Computer Graphics 1

2 Transformation

Summer Semester 2021

Ludwig-Maximilians-Universität München

Tutorial 2: Transformation

- Linear Algebra Concepts Review
- Homogeneous Coordinates and Affine Transformation
- 3D Rotation
- Summary

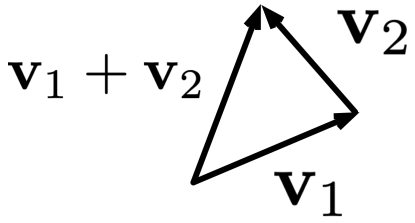
Point vs. Vector

- A point encodes a specific ***location***
  - An exact information
  - A reference is needed
  - In the *Cartesian coordinate system*, the reference point is the ***origin***
    - *"The metro station is 100 meters away to the south of the office"* ⇒ *Point*
      - *Reference point: the office*
      - *Location: 100 meters away to the south*

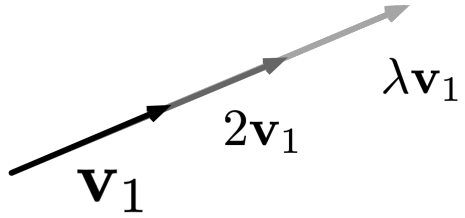
- A vector encodes ***direction*** and ***magnitude***
  - Given a reference point, a vector can look like a point, e.g. $\mathbf{v} = (x_1, x_2, x_3)^\top \in \mathbb{R}^3$
    - *"The highest standing jump is 1.651 meters"* ⇒ *Vector*
      - *Direction: jump up*
      - *Magnitude: 1.651 meters*

Vector Operations

Vector Sum



Scalar-Vector Product



Span and Basis in Linear Space

A span is the space of all possible linear combinations of its basis vectors.

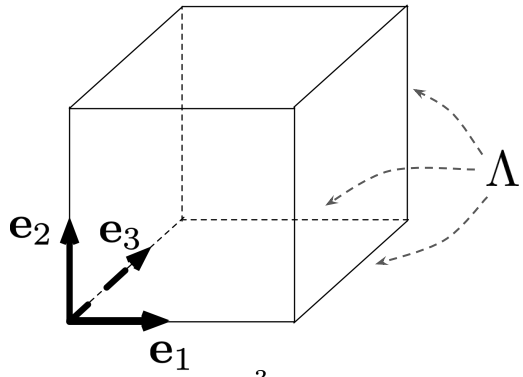
The basis is orthonormal if each basis vector is orthogonal to all others.

The dimension of the space is equal to the number of linearly independent basis vectors.

Most importantly:

The coordinates of a vector are defined by (if possible) the unique linearly combined scalar coefficients of a given basis.

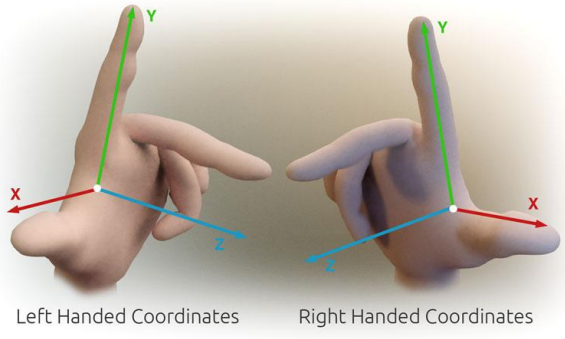
Cube Span



$$\text{span}\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} = \sum_{i=1}^{3} \lambda_i \mathbf{e}_i, \lambda_i \in [0, \Lambda]$$

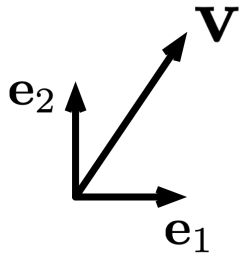
Cartesian Coordinate Systems

- In 3D, the Cartesian coordinate systems can be defined in two different ways: *Left handed* or *right handed*
  - Y-axis upward (both)
  - WebGL (also OpenGL): Right handed
    - ⇒ three.js is also right handed
  - Direct3D: Left handed
- Why?
  - Historical reason: personal preference, a random decision
- **Without ambiguity or otherwise specified, we use right handed systems**



Norm

If $\mathbf{e}_1$ and $\mathbf{e}_2$ are basis vectors, then the norm of a vector is defined as

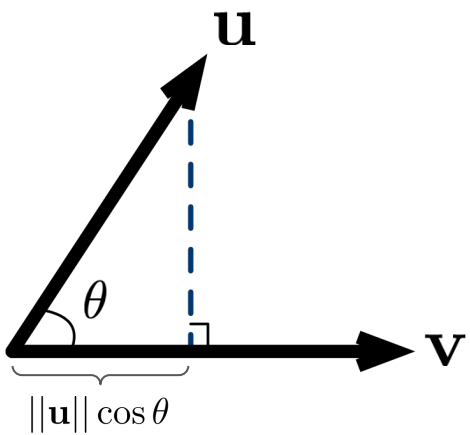


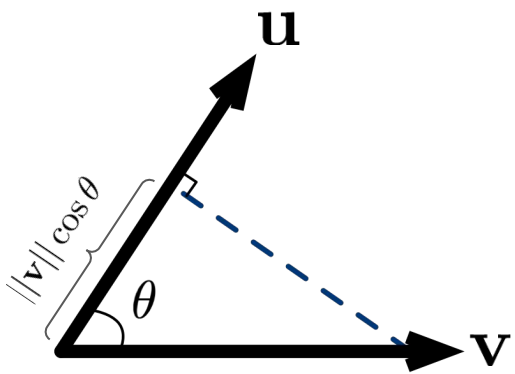
$$||\mathbf{v}|| = ||\lambda_1 \mathbf{e}_1 + \lambda_2 \mathbf{e}_2|| = \sqrt{\lambda_1^2 + \lambda_2^2}$$

