## Course Plan

lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Models**
lec. 7: Game **Textures**
lec. 9: Game **Materials**
lec. 8: Game **3D Animations**
lec. 10: **Networking** for 3D Games
lec. 11: **3D Audio** for 3D Games
lec. 12: **Rendering Techniques** for 3D Games
lec. 13: **Artificial Intelligence** for 3D Games

58

## Rotation composition? Quaternion multiplication!

$p, q_0, q_1 \in \mathbb{H}$
$q_0, q_1$ represent rotations
$p$ represents a point

p rotated by q1, then rotated by q0

p rotated by q1

$$q_0 \cdot (q_1 \cdot p \cdot \bar{q}_1) \cdot \bar{q}_0$$

product is associative
(like for complex numbers)

$$=$$

$$(q_0 \cdot q_1) \cdot p \cdot (\bar{q}_1 \cdot \bar{q}_0)$$

$\bar{r} \cdot \bar{s} = \overline{s \cdot r}$
(rules of quaternions)
(remember: product is not commutative)

$$=$$

$$(q_0 \cdot q_1) \cdot p \cdot \overline{(q_0 \cdot q_1)}$$

59

## Rotation inversion?
## Quaternion conjugation!

$\mathrm{q}, \mathrm{p} \in \mathbb{H}$
$\mathrm{q}$ represents a rotation
$\mathrm{p}$ represents a point

Just like with compex numbers:

$$\mathrm{q}^{-1} = \frac{\bar{\mathrm{q}}}{\|\mathrm{q}\|^2} = \bar{\mathrm{q}}$$

that is:

$$\bar{\mathrm{q}} \cdot \mathrm{q} = \|\mathrm{q}\|^2 = 1$$

because q is unitary

Proof:

p rotated by q, then rotated by $\bar{\mathrm{q}}$

p rotated by q

$$\bar{\mathrm{q}} \cdot (\mathrm{q} \cdot \mathrm{p} \cdot \bar{\mathrm{q}}) \cdot \bar{\bar{\mathrm{q}}}$$

product is associative (like for complex numbers)

$$= (\bar{\mathrm{q}} \cdot \mathrm{q}) \cdot \mathrm{p} \cdot (\bar{\mathrm{q}} \cdot \mathrm{q})$$

$$= \mathrm{p}$$

60

## Quaternions: geometric interpretation (complete)

- A quaternion $\mathrm{q} = (\vec{\mathrm{v}}, d)$ represents :
  - the 3D point / vector / versor $\vec{\mathrm{v}}$, when $d = 0$
  - a 3D rotation, when $\mathrm{q}$ is unit, i.e. $\|\mathrm{q}\|^2 = \|\vec{\mathrm{v}}\|^2 + d^2 = 1$
  - neither, otherwise
- If $\mathrm{q}$ is a rotation and $\mathrm{p}$ is a point ($\mathrm{q}, \mathrm{p} \in \mathbb{H}$ ) then…
  - $\mathrm{q} \cdot \mathrm{p} \cdot \bar{\mathrm{q}}$ is the rotated point / vector
  - $\bar{\mathrm{q}}$ is the inverse rotation
  - (so, $\bar{\mathrm{q}} \cdot \mathrm{p} \cdot \mathrm{q}$ is point $\mathrm{p}$ rotated… in the *other* direction)
  - $\mathrm{q}_0 \cdot \mathrm{q}_1$ is the composited rotation (first $\mathrm{q}_1$ then $\mathrm{q}_0$ )

61

## 3D Rotations as Quaternions

- quaternion $\mathbf{q}$ representing the 3D rotation of angle $\alpha$ around axis $\hat{a}$ :
  - $\mathrm{q} = \left( \sin\left(\frac{\alpha}{2}\right) \hat{a}\,,\ \cos\left(\frac{\alpha}{2}\right) \right)$

  that is
  - $\mathrm{q} = \sin\left(\frac{\alpha}{2}\right) \hat{a}_x i + \sin\left(\frac{\alpha}{2}\right) \hat{a}_y j + \sin\left(\frac{\alpha}{2}\right) \hat{a}_z k + \cos\left(\frac{\alpha}{2}\right)$
- Observe that $\|\mathrm{q}\|^2 = 1$

verify

62

## Exercise: are the following quaternions unitary?

- $\mathbf{q}_0 = (0,0,-1,0) = -j$
- $\mathbf{q}_1 = \left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right) = 0.5i + 0.5j + 0.5k + 0.5$
- $\mathbf{q}_2 = (1,1,1,1) = i + j + k + 1$

