Sample Code

Rotating a cube by transforming its vertices

Rotate a cube through a series of keyframes using quaternion interpolation to transition between them.

Download

macOS 13.3+ | Xcode 14.3+

Overview

Quaternions are fundamental to graphics programming and are often used as a compact representation of the rotation of an object in three dimensions. You can rotate a 3D object in space by applying unit quaternion actions to each of its vertices. The simd module includes functions to interpolate between a series of rotational keyframes — defined by unit quaternions — with either the `simd_slerp(_:_:_:)` (for linear interpolation) or the `simd_spline(_:_:_:_:_:)` (for smooth, spline-based interpolation) functions.

This sample code project defines a cube using eight vertices and transforms it through a series of rotations. The sample app provides a SwiftUI `Toggle` control that switches between a series of discrete spherical linear interpolations (that is, a series of separate arcs between each keyframe) and a continuous spline (that is, a single, smooth path between each keyframe).

Define a cube by its vertices

The sample code defines a cube with eight `simd_double3` vectors. Each vector specifies the 3D position of one of the cube's corners.

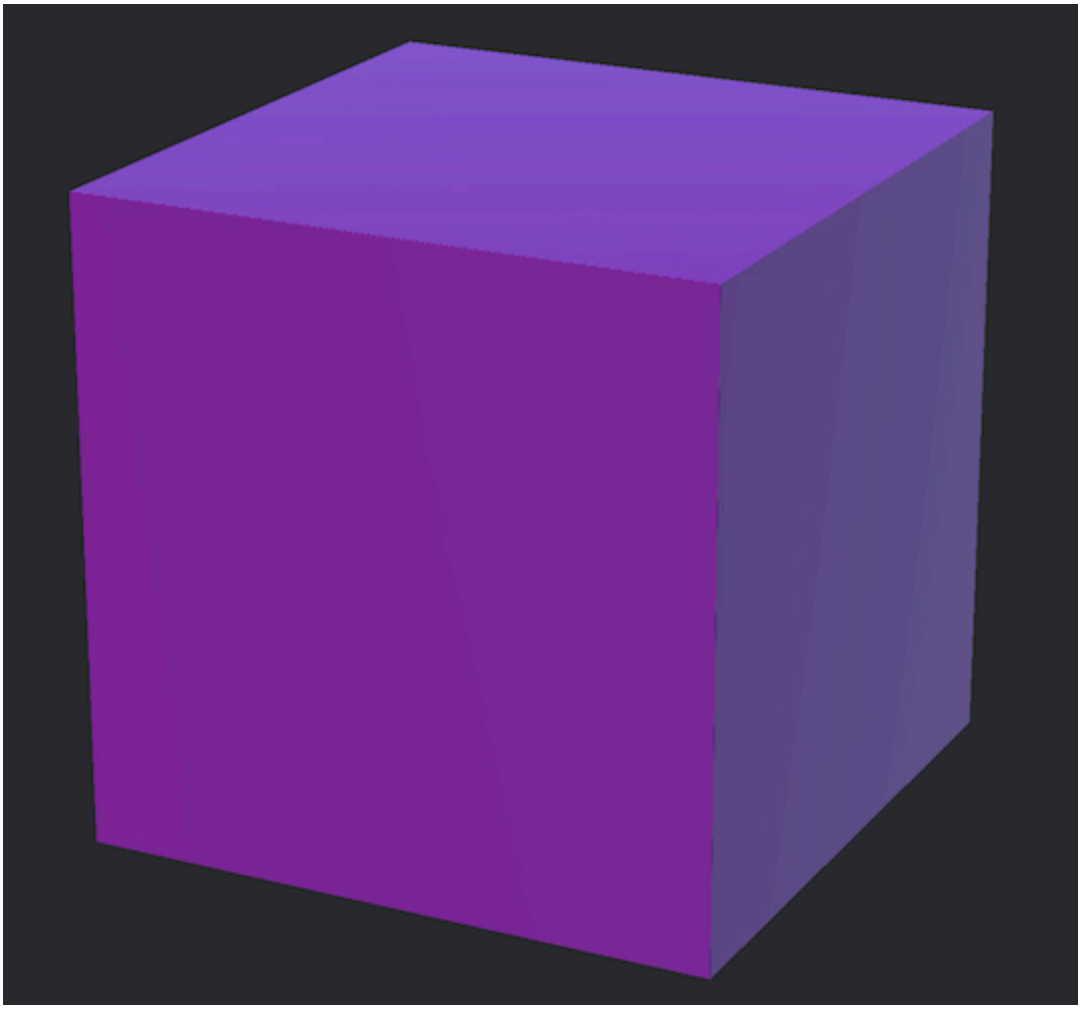
```
let cubeVertexOrigins: [simd_double3] = [
    simd_double3(x: -0.5, y: -0.5, z: 0.5),
```

```
    simd_double3(x: 0.5, y: -0.5, z: 0.5),
    simd_double3(x: -0.5, y: -0.5, z: -0.5),
    simd_double3(x: 0.5, y: -0.5, z: -0.5),
    simd_double3(x: -0.5, y: 0.5, z: 0.5),
    simd_double3(x: 0.5, y: 0.5, z: 0.5),
    simd_double3(x: -0.5, y: 0.5, z: -0.5),
    simd_double3(x: 0.5, y: 0.5, z: -0.5)
    ]
```

