Sample Code

Sharing texture data between the Model I/O framework and the vImage library

Use Model I/O and vImage to composite a photograph over a computer-generated sky.

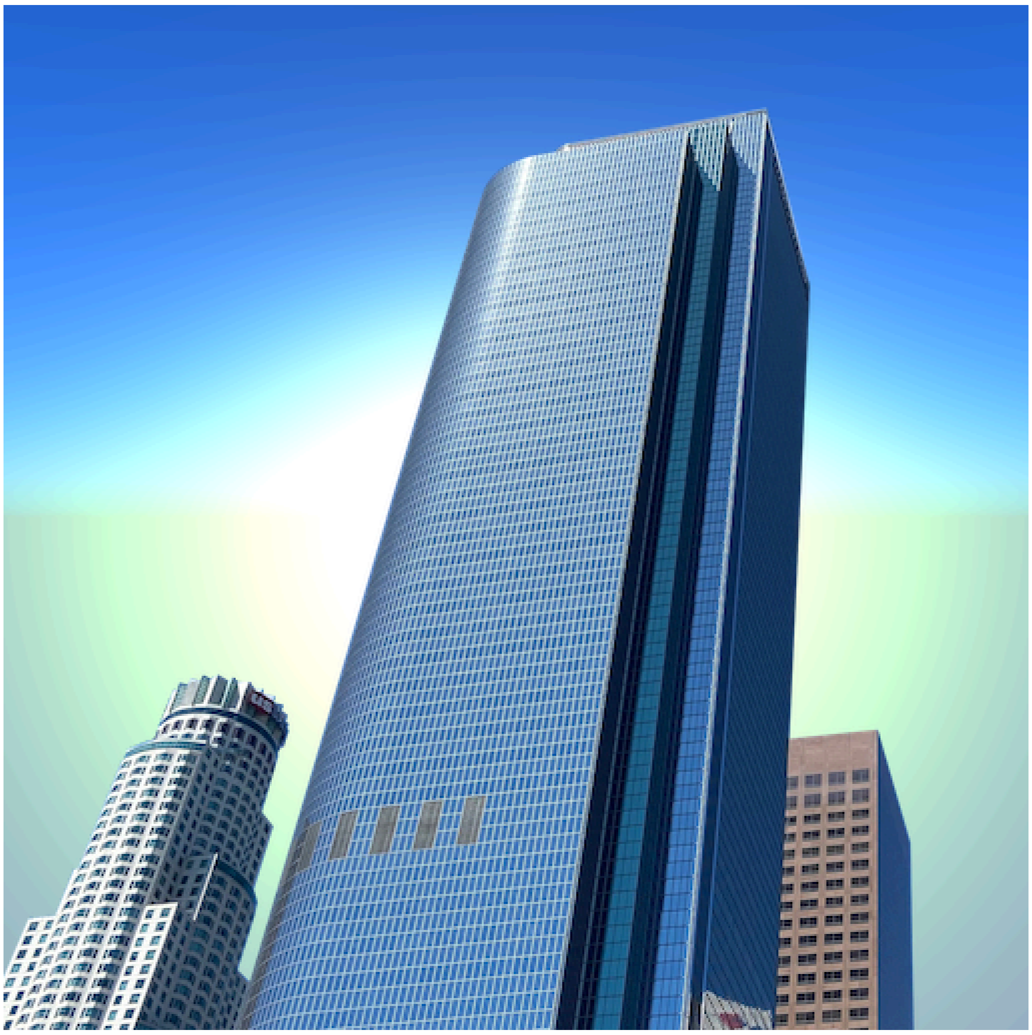
Download

macOS 14.0+ | Xcode 15.0+

Overview

The Model I/O framework provides the `MDLTexture` class and its subclasses to generate procedural textures such as noise, normal maps, and realistic sky boxes. This sample code project uses an `MDLSkyCubeTexture` instance to generate a physically realistic simulation of a sunlit sky. The code uses the generated sky image as the background and a photograph of a building as the foreground.

