Phép quay bằng Quaternion

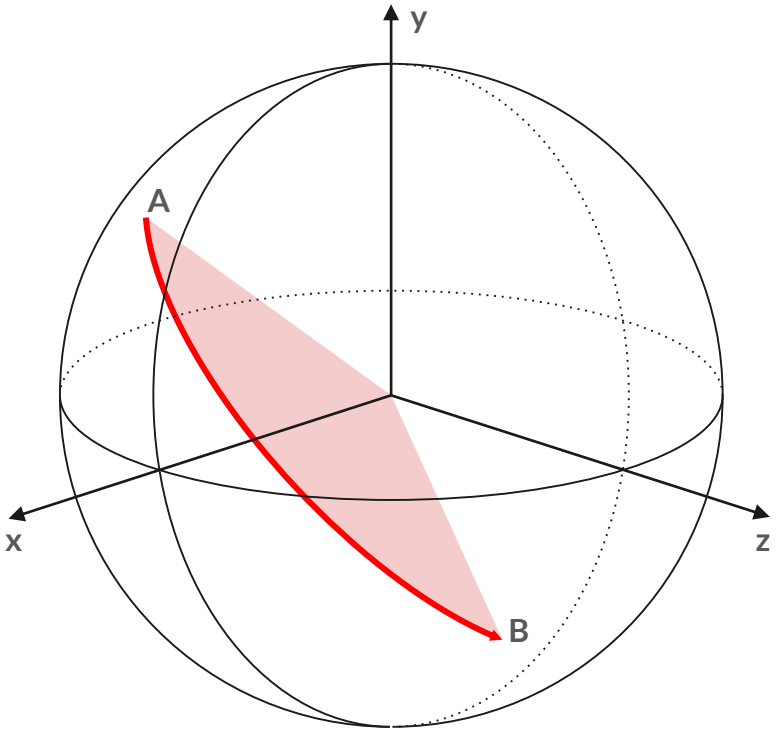
nguyenkimlong@savameta.com

Tổng quan

Đối với Euler, để quay một vật ta phải quay lần lượt các trục của nó (x -> y ->z)

Trong toán học, điều này rất khó tưởng tượng, kể cả khi ta biết điểm quay đầu và cuối

Do vậy, ta phải quy đổi sang một đơn vị khác, có tên là **Quaternion**

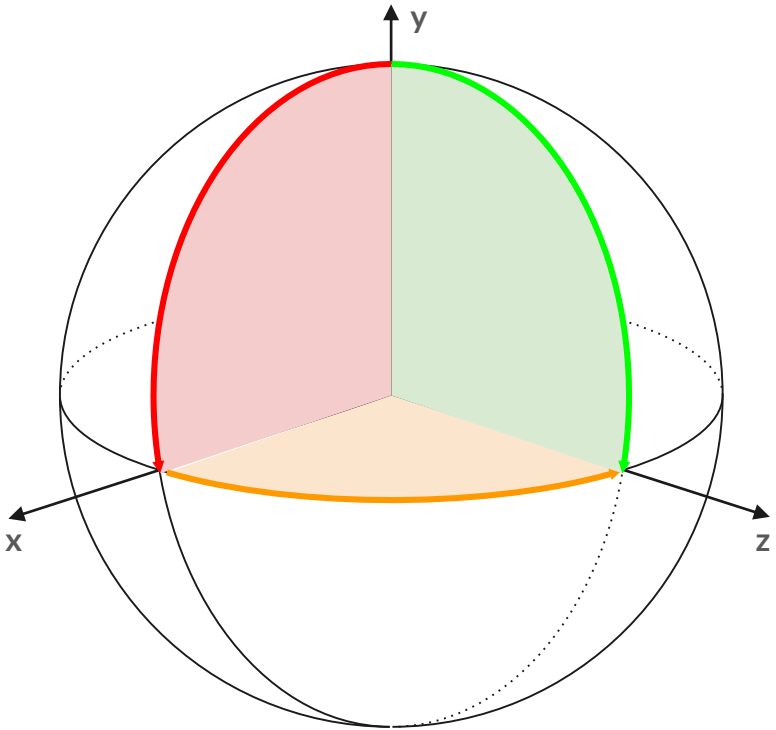


Quaternion

Quaternion là phép quay một vật, đại diện cho độ lớn và hướng quay của vật

Như hình bên, giả sử ta quay một góc 90 độ theo hướng y->x (màu **đỏ**), sau đó quay thêm một góc 90 độ nữa theo hướng x->z (màu **cam**), thì tương đương với việc ta quay một góc 90 độ theo hướng y->z (màu **xanh**)