63

## Example: turnabout rotation

(italian: un «dietrofront»)

- Find the quaternion $\mathbf{r}$ representing the rotation by 180° ($\pi$ radiants) around axis Y
  - $\hat{a} = (0,1,0)$
  - $\alpha = \pi, \sin\left(\frac{\alpha}{2}\right) = 1, \cos\left(\frac{\alpha}{2}\right) = 0$
  - $\mathbf{r} = (1\,\hat{a}, 0) = 0i + 1j + 0k + 0 = j$

imaginary vector real scalar

- Find the quaternion $\mathbf{q}$ representing point $\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$:
  - $\mathbf{q} = 2i + 3j + 4k$
- Rotate that point with that rotation:
  - $\mathbf{q}' = \mathbf{r}\,\mathbf{q}\,\bar{\mathbf{r}} = j\,(2i + 3j + 4k)(-j) = \ldots$ *(finish me!)*

65

## 3D Rotations as Quaternions: equivalent representations ☹

- Around axis $\hat{a}$ by angle $\alpha$ :

$$q = \left(\sin\left(\frac{\alpha}{2}\right)\hat{a}, \cos\left(\frac{\alpha}{2}\right)\right)$$

- Around axis $-\hat{a}$ by angle $(-\alpha)$ (it's the **same rotation**!) :

$$q' = \left(-\sin\left(\frac{-\alpha}{2}\right)\hat{a}, \cos\left(\frac{-\alpha}{2}\right)\right) = q$$

the **same** quaternion :-)

Nice! But:

360°

- Around axis $\hat{a}$ by angle $(\alpha + 2\pi)$ (again, it's the **same rotation**!) :

$$q'' = \left(\sin\left(\frac{\alpha}{2} + \pi\right)\hat{a}, \cos\left(\frac{\alpha}{2} + \pi\right)\right) =$$

180°

$$= \left(-\sin\left(\frac{\alpha}{2}\right)\hat{a}, -\cos\left(\frac{\alpha}{2}\right)\right) = -q$$

a **different** quaternion :-(

- Conclusion:
  quaternion $q$ and quaternion $-q$ encode the same rotation

66

## 3D Rotations as Quaternions: equivalent representations ☹

Given a quaternion $\mathbf{q}$ representing a rotation:

- Flip its imaginary part (getting $\bar{\mathbf{q}}$): invert rotation
- Flip its real part (getting $-\bar{\mathbf{q}}$): invert rotation
- Flip everything (getting $-\mathbf{q}$ ): same rotation

Every single rotation is encoded
by two different quaternions: $\mathbf{q}$ and $-\mathbf{q}$.
(the inverse rotation is also encoded
by two different quaternions: $\bar{\mathbf{q}}$ and $-\bar{\mathbf{q}}$ )

67

## Interpolating two quaternions (that represent two rotations)

Works well, but two *caveats*:

⚠ Take the "shortest path" (as usual):
flip 2nd quaternion first, if this makes them closer

- Distance defined as dot product in 4D
  (consider quaternions as 4D unit vectors for this)
  (remember: dot product between unitary vectors is a measure of similarity!)

⚠ Loss of normality

- Needs re-normalization (NLERP),
- Or SLERP
  (again, just consider quaternions as 4D unit vectors)

68

## Shortest path interpolation: the case of quaternions

- Let $\mathbf{p}$ and $\mathbf{q}$ be two rotations
- $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation.
  - Which one to choose?
- Which one is closer to $\mathbf{p}$ ?
  - Distance between $\mathbf{p}$ and $\mathbf{q}$ = $\mathrm{dot}(\mathbf{p}, \mathbf{q})$
  - Distance between $\mathbf{p}$ and $-\mathbf{q}$ = $\mathrm{dot}(\mathbf{p}, -\mathbf{q}) = -\mathrm{dot}(\mathbf{p}, \mathbf{q})$
- Conclusion:
  - If $\mathrm{dot}(\mathbf{p}, \mathbf{q})$ is positive, interpolate with $\mathbf{q}$
  - Otherwise, interpolate with $-\mathbf{q}$

