Article

Reducing CPU Utilization in Your RealityKit App

Target specific CPU metrics with adjustments to your app and its content.

Overview

If your app's main thread consumes more than 16.6 ms per frame, it won't be able to achieve 60 fps. You can use the CPU metrics measured by RealityKit, as described in Improving the Performance of a RealityKit App, to decide where you need to focus your efforts to reach this goal.

Note that with multithreaded rendering, keeping the main CPU thread below 16.6 ms doesn't guarantee 60 fps. If the render thread also takes a long time, RealityKit might occasionally drop a frame to prevent the main thread from getting too far ahead.

For the best user experience, keep both the main and render threads below about 12 ms on average. It's helpful to keep the average time consumed well below the 16.6 ms limit to handle situations where the CPU load increases suddenly, such as when the user triggers multiple complex effects simultaneously. Also, the closer you run to the hardware's limit, the more likely the system might have to throttle the frame rate to avoid overheating.

Improve render thread time by merging meshes

The bulk of CPU render time involves setting up draw calls and command buffers, preparing render targets, and encoding buffers for submission to the GPU. Because RealityKit invokes a draw call for each mesh, having more meshes consumes more CPU time. To reduce render thread time, merge meshes that share a material when possible.

RealityKit does this for you automatically to the extent that it can when you load an entity in a usdz file as a `ModelEntity`, for example using the `loadModel(named:in:)` method. But you can help by minimizing the number of distinct meshes in your models in the first place.

On the other hand, merging meshes may reduce opportunities for visibility culling and other optimizations, which can lead to increased GPU utilization. This is because while RealityKit avoids rendering off-screen meshes, it does render the entirety of any mesh that's even partially visible. So consider carefully how you merge meshes. For example, it usually doesn't make sense to merge two meshes that are physically separated by a large distance.

> **Note**
>
> While disabling certain effects primarily benefits GPU utilization, as described in Choose render effects carefully, disabling these effects may also help reduce CPU render time. For modest gains, experiment by turning off shadows, HDR, motion blur, depth of field, and environment texturing.

Improve ecs time by flattening your assets

The CPU does ECS work for rendering, animation, physics, assets, networking, and audio on every frame, for every entity with related components. You can reduce this work by flattening your assets as much as possible. That is, combine an entity and its attached descendants into a single entity.

When loading entities stored in usdz files as `ModelEntity` instances, RealityKit automatically flattens the model for you, as described in Loading entities from a file. Do this unless you need programmatic access to the descendant entities.

Improve physics time by reducing collisions

Model entities employ physics to move and interact in ways that appear natural to the user. The CPU carries out the necessary calculations, the cost of which comes primarily from detecting collisions. You can usually reduce the physics portion of your app's CPU utilization by minimizing the number of collisions in the scene and simplifying collision shapes:

- Reduce the number of rigid bodies. Design your app to minimize the number of objects in the scene that can collide.
- Isolate rigid bodies into distinct areas to cut down on the number of ways objects can collide. A box that holds 200 rigid bodies generates a lot more collisions than 4 boxes each containing 50 rigid bodies.
- Use care when calling `generateCollisionShapes(recursive:)` to recursively create collision shapes. You might be able to manually construct simpler collision shapes that are less computationally complex for some meshes.

- Choose the right collision shape. Some shapes, like those with complex hulls, can drastically increase computational effort. Experiment with different shapes to see how they affect physics computation time.

Improve network time by reducing synchronization traffic

Synchronizing data among devices enables rich multiuser experiences. However, network access comes at a cost. It's typically best to maximize locality, performing computations on the device where you need the data, and synchronizing as little as possible.

On the other hand, data synchronization can be a better option than repeated calculation of the same data. For example, it might be better to perform an expensive calculation that produces a small number of bytes only once, and share the result with other devices.

When using synchronization, try to minimize the number of synchronized entities. Each entity that requires synchronization consumes CPU time. You can turn off synchronization for a particular entity by deleting its `RealityKit/Entity/synchronization` component:

```
entity.synchronization = nil
```

Within an entity, the system automatically synchronizes properties of all built-in components, as well the properties of custom components that adopt the `Codable` protocol. If you want to avoid synchronizing data for a particular custom component, skip the codable conformance.

> **Note**
>
> RealityKit only synchronizes component properties. It doesn't synchronize any custom entity properties.

Improve audio time by reducing audio complexity

Audio sources tied to objects in the environment help to make a scene feel immersive. But if you find that audio processing consumes too much CPU time, take steps to minimize the performance impact of audio:

- Limit the number of audio sources in the scene at any given time. For example, you might remove distant audio sources, or merge multiple audio sources into one.
- Prefer static audio sources whenever possible. Audio sources that move require frequently updated audio transforms.

- Avoid adding a large number of audio components to a scene all at once, which can cause a spike in CPU activity. Instead, space out the initializations, or do them before the user begins interacting with the environment.

Improve asset time by streamlining the asset pipeline

Reduce or defer the performance cost of loading or creating assets:

- Prefer Reality files over usdz files. Reality files provide better performance.
- Avoid loading or creating assets during active AR operation. For example, consider preloading data while the user chooses from options on a menu.
- Use asynchronous loads, with methods like `loadModelAsync(named:in:)`, to prevent blocking your main thread.
- Share resources, like instances of `MeshResource`, rather than creating new instances of the same resource for each entity.
- Consider breaking large assets into a collection of smaller assets. Smaller assets load faster, giving you the flexibility to distribute asset loading over time, if it makes sense for your app.

For more information about loading entities, see Loading entities from a file.

See Also

Performance improvements

Improving the Performance of a RealityKit App

Measure CPU and GPU utilization to find ways to improve your app's performance.

Reducing GPU Utilization in Your RealityKit App

Prevent the GPU from limiting your app's frame rate by reducing the complexity of your render.

{} Construct an immersive environment for visionOS

Build efficient custom worlds for your app.

Passing Metal command objects around your application

Build a system that creates and passes Metal command objects to entities dispatching Metal compute shaders.

```
protocol Resource
```

A shared resource you use to configure a component, like a material, mesh, or texture.