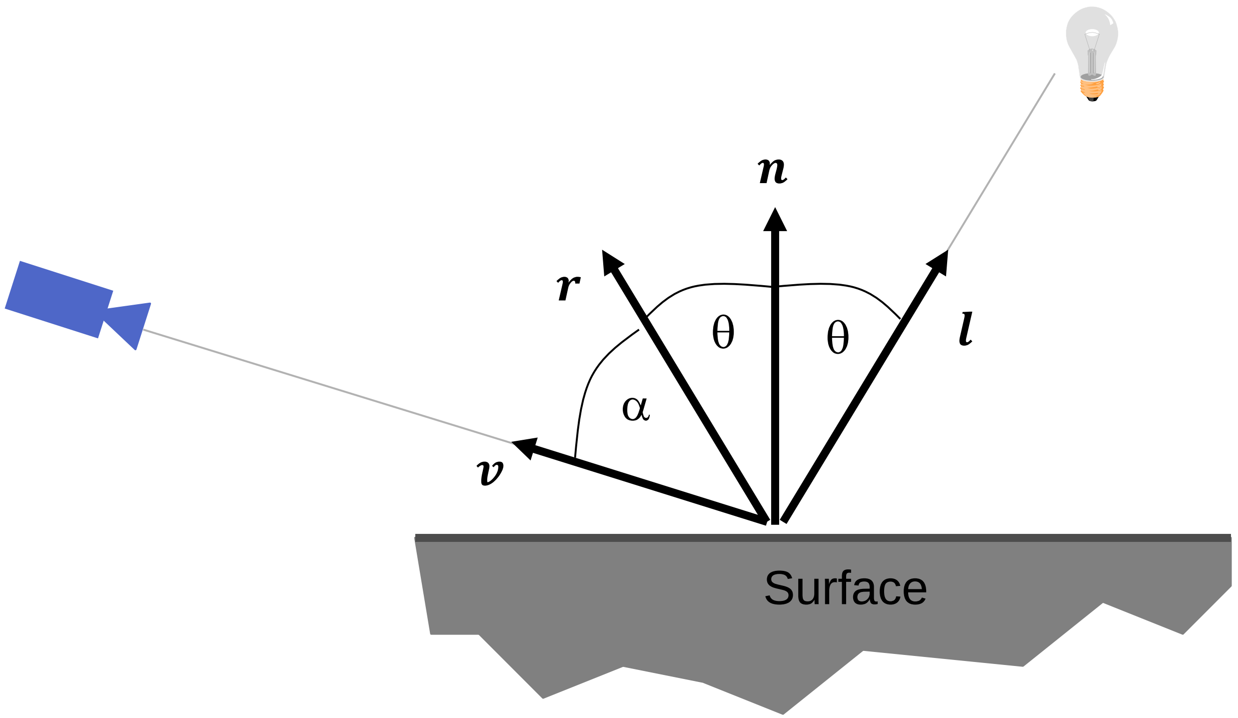


Phong Lighting Model

- Parameters
  - k_s: specular reflection coefficient
  - q : specular reflection exponent

$$L_0 = k_s(\cos(\alpha))^q \frac{L_i}{r^2}$$

$$L_0 = k_s(\boldsymbol{v} \cdot \boldsymbol{r})^q \frac{L_i}{r^2}$$



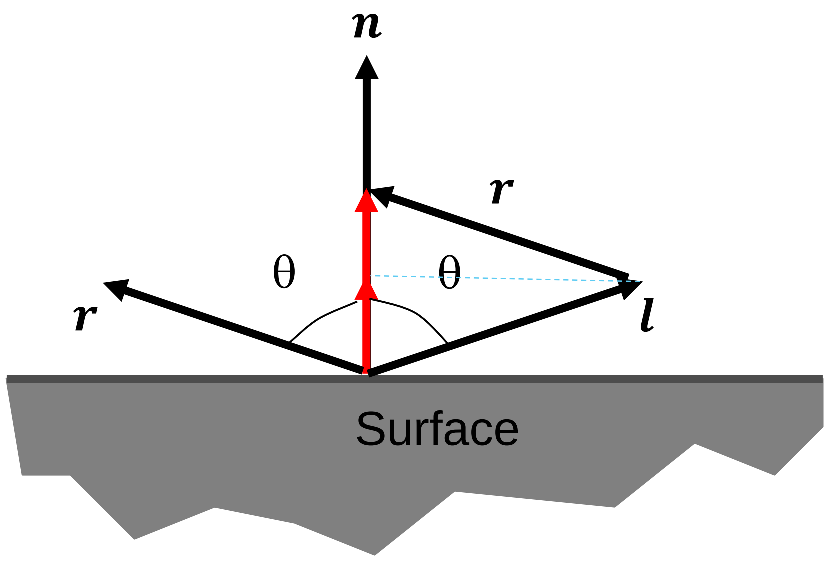
How to get the mirror direction?