The image below shows the final composition:



Using the UI, someone can define the parameters that control the sky simulation such as upper atmosphere scattering and sun elevation. Before exploring the code, try building and running the app to get familiar with the effect of the different parameters on the image.

Create the sky texture generator

The `ImageProvider` class declares constants for the source image's dimensions and the `MDLSkyCubeTexture` instance named `skyGenerator`:

```
let width: Int
var height: Int

let skyGenerator: MDLSkyCubeTexture
```

The initializer creates the sky generator instance that's the same size as the top layer image of the skyscraper:

```
width = foregroundImage.width
height = foregroundImage.height
```

```
skyGenerator = MDLSkyCubeTexture(name: nil,
                                 channelEncoding: .uInt8,
                                 textureDimensions: .init(x: Int32(width),
                                                          y: Int32(height)),
                                 turbidity: 0,
                                 sunElevation: 0,
                                 upperAtmosphereScattering: 0,
```

Update the sky texture generator parameters

With each change to the SwiftUI `Picker` controls that define the sky generator parameters, the app calls the `renderSky()` function. The function sets the sky generator parameters and calls `update()` to generate new texel data:

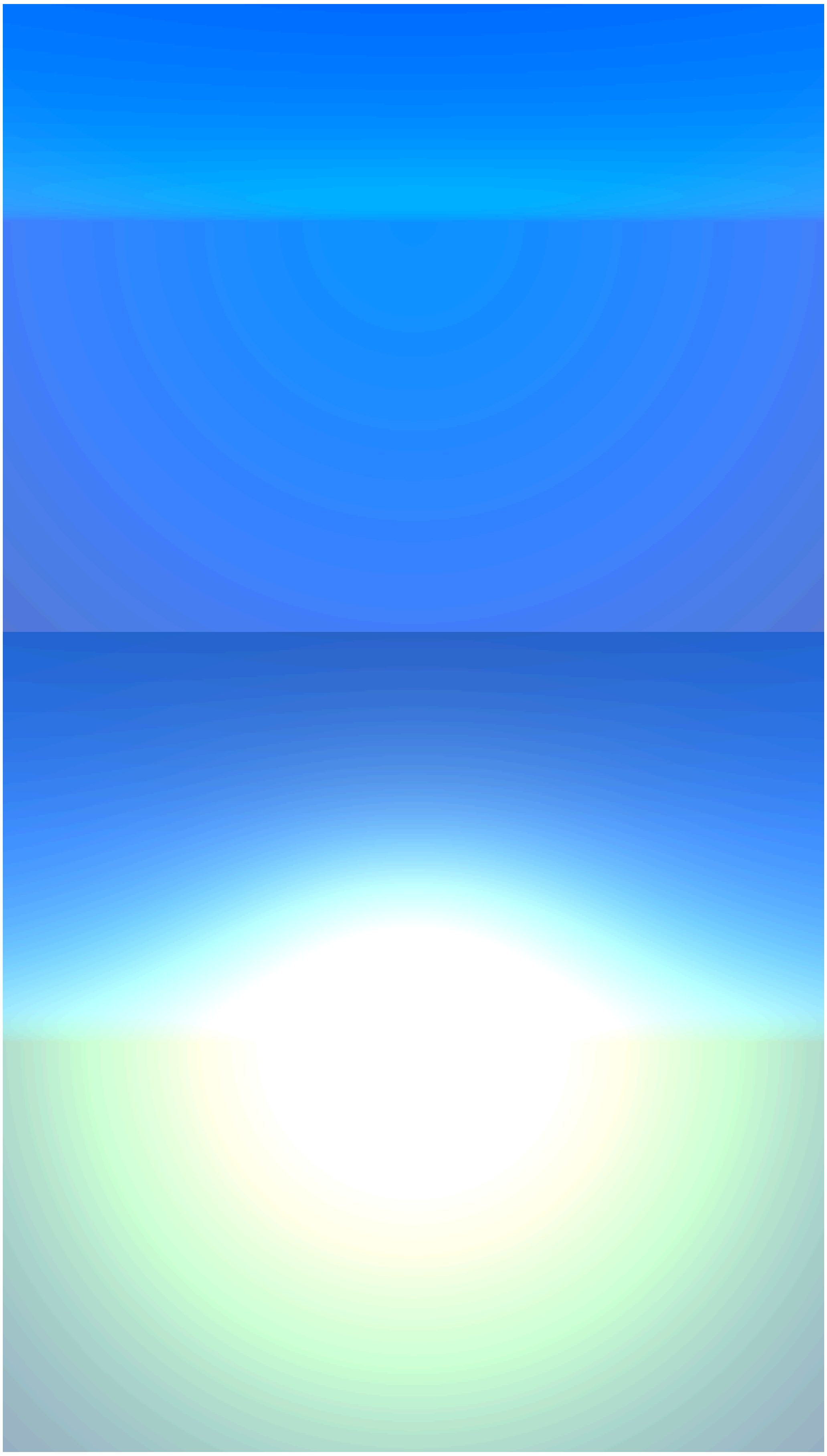
```
skyGenerator.turbidity = turbidity
skyGenerator.sunElevation = sunElevation
skyGenerator.upperAtmosphereScattering = upperAtmosphereScattering
skyGenerator.groundAlbedo = groundAlbedo

skyGenerator.update()
```

