Geometric Transformations

Computer Graphics and Visualization

Pedher Johansson
Department of Computer Science
Fall 2016

Scalars

- Ordinary real numbers and their operations are an example of a Scalar field
- Scalars have two fundamental operations between pairs, addition and multiplication
- They are associative, commutative and distributive.
- Each element α has an additative inverse $-\alpha$ and a multiplicative inverse α^{-1}

Vector space

- A vector space contains scalars and vectors
- Vectors have two operations
  - vector-vector addition

$$v = u + w$$

  - scalar-vector multiplication

$$v = \alpha u$$

- Every vector v has an additative inverse $-v$

Vector

- A vector is a direction and magnitude in space.
- It has no location.

Affine space

- In affine space Points are added
  - A point is a location in space.
  - It has neither size or shape.

point-vector addition A new point is formed by adding a vector to a point.

$$P = Q + v$$

point-point subtraction A subtraction of one point from an other, forms a vector.

$$v = P - Q$$

zero-vector A vector of no magnitude, thus an undefined direction.

Lines and Rays in Affine Space

$$P(\alpha) = P_0 + \alpha d$$

- Parametric form
  - We generate points on the line by varying the parameter α
- A *line* is infinite in both directions

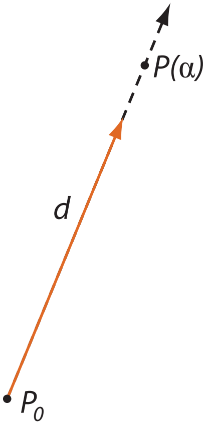
$$-\infty \leq \alpha \leq \infty$$

- A *ray* is infinite in one direction

$$\alpha \geq 0$$

- A *line segment* is finite

$$a \geq \alpha \geq b$$



Affine Sum

- No point-point addition
- No point-scalar multiplication

However!

$$
\begin{aligned}
&P = Q + \alpha v \\
&v = R - Q \\
&P = Q + \alpha(R - Q) = \alpha R + (1 - \alpha) Q
\end{aligned}
$$

- A point can be expressed as combination of two points and it is located on the line connecting the two.

Planes

- Using affine sums a plane can be defined by
$$T(\alpha,\beta) = P_0 + \alpha u + \beta v$$
if u and v are nonparallel.
- The *normal vector* to a plane can be found using the cross product.
$$n = u \times v$$

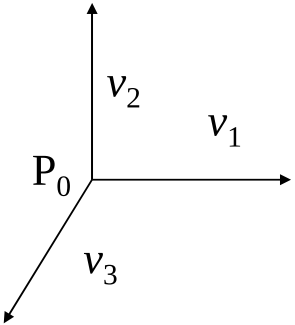
Coordinate Representation - Frames

- In affine space we have the ability to build *frames*
- Need a frame of reference to *relate* points and objects.
  - Object coordinate
  - World coordinates
  - Camera coordinates

Frames

In an affine space

a point in space together with the basis vectors can form a frame.



Euclidean Space

- No concept of distance or length in affine space.
- Euclidean space add the dot-product
  - Vector length $|v| = \sqrt{v \cdot v}$
  - Angles $u \cdot v = |u||v| \cos\theta$

- Affine space enough to define geometric models
- In a frame we use euclidean operations

Frames in Affine Space

- Frame determined by

$$(P_0, v_1, v_2, v_3)$$

- Within this frame, every vector can be written as

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$$

- Every point can be written as

$$P = P_0 + \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3$$

Going 4D

Add a dimension, w, so that the 3D affine space is a projection in a 4D space.
Benefits:

- Uniform representation of all transformations and projections.
- Efficient pipeline

Let us take a 2D example!

2D affine space in 3D

Take a 2D image

- Let it become a 2D projection in 3D space where $w = 1$ (affine space)
- All 2D points becomes a 3D line passing through origo
- All 2D lines becomes a plane also passing through origo.

Homogenous Coordinates

- All 3D points along the 3D line represents the same 2D point
- Changing w for the projection plane, does not effect the projected image.
  - All point will have the sema relative distance
  - we do not have a notion of legth, so it is not "bigger" or "smaller"
- The *homogenous coordinate* (or projective coordinate) of a 2D point $[x\ y]^T$ is $[wx\ wy\ w]^T$, where $[x\ y\ 1]^T$ is the point's normalized form.