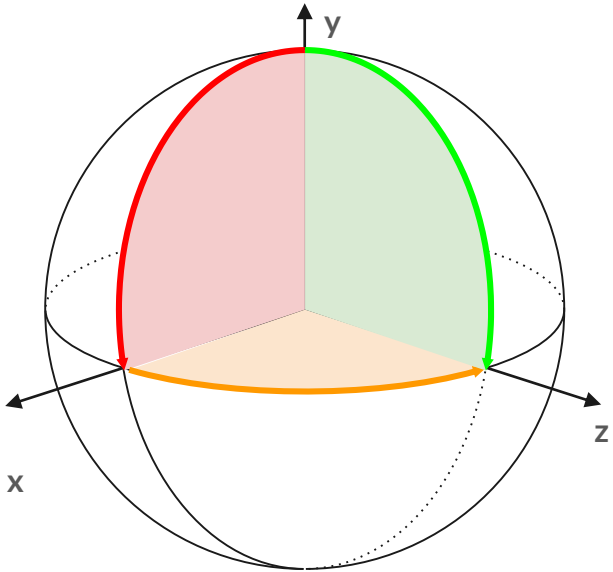
Quaternion dùng ma trận 4x4 để tính toán kết quả cuối cùng, chính là phép quay ngắn nhất sau một chuỗi các phép quay



Quaternion

Trong Unity, để có phép quay YZ thì ta nhân phép quay YX với phép quay XZ

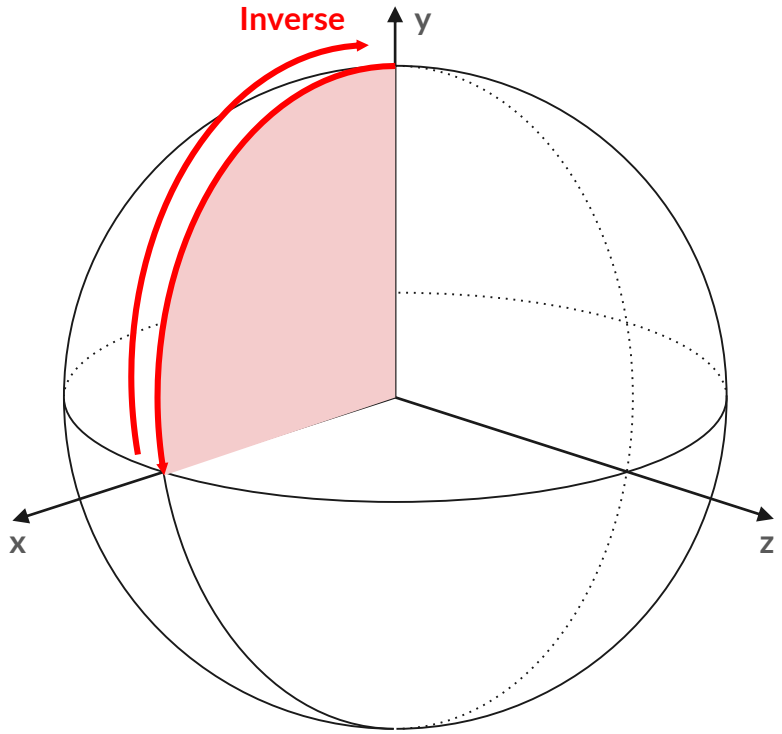
*** Điều này gần giống việc cộng 2 vector, ta chỉ việc thay phép cộng thành phép nhân**




```
public static Quaternion Plus(this Transform A, Transform B)
{
    return A.rotation * B.rotation;
}
```