69

## Quaternions, exercise:
### Experiment with cumulation rotations

1. Take the quaternion q0 that encodes the 180° rotation around the Y axis (see exercises above)
2. Take the quaternion q1 that encodes the 180° rotation around the X axis (see exercises above)
3. Compute the quaternion q2 that does the two rotations in succession, in that order (using q0 and q1)
4. Which rotation is encoded by q2? Verify with a real 3D object (e.g. a cellphone) that q2 encoded the status that is reached if you rotate by q0 and then by q1

70

## Quaternions, exercise:
### Experiment with interpolating rotations

1. Take (again) the quaternion q0 that encodes the 180° rotation around the Y axis
2. Take the quaternion q1 that encodes identity rotation (or, rather, ***one*** quaternion that encodes it)
3. Compute the quaternion q2 that interpolates the two quaternions q0 and q1
   - (what happens with the shortest path?
     why do you think that happens?)
4. Which rotation is encoded by q2? To help with the answer, the sin and cos for PI/4 radiants (45°) is…

71

## Quaternion Product

| × | a $i$ | + | b $j$ | + | c $k$ | + | d | |
|---|---|---|---|---|---|---|---|---|
| e $i$ | | + | | + | | + | | + |
| + | | | | | | | | |
| f $j$ | | + | | + | | + | | + |
| + | | | | | | | | |
| g $k$ | | + | | + | | + | | + |
| + | | | | | | | | |
| h | | + | | + | | + | | |

73

## Quaternion Product

| $\vec{w}$ \ $\vec{v}$ | a $i$ | + | b $j$ | + | c $k$ | + | d |
|---|---|---|---|---|---|---|---|
| e $i$ | -1 ae | + | $k$ be | + | $-j$ ce | + | $i$ de |
| + f $j$ | $-k$ af | + | -1 bf | + | $i$ cf | + | $j$ df |
| + g $k$ | $j$ ag | + | $-i$ bg | + | -1 cg | + | $k$ dg |
| + h | $i$ ah | + | $j$ bh | + | $k$ ch | + | hd |

$$(\vec{w}, h) \cdot (\vec{v}, d) = (\text{some vector}, \text{some scalar})$$

74

## Quaternion Product

| $\vec{w}$ \ $\vec{v}$ | a $i$ | + | b $j$ | + | c $k$ | + | d |
|---|---|---|---|---|---|---|---|
| e $i$ | -1 ae | + | $k$ be | + | $-j$ ce | + | $i$ de |
| + f $j$ | $-k$ af | + | -1 bf | + | $i$ cf | + | $j$ df |
| + g $k$ | $j$ ag | + | $-i$ bg | + | -1 cg | + | $k$ dg |
| + h | $i$ ah | + | $j$ bh | + | $k$ ch | + | hd |

$$(\vec{w}, h) \cdot (\vec{v}, d) = (\vec{w}\, d + \vec{v}\, h + \vec{w} \times \vec{v}, \; h\, d - \vec{w} \cdot \vec{v})$$

76

## Quaternion multiplication: notes

- The previous slide shows how to compactly write (and compute! and code!) a product between two quaternions that are expressed as pairs (imaginary-vector , real-scalar)
- Quaternion multiplication is the basic operation, used (twice) to apply a rotation to a point/vector/versor, and (once) to cumulate two quaternions
- For example (next slide): let's use this in the formula to rotate to a point/vector/versor with quaternions
  - and let's see what simplifies
- Note: do not confuse the **quaternion product** with the **dot product** of two quaternions seen as 4D vectors

77

## Applying rotations (as quaternions)

$$\underbrace{(\vec{w}\,,a)}_{\text{quaternion representing a rotation}}\ \underbrace{(\vec{v},0)}_{\text{quaternion representing 3D point or vector } \vec{v}}\ \underbrace{(-\vec{w}\,,a)}_{\text{conjugate of } (\vec{w}\,,a)} =$$

$$= (\vec{w}\,,a)\,(a\vec{v} - \vec{v}\times\vec{w}, \vec{v}\cdot\vec{w}) =$$

$$= \begin{pmatrix} a\,\vec{w}\times\vec{v} - \vec{w}\times\vec{v}\times\vec{w} + (\vec{v}\cdot\vec{w})\vec{w} + a^2\,\vec{v} - a\,\vec{v}\times\vec{w}, \\ \cancel{a\vec{v}\cdot\vec{w}} - \cancel{a\vec{v}\cdot\vec{w}} + \cancel{\vec{w}\cdot(\vec{v}\times\vec{w})} \end{pmatrix} =$$

why?

