Image Editing with Depth

Add a new dimension to your images

Session 508

Etienne Guerard, Image Editor-in-Chief

What is depth?

Loading depth data

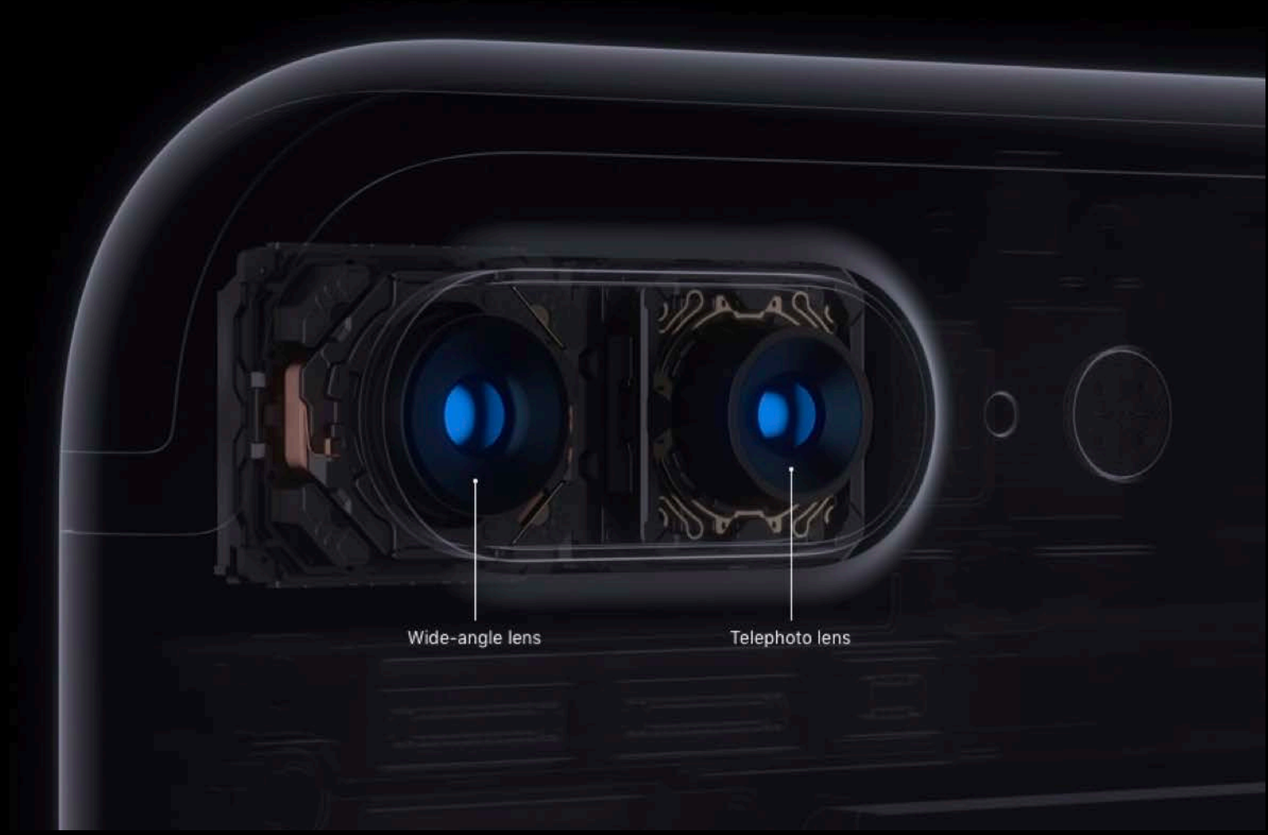
Filtering with depth data

Saving depth data

What Is Depth?

What Is Depth?