Dot Product

Dot product of two vectors $\mathbf{u}, \mathbf{v}$ is defined the scalar multiplication of the *projection magnitude of one to the other* and *the magnitude of the other*:

$$\langle \mathbf{u}, \mathbf{v} \rangle = ||\mathbf{u}||||\mathbf{v}|| \cos\theta$$



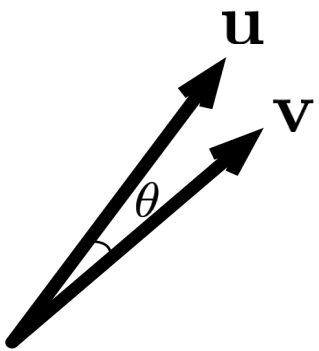


Dot Product: Geometric Meaning

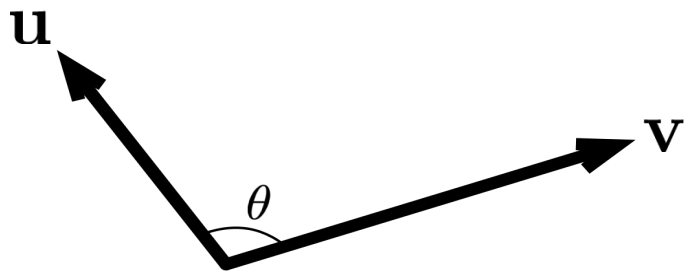
Dot product can be used for a similarity measurement of two vectors:

$$\text{“similarity”} = \cos\theta = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{||\mathbf{u}||||\mathbf{v}||}$$

"quite similar": two vectors have quite similar magnitude and quite similar direction



"quite similar"



"quite different"

Dot Product (with Orthonormal Basis)

With orthonormal basis, dot product of two vectors $\langle \mathbf{u}, \mathbf{v} \rangle$ is simplified* to the sum of element-wise multiplications:

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{n} u_i v_i$$

where u_i, v_i are the i-th component of $\mathbf{u}, \mathbf{v}$.

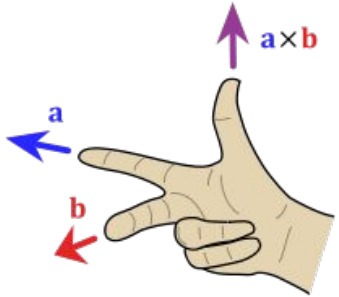
This is how to computers deal with dot product.

*The definition of a dot product does not rely on a coordinate system, the formula is true if and only if the basis are orthogonal.

Cross Product

For 3D vectors, by *definition*:

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

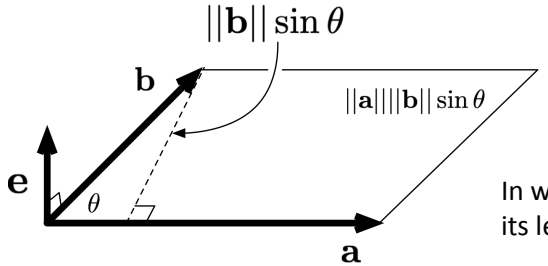


"right-handed"

Naturally: What's the geometric meaning of this definition?!?

If $\mathbf{e}$ is a unit vector orthogonal w.r.t. $\mathbf{a}$ and $\mathbf{b}$, then:

$$\mathbf{a} \times \mathbf{b} = \mathbf{e}||\mathbf{a}||||\mathbf{b}|| \sin\theta$$



In words: the result is a vector orthogonal to **a** and **b** and its length corresponds to the area between **a** and **b**.

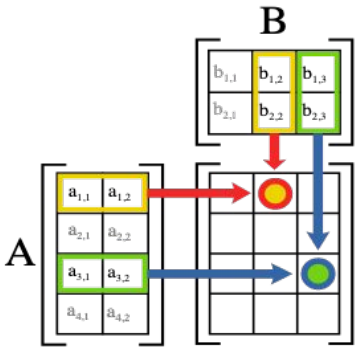
Matrix

Addition, subtraction, scalar multiplication are computed element-wise.

Matrix transpose *flips* elements over diagonal, e.g. $\begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \\ x_{20} & x_{21} \end{pmatrix}^\top = \begin{pmatrix} x_{00} & x_{10} & x_{20} \\ x_{01} & x_{11} & x_{21} \end{pmatrix}$

Matrix multiplication is more interesting to us:

$$\mathbf{C}_{m\times n} = \mathbf{A}_{m\times p} \cdot \mathbf{B}_{p\times n} \text{ where } c_{i,j} = \sum_{k=1}^{p} a_{i,k} b_{k,j}, 1 \le i \le m, 1 \le j \le n$$



Computation process is labor extensive, and tedious. ⇒ *program it!*

What happens $\mathbf{A}_{m\times p_1} \cdot \mathbf{B}_{p_2\times n}$ where $p_1 \neq p_2$*?*

A: Undefined.

What is the result of $\mathbf{M}_{1\times 3} \cdot \mathbf{N}_{3\times 1}$*?*

A: 1x1 matrix**.

*Note the difference of the $\bullet$ and the $\langle , \rangle$: Matrix multiplication always result in a matrix but dot product only result in a scalar.