Vanishing points

- So how about a 3D point $[2\ 3\ \varepsilon]^T$ if $\varepsilon \to 0$?
- In affine space this point will vanish into infinity.
- Points where $w = 0$ is caled *vanishing points*
- In the affine space they represent *points in infinity*
- Can also be seen as directions or *vectors*.

Frames in homogenous coordinates

- A frame can be represented as a point where $w = 1$ and a set of basis vectors where $w = 0$.
- We represent it as $[v_1 \ v_2 \ v_3 \ P_0]^T$
- Using that frame a point Q and vector u can then be represented as

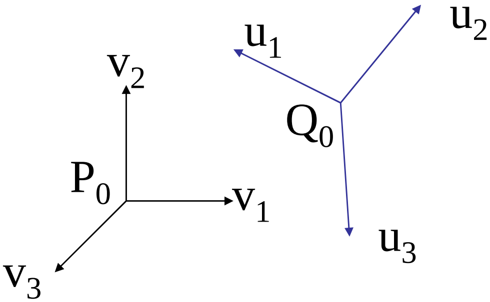
$$u = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$$
$$Q = P_0 + \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3$$

or for short

$$u = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 0]^T$$
$$Q = [\beta_1 \ \beta_2 \ \beta_3 \ 1]^T$$

Change of Frames

- Consider two frames: (v_1, v_2, v_3, P_0) and (u_1, u_2, u_3, Q_0)



- Any point or vector can be represented in either frame
- We can represent u_1, u_2, u_3, Q_0 in terms of v_1, v_2, v_3, P_0.

Representing One Frame in Terms of the Other

$$
\begin{aligned}
u_1 &= \gamma_{11} v_1 + \gamma_{21} v_2 + \gamma_{31} v_3 + 0 \cdot P_0 \\
u_2 &= \gamma_{12} v_1 + \gamma_{22} v_2 + \gamma_{32} v_3 + 0 \cdot P_0 \\
u_3 &= \gamma_{13} v_1 + \gamma_{23} v_2 + \gamma_{33} v_3 + 0 \cdot P_0 \\
Q_0 &= \gamma_{14} v_1 + \gamma_{24} v_2 + \gamma_{34} v_3 + 1 \cdot P_0
\end{aligned}
$$

defining a 4×4 matrix

$$
[u_1 \; u_2 \; u_3 \; Q_0] = [v_1 \; v_2 \; v_3 \; P_0] \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & \gamma_{14} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & \gamma_{24} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & \gamma_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Change of Coordinate Systems

- Within the two frames any point or vector has a representation of the same form
  - $\mathbf{a} = [\alpha_1\ \alpha_2\ \alpha_3\ w]^T$ in the first frame
  - $\mathbf{b} = [\beta_1\ \beta_2\ \beta_3\ w]^T$ in the second frame

  where $w = 1$ for points and $w = 0$ for vectors
- Hence

$$[v_1\ v_2\ v_3\ P_0]\mathbf{a}\ = [u_1\ u_2\ u_3\ Q_0]\mathbf{b}\ = [v_1\ v_2\ v_3\ P_0]\mathbf{Mb}$$

  or

$$\mathbf{a} = \mathbf{Mb} \Rightarrow \mathbf{M}^{-1}\mathbf{a} = \mathbf{b}$$

Affine Transformations

The matrix M is 4×4 and has 12 degrees of freedom (since 4 elements are fixed). It specifies all the **affine transformations** in *homogeneous coordinates*. Affine transformations is a subset of all linear transformations.

Affine transformations

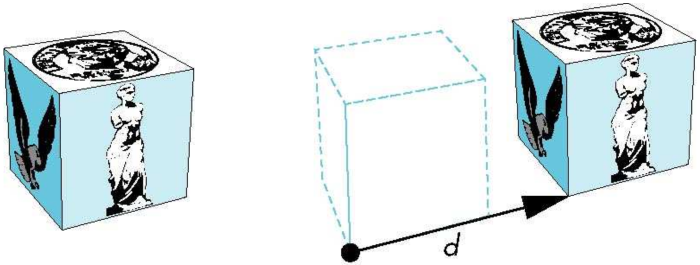
- Equivalent to a change in frames
- Preserves parallel lines
- Preserves ratios of vectors along a line

Translation

- Move (translate, displace) a point to a new location
- Displacement determined by a vector d
  - Three degrees of freedom

$$P\delta = P + d$$

- Although we can move a point to a new location in infinite ways, when we move many points there is usually only one way



Affine Translation

- A translation in positive direction of an object is equivalent with an translation in negative direction of the frame.
- Applying the *inverse* of **M** on an object is equivalent to applying **M** on the frame.