$$\boldsymbol{r} + \boldsymbol{l} = 2\,\boldsymbol{n}\cos(\theta) = 2\boldsymbol{n}(\boldsymbol{n} \cdot \boldsymbol{l})$$

$$\boldsymbol{r} = 2(\boldsymbol{n} \cdot \boldsymbol{l})\boldsymbol{n} - \boldsymbol{l}$$

$$L_0 = k_s(\boldsymbol{v} \cdot \boldsymbol{r})^q \frac{L_i}{r^2}$$

$$= k_s(\boldsymbol{v} \cdot (2\boldsymbol{n}(\boldsymbol{n} \cdot \boldsymbol{l}) - \boldsymbol{l})^q \frac{L_i}{r^2}$$



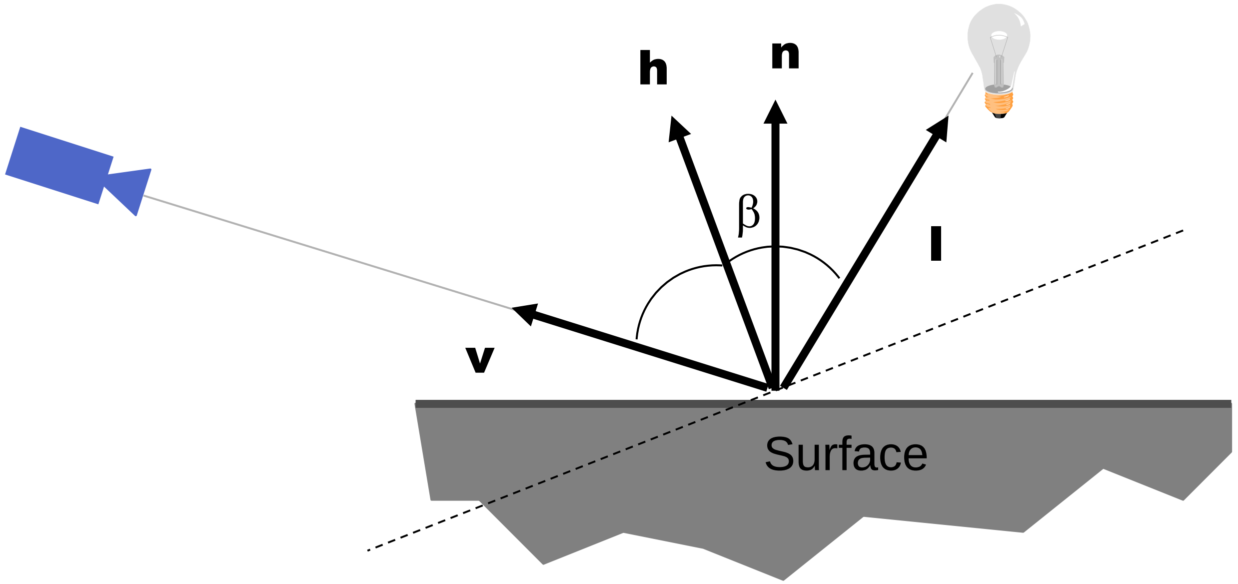
Blinn Lighting Variation

- Uses the halfway vector **h** between **l** and **v**.

$$\boldsymbol{h} = \frac{\boldsymbol{l} + \boldsymbol{v}}{\|\boldsymbol{l} + \boldsymbol{v}\|}$$