Quaternion Inverse

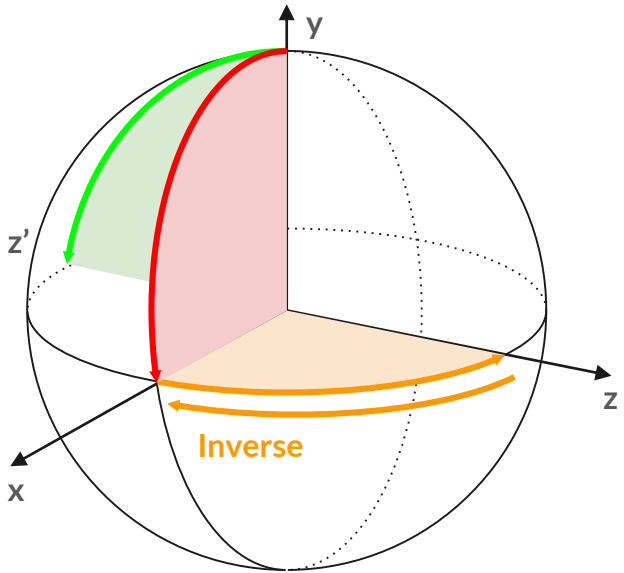
Nếu như ta quay một góc 90 độ theo hướng y->x, thì inverse của nó cũng là một góc 90 độ nhưng hướng ngược lại từ x->y



Quaternion Inverse

Trong Unity, để có Quaternion YZ’ thì ta nhân Quaternion YX với Quaternion Inverse XZ

*** Điều này gần giống việc trừ 2 vector, ta chỉ việc thay phép trừ thành phép nhân với inverse**

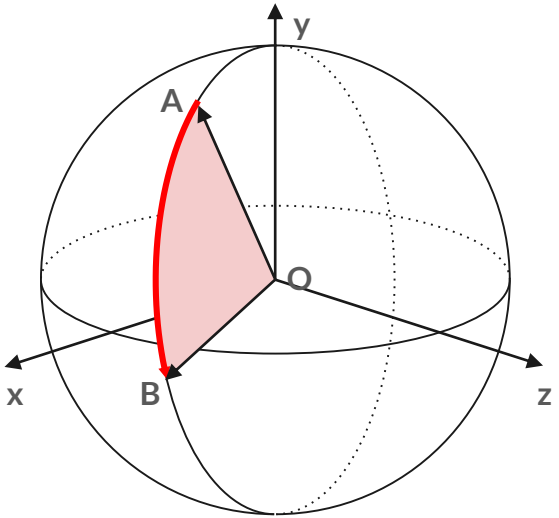



```
public static Quaternion Minus(this Transform A, Transform B)
{
    return A.rotation * Quaternion.Inverse(B.rotation);
}
```

Quaternion Vector

Khi ta nhân Quaternion với Vector, ta sẽ quay Vector đó theo hướng của Quaternion

Như hình bên, để biến vector OA thành vector OB, ta chỉ cần nhân quaternion OAB với vector OA




```
public static Vector3 Rotate(this Vector3 V, Quaternion Q)
{
    return Q * V;
}
```

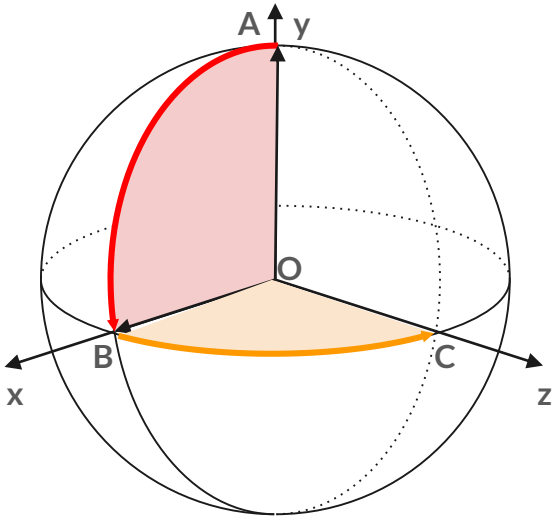
Quaternion Vector

Chú ý:

Khi nhân liên tiếp quaternion với vector, ta cần chú ý tới thứ tự thực hiện

Giả sử vector OA biến đổi thành vector OC nhờ 2 phép quay: dọc theo Oyx và ngang theo Oxz

Nhưng nếu ta quay ngang trước rồi mới dọc sau, thì kết quả OA chỉ tới được OB

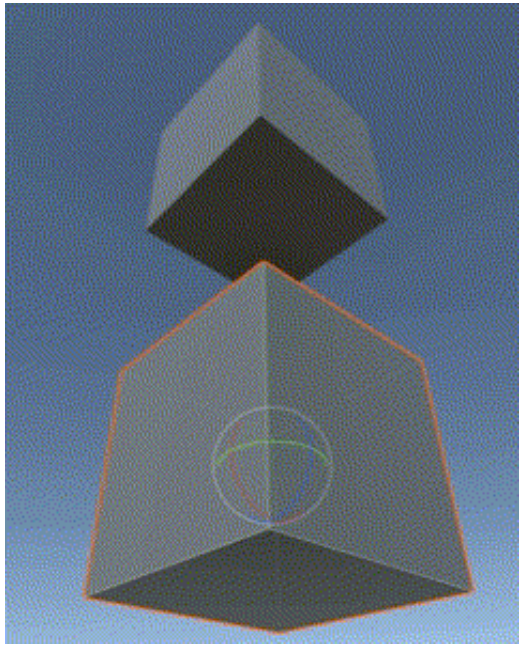


Quaternion Offset

Trong bài toán thực tế, chúng ta thường không có một Quaternion cụ thể, mà dựa vào độ lệch Quaternion của 2 đối tượng để dùng vào mục đích khác nhau:

- Xoay vật theo vật khác
- Di chuyển vật theo tay
- Điều khiển cánh tay robot theo IK
- ...

Quaternion Offset



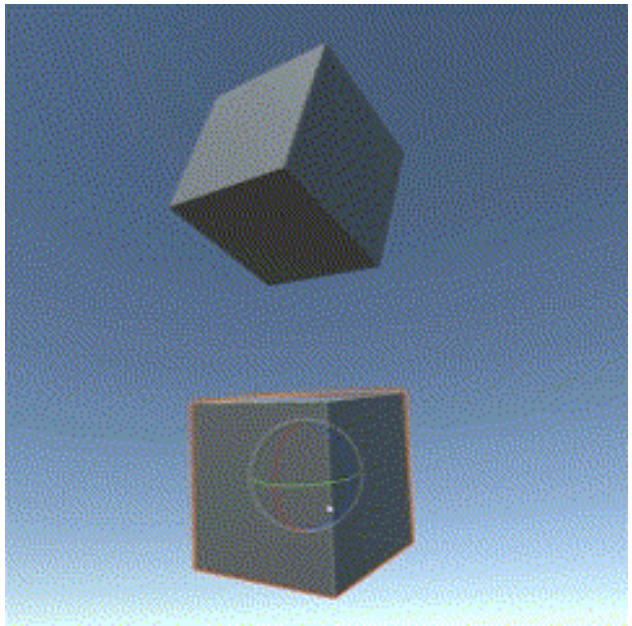
```
public class QuaternionOffset : MonoBehaviour
{
    [SerializeField] private Transform current;
    [SerializeField] private Transform target;

    private Quaternion rotationOffset;

    private void Awake()
    {
        rotationOffset = target.rotation *
Quaternion.Inverse(current.rotation);
    }

    private void Update()
    {
        target.rotation = current.rotation * rotationOffset;
    }
}
```

Quaternion Follower

Ta có thể biến tấu một chút, ngoài việc lưu độ lệch về góc xoay, ta sẽ lưu cả độc lệch về khoảng cách

