GDC

Practical Quaternions – an easy guide to 3d rotations for non-mathematicians

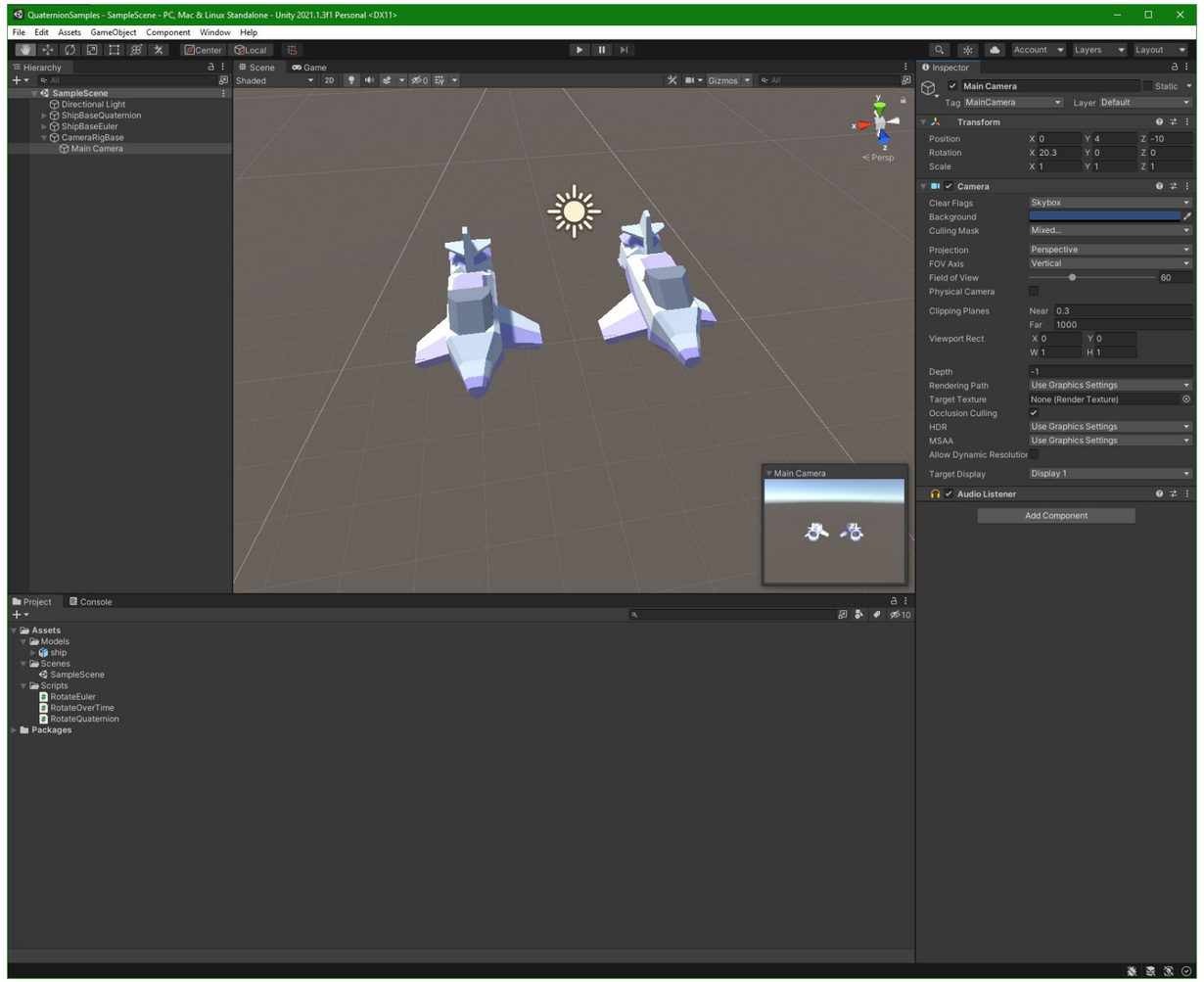
Patrick Martin
Developer Relations Engineer @ Google

Who am I?

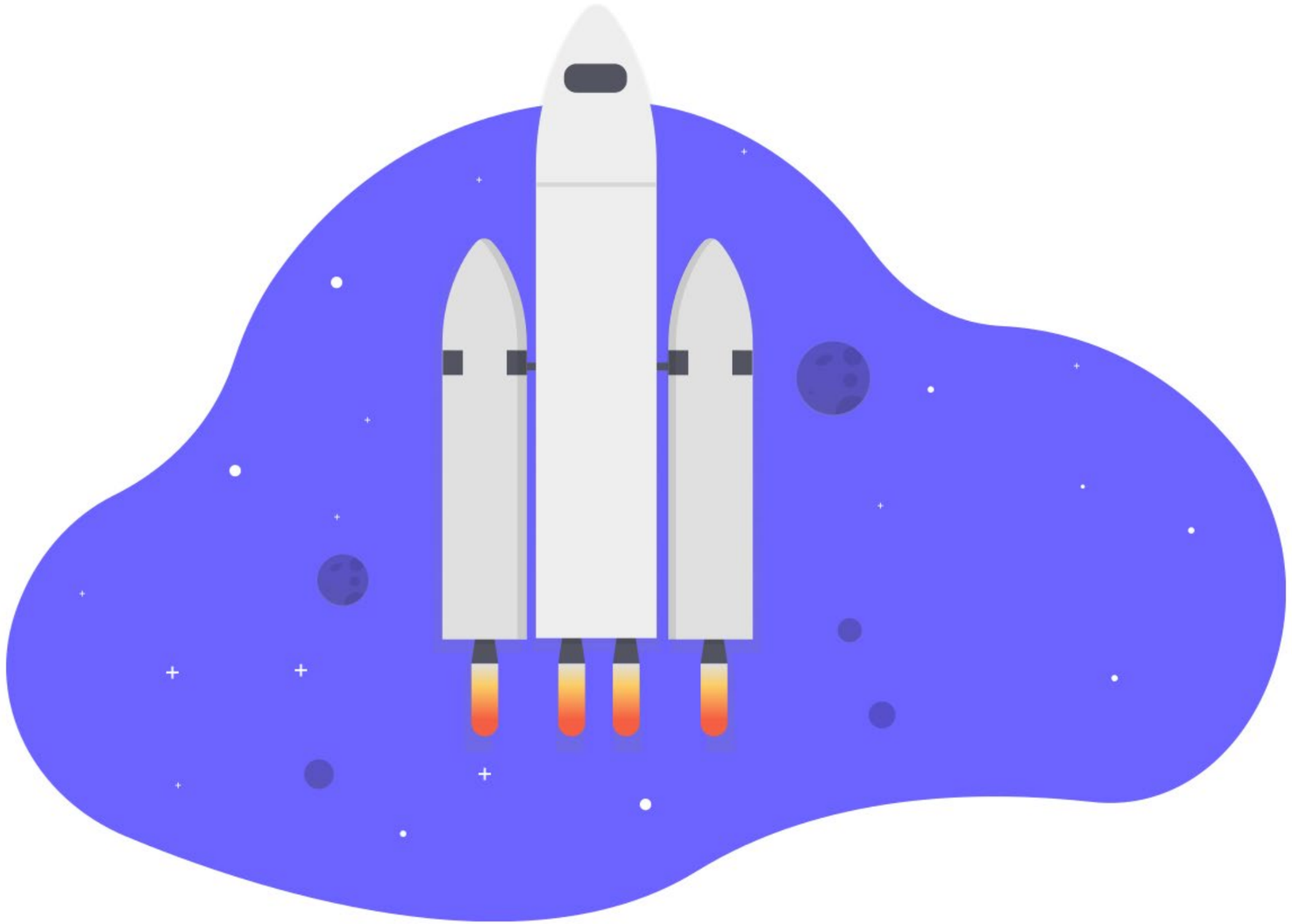
- Mobile games at Venan
- Robotics and AR at Firebase
- Developer Advocate for Firebase and Android


BANK HOURS
SATURDAY: CLOSED

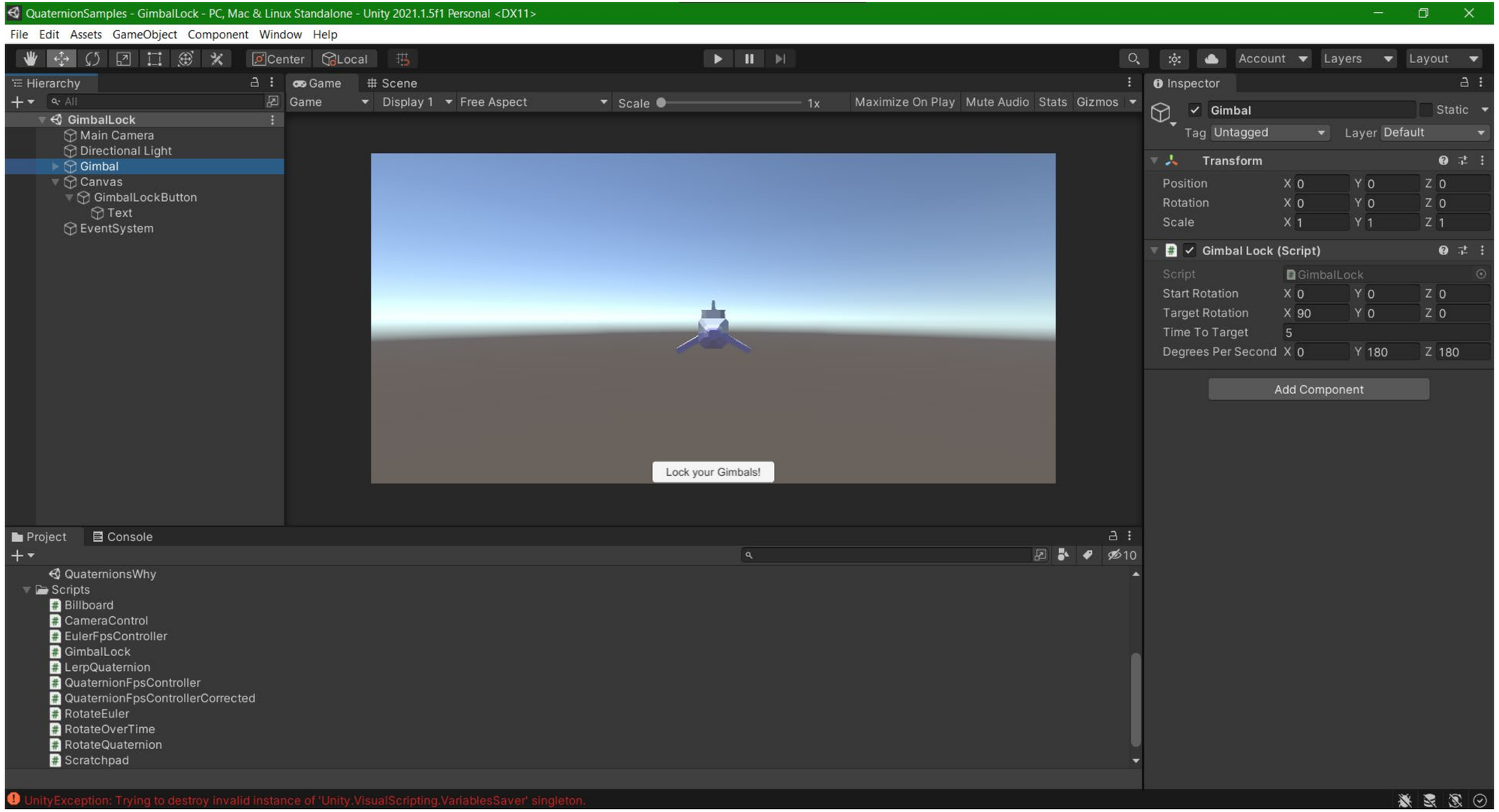
Demo



```
public class RotateQuaternion : MonoBehaviour
{
    [SerializeField] private Vector3 _euler;
    [SerializeField] private float _interpolationTime = 1f;
    private Quaternion _sourceRotation;
    private Quaternion _targetRotation;

    // Start is called before the first frame update
    void Start()
    {
        _sourceRotation = transform.rotation;
        _targetRotation = Quaternion.Euler(_euler);
    }

    // Update is called once per frame
    void Update()
    {
        float amount = (Time.timeSinceLevelLoad / _interpolationTime) % 2f;
        if (amount > 1)
        {
            amount = 2f - amount;
        }
        transform.localRotation = Quaternion.Slerp(_sourceRotation, _targetRotation, amount);
    }
}
```



Demo



The Rules

7 simple rules for understanding quaternions

You won't believe number 4

The Rules

The “imaginary component” is a scary way of saying “the axis of rotation”.

The Rules

For convenience, half of the rotation is in the real part and half is in the imaginary.