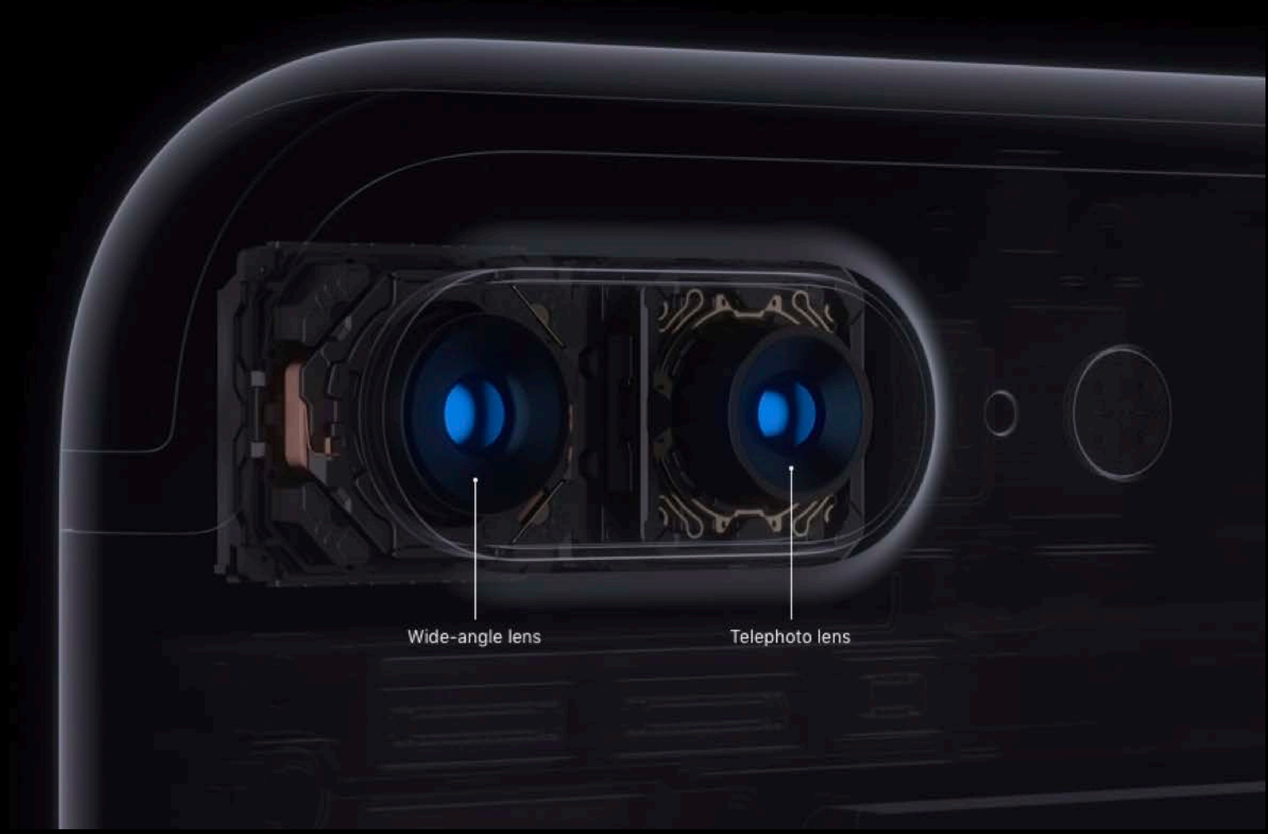
iPhone 7+



What Is Depth?