Determinant

The determinant of a 2x2 matrix

$$\det(\mathbf{B}) = \begin{vmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{vmatrix} = b_{00}b_{11} - b_{10}b_{01}$$

And the determinant of 3x3 matrix:

$$\det(\mathbf{C}) = \begin{vmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{vmatrix} = c_{00} \begin{vmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{vmatrix} - c_{01} \begin{vmatrix} c_{10} & c_{12} \\ c_{20} & c_{22} \end{vmatrix} + c_{02} \begin{vmatrix} c_{10} & c_{11} \\ c_{20} & c_{21} \end{vmatrix}$$

Determinant: Geometric Meaning

Cross product can be rewritten as:

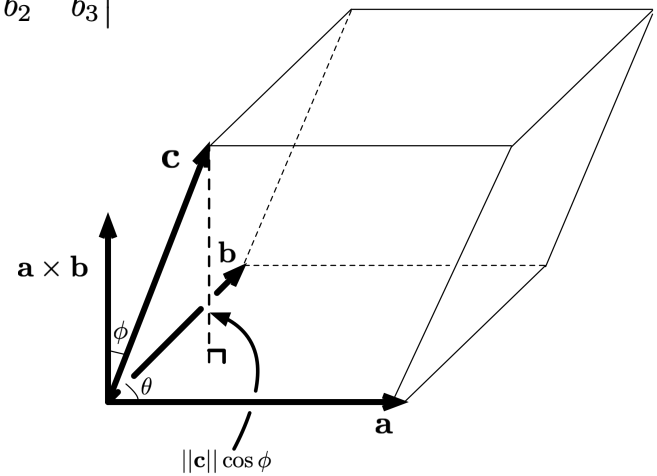
$$\mathbf{a}\times\mathbf{b} = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix} = (a_2b_3 - a_3b_2)\begin{pmatrix}1\\0\\0\end{pmatrix} + (a_3b_1 - a_1b_3)\begin{pmatrix}0\\1\\0\end{pmatrix} + (a_1b_2 - a_2b_1)\begin{pmatrix}0\\0\\1\end{pmatrix}$$

$$= \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix}\mathbf{e}_1 - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix}\mathbf{e}_2 + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix}\mathbf{e}_3 = \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \quad \textit{(mnemonic!)}$$

The calculation reveals geometric meaning of a determinant:

$$\langle \mathbf{c}, (\mathbf{a}\times\mathbf{b})\rangle = \langle \mathbf{c}, \mathbf{e}\rangle ||\mathbf{a}||||\mathbf{b}|| \sin\theta = \overbrace{||\mathbf{c}||\cos\phi}^{\text{height}} \overbrace{||\mathbf{a}||||\mathbf{b}||\sin\theta}^{\text{bottom surface}}$$

$$\langle \mathbf{c}, (\mathbf{a}\times\mathbf{b})\rangle = \left\langle (c_1,c_2,c_3)^\top, \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \right\rangle = \begin{vmatrix} c_1 & c_2 & c_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

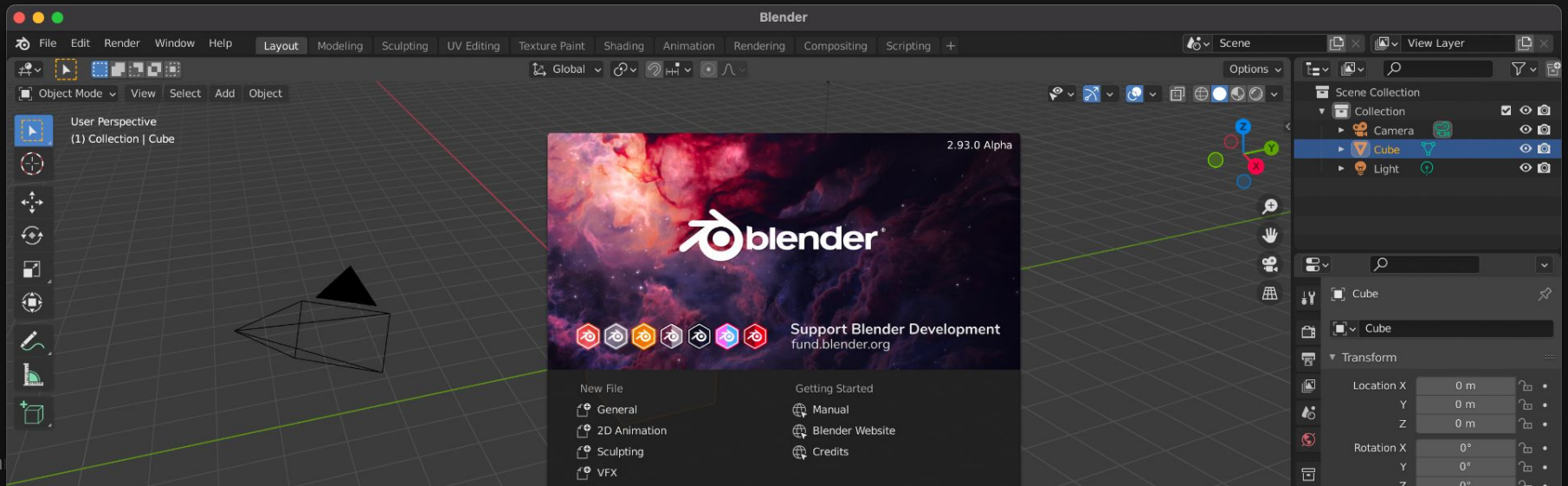


determinant is a volume!