Affine Translation

- A translation is equivalent to changing frame, with
  - preserved base vectors
  - a moved origin.

$$
\begin{aligned}
u_1 &= 1 \cdot v_1 + 0 \cdot v_2 + 0 \cdot v_3 + 0 \cdot P_0 \\
u_2 &= 0 \cdot v_1 + 1 \cdot v_2 + 0 \cdot v_3 + 0 \cdot P_0 \\
u_3 &= 0 \cdot v_1 + 0 \cdot v_2 + 1 \cdot v_3 + 0 \cdot P_0 \\
Q_0 &= -d_x \cdot v_1 - d_y \cdot v_2 - d_z \cdot v_3 + 1 \cdot P_0
\end{aligned}
$$

$$
\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & -d_x \\ 0 & 1 & 0 & -d_y \\ 0 & 0 & 1 & -d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

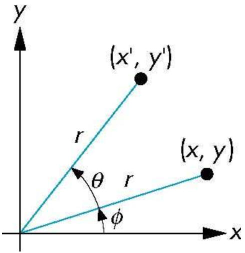
Affine Translation

- The matrix to apply to the object is

$$\mathbf{M}^{-1} = \mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation (2D)

- Consider rotation about the origin by θ degrees
  - radius stays the same, angle increases by θ

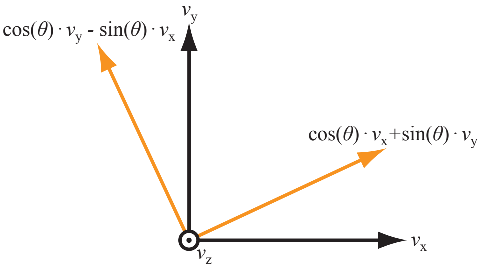


$$x = r \cdot \cos(\phi)$$
$$y = r \cdot \sin(\phi)$$

$$x' = r \cdot \cos(\phi + \theta) = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$
$$y' = r \cdot \sin(\phi + \theta) = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

Rotation about the *z*-axis

- A counterclockwise rotation around the z-axis is equivalent to changing frame, with
  - x and y vectors rotated clockwise around the z axis and
  - a preserved origin



Rotation about the z-axis

$$
\begin{aligned}
u_x &= \cos(-\theta) \cdot v_x + \sin(-\theta) \cdot v_y + 0 \cdot v_z + 0 \cdot P_0 \\
u_y &= -\sin(-\theta) \cdot v_x + \cos(-\theta) \cdot v_y + 0 \cdot v_z + 0 \cdot P_0 \\
u_z &= 0 \cdot v_x + 0 \cdot v_y + 1 \cdot v_z + 0 \cdot P_0 \\
Q_0 &= 0 \cdot v_z + 0 \cdot v_y + 0 \cdot v_z + 1 \cdot P_0
\end{aligned}
$$

$$
\mathbf{M} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Rotation about the z-axis

- The matrix to apply to the object is

$$\mathbf{M}^{-1} = \mathbf{R}_z = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about the x and y axis

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

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

Scaling

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Concatenation

- We can form arbitrary affine transformation matrices by multiplying rotation, translation, and scaling matrices
- Because the same transformation is applied to many vertices, the cost of forming a matrix $\mathbf{M} = \mathbf{DCBA}$ is not significant compared to the cost of computing $\mathbf{M}p$ for many vertices p