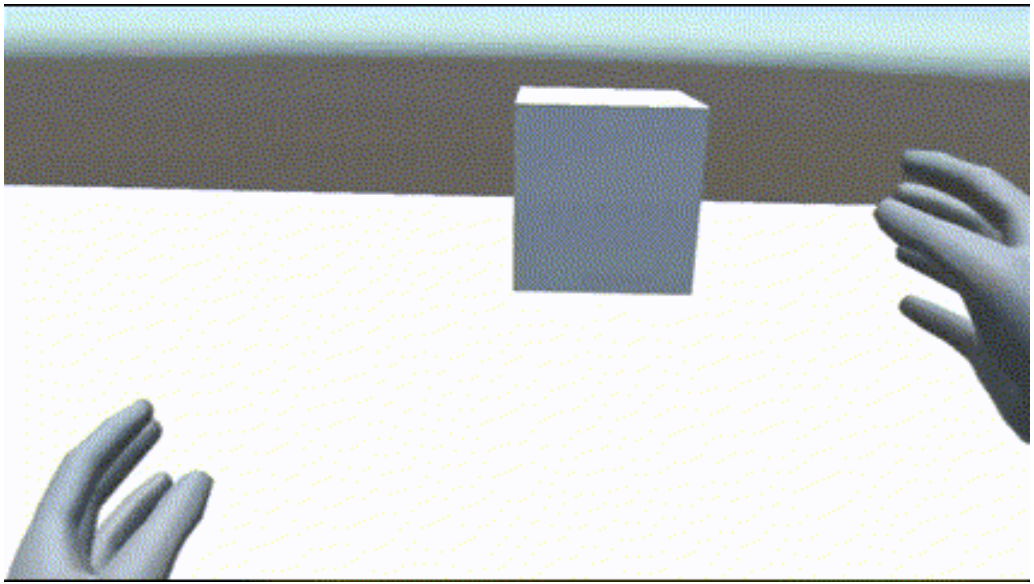
Kết quả là **Target** không chỉ xoay mà còn đi theo **Current**

Quaternion Follower



```
public class QuanternionFollower : MonoBehaviour
{
    [SerializeField] private Transform current;
    [SerializeField] private Transform target;

    private Vector3 posOffset;
    private Quaternion rotOffset;

    private void Awake()
    {
        rotOffset = Quaternion.Inverse(current.rotation) * target.rotation;
        posOffset = Quaternion.Inverse(current.rotation) * (target.position -
current.position);
    }

    private void Update()
    {
        target.rotation = current.rotation * rotOffset;
        target.position = current.position + current.rotation * posOffset;
    }
}
```

Quaternion Hand VR



Áp dụng **Quaternion Follower** vào Hand VR

Quaternion và các hàm thường dùng

Public Methods:

- ToAngleAxis

Static Methods:

- Angle
- AngleAxis
- Euler
- FromToRotation
- Lerp
- LookRotation

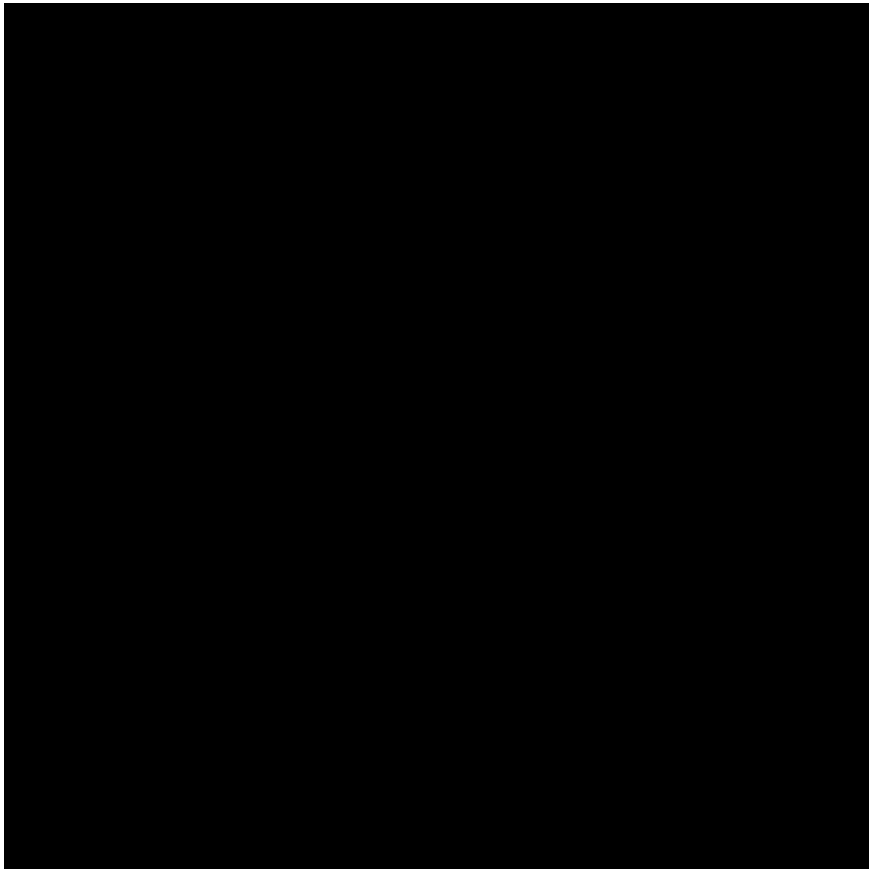
ToAngleAxis

public void **ToAngleAxis**(out float **angle**, out Vector3 **axis**);

Trả về góc (**angle**) và trục xoay (**axis**) của một quaternion

Như hình bên:

- Góc đã xoay là **(60, 90, 0)**
- Màu đỏ là **angle**
- Màu xanh dương là **axis**
- Sau khi bấm nút **Rotate** thì cube sẽ quay ngược lại góc **angle** theo trục **axis** để về vị trí ban đầu



Angle

public static float **Angle**(Quaternion **a**, Quaternion **b**);

Trả về góc giữa hai quaternion

```
var angle = Quaternion.Angle(A, B);
```

AngleAxis

public static Quaternion **AngleAxis**(float **angle**, Vector3 **axis**);

Trả về phép quay dựa trên góc và trục xoay

```
transform.rotation = Quaternion.AngleAxis(30, Vector3.up);
```

Euler

public static Quaternion **Euler**(float **x**, float **y**, float **z**);

Trả về phép quay dựa trên góc euler

```
var rotation = Quaternion.Euler(0, 30, 0);
```

FromToRotation

public static Quaternion **FromToRotation**(Vector3 **fromDirection**, Vector3 **toDirection**);

Trả về phép quay giữa 2 vector

```
var rotation = Quaternion.FromToRotation(current.forward, target.forward);
```

Lerp

public static Quaternion **Lerp**(Quaternion **a**, Quaternion **b**, float **t**);

Trả về phép quay được nội suy giữa 2 quaternion

```
current = Quaternion.Lerp(current, target, Time.deltaTime * speed);
```

LookRotation

public static Quaternion **LookRotation**(Vector3 **forward**, Vector3 **upward** = Vector3.up);

Trả về phép quay được xác định bởi 2 hướng forward và upward

```
var rotation = Quaternion.LookRotation(target.position - transform.position, Vector3.up);
```

Tổng kết

- Quaternion là phép quay một vật, đại diện cho độ lớn và góc quay vật đó
- Quaternion có thể nhân với nhau
- Quaternion có thể nghịch đảo bằng hàm Inverse
- Khi thực hiện các phép nhân Quaternion liên tiếp, cần chú ý tới thứ tự
- Có thể nhân Quaternion với một Vector để thực hiện phép quay cho Vector đó
- Có thể lưu độ lệch giữa 2 Quaternion để dùng vào các mục đích khác nhau
- Unity hỗ trợ rất nhiều hàm để quay, cũng như quy đổi sang dạng euler