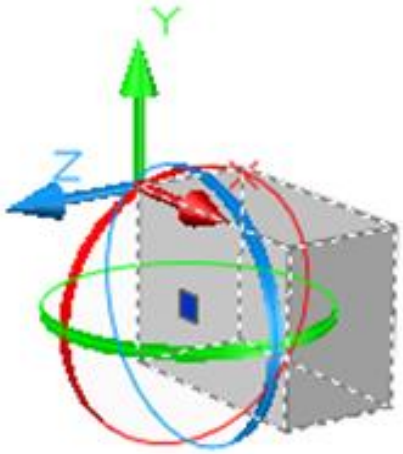
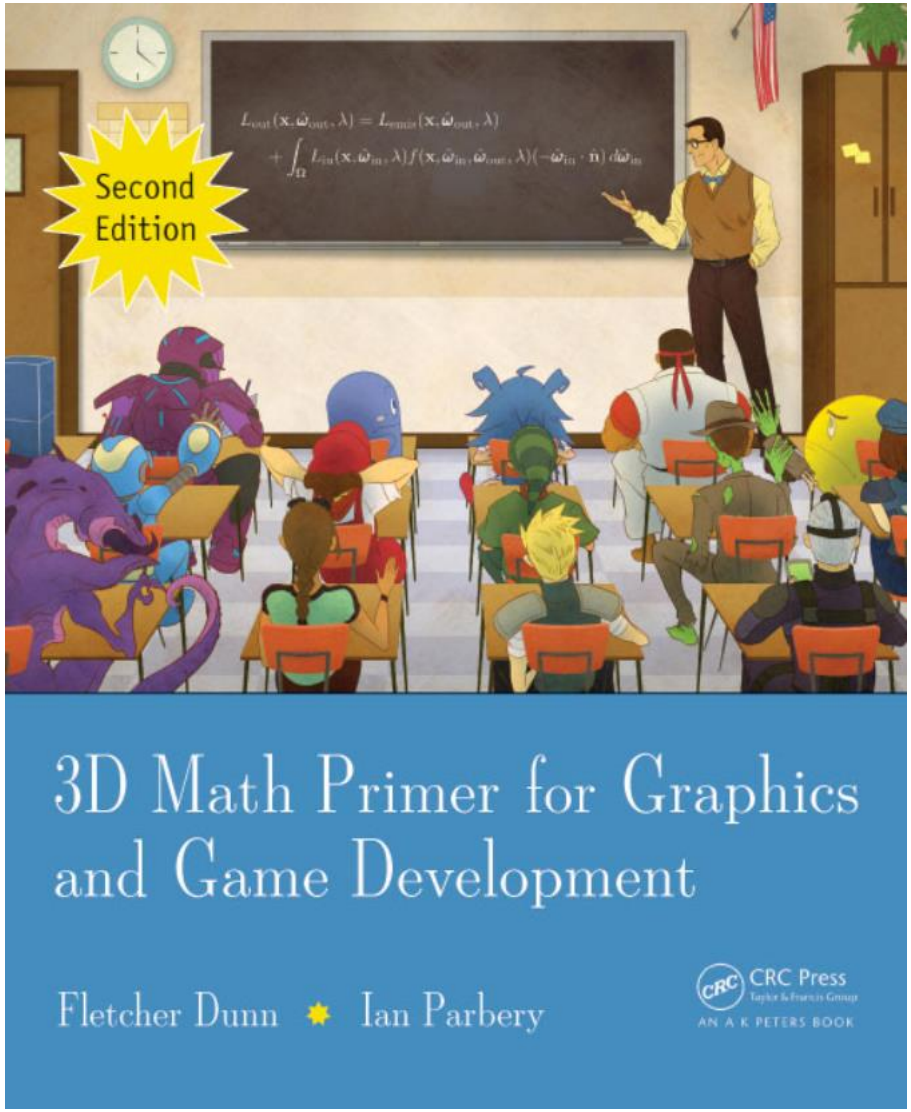
Computational geometry
IV –Rotation and Orientation



Stefan BORNHOFEN

3D Math Primer

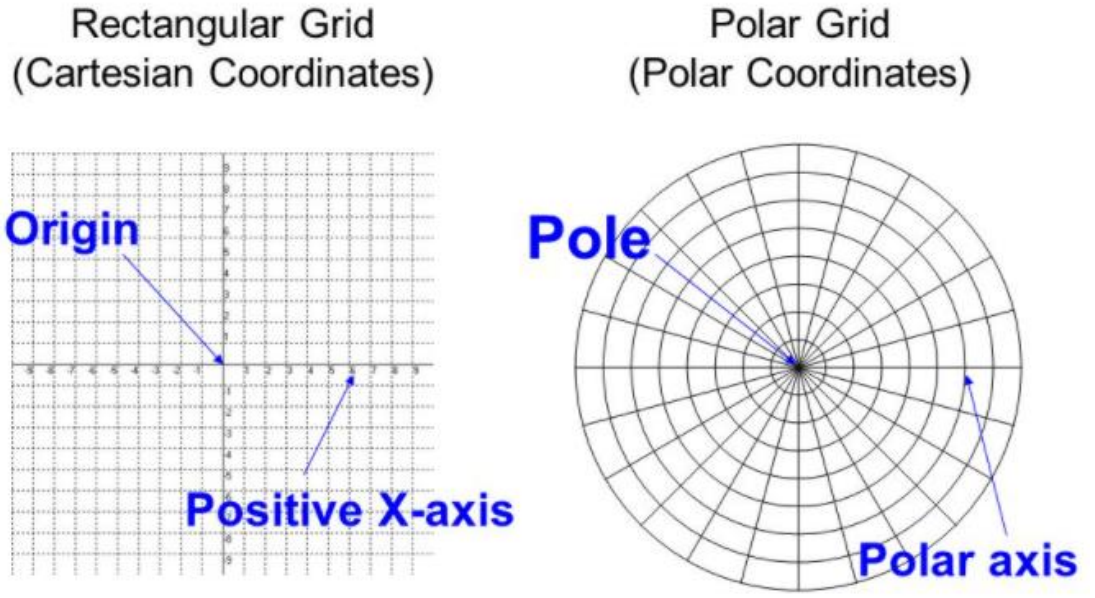
https://gamemath.com



Polar coordinate system

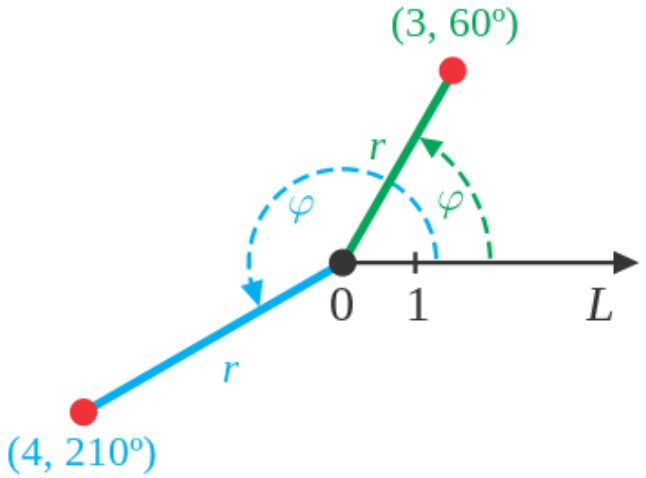
- A 2D coordinate system in which each point on a plane is determined by a distance from a reference point and an angle from a reference direction (“azimuth”).
- The reference point is called the **pole**, and the ray from the pole in the reference direction is the **polar axis**.
- Angles are expressed in either degrees or radians (360° = 2π rad).