Breakout: Hands-on

Before we move on, install Blender (an open source 3D modeling software that can help us to learn computer graphics and easily verify well implemented algorithms). Watch the official blender tutorial videos (from 1 to 5), try to answer these two questions:

1. How to translate, scale, and rotate the initial cube?
2. How to add a monkey (Suzanne) ?

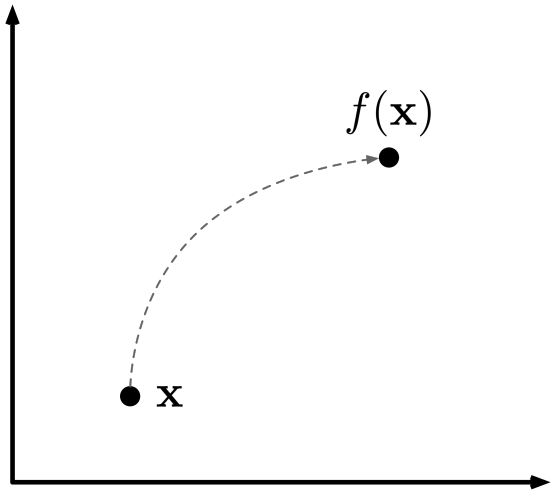


Tutorial 2: Transformations

- Linear Algebra Concepts Review
- Homogeneous Coordinates and Affine Transformation
- 3D Rotation
- Summary

Transformation

Transform one point into another using a transformation function *f*



Linear Transformations

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$f(a\mathbf{x}) = af(\mathbf{x})$$

Matrix multiplication $\mathbf{x}' = \mathbf{A}\mathbf{x}$ is a linear transformation because:

$$\mathbf{A}(\mathbf{x}+\mathbf{y}) = \mathbf{A}\mathbf{x} + \mathbf{A}\mathbf{y}$$
$$\mathbf{A}(a\mathbf{x}) = a\mathbf{A}\mathbf{x}$$

In 3D: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

E.g. scaling: $\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$

Is Translation linear?

No. We can also not write it as a matrix multiplication:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

As we cannot distinguish a vector or a point in three dimension:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Thus, we need a "*unified theory*".

Homogeneous Coordinates

Go to a higher dimension (Add the fourth component):

Point = $(x, y, z, \mathbf{1})^\top$ Vector = $(x, y, z, \mathbf{0})^\top$ Matrix = $\begin{pmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ w_1 & w_2 & w_3 & \mathbf{1} \end{pmatrix}$

With the extension, we have the following intuition matching properties:

- vector + vector = vector
- vector - vector = vector
- vector + point = point
- point - point = vector

Convert to 1 when instantiating it to a point

What about point + point?

$$(x_1, y_1, z_1, 1)^\top + (x_2, y_2, z_2, 1)^\top = (x_1 + x_2, y_1 + y_2, z_1 + z_2, 2)^\top \Rightarrow (\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}, \frac{z_1 + z_2}{2}, 1)$$ (i.e. midpoint)

Transformation using Homogeneous Coordinates

- Homogeneous form of translation:

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

Point remains a point (location dependent)

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

Vector remains a vector (location independent)

- Homogeneous form of scaling*:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & s_w \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \\ s_w \end{pmatrix} = \begin{pmatrix} (s_x/s_w)x \\ (s_y/s_w)y \\ (s_z/s_w)z \\ 1 \end{pmatrix}$$

Point remains the same (a point has no size)

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & s_w \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \\ 0 \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \\ 0 \end{pmatrix}$$

Vectors gets scaled (vectors can scale on different directions)

* s_w usually set as 1.

Why Homogeneous Coordinates?

- We can combine translation (and more) with linear transformations
- Homogeneous coordinates enable us to apply non-linear transformations as matrix multiplication, e.g. translation
- Homogeneous transformation can be *inverted*

Inverse Transformation

Inverse transformation $\mathbf{T}^{-1}$ can be considered as an "*undo*". Thus:

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{T}\mathbf{x} = \mathbf{I}\mathbf{x} = \mathbf{x}$$

One can use this property to find the inverse of a transformation. For example, the inverse of a scaling transformation can be deduced:

$$\mathbf{T}_s^{-1}\mathbf{T}_s = \begin{pmatrix} s'_x & 0 & 0 & 0 \\ 0 & s'_y & 0 & 0 \\ 0 & 0 & s'_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{I}$$

$$\implies s'_x = \frac{1}{s_x}, s'_y = \frac{1}{s_y}, s'_z = \frac{1}{s_z} \implies \mathbf{T}_s^{-1} = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Affine Transformation

With homogeneous coordinates, one can unify an *affine transformation* as the combination of linear transformation and translation in a single 4x4 matrix.

Moreover, it is important to notice that affine transformations always translation lastly, e.g. scale + translation:

$$\mathbf{T}_t\mathbf{T}_s = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & t_x \\ 0 & s_y & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

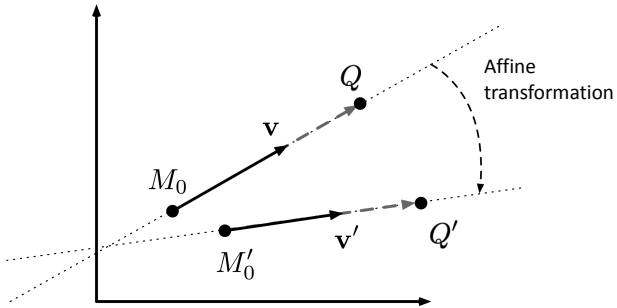
$$\mathbf{T}_s\mathbf{T}_t = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & s_x t_x \\ 0 & s_y & 0 & s_y t_y \\ 0 & 0 & s_z & s_z t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(translation is unexpectedly scaled)