The Rules

Multiplication is a fancy way of saying “apply a rotation”

The Rules

“inverse” and “conjugate” are both fancy ways of saying negate the imaginary component.

The Rules

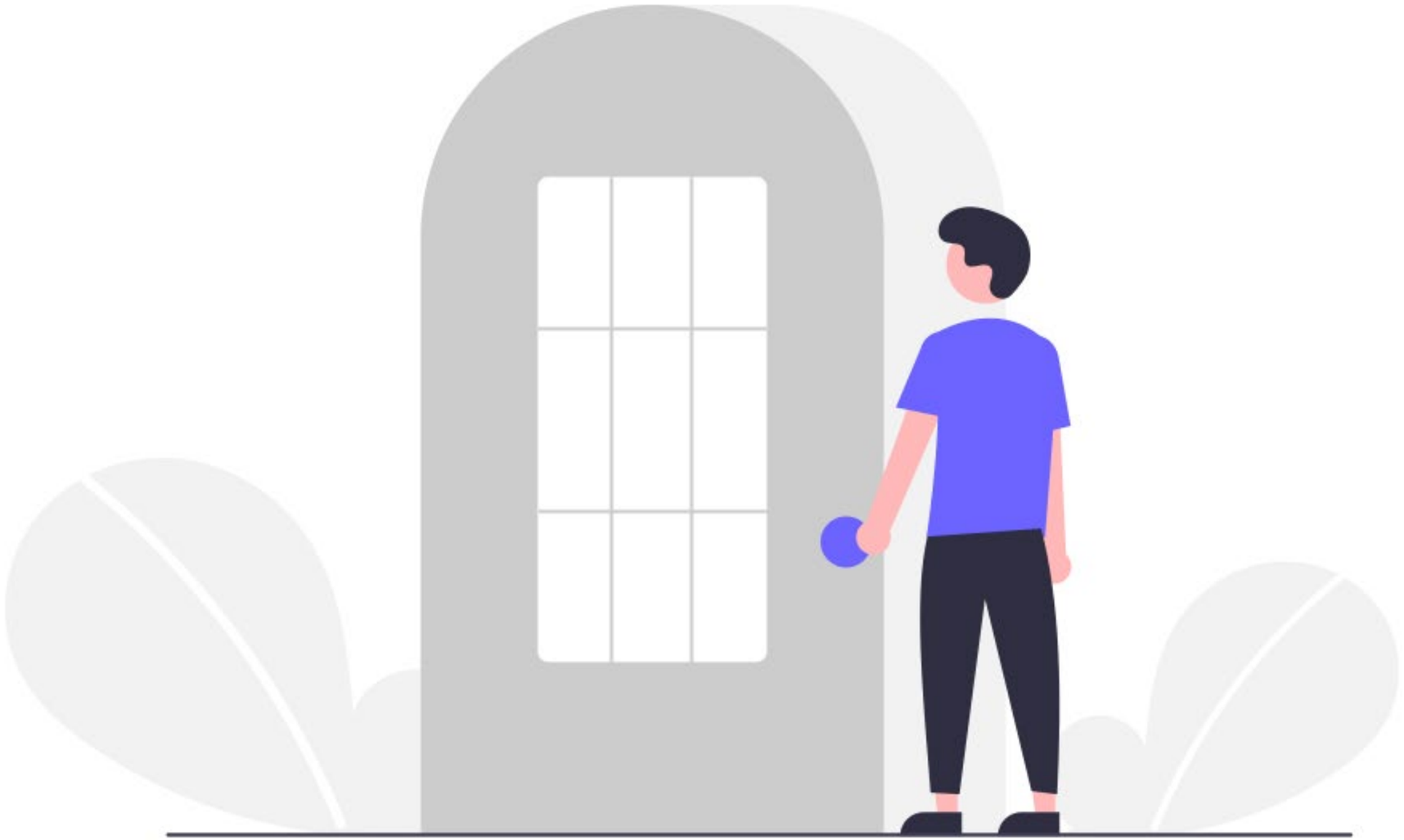
Just like LERPing between vectors you SLERP between quaternions

The Rules

If you go from a right handed coordinate system to a left, you’re negating one of your imaginary components.

The Rules

Trust the library.



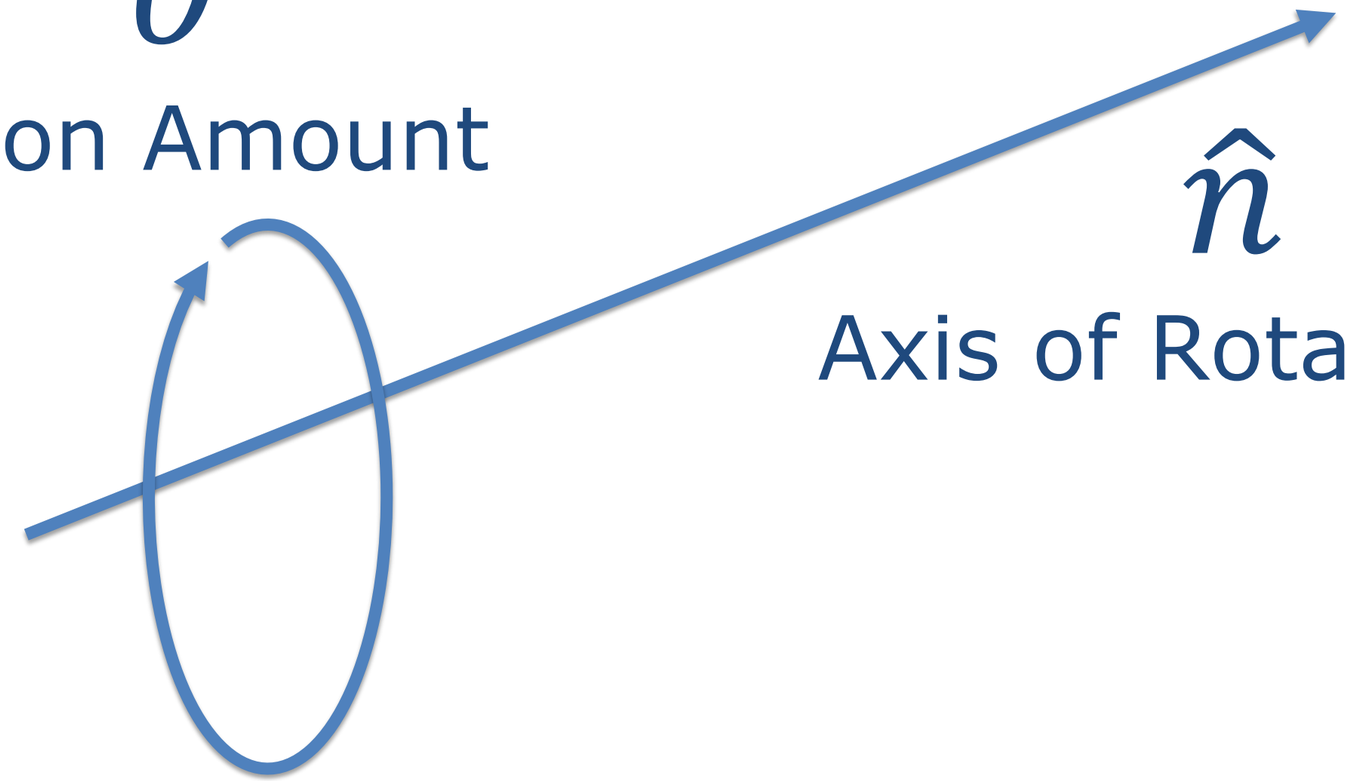
Visualizing a Quaternion

- “The imaginary part is the axis of rotation”
- “Half the rotation is in the real part and half is in the imaginary”

Visualizing a Quaternion

θ

Rotation Amount



$\hat{n}$

Axis of Rotation

Anatomy of a Quaternion

Anatomy of a Quaternion

$$\begin{bmatrix} s & \hat{n_x} & \hat{n_y} & \hat{n_z} \end{bmatrix}$$

Real

Imaginary
(axis of rotation)

Anatomy of a Quaternion

$$\begin{bmatrix} s & \hat{n} \end{bmatrix}$$

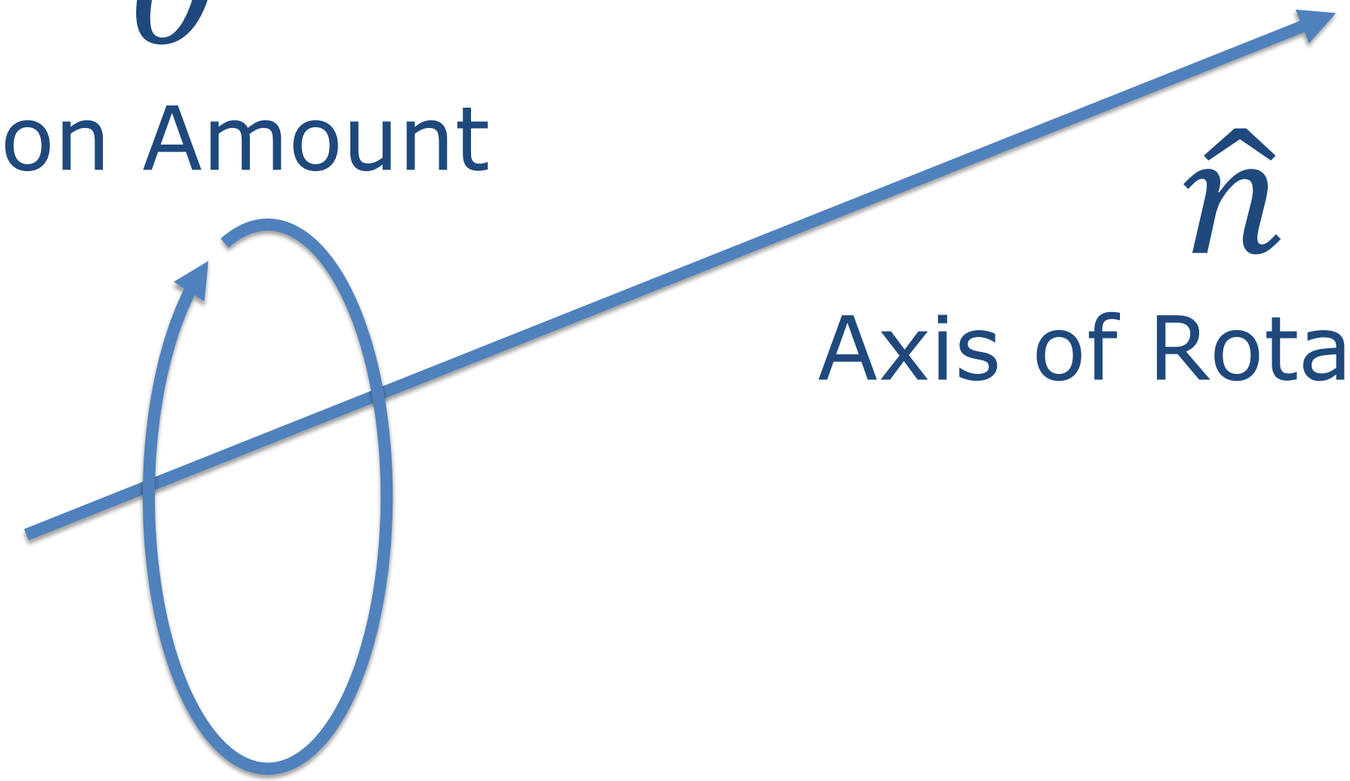
Real

Imaginary
(axis of rotation)

Visualizing a Quaternion

θ

Rotation Amount



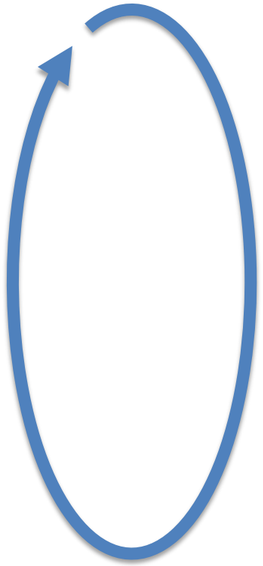
$\hat{n}$

Axis of Rotation

Visualizing a Quaternion

θ

Rotation Amount



Anatomy of a Quaternion

$$\left[\underbrace{\cos\left(\frac{\theta}{2}\right)}_{\text{Real}} \quad \underbrace{\sin\left(\frac{\theta}{2}\right)\hat{n}}_{\text{Imaginary}}\right]$$