In the polar system, it is easier to work with rotations and orientations than in the Cartesian system.



Polar coordinate system

- In the Cartesian system we describe a 2D point using the using two signed distances (*x*,*y)*.
- The polar coordinate pair (*r*, θ) species a point in 2D space as follows:
  1. Start at the origin, facing in the direction of the polar axis, and rotate by angle θ. Positive values of θ usually mean counterclockwise rotation,
  2. Move forward from the origin a distance of *r* units.



Aliasing

For any given point, there are infinitely many polar coordinate pairs that can be used to describe that point. This phenomenon is called **aliasing.** Aliasing does not happen in Cartesian space.

Two polar coordinate pairs are said to be *aliases* of each other if they have different numeric values but refer to the same point in space.

For any point (r, θ) other than the origin, all aliases of the polar coordinates (r, θ) can be expressed as:

$$\left((-1)^k\, r, \theta + k180^\circ\right)$$

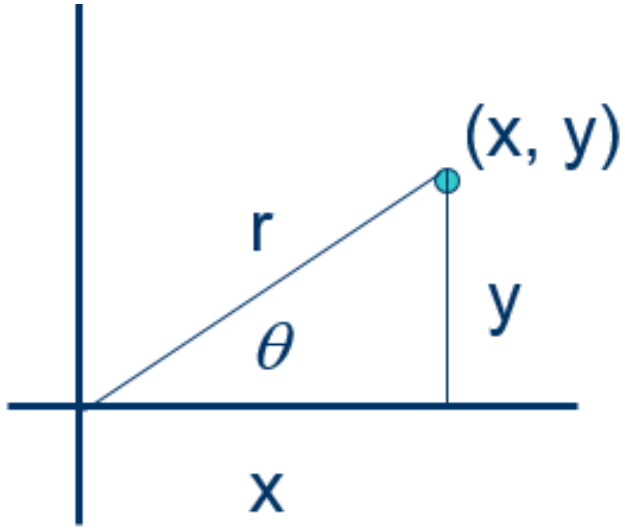
Canonical Polar Coordinates

A polar coordinate pair (r, θ) is in canonical form if

$r \geqslant 0$	We don't measure distances "backwards."
$-180^\circ < \theta \leqslant 180^\circ$	The angle is limited to 1/2 revolution, and use $+180^\circ$ for "West."
$r = 0 \quad \Rightarrow \quad \theta = 0$	At the origin, set the angle to zero.

Conversion Polar ⇔ Cartesian

Polar to Cartesian: (r, θ)=>(x,y)



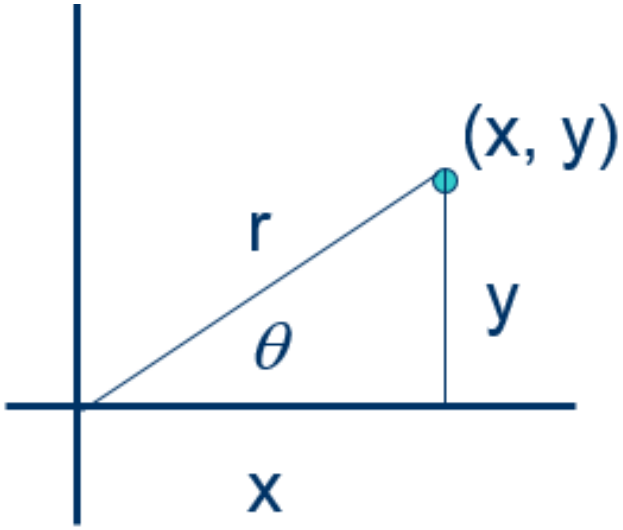
Cartesian to Polar: (x,y)=>(r, θ)

Conversion Polar ⇔ Cartesian

Polar to Cartesian: (r, θ)=>(x,y)

cos(θ) = x/r => x = r*cos(θ)

sin(θ) = y/r => y = r*sin(θ)



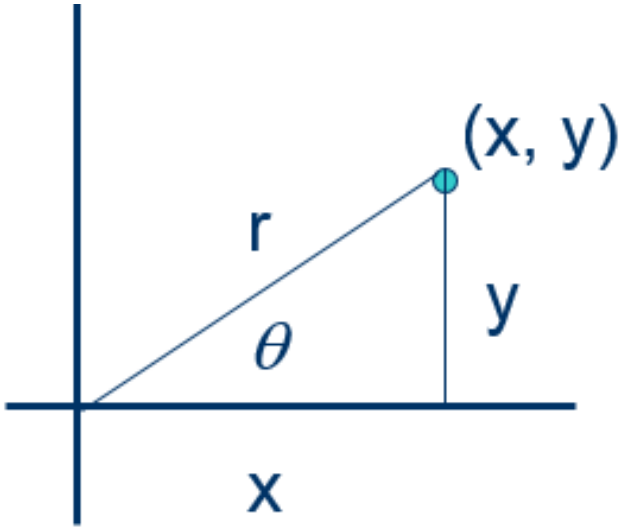
Cartesian to Polar: (x,y)=>(r, θ)

Conversion Polar ⇔ Cartesian

Polar to Cartesian: (r, θ)=>(x,y)

$\cos(\theta) = x/r \quad => \quad x = r*\cos(\theta)$

$\sin(\theta) = y/r \quad => \quad y = r*\sin(\theta)$



Cartesian to Polar: (x,y)=>(r, θ)

$r^2 = x^2 + y^2 \quad => \quad r = \text{sqrt}(x^2+y^2)$

$\tan(\theta) = y/x \quad => \quad \theta = \text{atan}(y/x)$

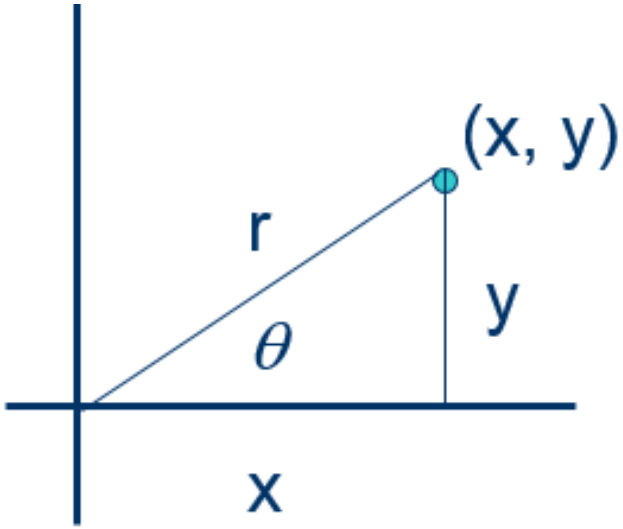
Conversion Polar ⇔ Cartesian

Polar to Cartesian: (r, θ)=>(x,y)

cos(θ) = x/r => x = r cos(θ)

sin(θ) = y/r => y = r sin(θ)

- What if x=0?
- When y/x >0, is there x>0 and y>0, or x<0 and y<0 ?
- atan is confined to [-90°,+90°]



Cartesian to Polar: (x,y)=>(r, θ)

$r^2 = x^2 + y^2$ => $r = sqrt(x^2+y^2)$

tan(θ) = y/x => θ = atan(y/x)