⇒ equal if and only if T_s is an identity matrix ⇒ **Order matters!**

Properties of Affine Transformation

1. Collinearity: Lines remain lines



2. Parallelism: Parallels remain parallel

3. Convexity: Convex curves remain convex curves

⇒ (line) proportion preserving

Types of Transformations

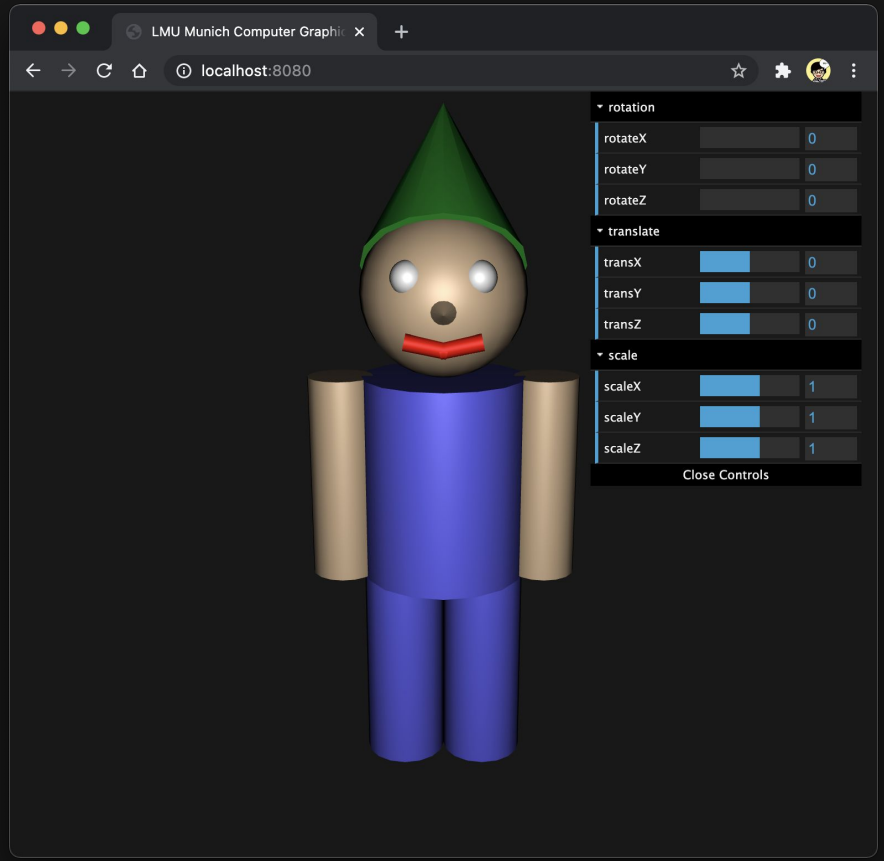
- Linear: Scale, rotation, reflection, shear, ...
- Non-linear: translation, ...
- **Affine: linear transformation + translation**
  - *line proportion preserving*
- *Isometric*: Translation, rotation, reflection
  - *distance preserving*
- Non-affine? Not now.

Aside: special transformation categories preserve different types of geometric properties, which is one of the key studies in computer graphics.

Breakout: Transform the Tinman

1. In Blender, open the `tinman.blender` file:
   - apply coarse transform the tinman using keyboard shortcut <s>, <g>, <r>;
   - apply precise transform the tinman using the item menu and transform the tinman back to the original model.

2. Open the provided code skeleton, look for TODO: comments and implement translation and scale transformation in three.js
   - a live demo can be found here

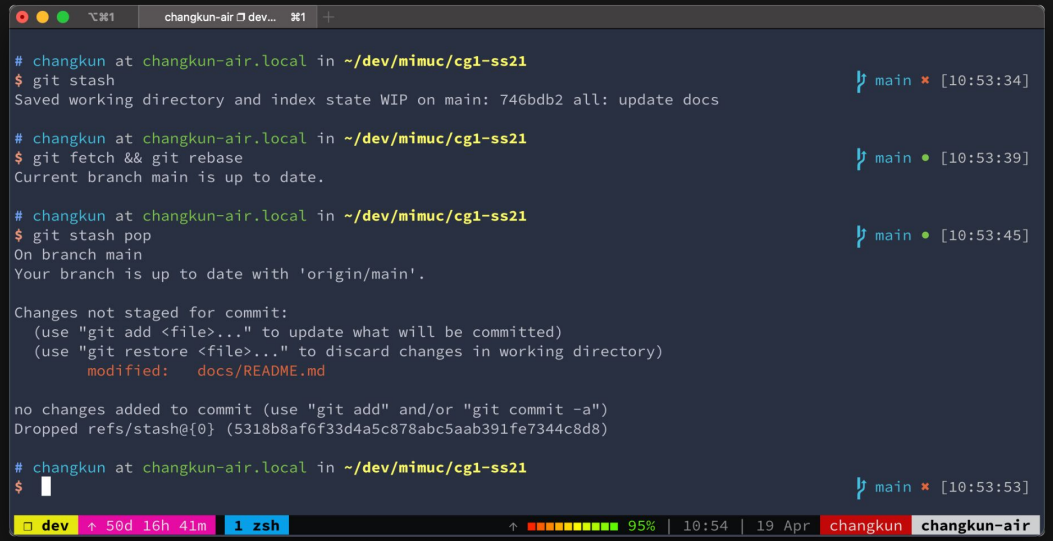


Aside: Synchronize GitHub Repository to Local Clone

Once the remote Github repository has new commits, it is a good idea to sync the latest changes from remote to local.