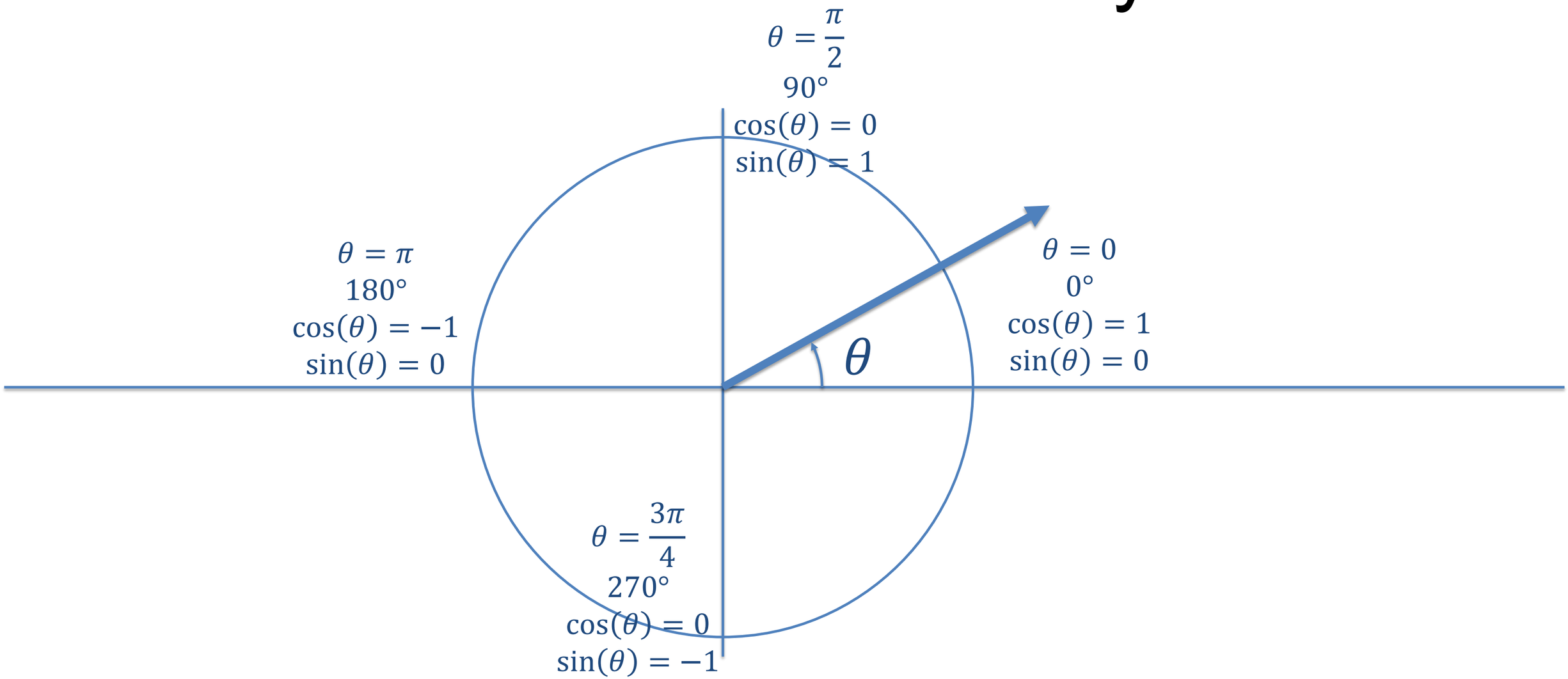
Anatomy of a Quaternion

$$\left(\underbrace{\cos\left(\frac{\theta}{2}\right)}_{\text{Real}} \quad \underbrace{\begin{bmatrix} \sin\left(\frac{\theta}{2}\right)\widehat{n_x} \\ \sin\left(\frac{\theta}{2}\right)\widehat{n_y} \\ \sin\left(\frac{\theta}{2}\right)\widehat{n_z} \end{bmatrix}}_{\text{Imaginary}} \right)$$

Anatomy of a Quaternion

You lied, we’re doing math!

Unit Circle – screenshot me if you need!



Anatomy of a Quaternion

$$\left[\underbrace{\cos\left(\frac{\theta}{2}\right)}_{\text{Real}} \quad \underbrace{\sin\left(\frac{\theta}{2}\right)\hat{n}}_{\text{Imaginary}}\right]$$

```
var identity = Quaternion.identity;
var identityAngleAxis = Quaternion.AngleAxis(0, Vector3.up);
Debug.Log($"Testing identity {identity} and {identityAngleAxis}");
```

Visualize a Quaternion

Anatomy of a Quaternion

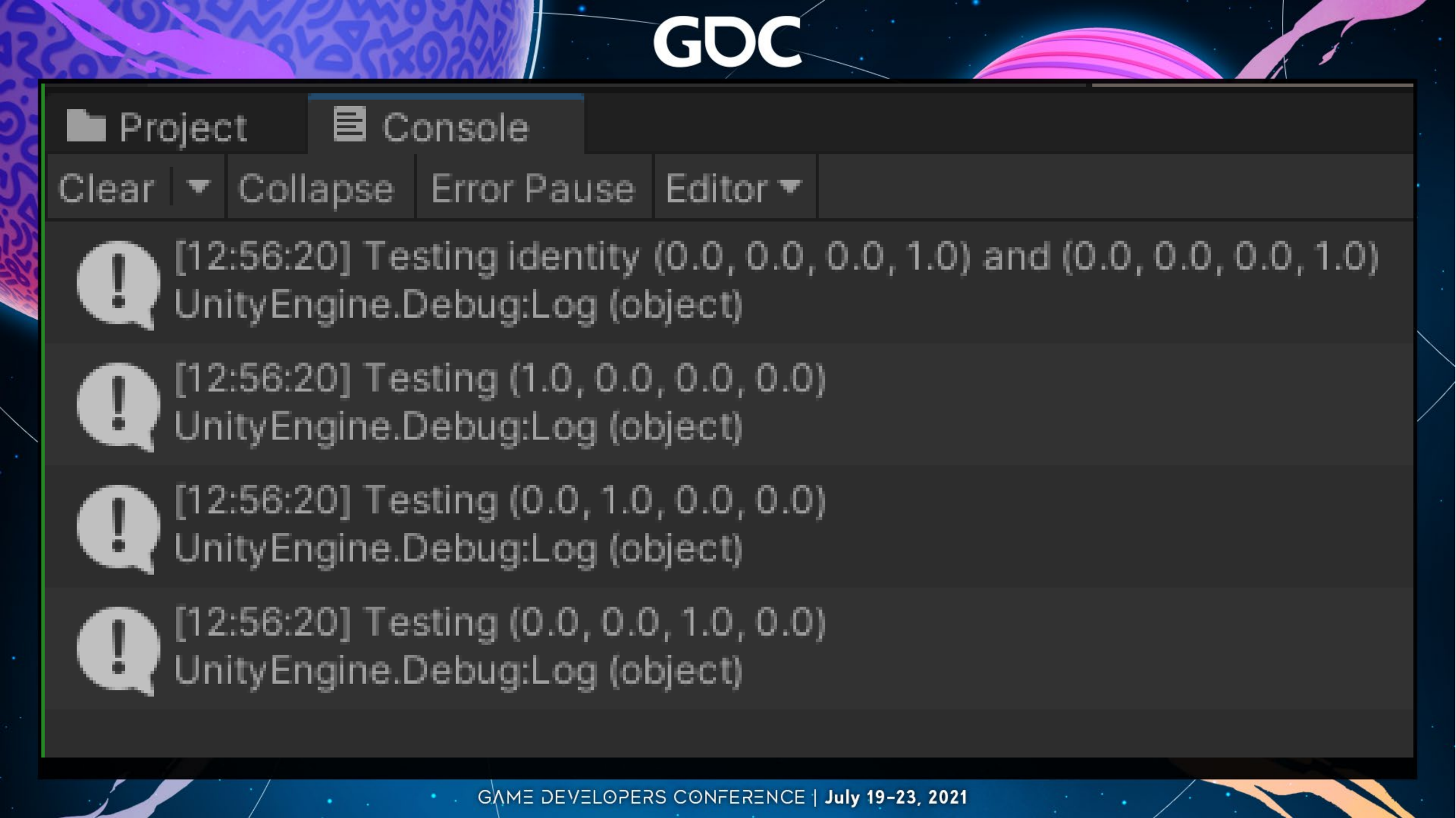
$$\left[\underbrace{\cos\left(\frac{\theta}{2}\right)}_{\text{Real}} \quad \underbrace{\sin\left(\frac{\theta}{2}\right)\hat{n}}_{\text{Imaginary}}\right]$$

```
var pitch = Quaternion.AngleAxis(180, Vector3.right);
Debug.Log($"Testing {pitch}");
```

Visualize a Quaternion

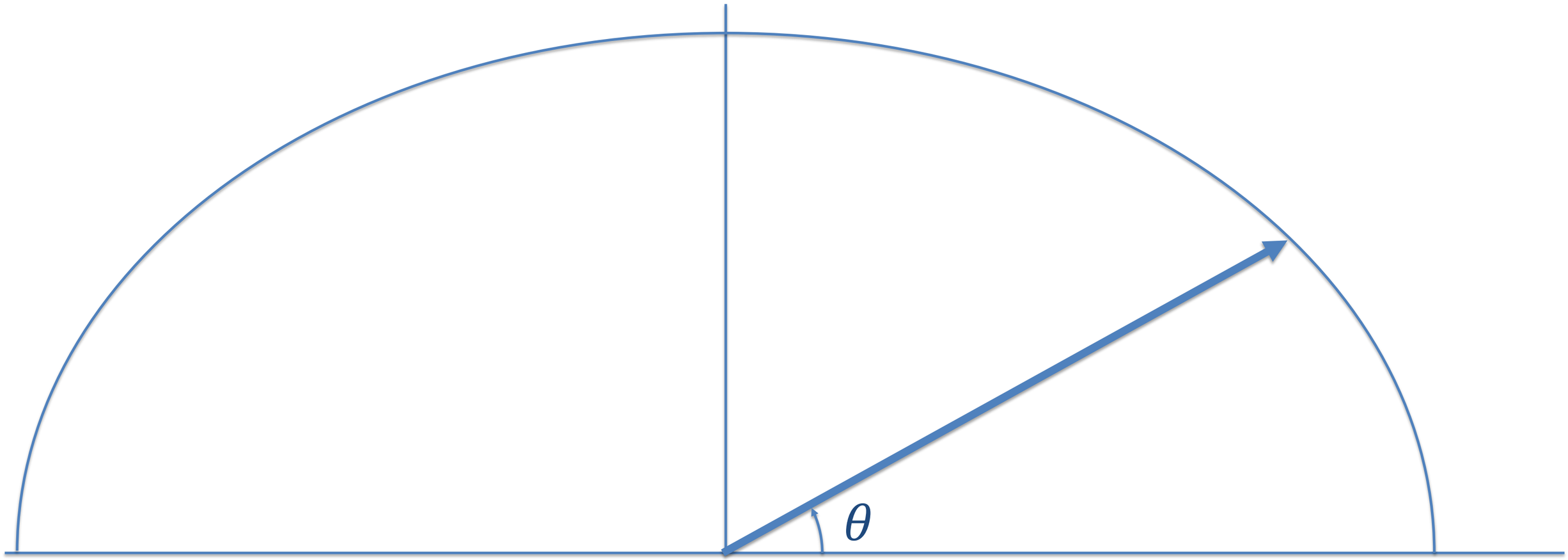
$$\begin{bmatrix} \underbrace{0}_{\text{Real}} & \underbrace{1 \quad 0 \quad 0}_{\text{Imaginary}} \end{bmatrix}$$

```
var yaw = Quaternion.AngleAxis(180, Vector3.up);
Debug.Log($"Testing {yaw}");

var roll = Quaternion.AngleAxis(180, Vector3.forward);
Debug.Log($"Testing {roll}");
```


GDC
Project
Console
Clear
Collapse
Error Pause
Editor
[12:56:20] Testing identity (0.0, 0.0, 0.0, 1.0) and (0.0, 0.0, 0.0, 1.0)
UnityEngine.Debug:Log (object)
[12:56:20] Testing (1.0, 0.0, 0.0, 0.0)
UnityEngine.Debug:Log (object)
[12:56:20] Testing (0.0, 1.0, 0.0, 0.0)
UnityEngine.Debug:Log (object)
[12:56:20] Testing (0.0, 0.0, 1.0, 0.0)
UnityEngine.Debug:Log (object)
GAME DEVELOPERS CONFERENCE | July 19-23, 2021