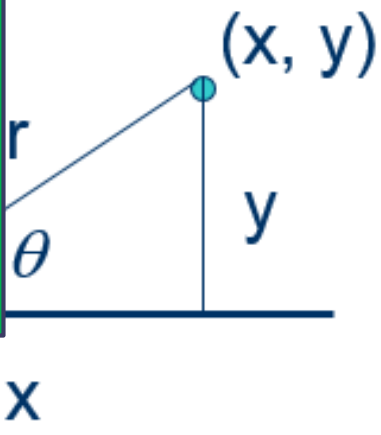
Conversion Polar ⇔ Cartesian

Polar to

cos(θ) =

sin(θ) = y

$$\operatorname{atan2}(y,x) = \begin{cases} 0, & x=0, y=0, \\ +90^{\circ}, & x=0, y>0, \\ -90^{\circ}, & x=0, y<0, \\ \arctan(y/x), & x>0, \\ \arctan(y/x)+180^{\circ}, & x<0, y\geq 0, \\ \arctan(y/x)-180^{\circ}, & x<0, y<0. \end{cases}$$



Cartesian to Polar: (x,y)=>(r, θ)

$r^2 = x^2 + y^2$ => r = sqrt(x^2+y^2)

tan(θ) = y/x => θ = `atan2(y,x)`

3D Polar Coordinate System

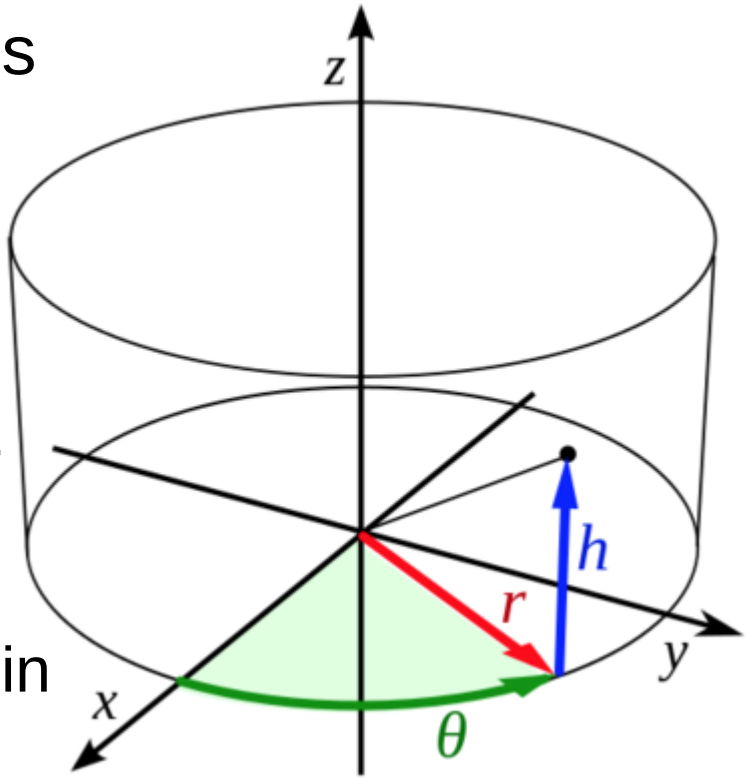
The 3rd dimension adds a 3rd coordinate.

It can be:

- another distance (like r): **cylindrical coordinates**
- another angle (like θ): **spherical coordinates**

Cylindrical coordinate system

A coordinate pair (r, θ, z) denoting
r: the axial distance from the z-axis
θ: the azimuth angle
z: the height over the xy-plane



Cylindrical coordinates are useful for objects and phenomena that have some rotational symmetry about the longitudinal axis, such as water flow in a pipe, or electromagnetic fields produced by in straight wire.

Spherial coordinate system

A coordinate pair (r, θ, φ) denoting

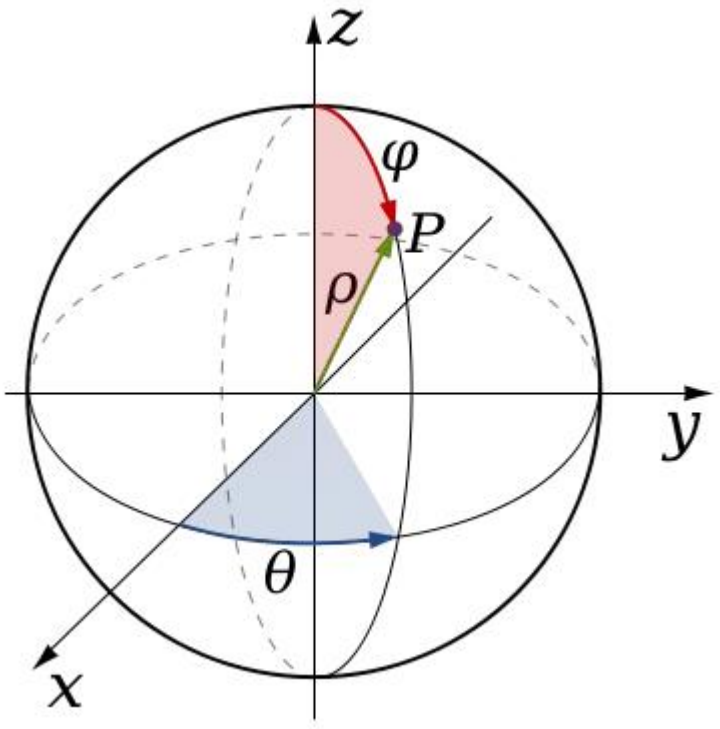
r: the radial distance

θ: the azimuth angle

φ: the zenith angle

r is sometimes written ρ.

The angles θ and φ are sometimes interchanged (physics standard).