One can accomplish the synchronization via:

```
$ git stash                 # temporarily shelves (or stashes) changes you've made locally
$ git fetch && git rebase   # fetch remote changes and synchronize it to the local copy
$ git stash pop             # re-apply previously stashed changes
```

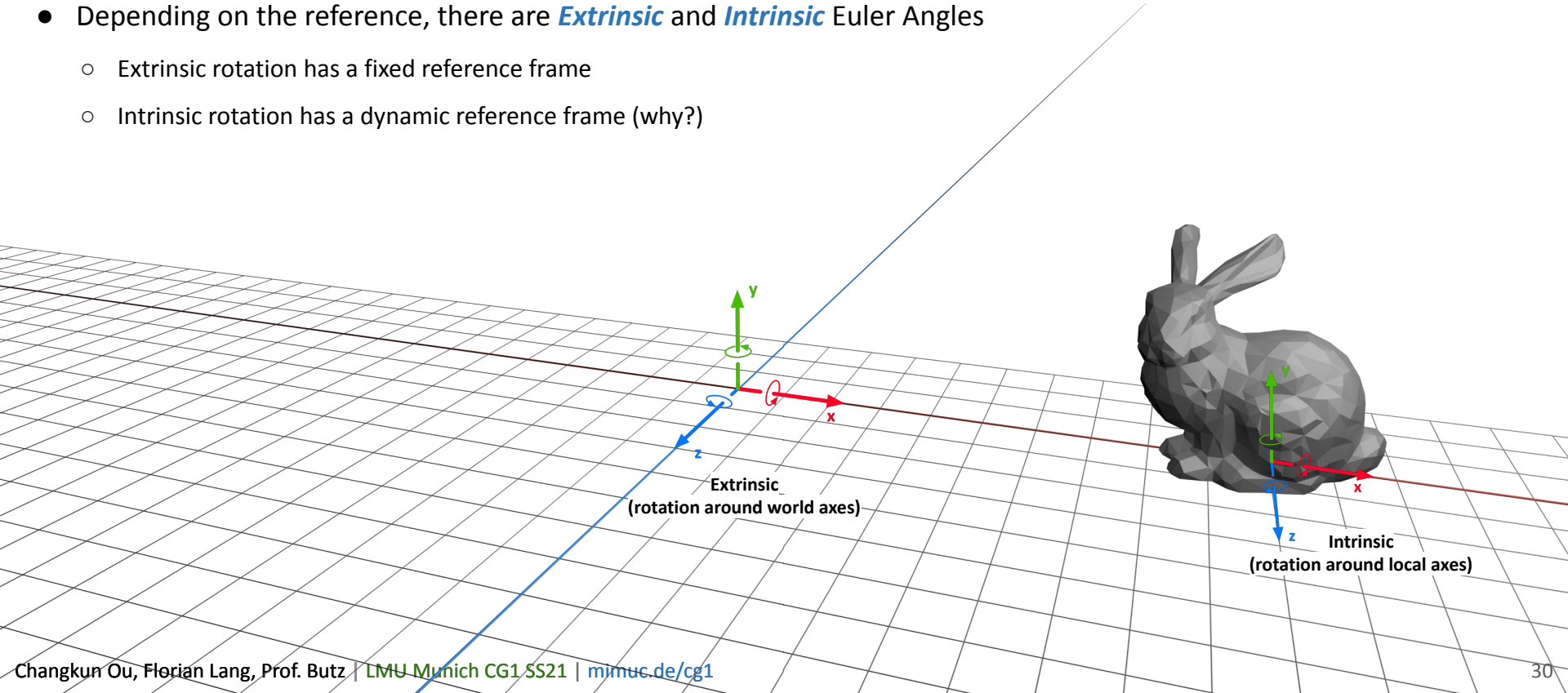


Tutorial 2: Transformations

- Linear Algebra Concepts Review
- Homogeneous Coordinates and Affine Transformation
- 3D Rotation
- Summary

Euler Angles

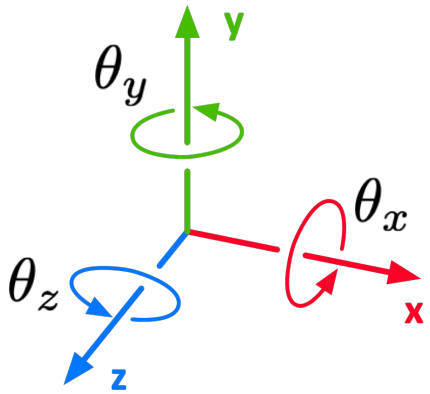
- A rotation around axes can be expressed using the so called ***Euler Angles***
- Depending on the reference, there are ***Extrinsic*** and ***Intrinsic*** Euler Angles
  - Extrinsic rotation has a fixed reference frame
  - Intrinsic rotation has a dynamic reference frame (why?)



Extrinsic Rotation

Rotation matrices around x-, y-, and z-axis:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix} \quad \mathbf{R}_y = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix} \quad \mathbf{R}_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Euler Angle Sequences

- Depends on the order of rotation, there are ***Tait-Bryan angles*** or ***Proper Euler angles***
- **Tait-Bryan angles**
  - 6 possible sequences: xyz, xzy, yxz, yzx, zxy, zyx.
- **Proper Euler angles**
  - 6 possible sequences: xyx, xzx, yxy, yzy, zxz, zyz.

The Euler in three.js uses intrinsic Tait-Bryan angles by default order: `xyz`.