$$= (a^2\,\vec{v} + 2a\,\vec{w}\times\vec{v} + (\vec{w}\cdot\vec{v})\,\vec{w} - \vec{w}\times\vec{v}\times\vec{w}\,, \qquad 0)$$

of course

78

## Exercise: quaternion norm as a quaternion product

- As you may remember,
  given a complex number $\mathbf{c} \in \mathbb{C}$, $\mathbf{c} = a + ib$
  its magnitude $\|\mathbf{c}\| = \sqrt{a^2 + b^2}$
  can be expressed as
  $$\|\mathbf{c}\|^2 = \mathbf{c}\,\bar{\mathbf{c}}$$
- Does the same hold for quaternions?
  Given $\mathbf{q} \in \mathbb{H}$ :
  $$\|\mathbf{q}\|^2 = \mathbf{q}\,\bar{\mathbf{q}}$$
- Verify, using the multiplication formula we learnt

79

## Quaternions as rotations: summary

- Compact to store (4 scalars, almost the minimum)
- Trivial to invert (just conjugate)
- Fast to composite (just multiply: $2^{nd} * 1^{st}$ )
- Fast to apply (multiply twice, with natural and conjugated)
- Easy to enforce that it stays a rotation (just renormalize)
  - Even after long sequences of cumulations, unlike matrices
- Behaves well under interpolation
  - Just use NLERP – even better with SLERP
  - Remember to take the shortest path (=> flip sign if necessary)
- The favorite representation in 3D games
  - but, other solutions still useful in one context or another

80

## Recap: representing rotations 1/2

| | 3x3 Matrix | Euler Angles |
|---|---|---|
| Space efficient? (in RAM, GPU, storage…) | ★☆☆☆☆ 9 scalars | ★★★★★ 3 scalars (even as small int!) |
| Efficient / easy to: Apply (to points/vectors) | ★★★★☆ 9 products (3 dot products) | ★★☆☆☆ requires trigonometry sin/cos |
| Efficient / easy to: Invert (produce inverse) | ★★★★★ just transpose | ★☆☆☆☆ |
| Efficient / easy to: Composite (with another rotation) | ★★☆☆☆ Matrix multipl (9 dots) Numerical errors | ★☆☆☆☆ |
| Efficient / easy to: Interpolate (with another rotation) | ★☆☆☆☆ Introduces shear/scale | ★☆☆☆☆ easy to do, unintuitive result (⚠ shortest-path required! |
| Intuitive? (e.g. to manually set) | ★★★☆☆ | ★★★★★ roll yaw pitch |
| Notes… | Free extra shear + scale. Useful to extract local axes. | ⚠ GIMBAL LOCK |