The quaternion keyframes act upon the vertex origins and mutate `cubeVertices` to rotate the cube.

```
lazy var cubeVertices = cubeVertexOrigins
```

This sample uses SceneKit to render the cube that vertices in the `cubeVertices` array define. You can also use the technique that the sample code uses to rotate geometry in other technologies such as Metal. The following image shows the cube, defined by the vertices above, rendered in SceneKit:



Define the quaternion rotation keyframes

As discussed in Working with Quaternions, spline interpolation requires a quaternion before the current value and a quaternion after the next value to compute the interpolated value. To support this, the following code defines the series of rotations with additional values at the beginning and end. The following declaration duplicates the first and last elements.

```
let vertexRotations: [simd_quatd] = [
    simd_quatd(angle: 0,
               axis: simd_normalize(simd_double3(x: 0, y: 0, z: 1))),
    simd_quatd(angle: 0,
               axis: simd_normalize(simd_double3(x: 0, y: 0, z: 1))),
    simd_quatd(angle: .pi * 0.05,
               axis: simd_normalize(simd_double3(x: 0, y: 1, z: 0))),
    simd_quatd(angle: .pi * 0.1,
               axis: simd_normalize(simd_double3(x: 1, y: 0, z: -1))),
    simd_quatd(angle: .pi * 0.15,
               axis: simd_normalize(simd_double3(x: 0, y: 1, z: 0))),
    simd_quatd(angle: .pi * 0.2,
               axis: simd_normalize(simd_double3(x: -1, y: 0, z: 1))),
    simd_quatd(angle: .pi * 0.15,
               axis: simd_normalize(simd_double3(x: 0, y: -1, z: 0))),
    simd_quatd(angle: .pi * 0.1,
               axis: simd_normalize(simd_double3(x: 1, y: 0, z: -1))),
    simd_quatd(angle: .pi * 0.05,
               axis: simd_normalize(simd_double3(x: 0, y: 1, z: 0))),
    simd_quatd(angle: 0,
               axis: simd_normalize(simd_double3(x: 0, y: 0, z: 1))),
    simd_quatd(angle: 0,
               axis: simd_normalize(simd_double3(x: 0, y: 0, z: 1)))
]
```

Animate between keyframes with spherical interpolation

This sample uses a CVDisplayLink instance to schedule updates to the cube's vertices and calls the `vertexRotationStep()` function every frame.

```
CVDisplayLinkCreateWithCGDisplay(CGMainDisplayID(), &displayLink)

let displayCallback: CVDisplayLinkOutputCallback = { _, _, _, _, _, displayLinkConte

    if let displayLinkContext = displayLinkContext {
        DispatchQueue.main.async {
```

```
            let cubeRotation = Unmanaged<CubeRotation>.fromOpaque(displayLinkContext
            cubeRotation.vertexRotationStep()
        }
    }

    return kCVReturnSuccess
}

CVDisplayLinkSetOutputCallback(displayLink,
                               displayCallback,
                               Unmanaged.passUnretained(self).toOpaque())

CVDisplayLinkStart(displayLink)
```

The following variables define the current index in `vertexRotations` and the time, between `0.0` and `1.0`, for the current interpolation:

```
var vertexRotationIndex = 1
var vertexRotationTime: Double = 0
```

With each display link notification, the `vertexRotationStep` function increments the vertex rotation time variable by a small amount.

```
let increment: Double = 0.02
vertexRotationTime += increment
```

The `simd_slerp(_:_:_:)` function returns a quaternion that's spherically interpolated between the current and next quaternion keyframe at the specified time:

```
quaternion = simd_slerp(
    vertexRotations[vertexRotationIndex],
    vertexRotations[vertexRotationIndex + 1],
    vertexRotationTime)
```

The quaternion acts upon each of the cube's vertices and rotates the cube around its center:

```
cubeVertices = cubeVertexOrigins.map {
    return quaternion.act($0)
}
```

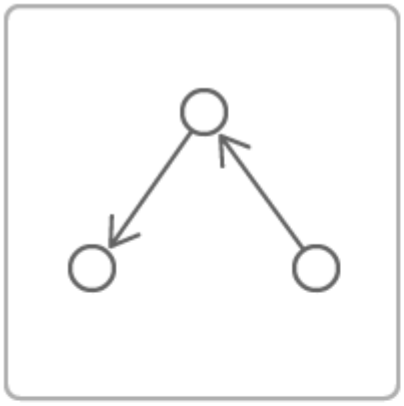
If the vertex rotation time is greater than or equal to one, the code progresses to the next keyframe, increments the index to the rotations array, and resets the rotation time to zero. When the code has reached the last usable quaternion in the array of rotations, it ends the animation.