What about xxy, xxz, yyx, yyz, zzx, zzy, xyy, xzz, yxx, yzz, zxx, zyy?

xx, yy, zz can be merged to a single X, Y, or Z rotation.

These orders can collapse to xy, xz, yx, yz, zx, zy (which are not *three* composed rotations)

Breakout: 3D Rotation using Euler Angles

In Blender:

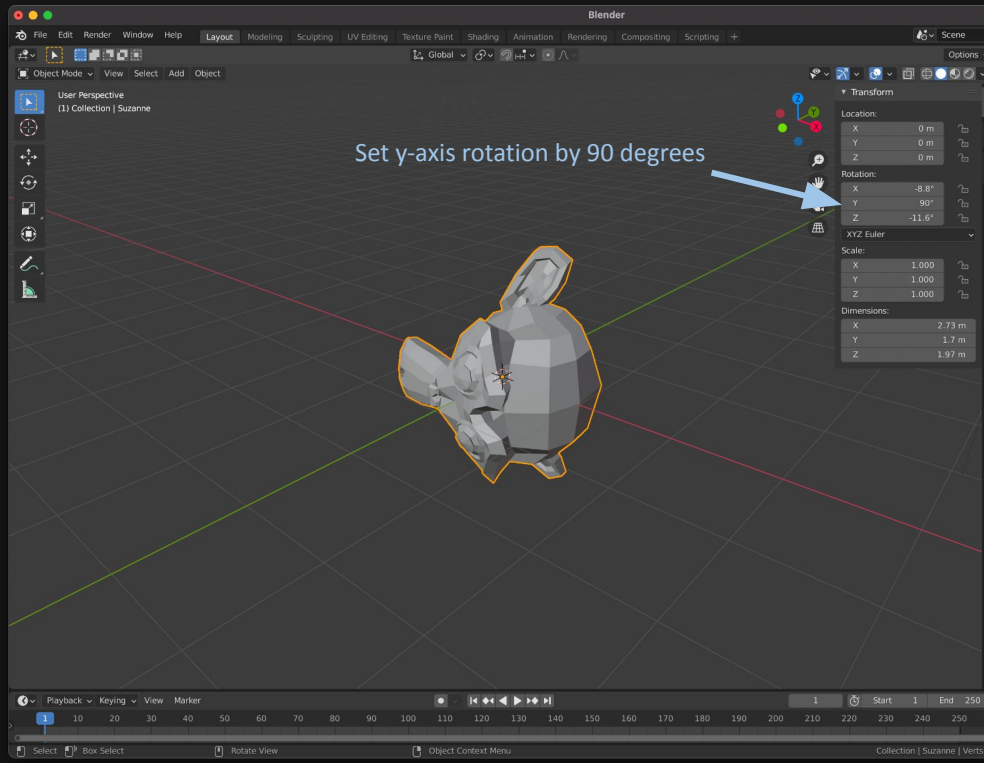
1. Add a Suzanne
2. Rotate y-axis by 90 degrees
3. Rotate x- or z-axis by *any* angle

Or in provided demo src/02-transform/1-wichtelsolo:

1. Run the provided demo [or live demo]
2. Rotate around y-axis by 90 degrees
3. Rotate x- or z-axis by *any* angle

The observation is:

The object can only be rotated in x-y plane after the second step (in XYZ Euler order)



Limitation of Euler Angles: *Gimbal Lock*

If $\theta_y = \frac{\pi}{2}$ then $\cos\theta_y = 0, \sin\theta_y = 1$. For any point $P = (x, y, z)^\top$, one can calculate:

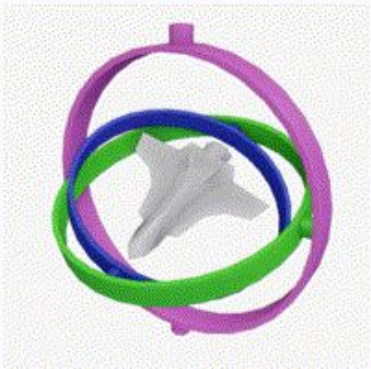
$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z P = \begin{pmatrix} 0 & 0 & 1 \\ \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z - \sin\theta_x \sin\theta_z & 0 \\ -\cos\theta_x \cos\theta_z + \sin\theta_x \sin\theta_z & \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_z & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} z \\ \dots \\ \dots \end{pmatrix}$$

No matter what we do with θ_x, θ_z, we will always land on a the same plane.

The *x*-axis is fixed to *z*, meaning a *single axis rotation*:

This is called the **Gimbal Lock.**

It is a principal limitation in Euler angles.



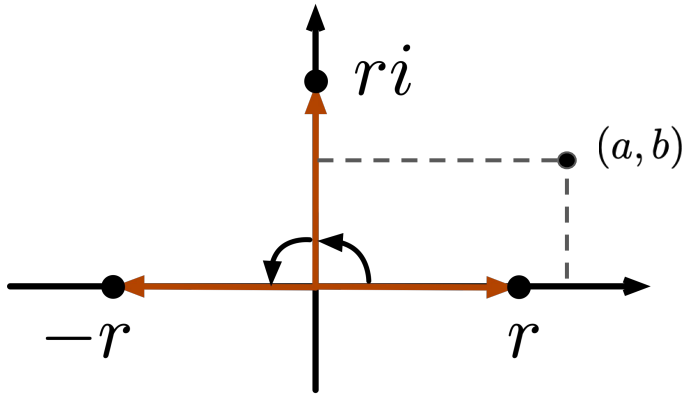
https://en.wikipedia.org/wiki/Gimbal_lock

Recall: Complex Numbers

- Consist of a real part, and an imaginary part:

$$a + bi \in \mathbb{C}, a, b \in \mathbb{R} \qquad i^2 = -1$$

- Complex number multiplication represents a 2D rotation, simple case:



- Complex numbers looks like points on 2D plane. What's the equivalent in 3D?