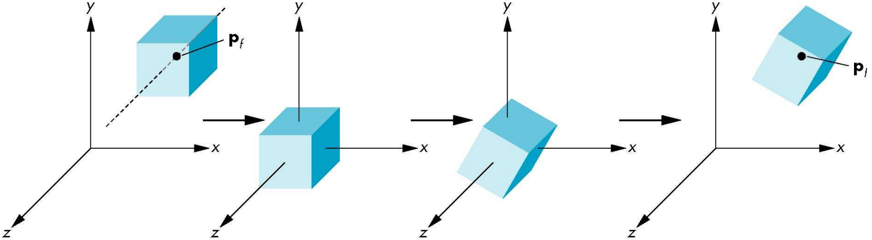
$$p' = \mathbf{CBA}p = \mathbf{M}p$$

- Note that the matrix on the right is the first applied
- The difficult part is how to form a desired transformation from the specifications in the application

Rotation About a Fixed Point other than the Origin

- Move fixed point to origin
- Rotate
- Move fixed point back

$$M = T(P_x, P_y, P_z) \cdot R(\theta) \cdot T(P_x, P_y, P_z)^{-1}$$



Why Homogenous Coordinates Again?

Why use 4D instead of 3D

- Gives a unique representation of both points and vectors.
- All affine transformations can be done with matrix multiplications.
  - Translations can not be done as a matrix multiplication using 3D coordinates
- Can be extended to perspective transformations

General Rotation to a New Frame

- Assume we have a vector, u_1 and $||u_1|| = 1$.
- If we want u_1 to become the x unit vector in a new frame we need a matrix $\mathbf{M}$ such that

$$\mathbf{M}u_1 = [1\ 0\ 0\ 0]^T$$

- This implies we also need two more vectors u_2 and u_3 that will become the new y and z unit vectors in a new frame.
- Then $u_1 \perp u_2 \perp u_3$ and $u_2 \perp u_3$, and $||u_1|| = ||u_2|| = ||u_3|| = 1$ must be fulfilled.
- With an unchanged origo, then also $\mathbf{M}[0\ 0\ 0\ 1]^T = [0\ 0\ 0\ 1]^T$.

General Rotation to a New Frame

Then

$$\mathbf{M}\begin{bmatrix} u_{1x} & u_{2x} & u_{3x} & 0 \\ u_{1y} & u_{2y} & u_{3y} & 0 \\ u_{1z} & u_{2z} & u_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I_4$$

Thus

$$\mathbf{M} = \begin{bmatrix} u_{1x} & u_{2x} & u_{3x} & 0 \\ u_{1y} & u_{2y} & u_{3y} & 0 \\ u_{1z} & u_{2z} & u_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \Rightarrow \mathbf{M} = \begin{bmatrix} u_{1x} & u_{2x} & u_{3x} & 0 \\ u_{1y} & u_{2y} & u_{3y} & 0 \\ u_{1z} & u_{2z} & u_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{T}$$

Because the matrix is *special orthogonal*.

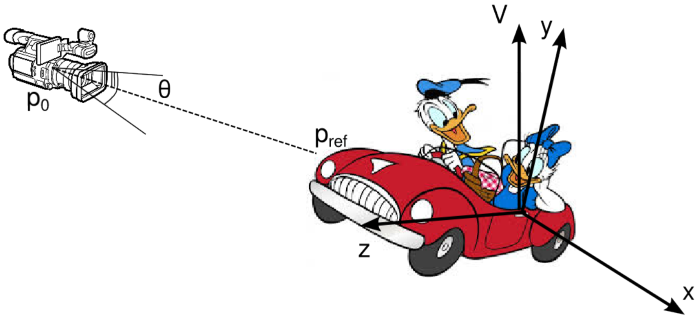
Camera Model

From point - p_0: The position of the camera, Center of projection, COP.

Look-at Point - p_{ref}: Where the camera is aimed.

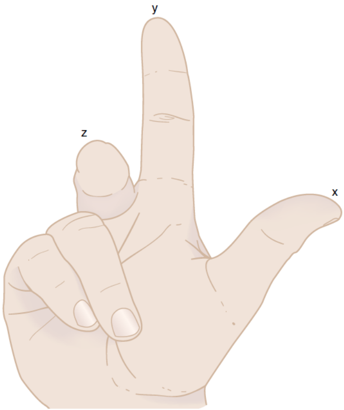
Up vector - V: Defines the up direction.

View angle - θ: Field of view.