$$L_0 = k_s(\cos\beta)^q \frac{L_i}{r^2}$$

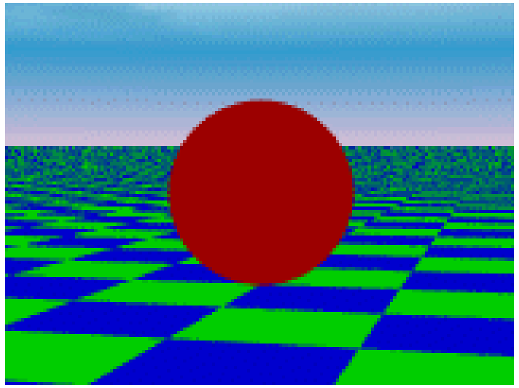
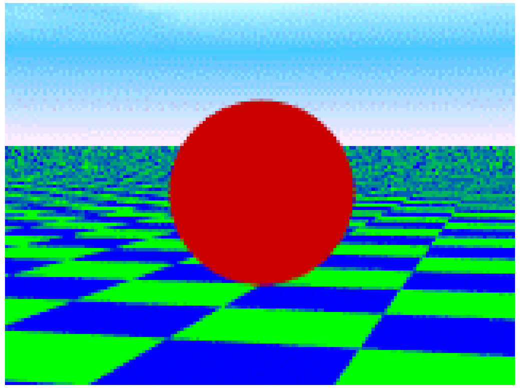
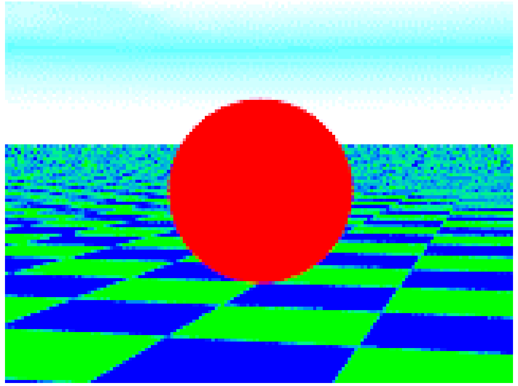
$$= k_s(\boldsymbol{n} \cdot \boldsymbol{h})^q \frac{L_i}{r^2}$$



Ambient Illumination

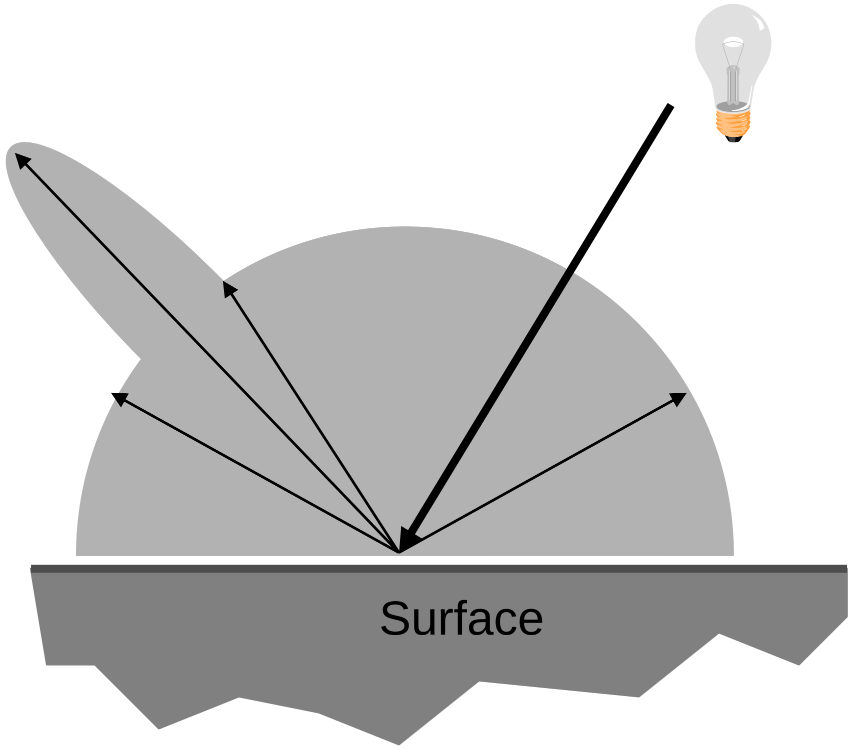
- Represents the reflection of all indirect illumination.
- This is a total hack!
- Avoids the complexity of global illumination.