Quaternion

A ***quaternion*** has one real, three imaginary parts:

$$\mathbf{q} = a + bi + cj + dk, i^2 = j^2 = k^2 = ijk = -1$$

Hard to imagine something in 4D! Consider a quaternion is a combination of a scalar and a 3D vector:

$$\mathbf{q} = (a, \mathbf{v}) \text{ where } \mathbf{v} = (b, c, d)^\top \in \mathbb{R}^3$$

The multiplication of two quaternions can be calculated by

$$\mathbf{pq} = (ea - \mathbf{w}^T \cdot \mathbf{v}, e\mathbf{v} + a\mathbf{w} + \mathbf{w} \times \mathbf{v})$$

where $\mathbf{q} = (a, \mathbf{v}), \mathbf{v} = (b, c, d)^\top$ and $\mathbf{p} = (e, \mathbf{w}), \mathbf{w} = (f, g, h)^\top$.

3D Rotation using Quaternions

Rotation in 3D can be expressed using quaternions.

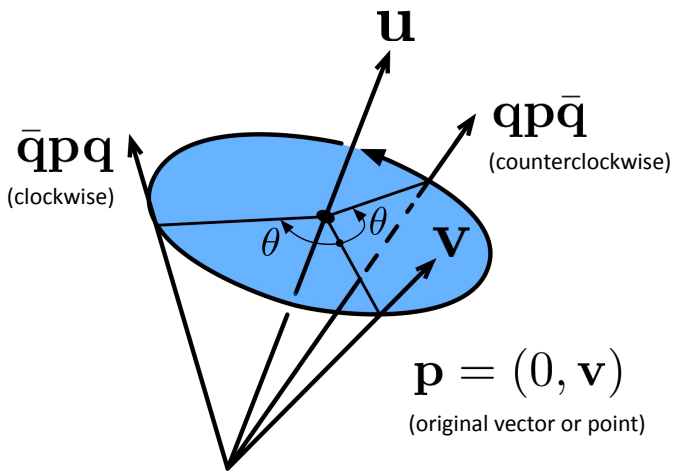
Given an ***unit*** axis $\mathbf{u}$ to an arbitrary direction and an angle θ, a rotation can be expressed by the **multiplication of two** *conjugate* quaternions:

angle **axis**

$$\mathbf{q} = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2})$$

$$\bar{\mathbf{q}} = (\cos\frac{\theta}{2}, -\mathbf{u}\sin\frac{\theta}{2})$$

negative



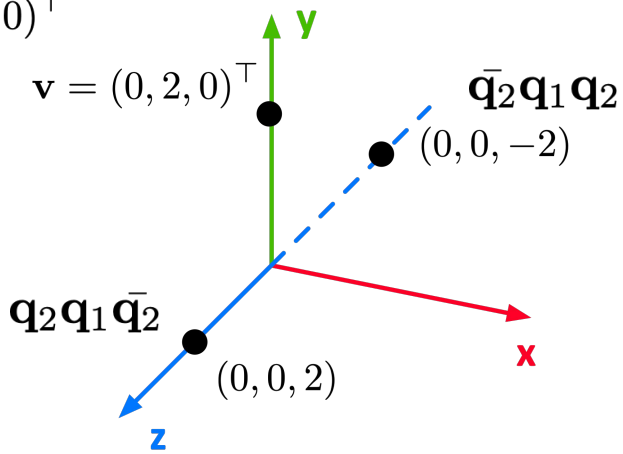
Example: Axis Rotations using Quaternions

The quaternions for the rotation around the x-axis, y-axis, and z-axis:

$$\mathbf{q}_x = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}), \mathbf{u} = (1,0,0)^\top$$
$$\mathbf{q}_y = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}), \mathbf{u} = (0,1,0)^\top$$
$$\mathbf{q}_z = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}), \mathbf{u} = (0,0,1)^\top$$

Specifically, $\mathbf{q}_2 = (\cos\frac{\pi}{4}, \mathbf{u}\sin\frac{\pi}{4})$ rotates 90 degrees on the x-axis $\mathbf{u} = (1,0,0)^\top$

- The clockwise rotation is represented by $\bar{\mathbf{q}_2}\mathbf{q}_1\mathbf{q}_2$
- The counterclockwise rotation is represented by $\mathbf{q}_2\mathbf{q}_1\bar{\mathbf{q}_2}$



Euler Angles vs. Quaternions

- Euler angles
  - are intuitive but complicated because one must communicate well by define a) extrinsic or intrinsic b) order sequence
  - are suffering the gimbal lock issue
- Quaternions
  - are much less intuitive than Euler angles
  - are convenient to express a rotation around an arbitrary direction
  - do not have the gimbal lock issue

Tutorial 2: Transformations

- Linear Algebra Concepts Review
- Homogeneous Coordinates and Affine Transformation
- 3D Rotation
- Summary

Summary

- We covered:
  - Review linear algebra but more importantly revealed the *real world geometric meaning* behind all these formulas
  - Using *homogeneous coordinates* representation to unify linear transformation and translation as *affine transformation*
  - Different types of *Euler angles* for 3D rotations
  - The *gimbal lock* issue caused by Euler angles and how to use *quaternions* to solve it

Next

Geometry Structures