Spherial coordinate system

A spherical coordinate pair (r, θ, φ) is in canonical form if

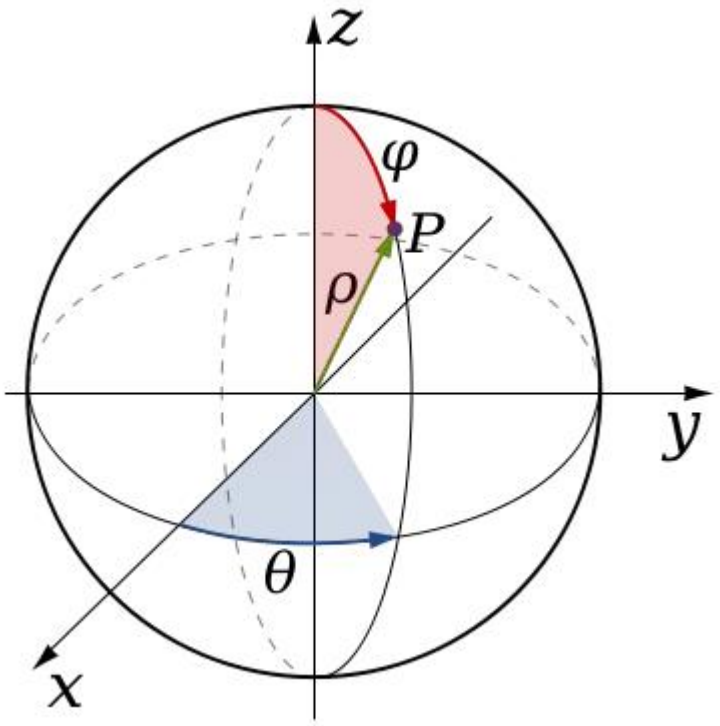
$r \geq 0$

$-180° < \theta \leq 180°$

$0° \leq \varphi \leq 180°$

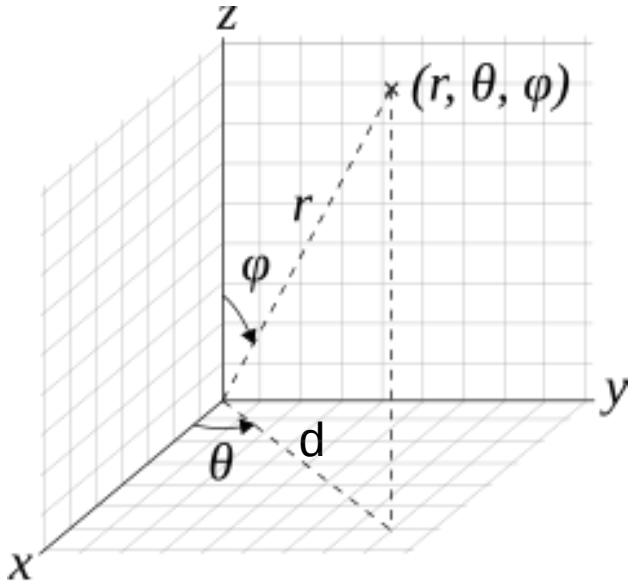
$r = 0 \Rightarrow \theta = \varphi = 0°$

$\varphi = 0° \Rightarrow \theta = 0°$



Conversion Spherical ⇔ Cartesian

Spherical to Cartesian (r, θ, φ) => (x,y,z)



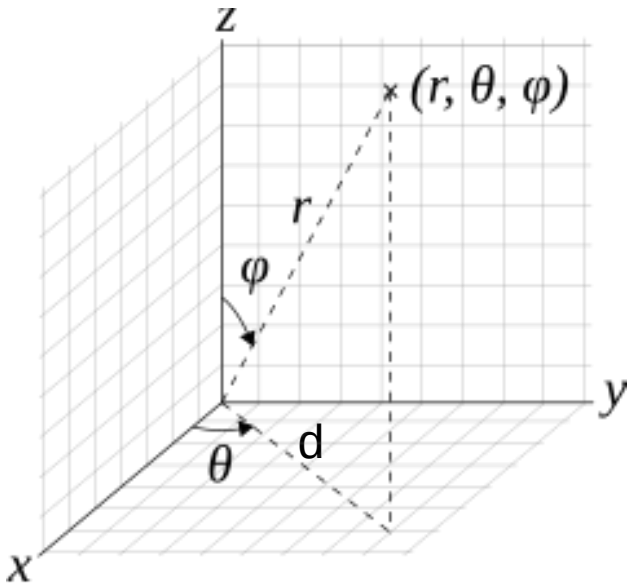
Conversion Spherical ⇔ Cartesian

Spherical to Cartesian (r, θ, φ) => (x,y,z)

cos(θ) = x/d
sin(θ) = y/d
cos(φ) = z/r
sin(φ) = d/r

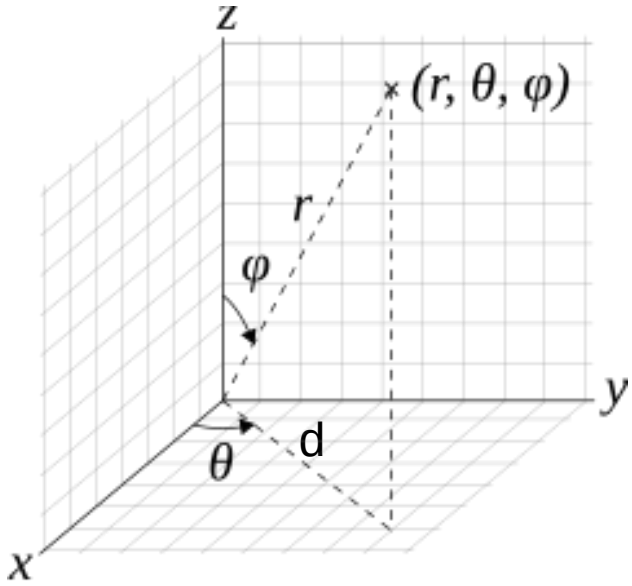
=>

x= r*sin(φ)*cos(θ)
y= r*sin(φ)*sin(θ)
z= r*cos(φ)



Conversion Spherical ⇔ Cartesian

Cartesian to Spherical (x,y,z) => (r, θ, φ)



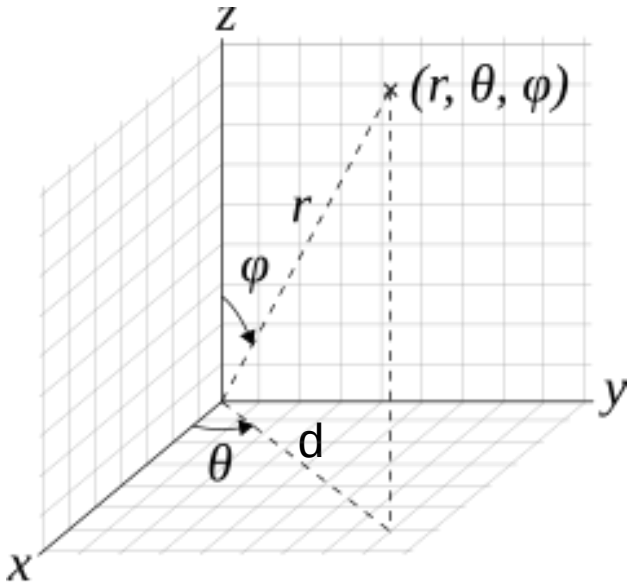
Conversion Spherical ⇔ Cartesian

Cartesian to Spherical (x,y,z) => (r, θ, φ)

$r = \sqrt{x^2+y^2+z^2}$

$\tan(\theta) = y/x$
$\Rightarrow \theta = \operatorname{atan2}(y,x)$

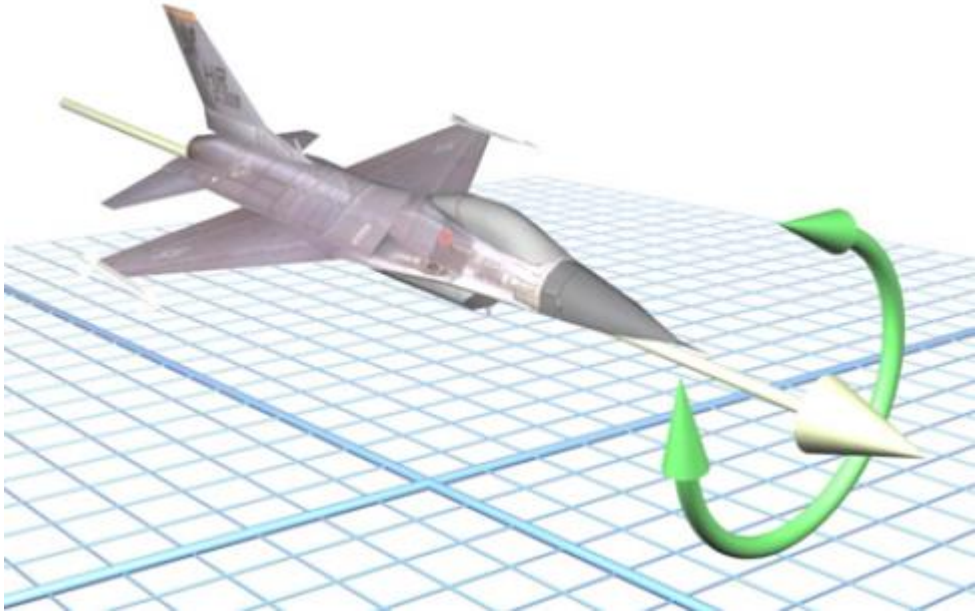
$\tan(\varphi) = d/z = \sqrt{x^2+y^2}/z$
$\Rightarrow \varphi = \operatorname{atan2}(\sqrt{x^2+y^2},z)$



Direction vs. Orientation

A direction can be specified using 2 values
(cf. polar coordinates: (θ, φ) without r).