$$L_a = k_a L_i$$

Putting it all together

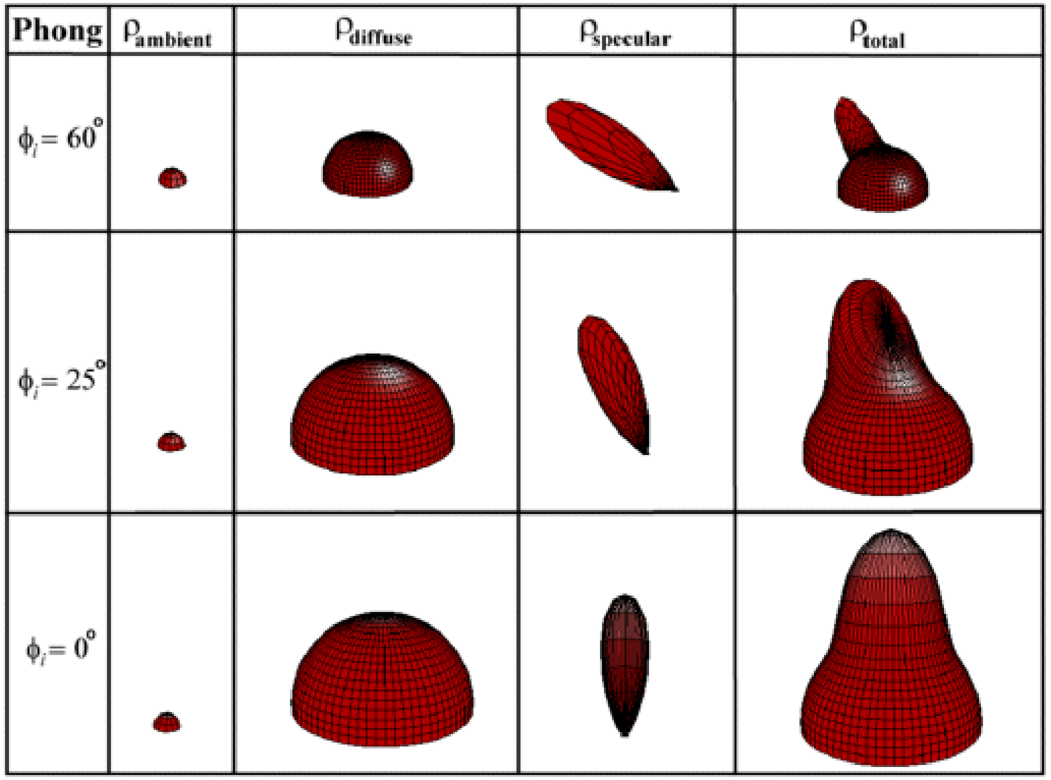
- Sum of three components:
  - diffuse reflection +
  - specular reflection +
  - ambient.



Putting it all together

- Phong Illumination Model

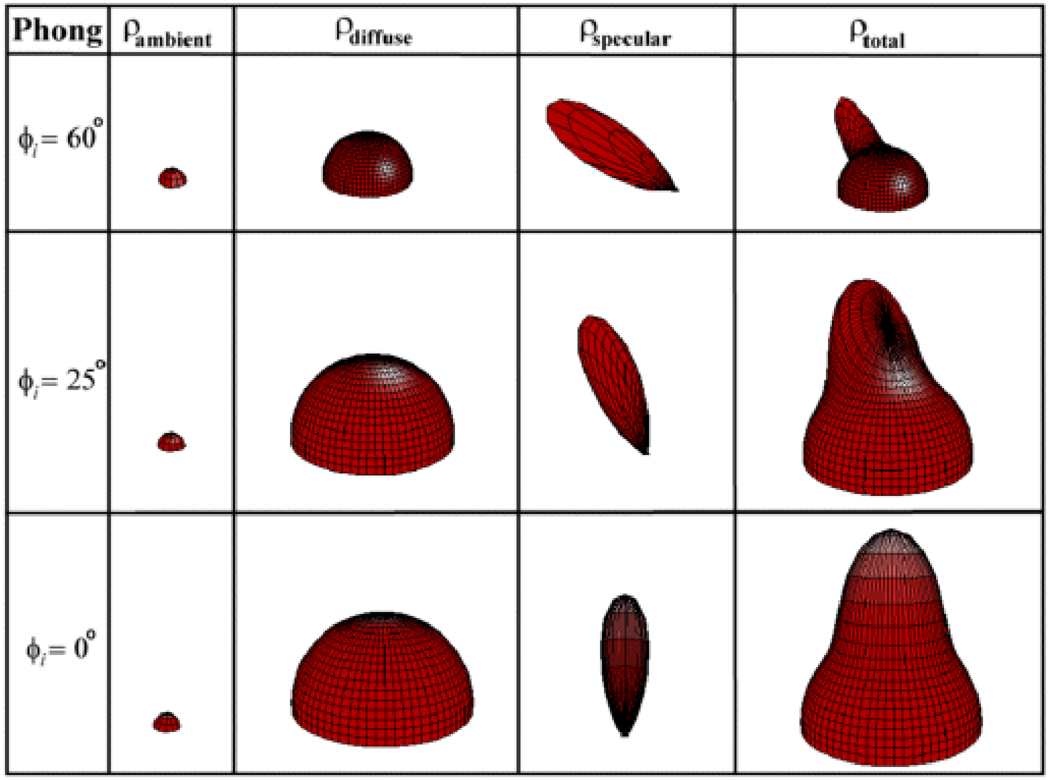
$$L_0 = k_a L_a + (k_d(\boldsymbol{n} \cdot \boldsymbol{l}) + k_s(\boldsymbol{v} \cdot \boldsymbol{r})^q)\frac{L_i}{r^2}$$



Putting it all together

- Blinn-Phong Illumination Model

$$L_0 = k_a L_a + (k_d(\boldsymbol{n} \cdot \boldsymbol{l}) + k_s(\boldsymbol{n} \cdot \boldsymbol{h})^q)\frac{L_i}{r^2}$$



Thank You!