iPhone 7+

iOS 11

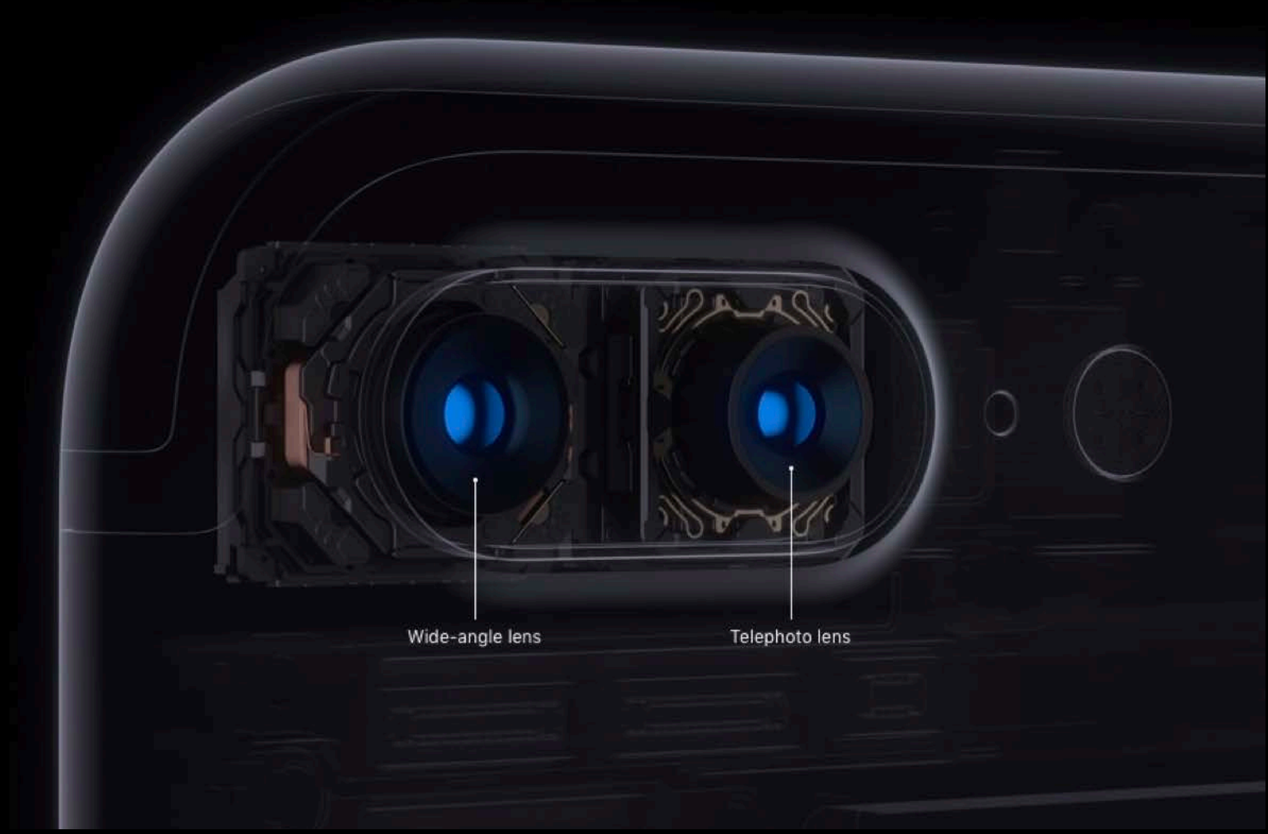


What Is Depth?

iPhone 7+

iOS 11

Dual camera system



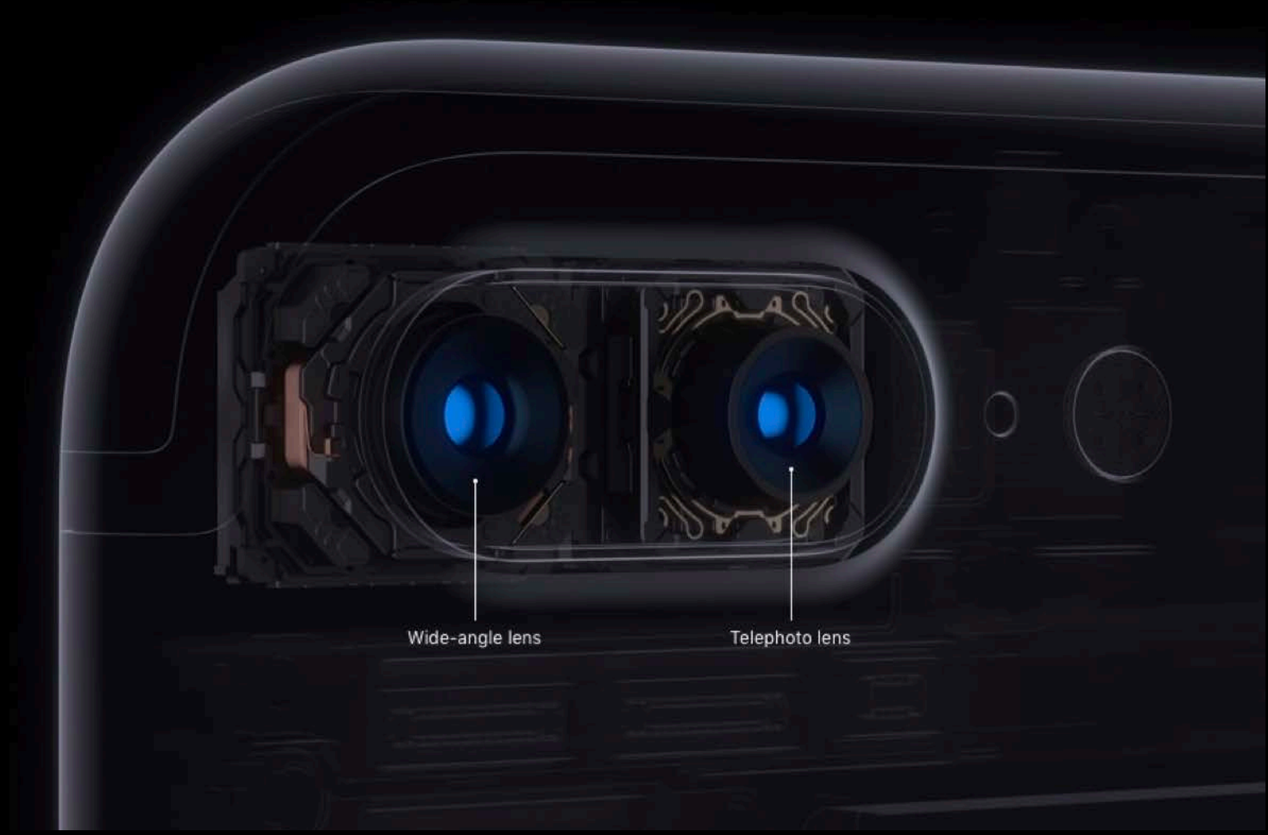
What Is Depth?