An orientation is a direction with a twist.
Specifying an orientation requires at least 3 values.



Rotation vs. Orientation

Same distinction as points and vectors:

- An orientation is a state in space
- A rotation is a motion, leading from one orientation to another.

Several representations exist for encoding rotation. We will look at:

- Rotation matrix
- Axis-Angle and Exponential Map
- Euler angles
- Quaternion

Rotation matrix

The brute force method of expressing rotations.

Rotations are a specific kind of linear transformation and can be modeled by a 3x3 matrix.

Rotation matrices are « orthonormal », i.e.

1) The basis vectors are mapped to orthogonal vectors
2) The determinant is 1

In other words, any right-handed basis is always mapped to another right-handed basis.

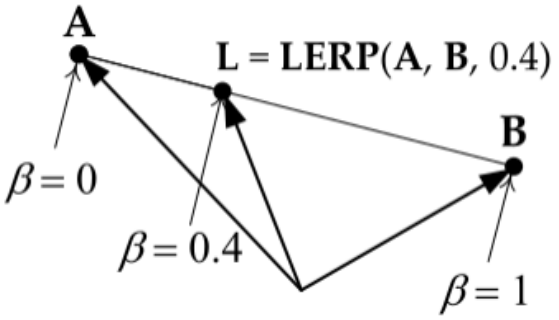
The inverse of a rotation matrix is equal of its transpose.

Rotation matrix

$$\begin{bmatrix} -0{,}1203 & -0{,}9429 & 0{,}3106 \\ 0{,}779 & -0{,}2835 & -0{,}5592 \\ 0{,}6153 & 0{,}1747 & 0{,}7687 \end{bmatrix}$$

Advantages

- Immediately applicable to vectors
- Used by all 3D API
- Easy to combine (matrix multiplication)
- Easy to undo (inverse matrix)



Drawbacks

- Difficult to interpret
- Memory footprint: 9 values
- Risk of malformed matrices
- Difficult to interpolate (LERP does not work)

Euler angles

Three composed elemental rotations (about the axes of a coordinate system) are sufficient to reach any orientation.

The three elemental rotations may be

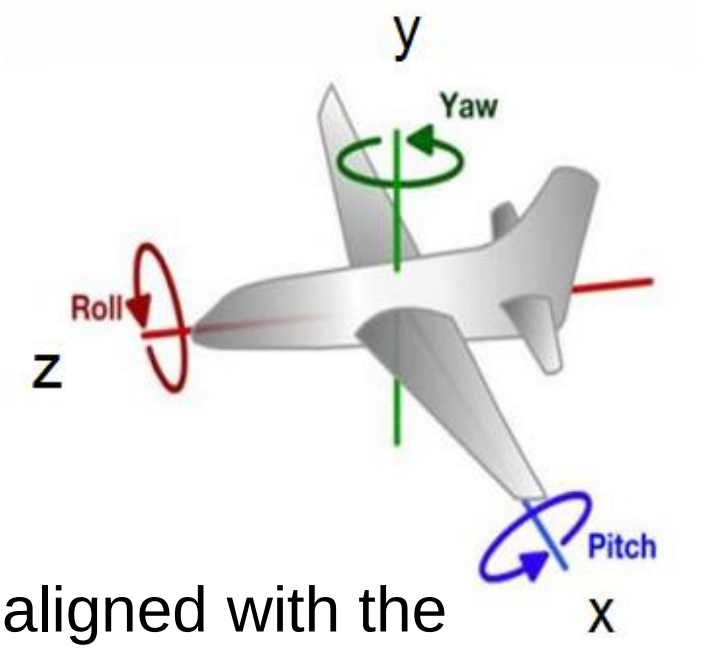
- **extrinsic** (rotations about the fixed axes XYZ of the original coordinate system),
- **intrinsic** (rotations about the rotating axes of the coordinate system XYZ, changing after each elemental rotation).

Moreover, there are twelve possible sequences of rotation axes, divided in two groups:

- Classic Euler angles (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y)
- Tait–Bryan angles (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z).

Euler angles

Our convention: **Yaw-Pitch-Roll** which is widely used in engineering.



Supposing that the object is initially aligned with the elementary axes and heading towards z, the sequence is:

- Intrinsic: Y -> X’ -> Z”
- Extrinsic: Z –> X -> Y

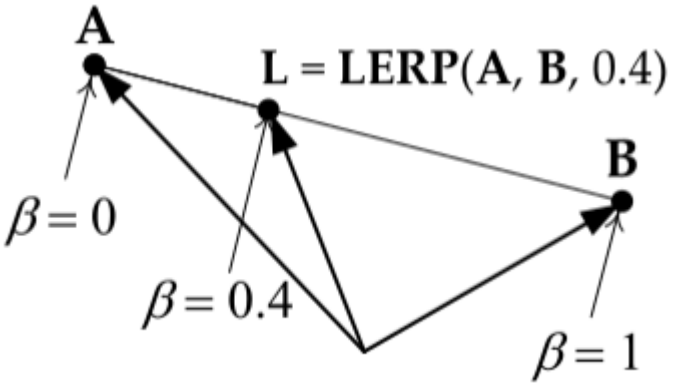
Euler angle rotations about moving axes written in reverse order are the same as the fixed axis rotations

Euler angles

Gimbal lock

When pitch=±90°, one degree of freedom is lost: yaw and roll both rotate about the same axis.

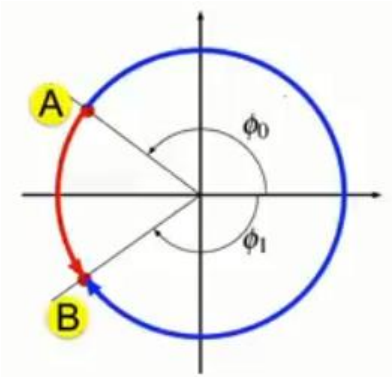
Euler angles



Interpolation

Stardard linear interpolation (LERP) applied to each Euler angle is very much possible, but often

- The path is not minimal



- Gimbal locks lead to wobbly and unnatural interpolation paths

Euler angles

Advantages

- Easy to understand
- Minimal memory footprint: 3 numbers
- No malformation: any 3 numbers are valid

Drawbacks

- No unique representation (aliasing)
- Difficult to combine
- LERP is problematic
- Gimbal lock

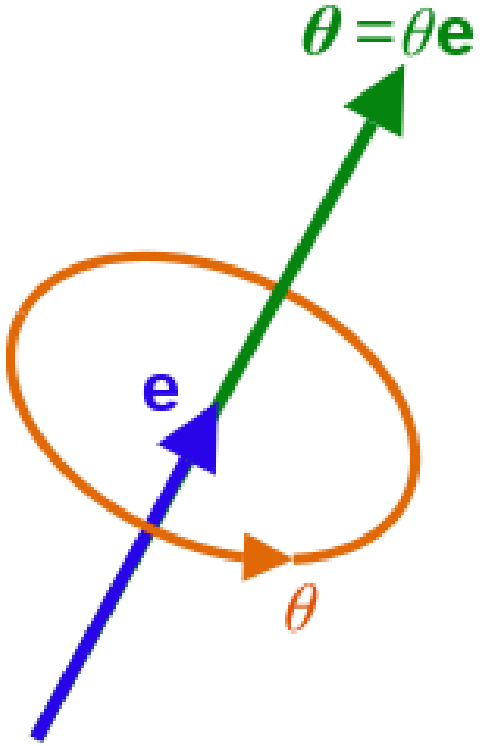
Axis-Angle and Exponential Map

Any rotation or sequence of rotations in 3D space is equivalent to a pure rotation about a single fixed axis through the origin (Euler's rotation theorem).