Right-Handed Coordinate System

- In OpenGL all coordinate systems are Right-Handed



Viewing Transformation

1. Place an object in world coordinates.
   - Transform object coordinates $p^{(obj)} = (x^{(obj)}, y^{(obj)}, z^{(obj)}, w)$ to world coordinates $p^{(w)} = (x^{(w)}, y^{(w)}, z^{(w)}, w)$.
2. Place to camera in the world
   - Transform world coordinates $p^{(w)}$ to camera (eye) coordinates $p^{(c)}$.
   - p_0 ends up in the origin of the eye coordinate system. p_{ref} ends up on the *negative* z-axis. V vector ends up in the positive Y-Z plane.
3. Project eye coordinates $p^{(c)}$ to normal device coordinates $p^{(ndc)}$.
4. Orthogonal projection to the view port (handled by OpenGL).

1. Object to World Transformation

- Apply affine transformations to transform (translate, scale, rotate, shear, etc.) an object $w^{(obj)} = Tp^{(obj)}$
- Usually concatinated transformations $T = T_n ... T_1$
- In a scene these transformations are ordered in a hierarchy (part of a scene graph)

2. World to Camera Transformation

- Two transformations
  - A translation
    (to move the camera to origin)
  - A rotation of the basis axes to align them with the camera

2. Translation

- Move the world frame so that the camera is at origin.
- Apply $T(-p_0)$ to the object
  (Correspons to apply $T(p_0)$ to the frame)

2. Create rotation matrix

-

The camera's Z-axis in world cordinates

$$n = \frac{p_0 - p_{\mathrm{ref}}}{||p_0 - p_{\mathrm{ref}}||}$$

Z-axis should be mapped to $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ in eye coordinates.

2. Create rotation matrix

Vector perpendicular to $p_0 - p_{\mathrm{ref}}$ and V

$$u = \frac{V \times (p_0 - p_{\mathrm{ref}})}{||V \times (p_0 - p_{\mathrm{ref}})||} = \frac{V \times n}{||V \times n||}$$

X-axis should be mapped to $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ in eye coordinates.

2. Create rotation matrix

Vector perpendicular to n and u

$$v = n \times u$$

Y-axis should be mapped to $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ in eye coordinates.

2. Create rotation matrix

Combining all tree conditions

$$\mathbf{M}_{wc} \begin{bmatrix} u & v & n \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{M}_{wc}$ is orthonormal, so

$$\mathbf{M}_{wc}^T = \mathbf{M}_{wc}^{-1} = \mathbf{M}_{cw}$$

2. In Homogenous Coordinates

$$\mathbf{M}_{wc}^{-1} = \begin{bmatrix} \vdots & \vdots & \vdots & 0 \\ u & v & n & 0 \\ \vdots & \vdots & \vdots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{M}_{wc}^{-1}$ is special orthogonal, so $\mathbf{M}_{wc}^{-1} = \mathbf{M}_{wc}^{T} = \mathbf{M}_{cw}$

2. The View Matrix

- The translation $\mathbf{T}(-p_0)$ and
- rotation $\mathbf{M}_{wc}$
- give the View Matrix $\mathbf{V}$ (Not the Up-vector)

$$\mathbf{V}(p_0, p_{\text{ref}}, V) = \mathbf{M}_{wc}\mathbf{T}(-p_0)$$

General Properties

$$\mathbf{M}_{wc}\mathbf{T}(-p_0)p^{(w)} = \mathbf{V}p^{(w)} = p^{(c)}$$

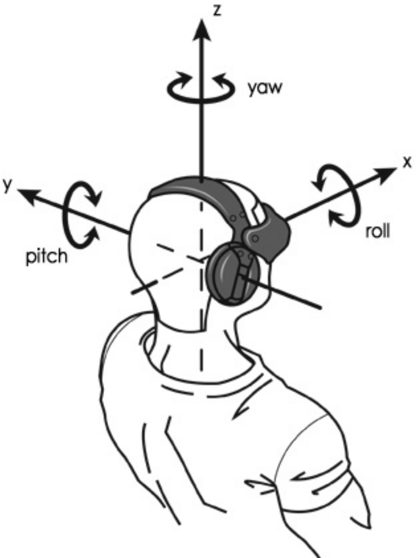
$$\mathbf{V}^{-1} = (\mathbf{M}_{wc}\mathbf{T}(-p_0))^{-1} = \mathbf{T}(p_0)\mathbf{M}_{wc}^T = \mathbf{T}(p_0)\mathbf{M}_{cw}$$

$$p^{(w)} = \mathbf{V}^{-1}p^{(c)} = \mathbf{T}(p_0)\mathbf{M}_{cw}p^{(c)}$$

Same properties for vectors!