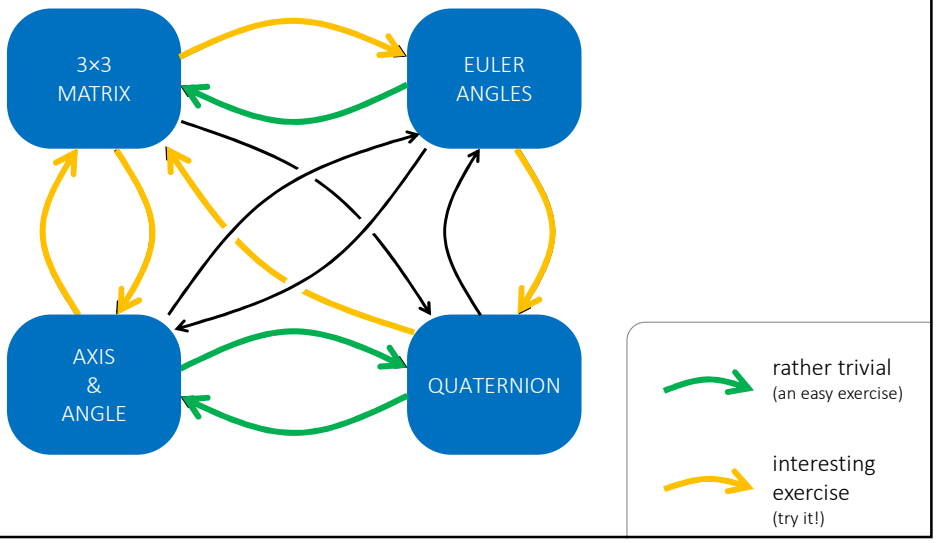
81

## Recap: representing rotations 2/2

| | axis , angle | (unitary) quaternion |
|---|---|---|
| Space efficient? (in RAM, GPU, storage…) | ★★★★☆ 4 scalars (or 3) (precision needed) | ★★★★☆ 4 scalars (precision needed) |
| Efficient / easy to: Apply (to points/vectors) | ★★★☆☆ Requires trigonometry | ★★★★★ Just 2 quat product |
| Efficient / easy to: Invert (produce inverse) | ★★★★★ Just flip axis OR angle | ★★★★★ super easy flip imaginary or real part |
| Efficient / easy to: Composite (with another rotation) | ★☆☆☆☆ | ★★★★★ super easy: 1 quat product |
| Efficient / easy to: Interpolate (with another rotation) | ★★★★☆ | ★★★★★ easy + good result (NLERP or SLERP) |
| Intuitive? (e.g. to manually set) | ★☆☆☆☆ no | ★☆☆☆☆ no |
| Notes… | ⚠ *two* representations for each rotation (flip all → no effect) (for different reasons) Require shortest path! | |

82

## Switching between representations




85

## What defines a rotation, for you?

*« Roll, pitch, and yaw! »*

then you are… a pilot, or an astronaut

*« X-angle, Y-angle, and Z-angle! »*

then you are… a digital artist (like an animator, or a scener)

*« An angle! »*

then you are… a flatland citizen

*« A vector! the dir is the axis the magnitude the angle »*

then you are… a physicist

*« A 3x3 matrix! the submatrix of a 4x4 transform »*

then you are… a computer graphicist, or a Graphics API

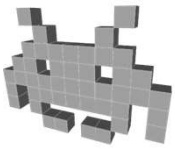
*« A quaternion! »*

then you are… a game developer

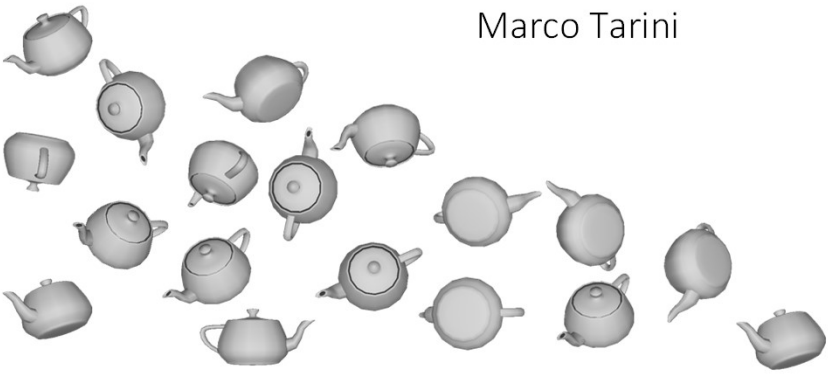
86

Master Game Dev

# Transformations in games: notes about game engines



Marco Tarini

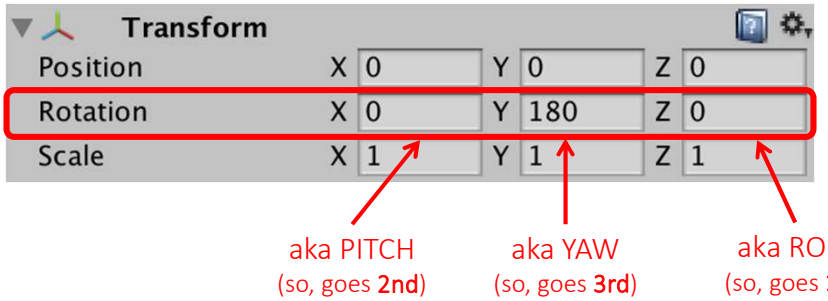


87

## Notes on transformation in unity
## What you see in the GUI

Rotations as Euler angles

- the intuitive choice! (perfect for a GUI)




note: exposed as degrees, not radians
--> even more intuitive

88

## Notes on rotations in unity internally

Rotations as Quaternion (a class)