The axis–angle representation of a rotation parameterizes a 3D rotation by two quantities:

- a unit vector e indicating the direction of an axis of rotation
- an angle θ describing the magnitude of the rotation about the axis.

Since e has unit length, without loss of information, we can multiply it by θ, yielding the single vector θe encoding a 3D rotation. This representation is called "exponential map".



Axis-Angle and Exponential Map

Advantages

- Small memory footprint: 4 values (axis-angle) or 3 values (exponential map)
- No malformation
- Easy to understand
- Easy to undo

Drawbacks

- Difficult to combine
- No unique representation (aliasing)
- LERP not at constant angular velocity

Quaternions

Invented by Sir William Rowan Hamilton in 1843 when he was searching for a division algebra over R which is greater than C.

$$q_0 + q_1 i + q_2 j + q_3 k$$

$$i^2 = j^2 = k^2 = -1$$

$$ij = k,\ ji = -k$$

$$jk = i,\ kj = -i$$

$$ki = j,\ ik = -j\ .$$

- Quaternions form a 4-dimensional vector space over the real numbers, with {1,i,j,k} as a basis.
- Quaternions do not form a field because multiplication is not commutative.

Quaternions

Addition

$$(q_0 + q_1 i + q_2 j + q_3 k) + (p_0 + p_1 i + p_2 j + p_3 k)$$
$$= (q_0 + p_0) + (q_1 + p_1)i + (q_2 + p_2)j + (q_3 + p_3)k$$

Multiplication

q1q2≠ q2q1

$$(q_0 + q_1 i + q_2 j + q_3 k)\,(p_0 + p_1 i + p_2 j + p_3 k)$$
$$= (q_0 p_0 + q_1 p_1 i^2 + q_2 p_2 j^2 + q_3 p_3 k^2) +$$
$$(q_0 p_1 i + q_1 p_0 i + q_2 p_3 jk + q_3 p_2 kj) +$$
$$(q_0 p_2 j + q_2 p_0 j + q_1 p_3 ik + q_3 p_1 ki) +$$
$$(q_0 p_3 k + q_3 p_0 k + q_1 p_2 ij + q_2 p_1 ji)$$
$$= (q_0 p_0 - q_1 p_1 - q_2 p_2 - q_3 p_3) +$$
$$(q_0 p_1 + q_1 p_0 + q_2 p_3 - q_3 p_2)i +$$
$$(q_0 p_2 + q_2 p_0 - q_1 p_3 + q_3 p_1)j +$$
$$(q_0 p_3 + q_3 p_0 + q_1 p_2 - q_2 p_1)k.$$

Dot product

$$\mathbf{p} \cdot \mathbf{q} = p_0 q_0 + p_1 q_1 + p_2 q_2 + p_3 q_3 = \left|\mathbf{p}\right|\left|\mathbf{q}\right| \cos\varphi$$

Quaternions

Conjugate $$q^* = q_0 - q_1 i - q_2 j - q_3 k$$

Norm $$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}\,.$$

Inverse

$$\begin{aligned}
q\,q^* &= (q_0 + q_1 i + q_2 j + q_3 k)(q_0 - q_1 i - q_2 j - q_3 k) \\
&= (q_0 q_0 - q_1 q_1 i^2 - q_2 q_2 j^2 - q_3 q_3 k^2) + \\
&\quad (-q_0 q_1 + q_1 q_0 - q_2 q_3 + q_3 q_2) i + \\
&\quad (-q_0 q_2 + q_2 q_0 + q_1 q_3 - q_3 q_1) j + \\
&\quad (-q_0 q_3 + q_3 q_0 - q_1 q_2 + q_2 q_1) k \\
&= q_0^2 + q_1^2 + q_2^2 + q_3^2\,. \\
q^{-1} &= \frac{q^*}{|q|^2}\,.
\end{aligned}$$

Quaternions

Quaternions can be used for 3D rotations (Ken Shoemake, 1985).

Quaternions don't have Gimbal lock. They allow smooth interpolation of orientations.

To represent rotations, we only need unit length quaternions which lie on the surface of a 4D hypersphere of radius 1.

A unit quaternion q = (w,x,y,z) represents a rotation by angle θ around a unit vector a:

$$\mathbf{q} = \left[\cos\frac{\theta}{2} \quad a_x \sin\frac{\theta}{2} \quad a_y \sin\frac{\theta}{2} \quad a_z \sin\frac{\theta}{2} \right] \quad or \quad \mathbf{q} = \left\langle \cos\frac{\theta}{2}, \mathbf{a}\sin\frac{\theta}{2} \right\rangle$$

