Article

Reducing GPU Utilization in Your RealityKit App

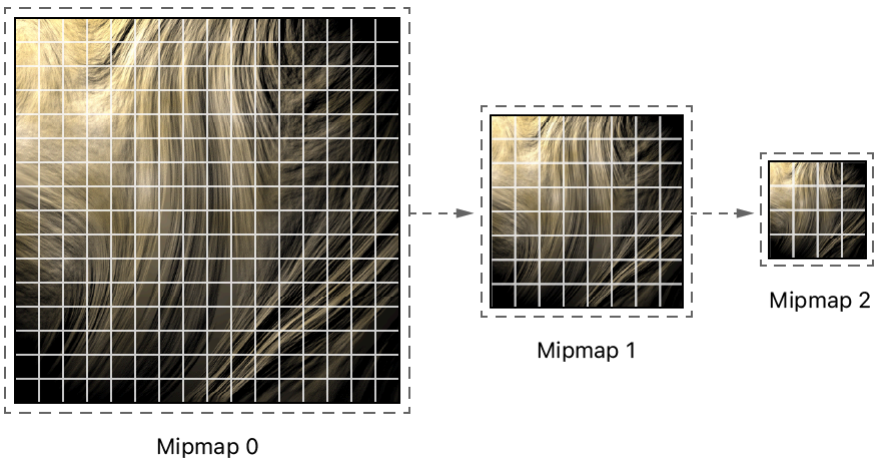
Prevent the GPU from limiting your app's frame rate by reducing the complexity of your render.

Overview

When you use RealityKit, the framework manages GPU access for you. However, your design decisions affect whether your app tries to use more than the available GPU capacity. If you find that your app is GPU limited, as described in Improving the Performance of a RealityKit App, consider ways to simplify your app's content, simplify the render, or both.

Avoid unnecessarily complex textures

RealityKit helps you get the most out of textures. For example, the framework automatically generates and uses *mipmaps*, which are a series of progressively lower resolution variants of your texture that improve render times when applied to distant objects.



You can take additional steps to minimize memory consumption and reduce GPU utilization:

- Use the smallest texture that still produces visually appealing results. Textures rendered on an iPhone or an iPad don't need more than about 2048 pixels on a side. Also, it's typically most efficient to use sizes that are powers of 2.
- Combine multiple textures into a single texture atlas, where possible. Having fewer individual texture files helps to reduce memory consumption.
- Use multiple textures of varying sizes to cover a mesh that doesn't require maximum resolution everywhere.

Reducing texture information also helps to reduce your app's memory footprint. To estimate the amount of memory an uncompressed texture consumes in bytes, use the following formula:

```
memory = channels × width × height × 4/3
```

The `channels` parameter indicates the number of color channels in the texture. The `width` times the `height` counts total pixels in each channel. The factor of `4/3` accounts for the lower resolution variants added by the mipmapping that RealityKit automatically performs.

For example, a texture describing a base color with four channels — one for each of red, green, blue, and alpha — stored in a usdz file as a PNG with dimensions of 2048 px × 2048 px consumes a little more than 22 MB.

Maintain a modest polygon count

One way to reduce the complexity of the render is to selectively reduce the polygon count of your models. Reserve high complexity for the models, or parts of models, that the user is likely to examine most closely. Reduce the complexity of background models, or regions of lesser importance.

A reduced polygon count can be especially helpful if you have a lot of shadows in the scene.

Choose render effects carefully

Polygon count doesn't affect the time needed for effects like depth of field or motion blur. Nevertheless, these effects do consume GPU time. So carefully select which effects to include by balancing your app's visual appearance requirements against its performance constraints.

You deactivate effects by adding options like `disableMotionBlur` to the `renderOptions` option set, and activate the effect by removing the option:

```
// Turn off motion blur.
arView.renderOptions.insert(.disableMotionBlur)
```

```
// Turn on motion blur.
arView.renderOptions.remove(.disableMotionBlur)
```

When you initialize a view, the system automatically inserts options to disable certain effects, depending on the GPU capabilities. But you can override this automatic selection by changing the option set at any time to suit your needs.

Experiment to figure out which combination of effects produces the best result. Be aware that disabling some of these effects can also benefit CPU utilization, specifically render thread time. For the complete list of effects that you can control, see `ARView.RenderOptions`.

Keep transparency and shadows to a minimum

Be careful to avoid one or more layers of transparent material appearing atop other rendered materials. This causes the fragment shader to run on the affected pixels once for each rendered layer. This is a particular problem when objects appear large in the display, affecting many pixels.

Likewise, the more lights and shadows you have in your scene, the more work the fragment shader does. Look for ways to reduce the number of distinct lights.

Simplify skeletal animations

Both skeletal animations and joint transforms result in deformer work, which runs on the GPU. This work can be very costly. Look for ways to minimize the number of joints, and the overall complexity of the animation.

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 CPU Utilization in Your RealityKit App
  Target specific CPU metrics with adjustments to your app and its content.
- 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.