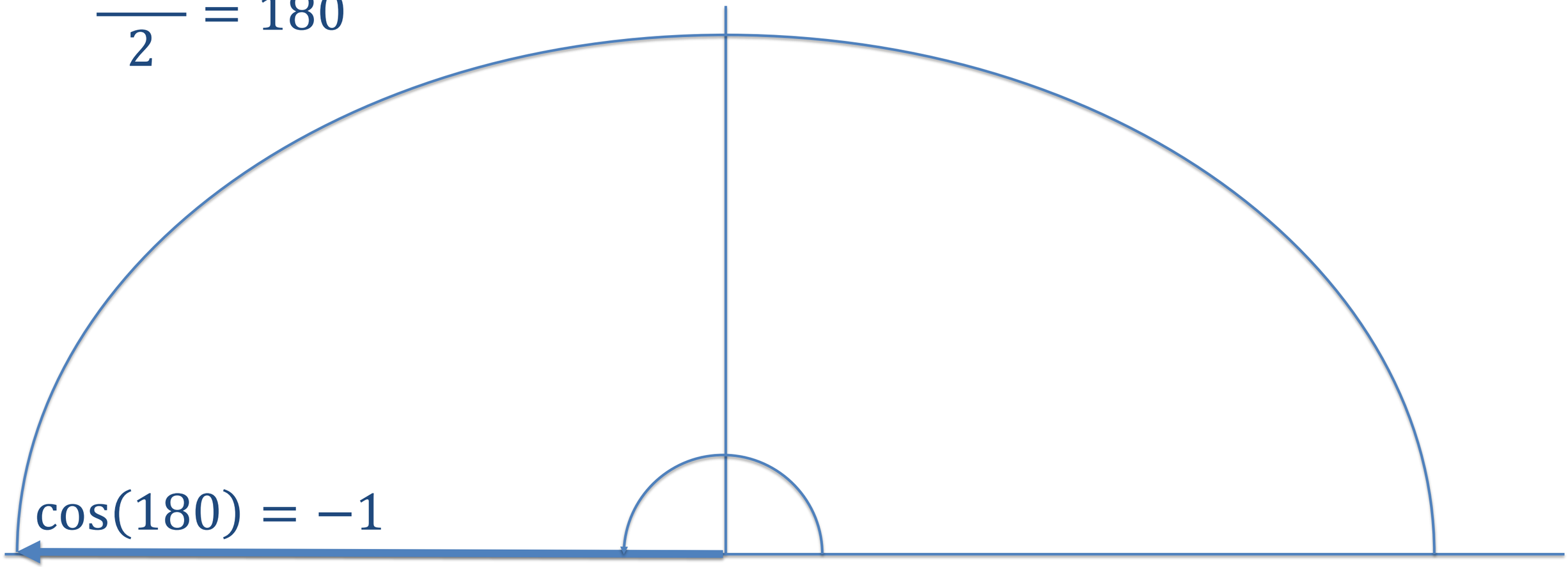
Visualize a Quaternion

Visualize a Quaternion



Visualize a Quaternion

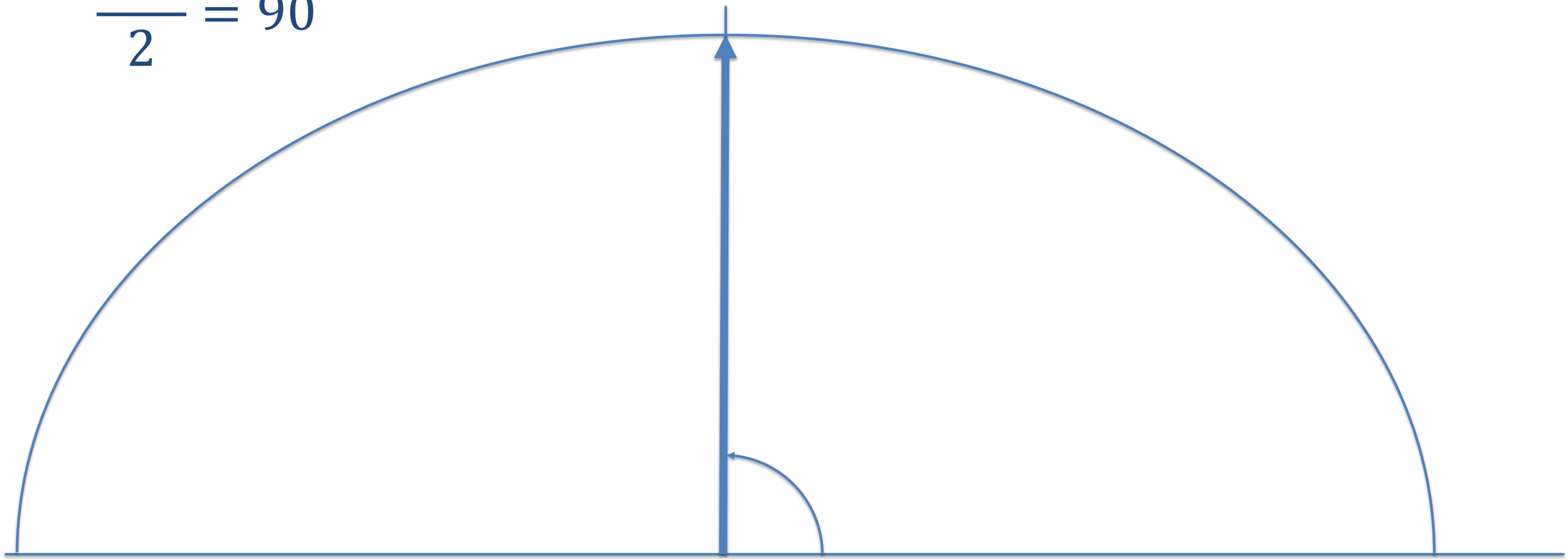
$$\frac{360}{2} = 180$$



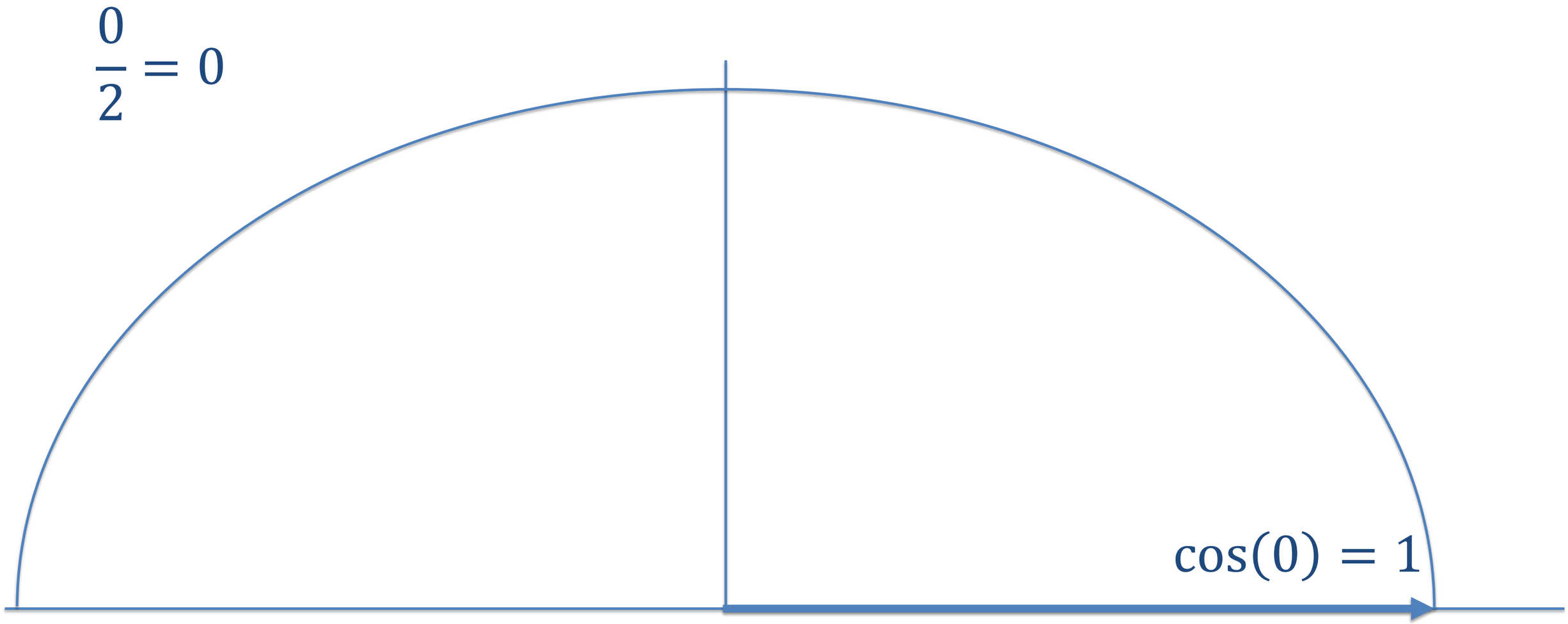
Visualize a Quaternion

$$\frac{180}{2} = 90$$

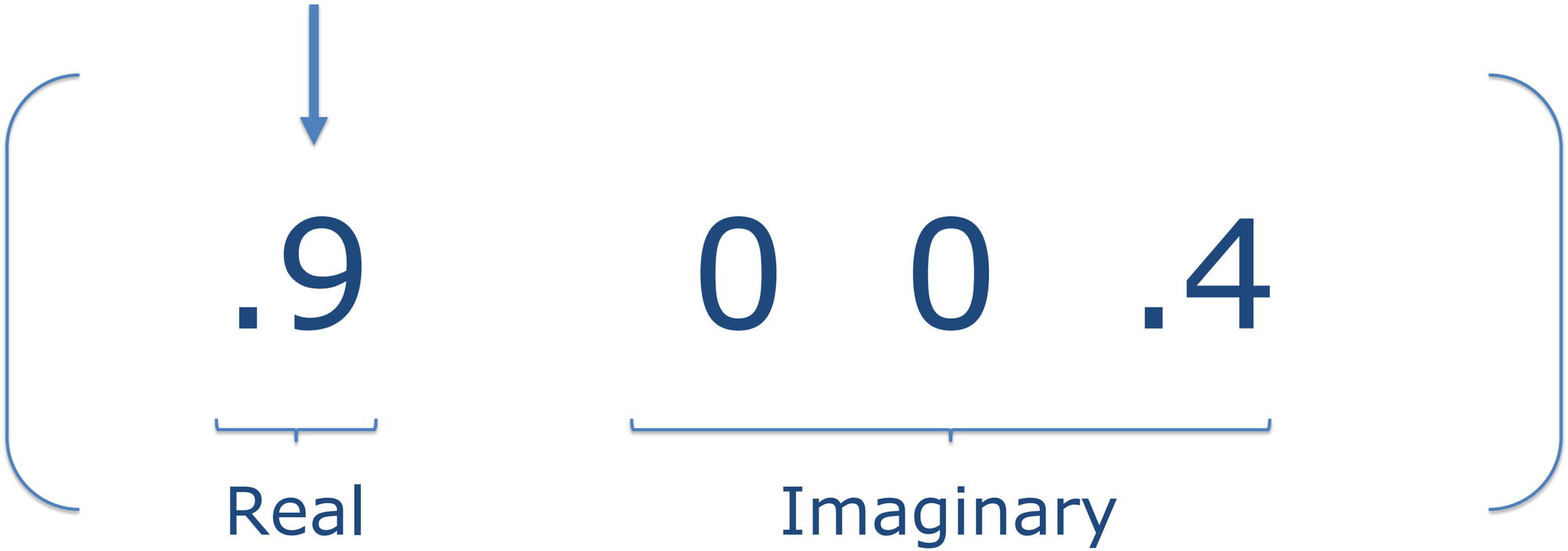
$$\cos(90) = 0$$



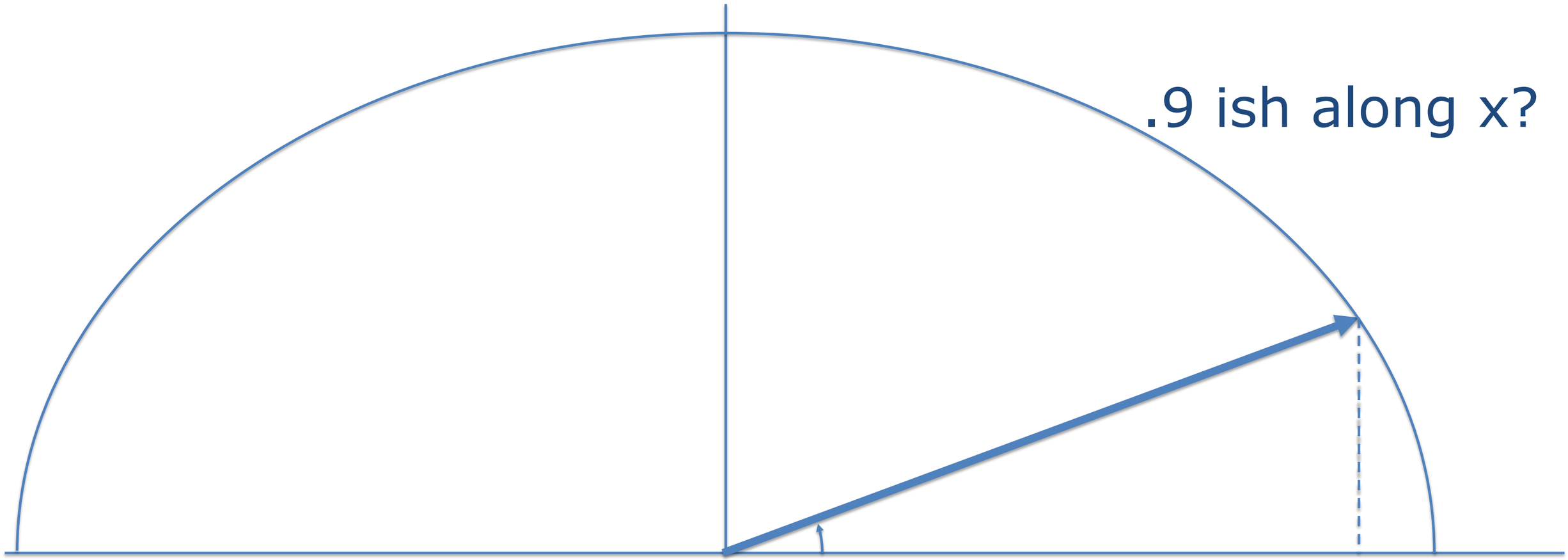
Visualize a Quaternion



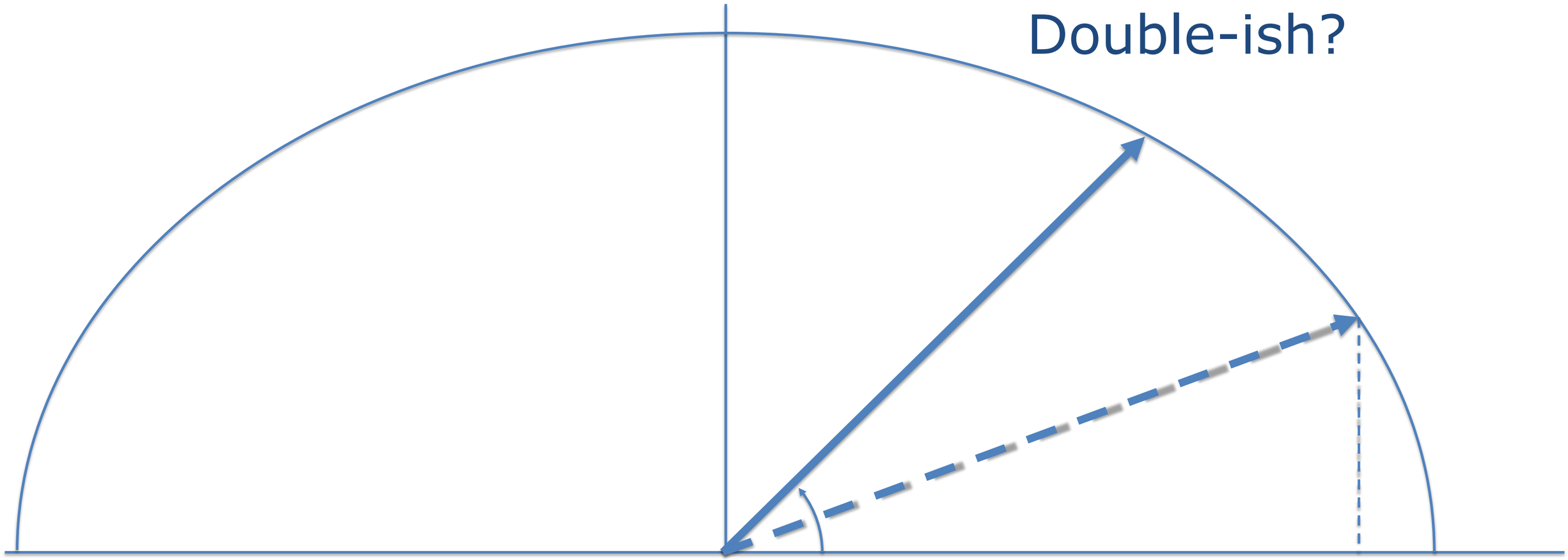
Visualize a Quaternion