Quaternions

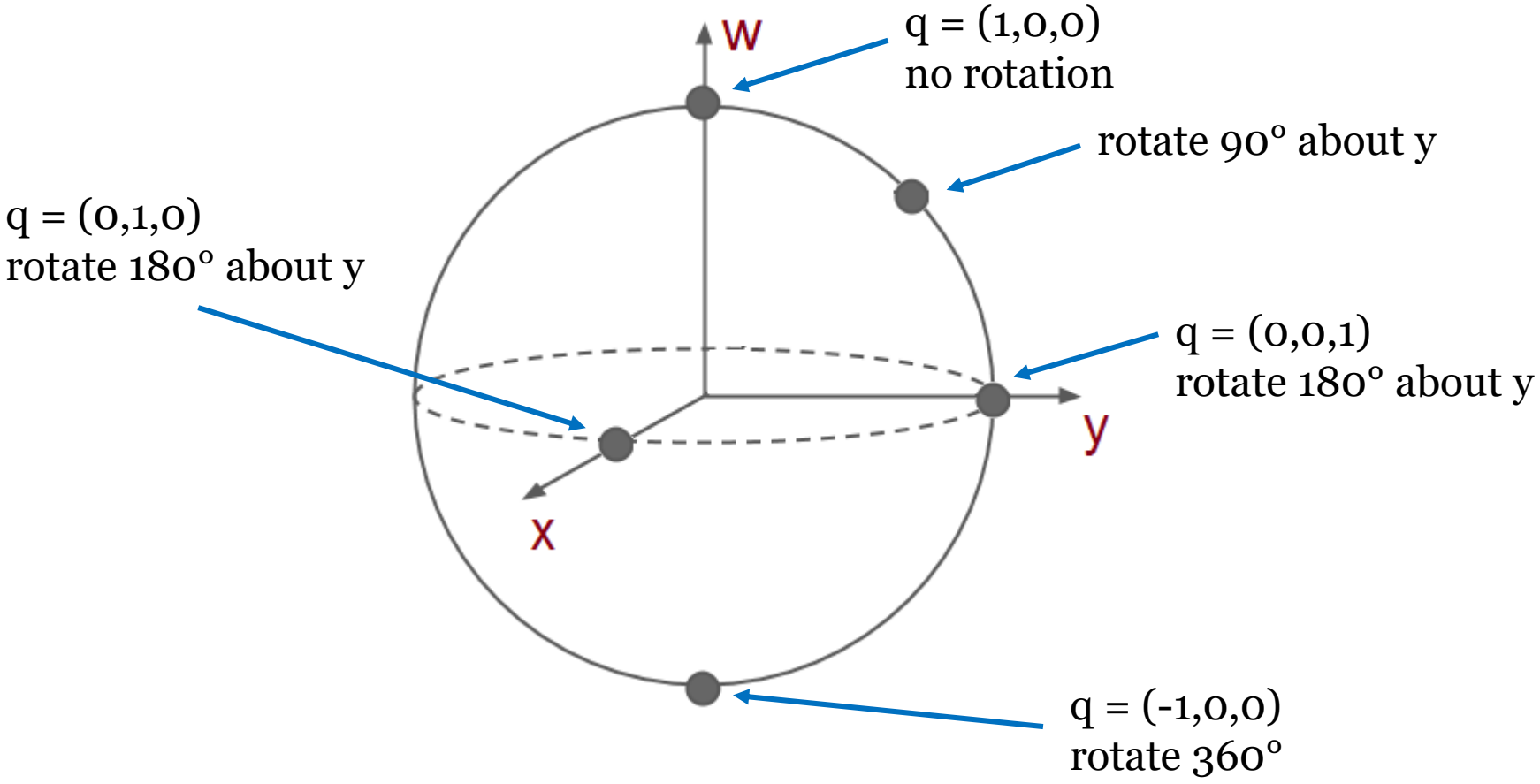
How to rotate a vector v with a quaternion?

Let $q = \cos(\theta/2) + \sin(\theta/2)\mathbf{u}$ be a unit quaternion
$|q| = |\mathbf{u}| = 1$

1) Convert the vector to a *pure quaternion* (no real part)
   v = (x,y,z) => p = 0 + xi + yj + zk
2) Compute p' = q p q^{-1} = q p q*, which will result in a quaternion which also has no real part
   => p' = 0 + x'i + y'j + z'k
3) The operation rotates the point p about axis u by angle θ
4) Convert p' back to a vector v'=(x',y',z')

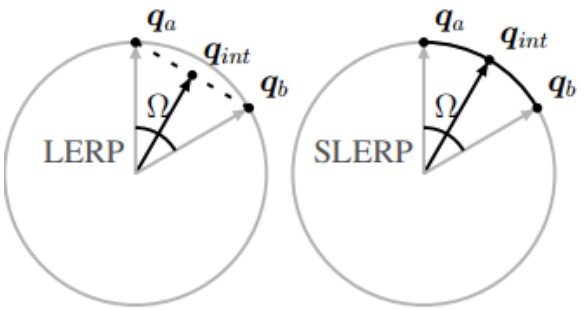
Quaternions

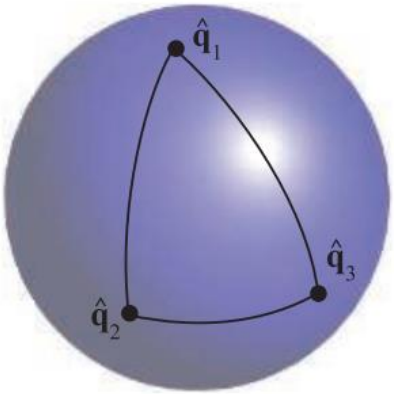
An attempt to visualize the rotation of a unit quaternion.
Let us consider only 2 rotation axes: q = (w, x, y)



Quaternions

Interpolation





Spherical linear interpolation (SLERP) always yields the shortest path, a single rotation around a fixed axis with uniform angular velocity.

The SLERP between two unit quaternions a and b is:

$$Slerp(t,\mathbf{a},\mathbf{b}) = \frac{\sin\left((1-t)\theta\right)}{\sin\theta}\mathbf{a} + \frac{\sin\left(t\theta\right)}{\sin\theta}\mathbf{b}$$

$$where: \theta = \cos^{-1}\left(\mathbf{a}\cdot\mathbf{b}\right)$$

Quaternions

Advantages

- Small memory footprint: 4 numbers
- Easy to combine and to undo
- No Gimbal Lock
- Interpolation is optimal (SLERP)

Drawbacks

- Difficult to understand
- Risk of malformed quaternions (but they can be projected on the unit sphere)

Exercise

In your 3D renderer, display the object orientation in the following forms

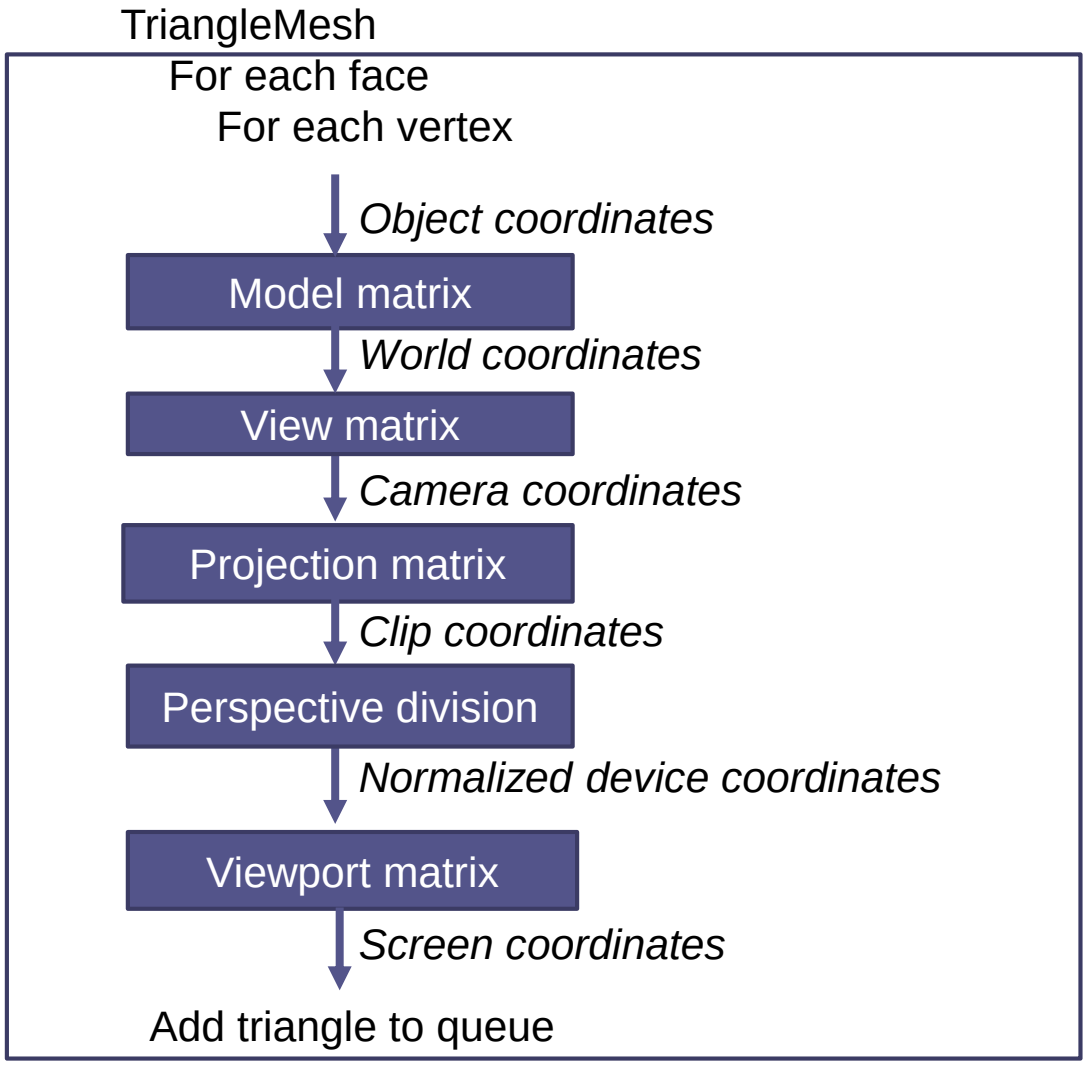
- Rotation matrix
- Euler angles (yaw-pitch-roll)
- Exponential map
- Quaternions

