Sample Code

Simulating physics joints in your RealityKit app

Create realistic, connected motion using physics joints.

Download

iOS 18.0+ | iPadOS 18.0+ | macOS 15.0+ | visionOS 2.0+ | Xcode 16.0+

Overview

This sample app demostrates how to create a `PhysicsRevoluteJoint` that simulates motion, like a pendulum swinging back and forth, as an alternative to animations.

You can create animations to depict objects in motion but simulations can be better for more complicated scenes because they can work with more variables, and support custom motion that react dynamically to external factors, such as a person's gestures.

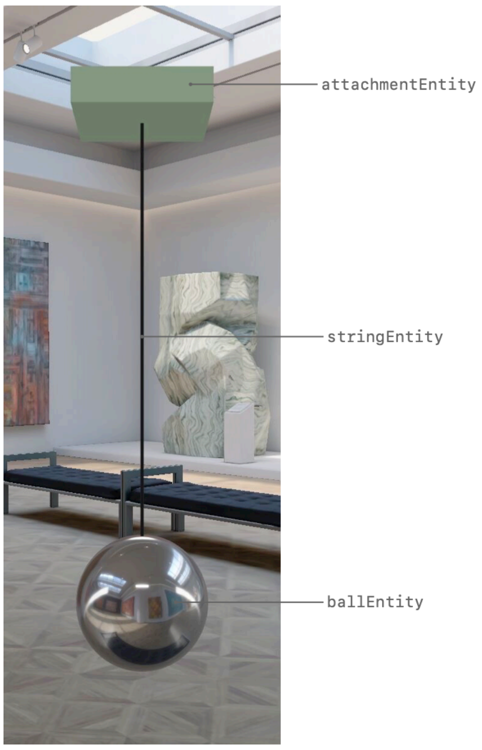
Play ▶

> **Note**
>
> All joint types in RealityKit conform to the `PhysicsJoint` protocol, including `PhysicsRevoluteJoint`.

Set up the app’s static scene

The app starts with a single pendulum, made up of a few entities:

- A green cuboid at the top with the name `attachmentEntity`
- A chrome ball at the bottom with the name `ballEntity`
- A gray cylinder connecting the cuboid and the ball, with the name `stringEntity`



Both `attachmentEntity` and `ballEntity` are children of a non-visible entity called `parentSimulationEntity`. This parent entity is the root of the simulation.

Additionally, `stringEntity` is a child of `ballEntity`, and only exists for a visual representation of the connection between `attachmentEntity` and `ballEntity`.

In the initial scene, each entity only has their default components, an updated y-position, and a `ModelComponent`.

Add physics components to the entities

Before you activate a physics joint, each entity must have both a `PhysicsBodyComponent` and a `CollisionComponent`.

This app adds a physics body and collision shape that matches the model's spherical shape, where `pendulumSettings.ballRadius` characterizes its radius.

To respond to forces like gravity and collisions, set the ball's `mode` to `PhysicsBodyMode.dynamic`. Set its material to have no friction and a restitution of 1, to result in completely elastic collisions:

```
let collisionShape = ShapeResource.generateSphere(
    radius: pendulumSettings.ballRadius)

var ballBody = PhysicsBodyComponent(
    shapes: [collisionShape],
    mass: pendulumSettings.ballMass,
    material: .generate(staticFriction: 0, dynamicFriction: 0, restitution: 1),
    mode: .dynamic
)
ballBody.linearDamping = 0
let ballCollision = CollisionComponent(shapes: [ballShape])
```

```
ballEntity.components.set([ballBody, ballCollision])
```

Because other forces can’t move the other end of the physics joint (`attachmentEntity`), set its `mode` to `PhysicsBodyMode.static`:

```
let attachmentShape = ShapeResource.generateBox(
    size: pendulumSettings.attachmentSize * pendulumSettings.ballRadius
)

var attachmentBody = PhysicsBodyComponent(
    shapes: [attachmentShape], mass: 1,
    material: .generate(staticFriction: 0, dynamicFriction: 0, restitution: 1),
    mode: .static
)
attachmentBody.linearDamping = 0
let attachmentCollision = CollisionComponent(shapes: [attachmentShape])

attachmentEntity.components.set([attachmentBody, attachmentCollision])
```