- can be initialized (via constructors) as a …
  quaternion, euler angles, axis+angle, or matrix
- can be converted to / accessed as… any of the above
  (thanks to methods, or to "properties" -
  basically setter/getter methods in disguise+
  that gives the illusion for a "quaternion" to be whichever type you think it is
- Support for cumulation (multiplication),
  inversion (conjugation), interpolation (SLERP) …
- Support for all the common problems
  (from-to rotation, etc)

89

## Notes on Rotations in UNREAL ENGINE

fields: `W X Y Z`

Class `FQuat` :

- convert from:
  - axis+angle, matrix4x4, Rotator, euler (vec3) (by constructors)
  - Euler angles (`makeFromEuler` method)
  - From-to vector pairs (`FindBetween` method)
- convert to:
  - `ToAxisAndAngle`, `Euler`, `Rotator`,
  - matrix columns `GetAxis`(`X`|`Y`|`Z`)
  - also, with names: `Get`(`Forward`|`Right`|`Up`)`Vector`,
- methods: invert with `Inverse`,
  blend with `FastSlerp`
  or `FastSlerpFullPath` (no shortest path)
  apply with `RotateVector` / `UnrotateVector`
  composite with `operator *`

Class `FRotator`
for "nautical" Euler angles:
fields: `Pitch Roll Yaw`

90

## Notes on Rotations in Godot (C# game engine)

Class **`Basis`**

- a rotation as a 3x3 matix
- Its x,y,z fields (of type Vector3) are the columns
  (that is, the x,y,z axis of local space define in global space)
- Warning: doesn't have to be orthonormal.
  For example, can include scale
- Has a method to enforce back orthonormality

Class **`Quaternion`**

- a rotation as a quaternion
- Includes method for SLERP-ing

The two classes can be converted to/from each other (via constructors)
Both include methods for cumulation (multiplication), application, inversion.
Both include methods for conversion to/from axis-angle.
Euler angles: no explicit support (but naturally you can go from 3 Euler angles to either class by multiplying rotations around x, y, z axis, but no built-in order)

91

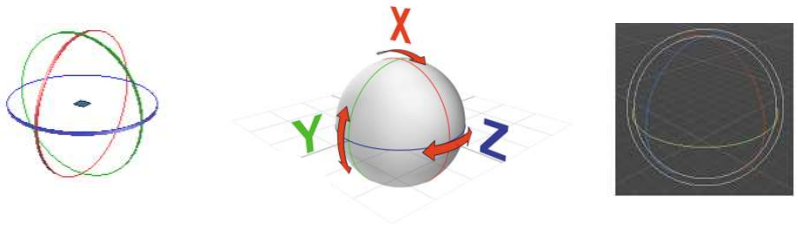
## Notes on rotations in OpenGL (Computer Graphics Library)

- In the «old school» API:
  (and now in many similar libraries)
  - API: glRotate3f
    - takes: angle & axis
  - Internally:
    - matrices
    - jointly as with any other spatial transform
    - separated in MODEL+VIEW+PROJECT transforms

92

## GUI: how do artists author 3D rotations?

- Typical way: rotation gizmo
  - (also: «arcball» or «trackball»)
  - 3 handles to control the three Euler angles
  - or "free", drag-n-drop mode (trackball metaphor)



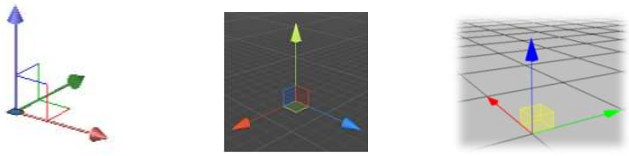

convention: Red = X Green = Y Blue = Z

93

## GUI: how do artists author 3D translations?

translation gizmo

- handles to traslate along axes or planes

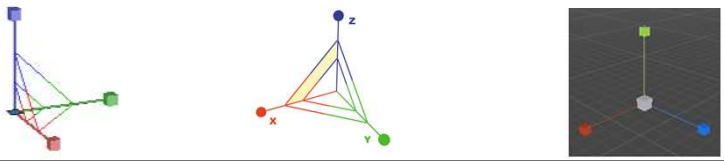


convention:
Red = X
Green = Y
Blue = Z

## GUI: how to author 3D scalings?

scale gizmo

- 3 handles for anisotropic scalings
  1 handle (middle) for uniform scalings



96