Visualize a Quaternion

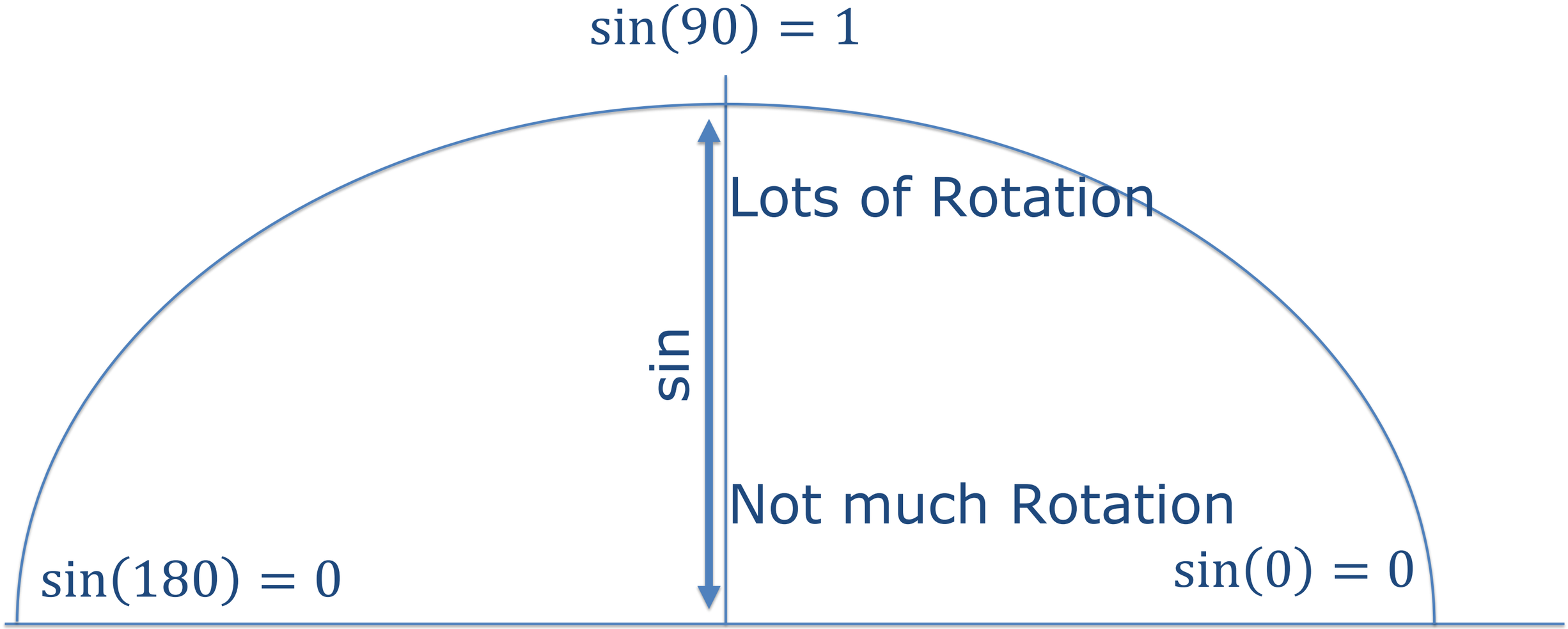


Visualize a Quaternion



Visualize a Quaternion

Visualize a Quaternion



Visualize a Quaternion

$$\cos^{-1}(.9) \cdot 2 \approx 51°$$

```
var randomRoll = Quaternion.AngleAxis(45, Vector3.forward);
Debug.Log($"Random roll: {randomRoll}");
```

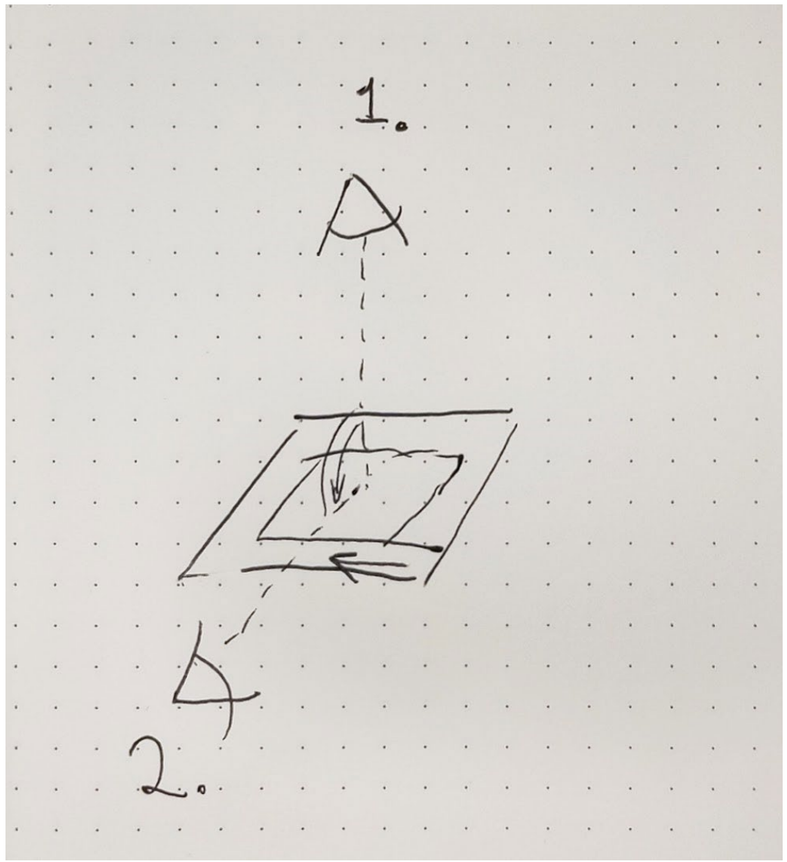
```
var randomRollNegated = Quaternion.AngleAxis(-45, Vector3.forward);
Debug.Log($"Random roll negated: {randomRollNegated}");
```