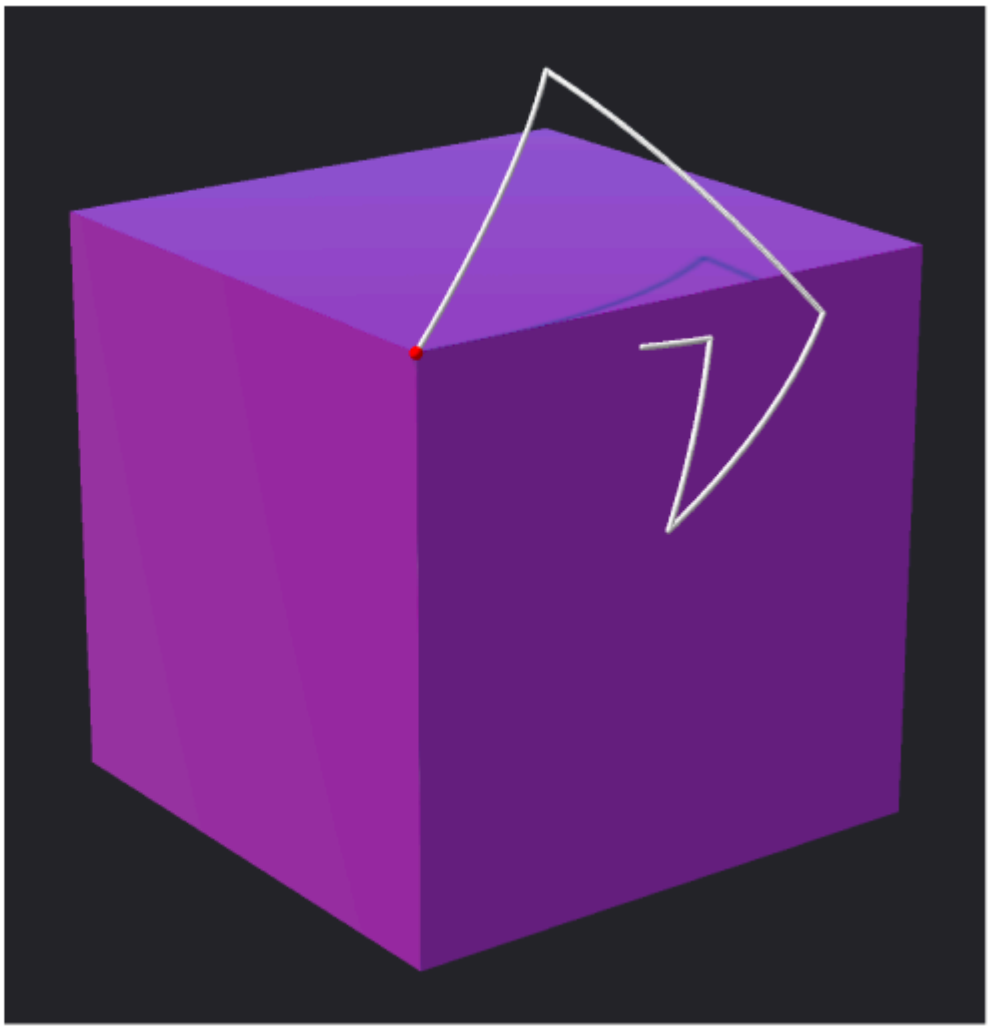
```
if vertexRotationTime >= 1 {
    vertexRotationIndex += 1
    vertexRotationTime = 0

    if vertexRotationIndex > vertexRotations.count - 3 {
        scene = setupSceneKit()

        vertexRotationIndex = 1
    }
}
```

Over time, the cube animates through the series of keyframes. The following image shows the sharp change in direction as the cube rotates between the keyframes:



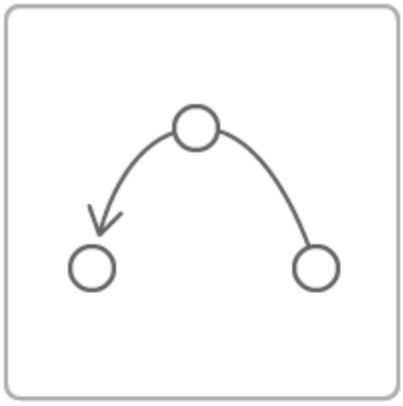


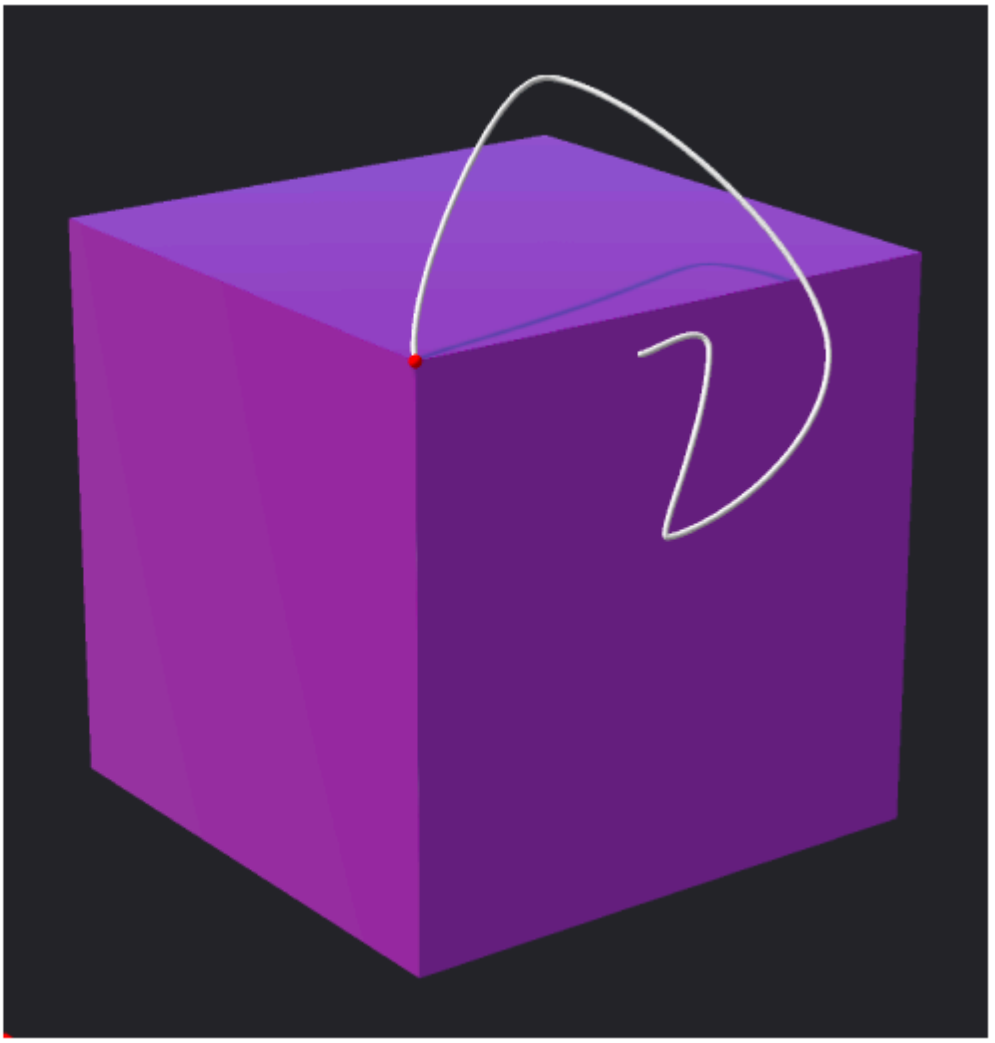
Animate between keyframes with spline interpolation

The sample code uses the identical code to the spherical interpolation sample for spline interpolation, apart from one difference: rather than generating the quaternion that acts upon the vertices with `simd_slerp(_:_:_:)`, it uses the `simd_spline(_:_:_:_:_:)` function.

```
quaternion = simd_spline(
    vertexRotations[vertexRotationIndex - 1],
    vertexRotations[vertexRotationIndex],
    vertexRotations[vertexRotationIndex + 1],
    vertexRotations[vertexRotationIndex + 2],
    vertexRotationTime)
```

The image below shows that the spline interpolation creates transitions between the quaternion keyframes that are smoother than the linear spherical interpolation.





See Also

Vectors, Matrices, and Quaternions

- Working with Vectors

  Use vectors to calculate geometric values, calculate dot products and cross products, and interpolate between values.

- Working with Matrices

  Solve simultaneous equations and transform points in space.

- Working with Quaternions

  Rotate points around the surface of a sphere, and interpolate between them.

- simd

  Perform computations on small vectors and matrices.

- vForce

  Perform transcendental and trigonometric functions on vectors of any length.