Add the simulation and physics joints components

The app adds the components `PhysicsSimulationComponent` and `PhysicsJointsComponent` to a common ancestor of `ballEntity` and `attachmentEntity`, to indicate where RealityKit adds the joints:

```
// Add physics simulation component to parent simulation entity.
parentSimulationEntity.components.set(PhysicsSimulationComponent())
// Add physics joints component to parent simulation entity.
parentSimulationEntity.components.set(PhysicsJointsComponent())
```

Create a new joint

`PhysicsRevoluteJoint` creates a hinge for the swinging motion in this example. A revolute joint, also known as a *hinge joint*, allows rotational movement in one axis, similar to a door swinging on its hinges.

Note

`PhysicsRevoluteJoint` conforms to `PhysicsJoint`, a protocol for all physics joints.

A joint needs two GeometricPin instances on separate entities to create a physics joint.

The app creates each pin with the method set(named:position:orientation:) on its respective entity. Use pins to access all pins an entity owns:

```
// Rotate hinge orientation from x to z-axis.
let hingeOrientation = simd_quatf(from: [1, 0, 0], to: [0, 0, 1])

// The attachment's pin is in the center of
// the attachment entity.
let attachmentPin = attachmentEntity.pins.set(
    named: "attachment_hinge",
    position: .zero,
    orientation: hingeOrientation
)

// The ball's pin is at the center of the
// attachment entity in local space.
let relativeJointLocation = attachmentEntity.position(
    relativeTo: ballEntity
)

let ballPin = ballEntity.pins.set(
    named: "ball_hinge",
    position: relativeJointLocation,
    orientation: hingeOrientation
)
```

Tip

PhysicsRevoluteJoint always rotates around the pins' local x-axis, but in this example the models rotate around the z-axis because the variable `hingeOrientation` realigns the pins' x-axis (`[1, 0, 0]`) to the z-axis (`[0, 0, 1]`).

Use each GeometricPin as the parameters to create a new PhysicsRevoluteJoint:

```
let revoluteJoint = PhysicsRevoluteJoint(pin0: attachmentPin, pin1: ballPin)
```

Add the physics joint to your simulation

Add the new joint to the simulation. The `addToSimulation()` method finds the closest entity ancestor of the joint's first pin that has a `PhysicsJointsComponent`, and adds the joint to its `joints` collection:

```
try revoluteJoint.addToSimulation()
```

Add motion to the scene

The simulation shows no effect until you add motion. One way to do so is via `ImpulseAction`. Give the ball a push in the negative x-direction to start the simulation:

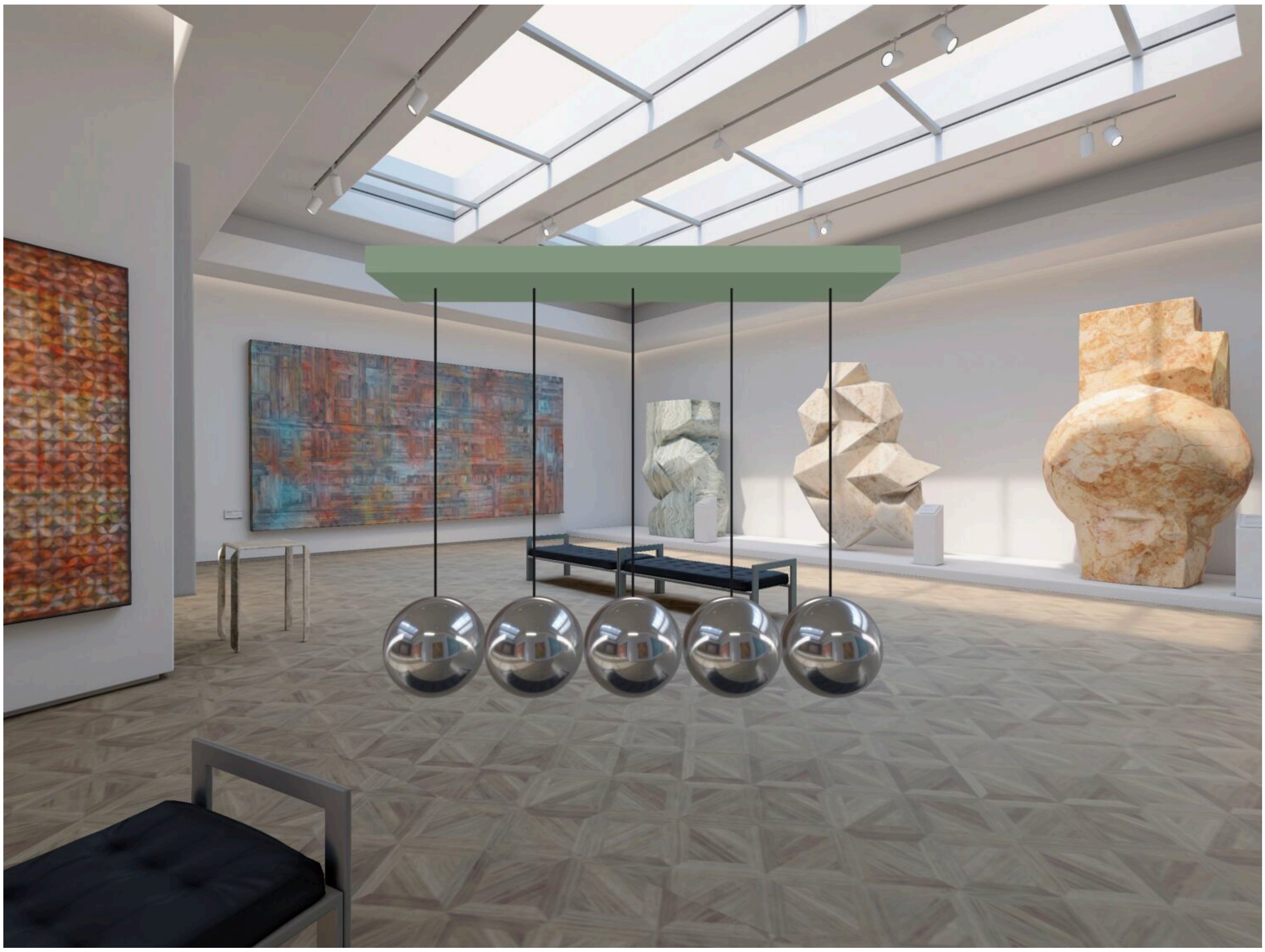
```
let impulseAction = ImpulseAction(
    targetEntity: .sourceEntity,
    linearImpulse: pendulumSettings.impulsePower)
let impulseAnimation = try AnimationResource.makeActionAnimation(
    for: impulseAction)

ballEntity.playAnimation(impulseAnimation)
```

The following video shows the resulting scene, with the `ballEntity` swinging left and right on the hinge:

Play ⊙

In the sample app, you can change the static value `PendulumSettings.pendulumCount` to a larger number such as 5 to place multiple pendulums in the app, and see them collide with each other:



Play

See Also

Pin and joint components

`struct GeometricPin`

A structure that identifies a local transform relative to an entity or entity's animating skeletal joint.

`struct GeometricPinsComponent`

A component that stores a sequence of geometric pins.

`protocol PhysicsJoint`

A type that describes physics joints.

`struct PhysicsJointsComponent`

A component that stores physics joints which RealityKit simulates.

`struct EntityGeometricPins`

A structure that wraps all geometric pins an entity owns.

`struct AttachedTransformComponent`

A component that stores an optional source pin owned by this entity and a target pin which this entity is attached to