.9 0 0 -.4

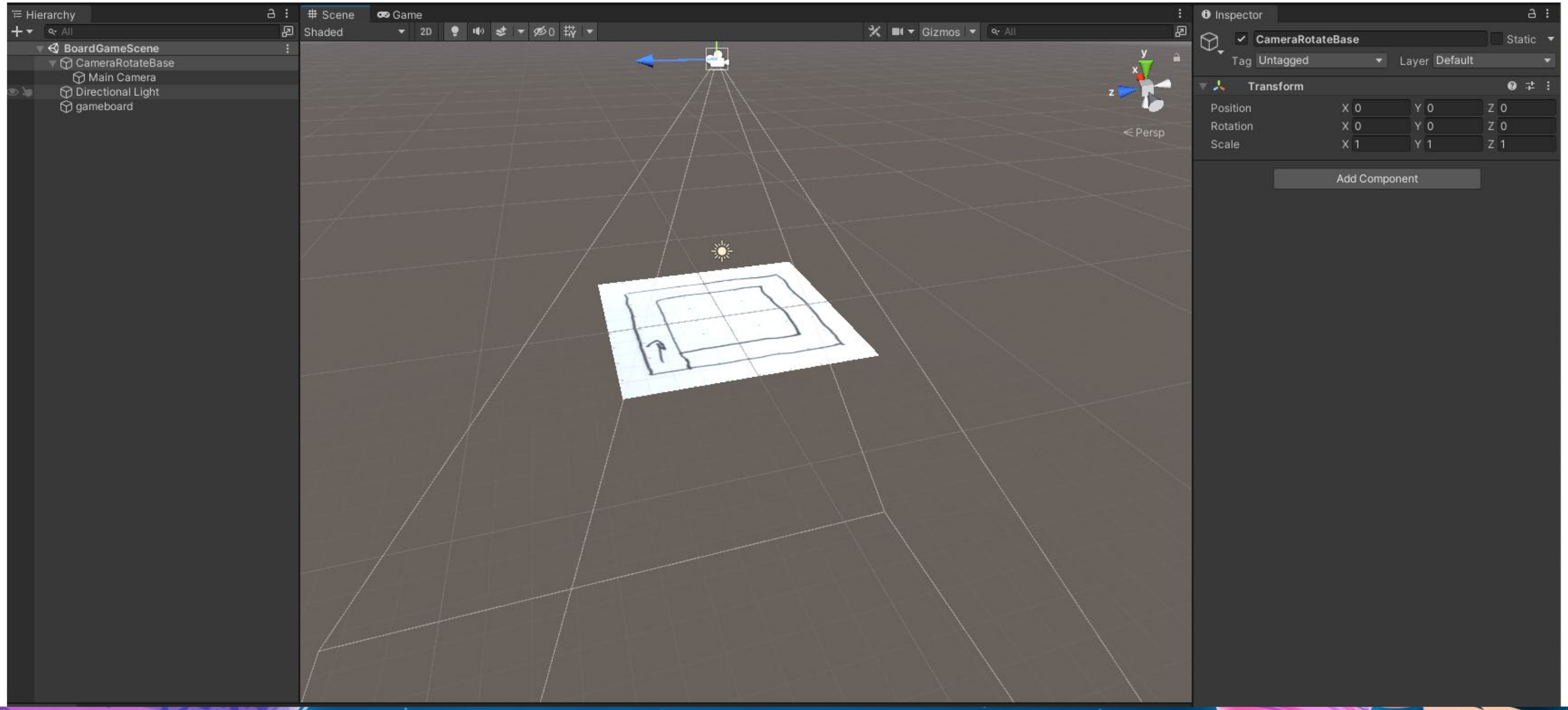
Real Imaginary

Visualize a Quaternion

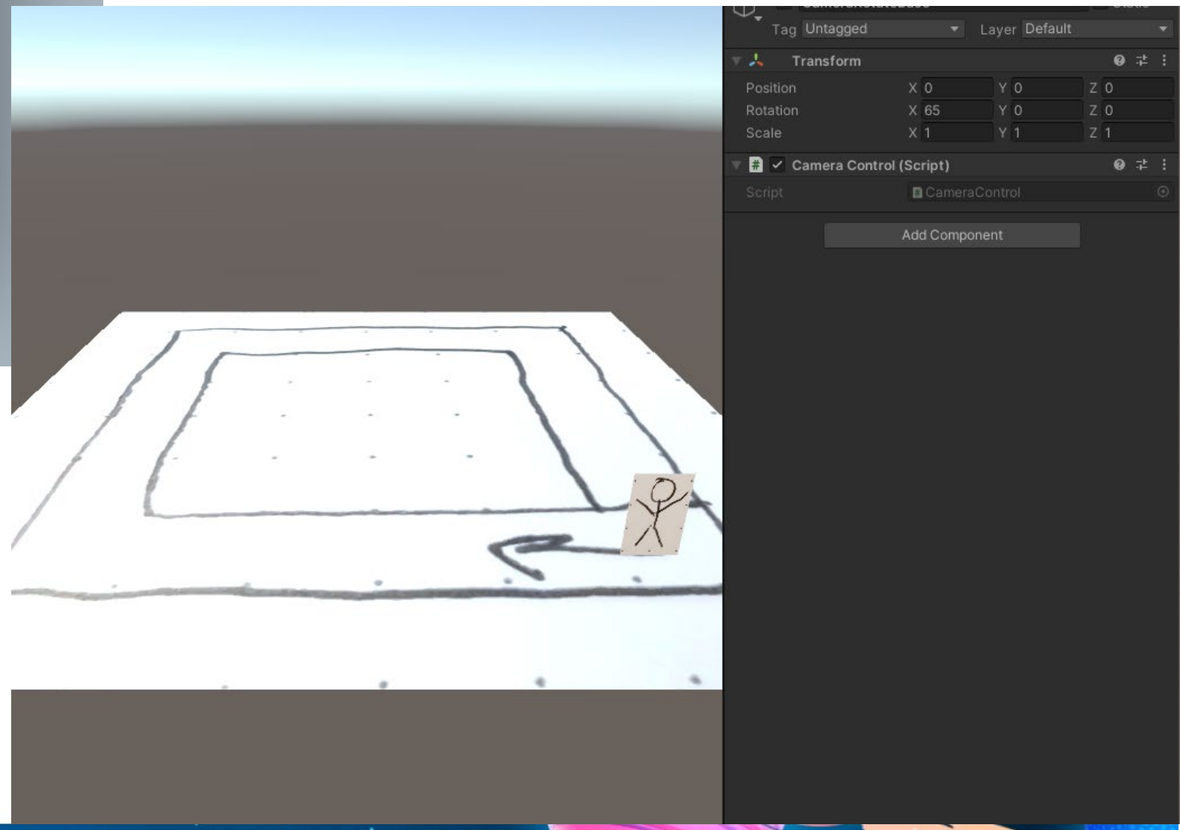
The board game problem



The board game problem

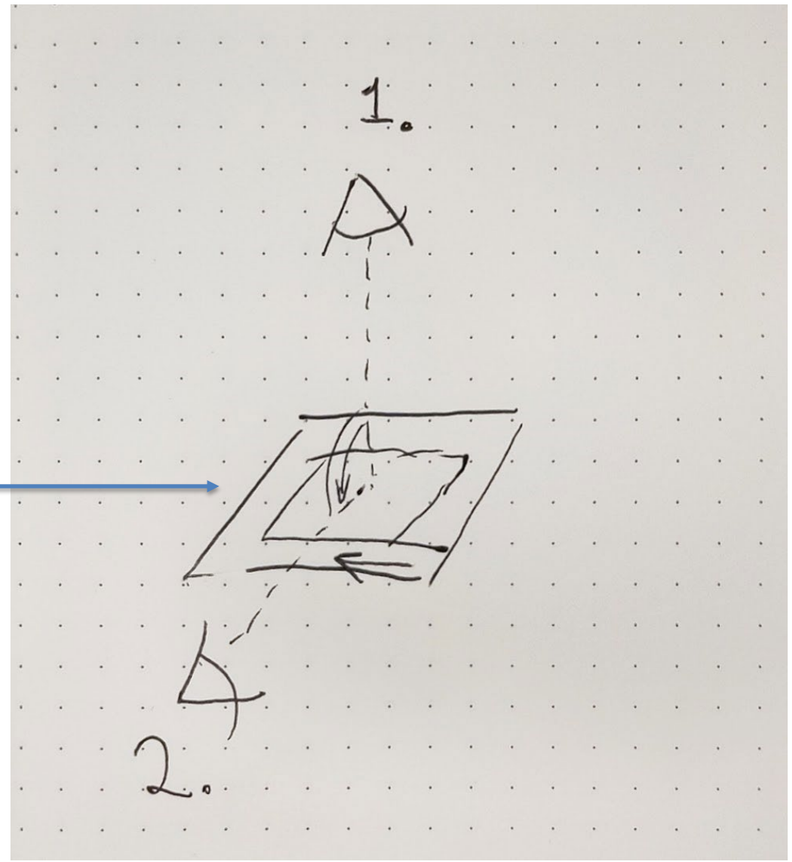


```
using UnityEngine;

public class CameraControl : MonoBehaviour
{
    Quaternion _origin = Quaternion.identity;
    Quaternion _homeRow = Quaternion.AngleAxis(65, Vector3.right);

    // Start is called before the first frame update
    void Start()
    {
        transform.localRotation = _homeRow;
    }
}
```



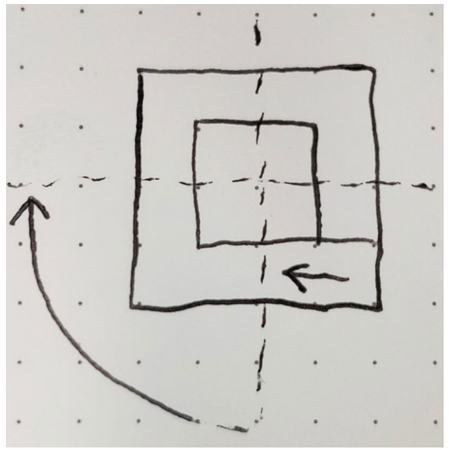
The board game problem

?

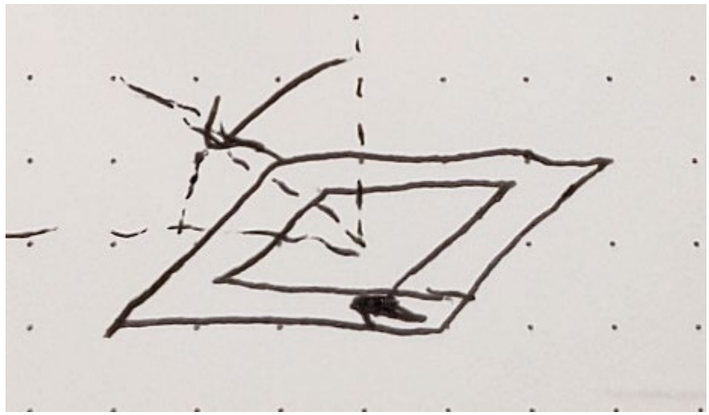


The board game problem

1



2



Multiplying Quaternions

- Multiplication is the same as “apply a rotation”

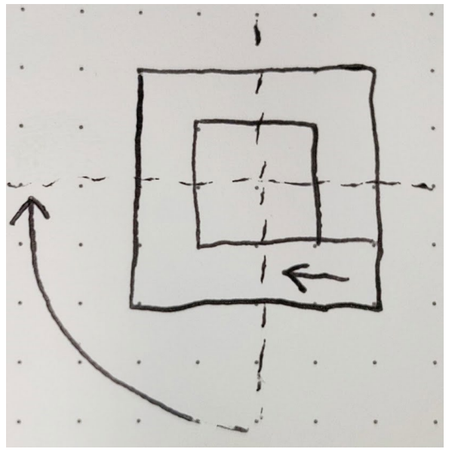
Multiplying Quaternions

$$p = [p_s\ p_v]$$
$$q = [q_s\ q_v]$$
$$pq = (p_s q_s - p_v \cdot q_v) + (p_s q_v + q_s p_v + p_v \times q_v)$$

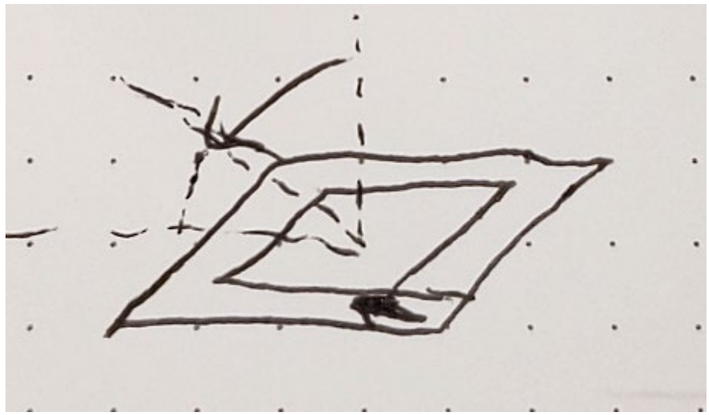
Source, Wikipedia: https://en.wikipedia.org/wiki/Quaternion#Quaternions_and_the_space_geometry

Multiplying Quaternions

q_0



q_1



$$q' = q_1 q_0$$

Multiplying Quaternions

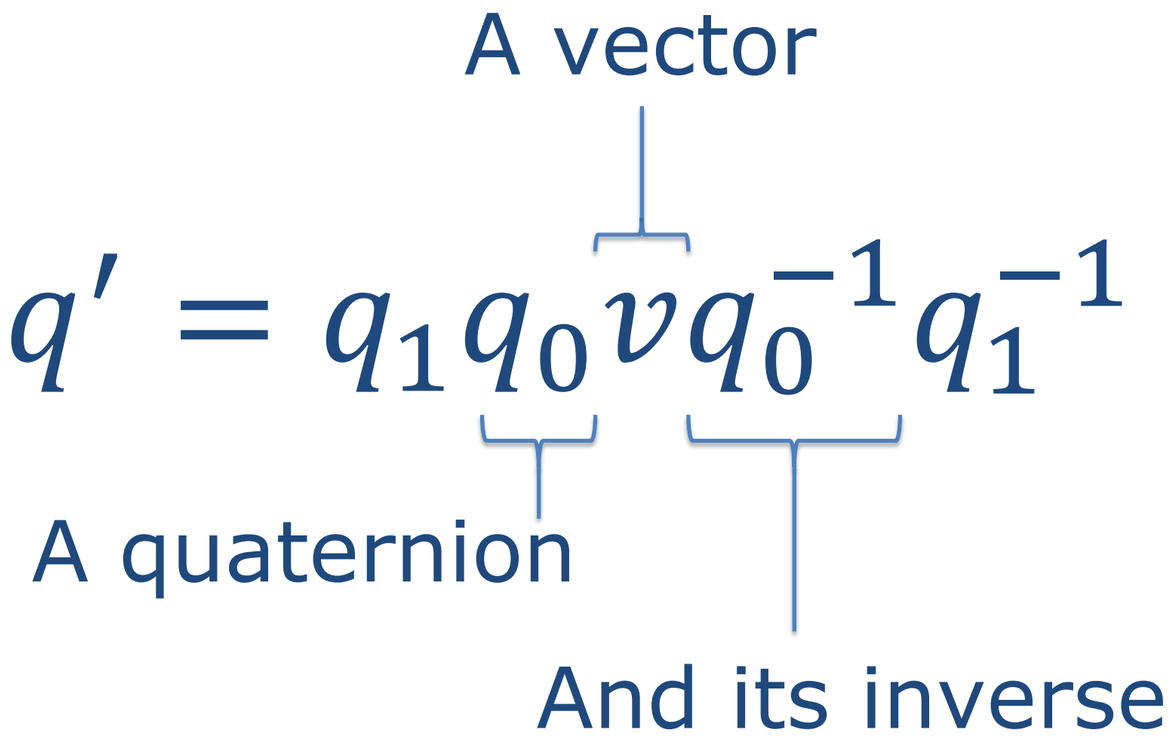
- Are you sure you know what you’re doing?
- Space is relative!