Create the composite image

The `texelDataWithTopLeftOrigin()` method returns the sky generator's image data organized such that its first pixel represents the top-left corner of the image. This layout matches the `vImage.PixelBuffer` layout. The code passes the texel data to the doc://com.apple.documentation/foundation/data/3139154-withunsafebytes function to work with the underlying bytes of the data's contiguous storage.

```
let img = skyGenerator.texelDataWithTopLeftOrigin()?.withUnsafeBytes { skyData in
```

The `MDLSkyCubeTexture` instance generates a cube texture that's represented as six sides, vertically stacked.



The code below calculates the range texels that correspond to the selected side (one of `["+X", "-X", "+Y", "-Y", "+Z", "-Z"]`) and binds those to `Pixel_8` values:

```
let imageIndex = ImageProvider.views.firstIndex(of: view) ?? 0
let imagePixelCount = width * height * format.componentCount
let range = imageIndex * imagePixelCount ..< (imageIndex + 1) * imagePixelCount

let values = skyData.bindMemory(to: Pixel_8.self)[ range ]
```

The code below creates a vImage.PixelBuffer structure from the values and, because the alphaComposite(_:topLayer:destination:) method expects ARGB data, permutes the channel order so that alpha channel is first:

```
let buffer = vImage.PixelBuffer(pixelValues: values,
                                size: .init(width: width, height: height),
                                pixelFormat: vImage.Interleaved8x4.self)

buffer.permuteChannels(to: (3, 0, 1, 2), destination: buffer)
```

Finally, the sample code project composites the skyscraper image, represented by foreground Buffer, over the sky image and returns a CGImage instance that contains the result:

```
    buffer.alphaComposite(.nonpremultiplied,
                          topLayer: foregroundBuffer,
                          destination: buffer)

    return buffer.makeCGImage(cgImageFormat: format)
} // Ends `skyGenerator.texelDataWithTopLeftOrigin()?.withUnsafeBytes`.
```

See Also

vImage Pixel Buffers

Using vImage pixel buffers to generate video effects

Render real-time video effects with the vImage Pixel Buffer.

Applying tone curve adjustments to images

Use the vImage library's polynomial transform to apply tone curve adjustments to images.

Adjusting the brightness and contrast of an image

Use a gamma function to apply a linear or exponential curve.

Adjusting the hue of an image

Convert an image to L*a*b* color space and apply hue adjustment.

{} Calculating the dominant colors in an image

Find the main colors in an image by implementing k-means clustering using the Accelerate framework.

`struct PixelBuffer`

An image buffer that stores an image’s pixel data, dimensions, bit depth, and number of channels.