iPhone 7+

iOS 11

Dual camera system

Disparity



What Is Depth?

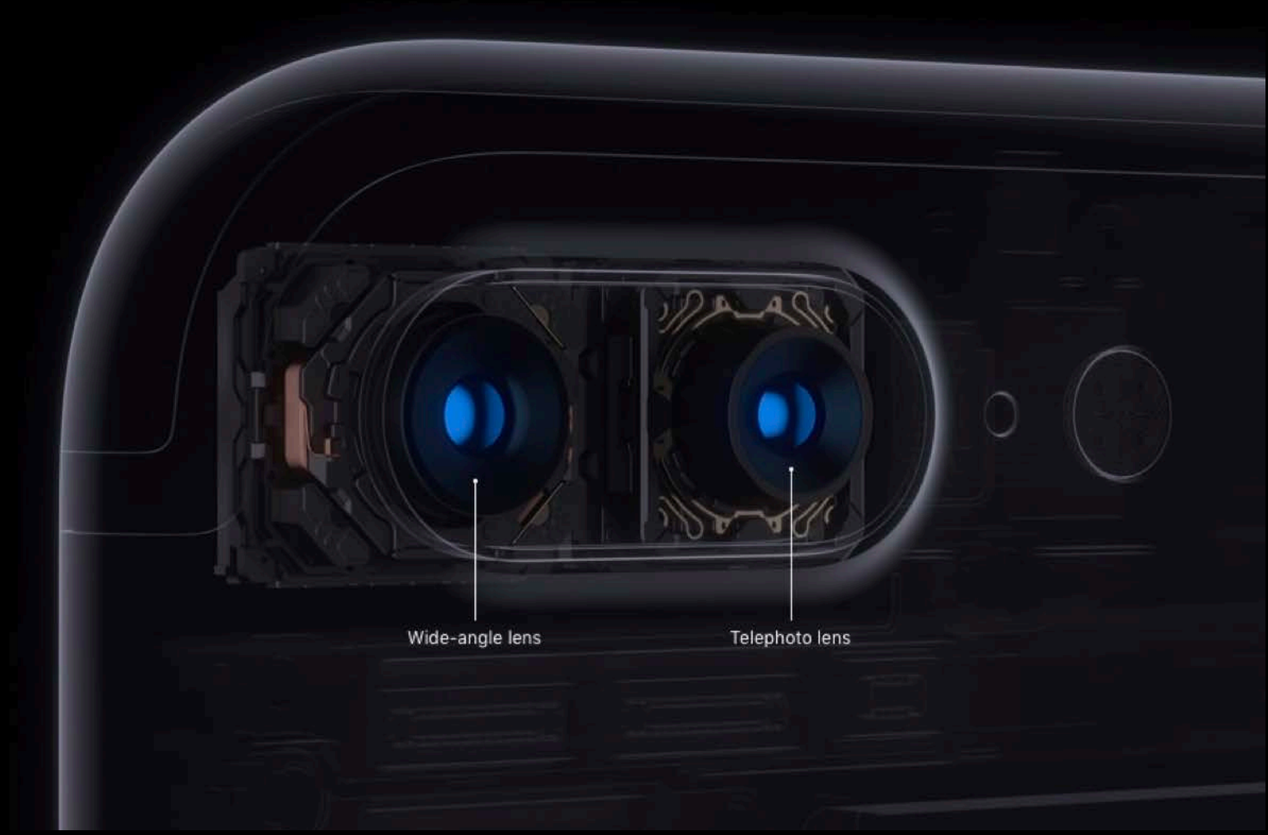
iPhone 7+

iOS 11

Dual camera system

Disparity

Depth = 1 / Disparity



What Is Depth?