Implement animations such that the object can anytime rotate back to its identity orientation using

- Euler angles LERP
- Exponential map LERP
- Quaternion SLERP

You need to find efficient conversions between these representations. Hint: take a look at the 3D Math Primer.

A more complex rendering pipeline

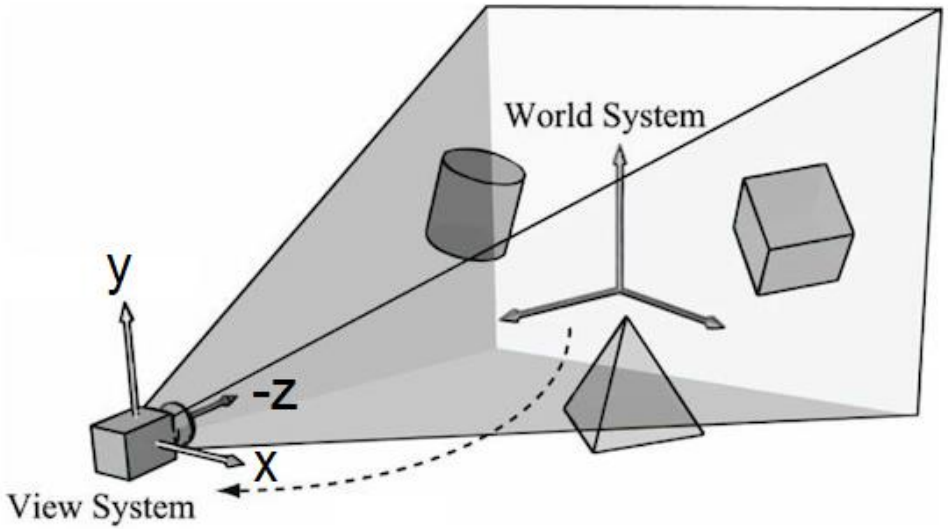


Render queue in wireframe mode

View matrix

Corresponds to placing and orienting the camera.

The viewing transformation transforms the world coordinates into the camera coordinate system of a camera that is placed at the origin of the coordinate system, points to the negative z axis and is put on the xz plane, i.e. the up-direction is given by the positive y axis.



This step rotates the entire world towards the camera, which is always looking at a fixed position from the origin.

View matrix

We can construct a transformation matrix C from the position t of the camera, the view direction d, and a world-up vector k (all in world coordinates) using the following algorithm.

1. Compute (in world coordinates) the direction z of the z axis of the view coordinate system as the negative normalized d vector: z = -d / |d|
2. Compute (in world coordinates) the direction x of the x axis of the view coordinate system by: x = d x k / |d x k|
3. Compute (in world coordinates) the direction y of the y axis of the view coordinate system: y = z x x

$$C = \begin{bmatrix} x_1 & y_1 & z_1 & t_1 \\ x_2 & y_2 & z_2 & t_2 \\ x_3 & y_3 & z_3 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

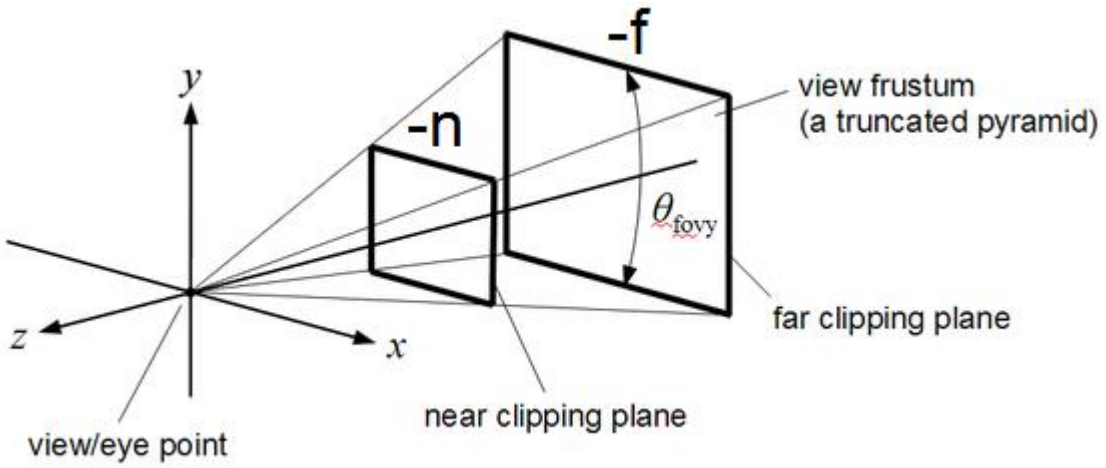
The view matrix is the inverse of the camera matrix C.

$$\mathbf{M}_{\mathbf{world} \to \mathbf{view}} = C^{-1}$$

Projection matrix

The standard **perspective projection** is characterized by

- an angle θ that specifies the field of view in y direction
- the distance n to the near clipping plane
- the distance f to the far clipping plane
- the aspect ratio a of the width to the height of a centered rectangle on the near clipping plane.



$$\mathrm{M}_{\text{projection}} = \begin{bmatrix} \frac{d}{a} & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{with } d = \frac{1}{\tan(\theta_{\text{fovy}}/2)}$$

The projection transformation maps view coordinates to clip coordinates, which are then mapped to normalized device coordinates by the **perspective division** (dividing by the fourth component of the clip coordinates).

Important classes

```
class Scene
 camera: mat4 // camera matrix
 view: mat4 // view matrix
 projection: mat4 // projection matrix
 viewport: mat4 // viewport matrix

class TriangleMesh
  vertices: List<vec3> // a set of vertex coordinates
  // a set of faces, i.e. three vertex indices
  // forming the corners of a triangle
  // (counter-clockwise order):
  faces: List<int[3]>
  model: mat4 // model matrix

class ScreenTriangle
v0, v1, v2: vec3
```

Exercise

Add the following cool features to your renderer.

1) Toggle between orthographic and perspective projection

2) Move and rotate the camera

3) Plot the reference world axes to better understand the movements of the object and of the camera