Multiplying Quaternions

- Look at the camera feed, this is a physical demo

Multiplying Quaternions

- Unity is the opposite of this for unknown reasons.



Multiplying Quaternions

- Unity is the opposite of this for unknown reasons.

From the inside out

$$q' = q_1 q_0 v q_0^{-1} q_1^{-1}$$

Inverting a Quaternion

- The inverse just means negating the imaginary part.

Inverting a Quaternion

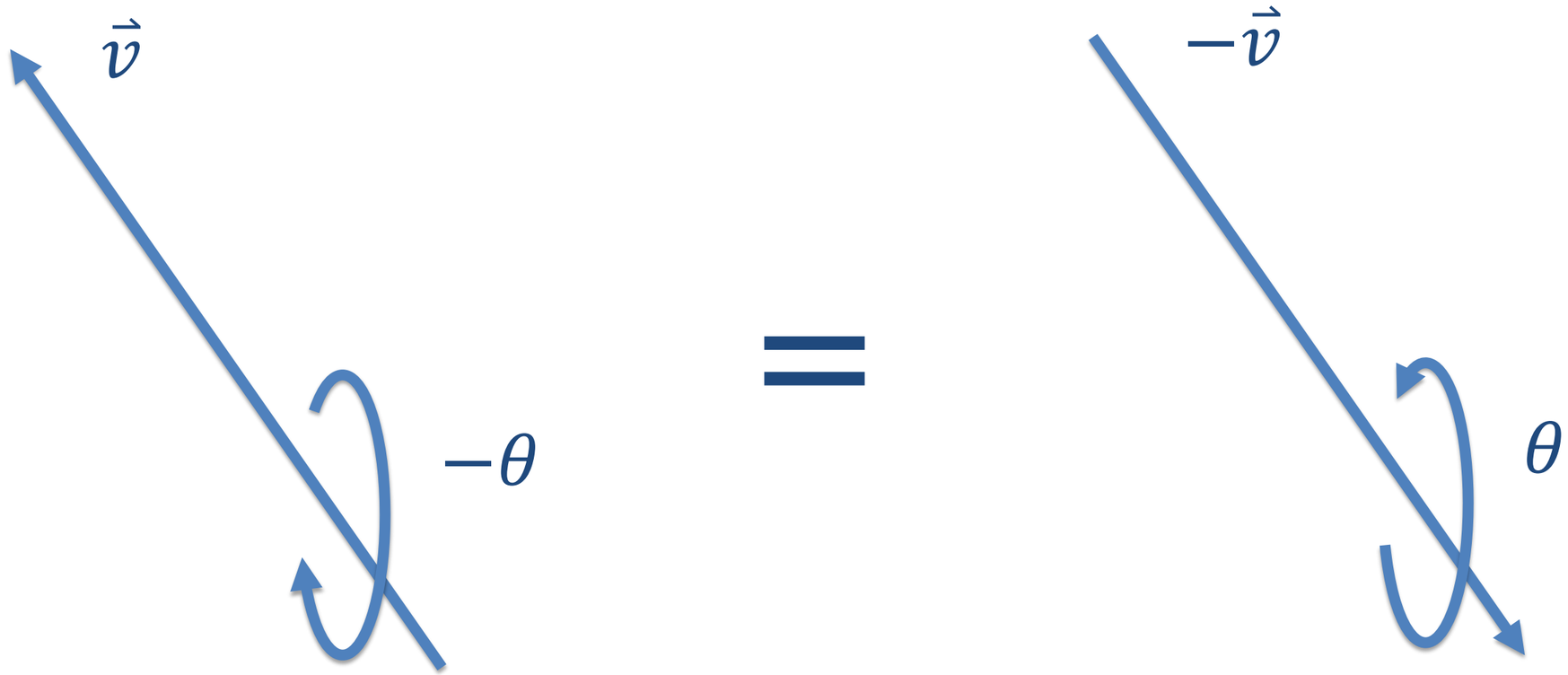
$$q^{-1} = \frac{[s \quad -i \quad -j \quad -k]}{s^2 + i^2 + j^2 + k^2}$$

Inverting a Quaternion

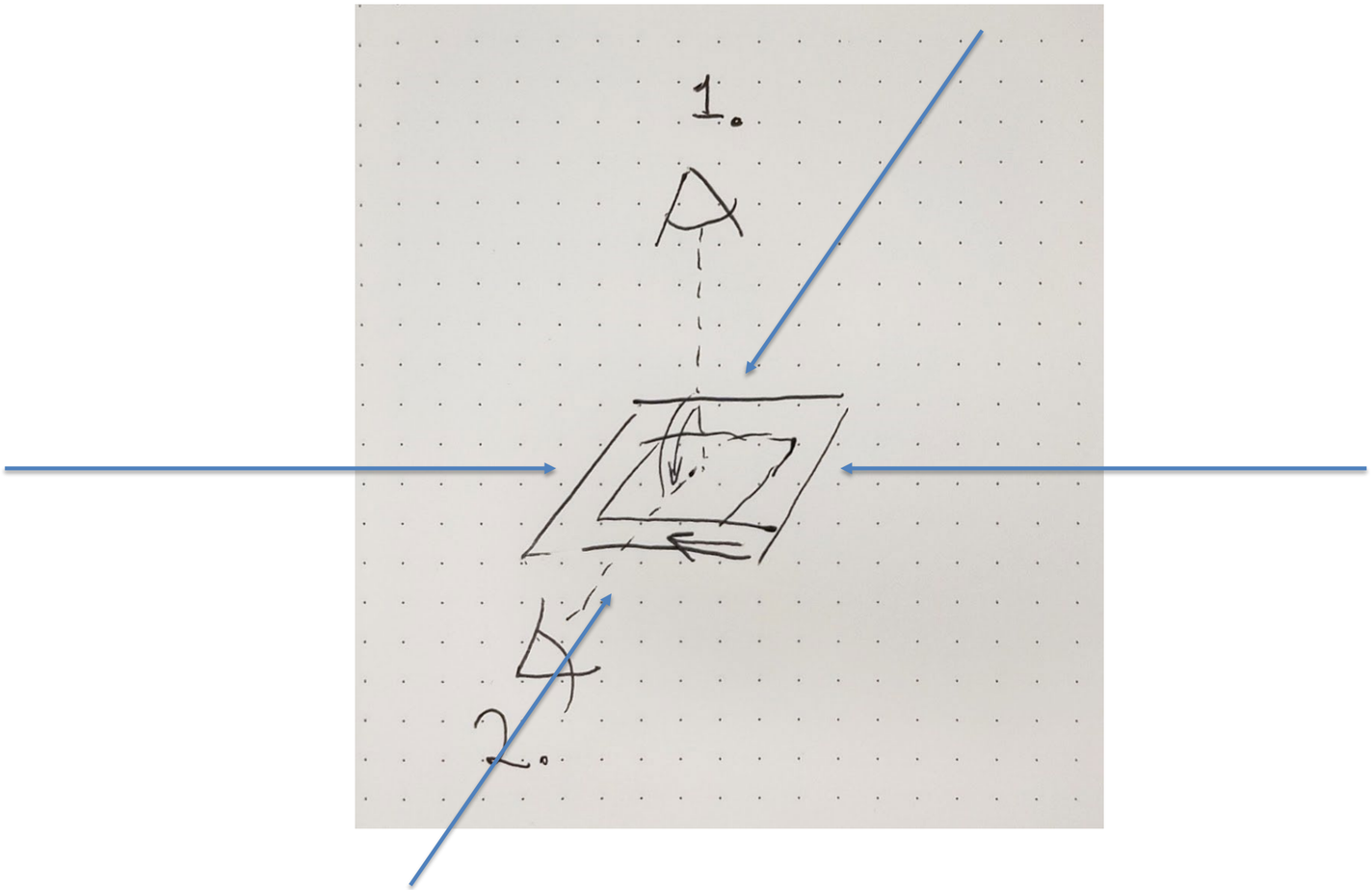
$$q^{-1} = [s \quad \underbrace{-i \quad -j \quad -k}]$$

Conjugate = negate the imaginary part

Inverting a Quaternion



The board game problem



The board game problem

```
_origin = Quaternion.identity;
_homeRow = Quaternion.AngleAxis(65, Vector3.right);
_leftSide = Quaternion.AngleAxis(90, Vector3.up) * Quaternion.AngleAxis(65, Vector3.right);
_backSide = Quaternion.AngleAxis(180, Vector3.up) * Quaternion.AngleAxis(65, Vector3.right);
_rightSide = Quaternion.AngleAxis(270, Vector3.up) * Quaternion.AngleAxis(65, Vector3.right);
```