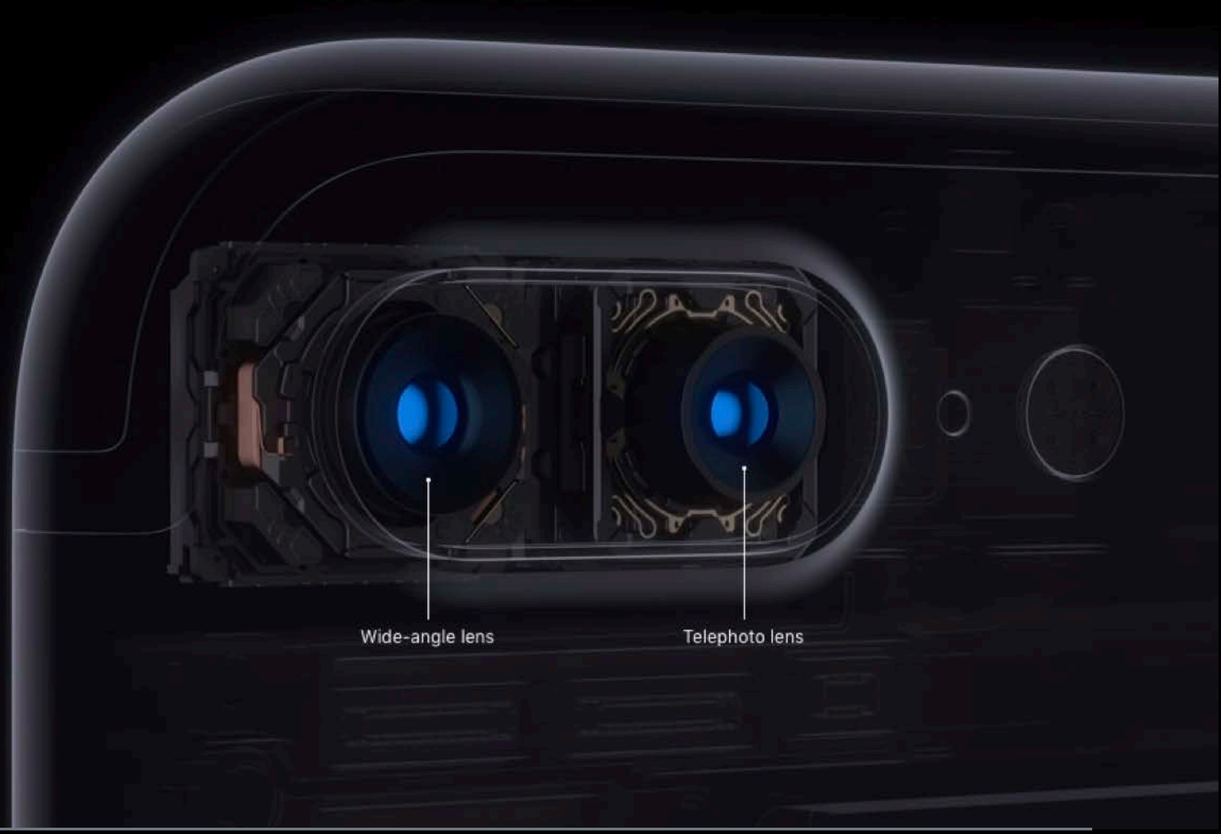
iPhone 7+

iOS 11

Dual camera system

Disparity

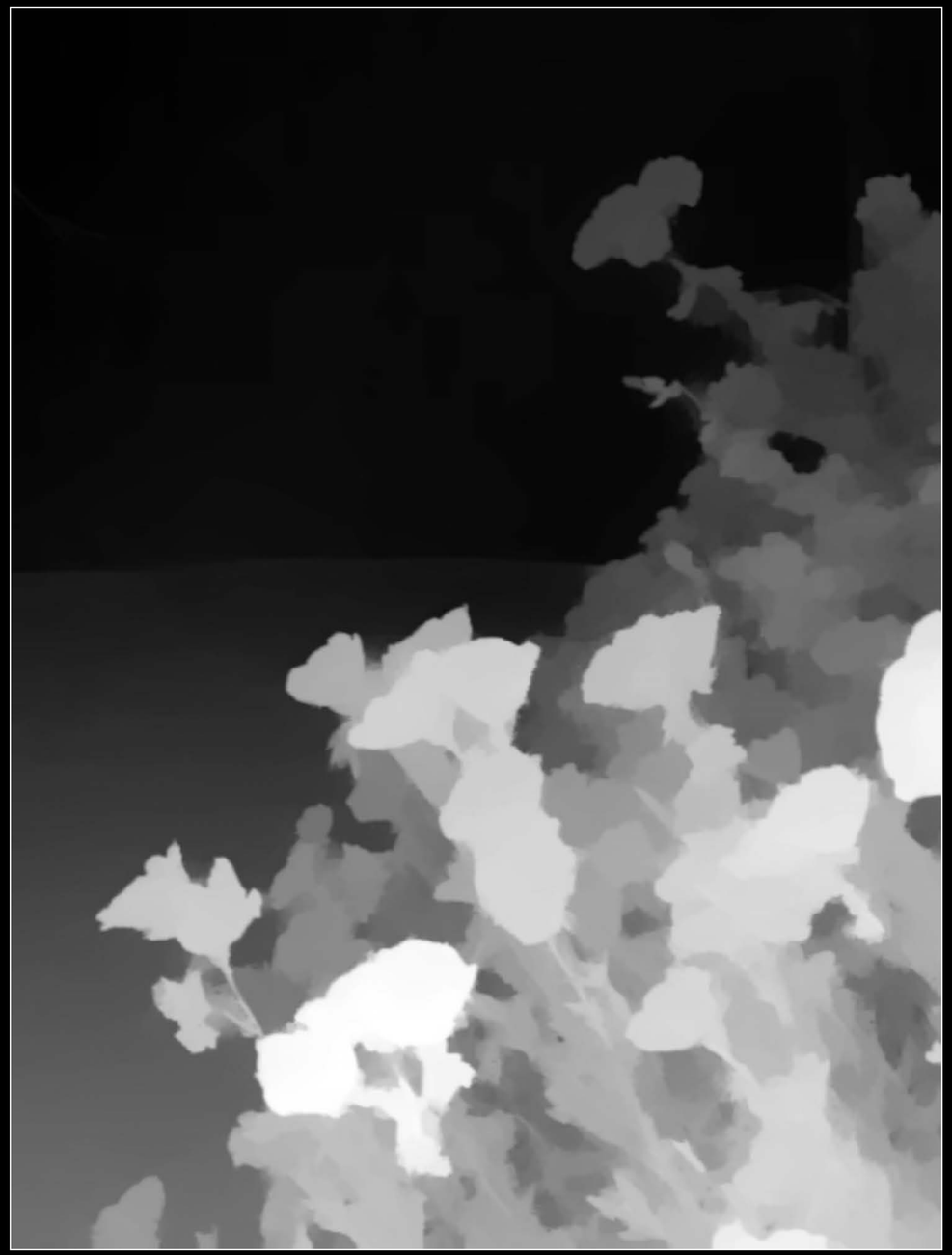
Depth = 1 / Disparity



Depth Explorer

Craig Milito, Senior Software Engineer

Depth Effects



Who Could Use Depth?