$$u^{(w)} = \mathbf{V}^{-1}u^{(c)} = \mathbf{T}(p_0)\mathbf{M}_{cw}u^{(c)}$$

Yaw-Pitch-Roll



Excersices

- How to change p_0, p_{ref}, and V when we go "forward" d with the camera?
- How to change p_0, p_{ref}, and V when we "roll" (turn right) θ with the camera?
- How to change p_0, p_{ref}, and V when we "pitch" θ with the camera?

- If we express these tranformations in camera coordinates, how do we update the $\mathbf{M}_{wc}$ matrix?

Smooth Rotation

From a practical standpoint, we often want to use transformations to move and reorient an object smoothly

Problem: find a sequence of model-view matrices $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_n$ so that when they are applied successively to one or more objects we see a smooth transition.

Smooth Rotation

Consider the two approaches

- For a sequence of rotation matrices $\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_n$, find the Euler angles for each and use

$$\mathbf{R}_i = \mathbf{R}_{ix}\mathbf{R}_{iy}\mathbf{R}_{iz}$$

  - Not very efficient
  - Risk of Gimbal lock
  - hard to get uniform incremental steps.

- Use the final positions to determine the axis and angle of rotation, then increment only the angle
  - Rotation about an arbitray axis
  - Use properties of eigenvalues

Smooth Rotation

Consider the two approaches

- For a sequence of rotation matrices $\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_n$, find the Euler angles for each and use

$$\mathbf{R}_i = \mathbf{R}_{ix}\mathbf{R}_{iy}\mathbf{R}_{iz}$$

  - Not very efficient
  - Risk of Gimbal lock
  - hard to get uniform incremental steps.

- Use the final positions to determine the axis and angle of rotation, then increment only the angle
  - Rotation about an arbitray axis
  - Use properties of eigenvalues

Quaternions can be more efficient than either

Quaternions

- Complex numbers can express rotations in 2D. Recalling Euler's identity
$$e^{\mathbf{i}\theta} = \cos(\theta) + \mathbf{i}\sin(\theta)$$
we can write a polar representation of a complex number as
$$c = a + \mathbf{i}b = re^{\mathbf{i}\theta}$$
where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1}\frac{b}{a}$.
- This polar representation gives an expression of rotations in the complex plane.

Quaternions

- Extension of imaginary numbers from two to three dimensions

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

- Requires one real and three imaginary components **i**, **j**, and **k**. We can define the quaternion r as

$$r = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = (q_0, \mathbf{q})$$

- For quaternions we have rules for addition, multiplication, magnitude and inverse.

Quaternions and Rotations

- Using the quaternion $q = (\cos\frac{\theta}{2}, \mathbf{v} \cdot \sin\frac{\theta}{2})$ describes a rotion of θ degrees around the unit vector $\mathbf{v}$.
- Representing a point $\mathbf{p} = (x, y, z)$ as the quaternion $\mathbf{p} = (0, \mathbf{p})$, then $\mathbf{p}' = (0, \mathbf{p}') = q\mathbf{p}q^{-1}$ where
  $$\mathbf{p}' = \cos^2\frac{\theta}{2}\mathbf{p} + \sin^2\frac{\theta}{2}(\mathbf{p}\cdot\mathbf{v})\mathbf{v} + \sin^2\frac{\theta}{2}(\mathbf{p}\times\mathbf{v}) + \sin^2\frac{\theta}{2}(\mathbf{p}\cdot\mathbf{v})\times\mathbf{v}$$
  gives us $\mathbf{p}$ rotated θ degrees around $\mathbf{v}$.

Quaternions vs Euler Angles

- Quaternions use less operations than euler angles
- Easier to interpolate points over a rotation with quaternions
- Euler angles can risk gimble locks