Remember that quaternion multiplication in Unity is backwards

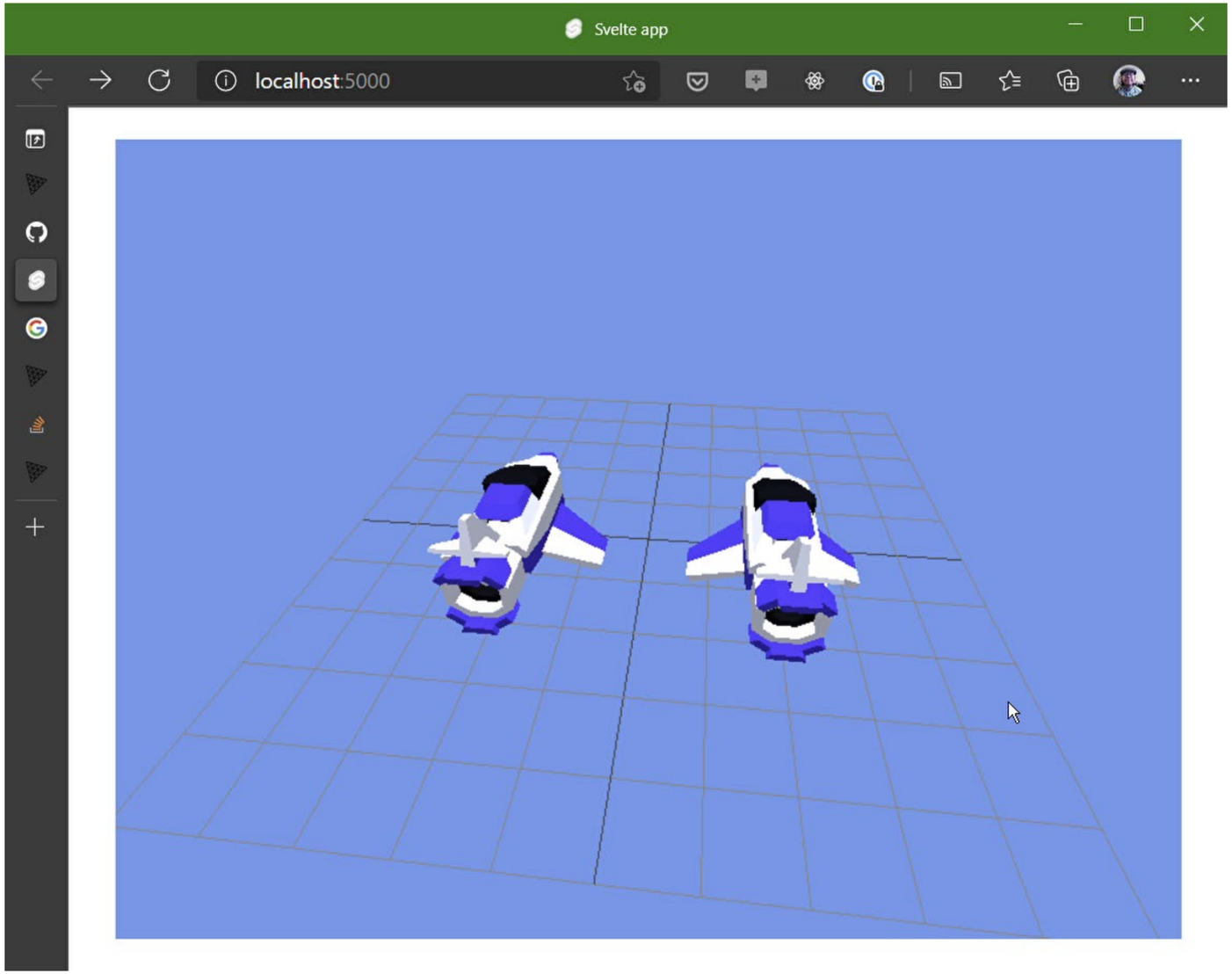
Interpolating Quaternions

- You LERP vectors and SLERP quaternions

Interpolating Quaternions

$$(1-t)q_0 + tq_1$$

Demo

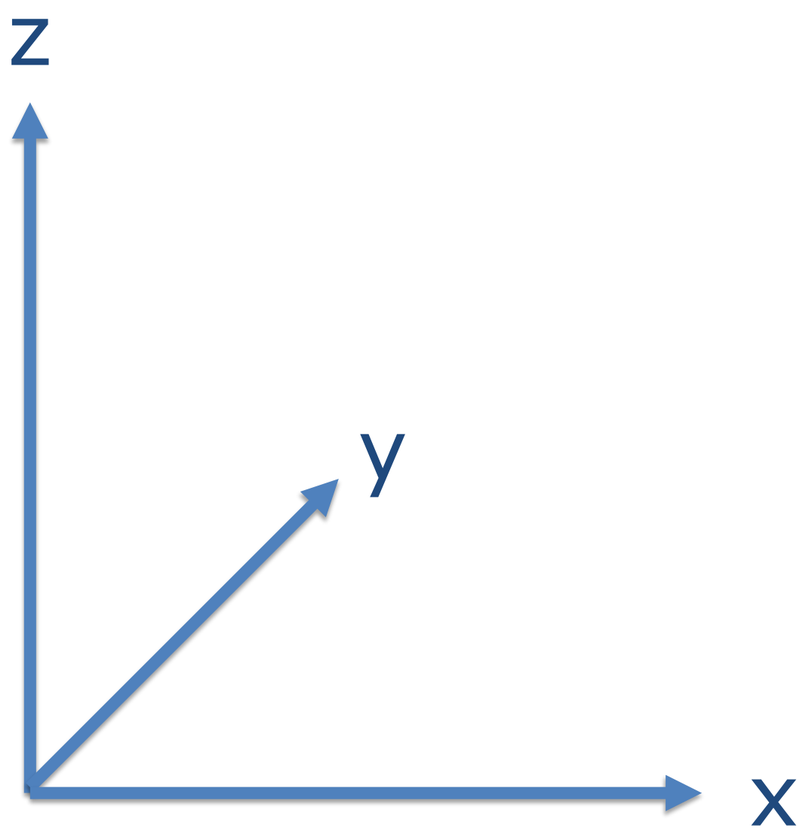


Interpolating Quaternions

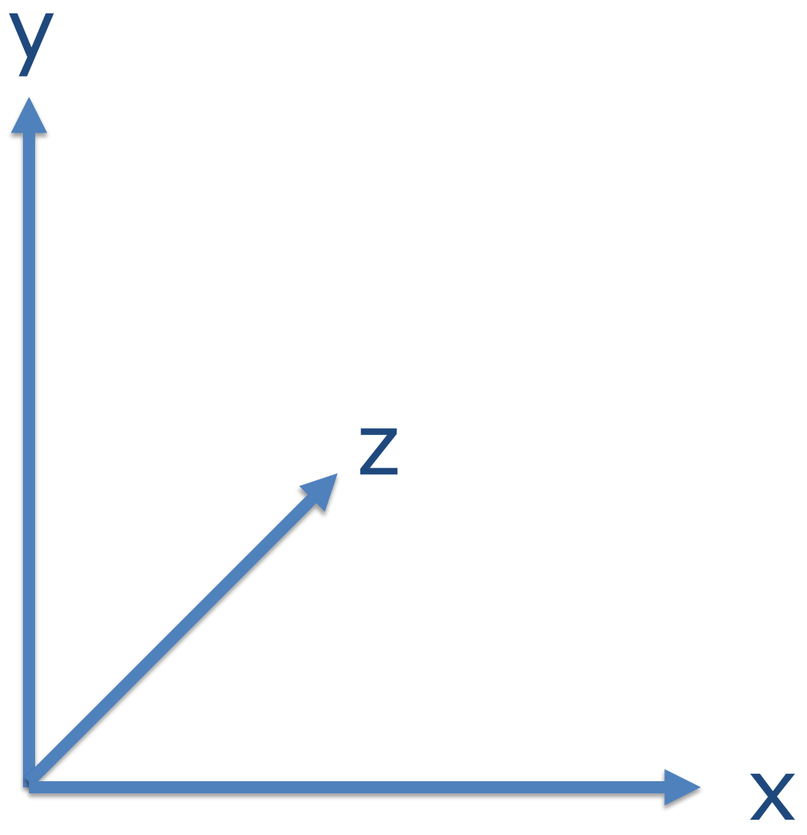
$$\frac{\sin\big((1-\mathrm{t})\theta\big)q_0 + \sin(t\theta)\,q_1}{\sin(\theta)}$$

$$\cos(\theta) = q_0 \cdot q_1$$

Coordinate Spaces



Right Handed



Left Handed

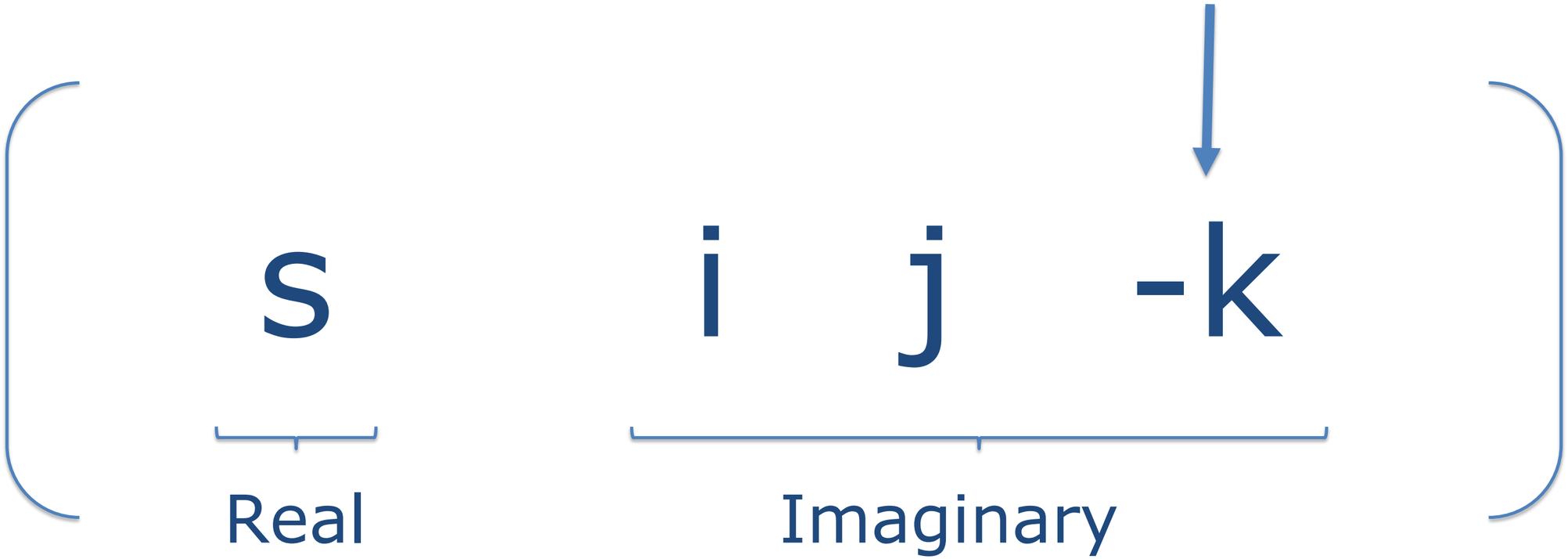
Handedness

“If you go from a right handed coordinate system to a left, you’re negating one of your imaginary components.”

Handedness

$$\begin{bmatrix} \underbrace{s}_{\text{Real}} & \underbrace{i \quad j \quad k}_{\text{Imaginary}} \end{bmatrix}$$

Handedness



Implementation

- “Trust the targeting computer.”

Thank you!

Patrick Martin
Follow me @pux0r3 on Twitter

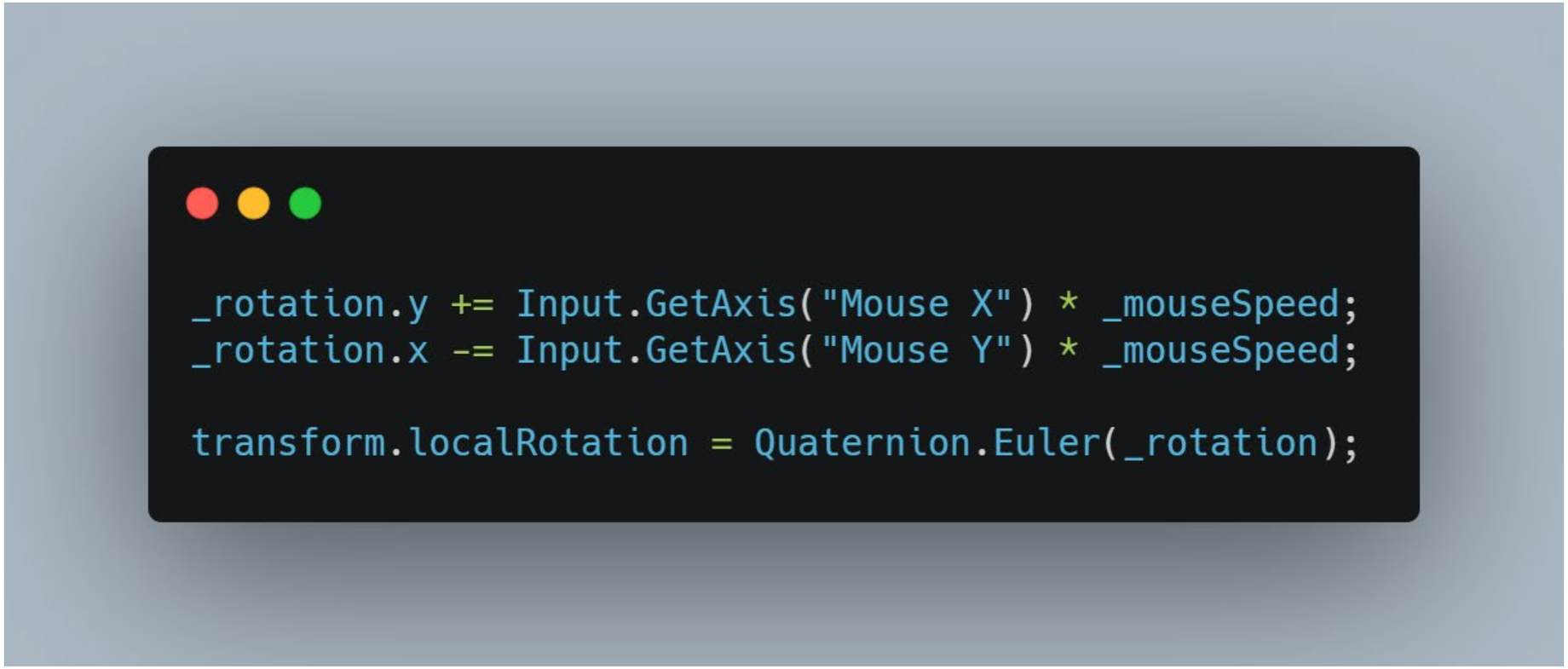
Practical Tips and Tricks

Remember to Renormalize

$$\|q\| = \sqrt{s^2 + i^2 + j^2 + k^2}$$

$$q = \left[\frac{s}{\|q\|} \quad \frac{i}{\|q\|} \quad \frac{j}{\|q\|} \quad \frac{k}{\|q\|}\right]$$

Mouselook



```
_rotation.y += Input.GetAxis("Mouse X") * _mouseSpeed;
_rotation.x -= Input.GetAxis("Mouse Y") * _mouseSpeed;

transform.localRotation = Quaternion.Euler(_rotation);
```

Mouselook

```
// This will look really weird
transform.localRotation = transform.localRotation * Quaternion.Euler(rotation);
```

Mouselook

```
transform.localRotation = Quaternion.AngleAxis(rotation.y, Vector3.up)
    * transform.localRotation
    * Quaternion.AngleAxis(rotation.x, Vector3.right);
```

Fix Up Vector

```
var refRight = Vector3.Cross(transform.forward, Vector3.up);
var targetUp = Vector3.Cross(refRight, transform.forward);
var angle = Mathf.Atan2(
    Vector3.Dot(transform.up, targetUp),
    Vector3.Dot(transform.up, refRight)) - Mathf.PI / 2f;
transform.localRotation = Quaternion.AngleAxis(angle * Mathf.Rad2Deg, transform.forward)
    * transform.localRotation;
```