Who Could Use Depth?

Editing app

Who Could Use Depth?

Editing app

Camera app

Who Could Use Depth?

Editing app

Camera app

Sharing app

What is depth?

Loading depth data

Filtering with depth data

Saving depth data

Loading Depth Data

Loading Depth Data

Image file

Loading Depth Data

Image file

Auxiliary data

Loading Depth Data

Image file

Auxiliary data

No depth in UIImage!

Accessing Image File Data with PhotoKit

Accessing Image File Data with PhotoKit

PHContentEditingInput

Accessing Image File Data with PhotoKit

PHContentEditingInput

```
let asset: PHAsset
asset.requestContentEditingInput(with: nil) { input, info in
    let imageURL = input?.fullSizeImageURL
    // ...
}
```

Accessing Image File Data with PhotoKit

PHContentEditingInput

```
let asset: PHAsset
asset.requestContentEditingInput(with: nil) { input, info in
    let imageURL = input?.fullSizeImageURL
    // ...
}
```

Accessing Image File Data with PhotoKit

PHContentEditingInput

PHImageManager

Accessing Image File Data with PhotoKit

PHContentEditingInput

PHImageManager

```
let asset: PHAsset
PHImageManager.default().requestImageData(for: asset, options: nil) {
    imageData, dataType, orientation, info in
    // ...
}
```

Accessing Image File Data with PhotoKit

PHContentEditingInput

PHImageManager

```
let asset: PHAsset
PHImageManager.default().requestImageData(for: asset, options: nil) {
    imageData, dataType, orientation, info in
    // ...
}
```

Testing for Depth Data with ImageIO

```swift
// Checking Whether an Image File Contains Depth Data

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Get top-level image source properties
let sourceProperties = CGImageSourceCopyProperties(source, nil)

print("Source properties: \(sourceProperties)")
```

```swift
// Checking Whether an Image File Contains Depth Data

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Get top-level image source properties
let sourceProperties = CGImageSourceCopyProperties(source, nil)

print("Source properties: \(sourceProperties)")
```

```swift
// Checking Whether an Image File Contains Depth Data

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Get top-level image source properties
let sourceProperties = CGImageSourceCopyProperties(source, nil)

print("Source properties: \(sourceProperties)")
```

```
// Example Image Source Properties

kCGImagePropertyFileContentsDictionary = {
     kCGImagePropertyImages = {
         kCGImagePropertyAuxiliaryData = {
                     kCGImagePropertyAuxiliaryDataType = kCGImageAuxiliaryDataTypeDisparity
                     kCGImagePropertyWidth = 768
                     kCGImagePropertyHeight = 576
                 }
             kCGImagePropertyWidth = 4032
             kCGImagePropertyHeight = 3024
    }
    ...
}
```

```
// Example Image Source Properties

kCGImagePropertyFileContentsDictionary = {
     kCGImagePropertyImages = {
          kCGImagePropertyAuxiliaryData = {
                    kCGImagePropertyAuxiliaryDataType = kCGImageAuxiliaryDataTypeDisparity
                    kCGImagePropertyWidth = 768
                    kCGImagePropertyHeight = 576
               }
            kCGImagePropertyWidth = 4032
            kCGImagePropertyHeight = 3024
    }
    ...
}
```

```
// Example Image Source Properties

kCGImagePropertyFileContentsDictionary = {
     kCGImagePropertyImages = {
         kCGImagePropertyAuxiliaryData = {
                     kCGImagePropertyAuxiliaryDataType = kCGImageAuxiliaryDataTypeDisparity
                     kCGImagePropertyWidth = 768
                     kCGImagePropertyHeight = 576
                 }
             kCGImagePropertyWidth = 4032
             kCGImagePropertyHeight = 3024
    }
    ...
}
```

```
// Example Image Source Properties

kCGImagePropertyFileContentsDictionary = {
     kCGImagePropertyImages = {
         kCGImagePropertyAuxiliaryData = {
                     kCGImagePropertyAuxiliaryDataType = kCGImageAuxiliaryDataTypeDisparity
                     kCGImagePropertyWidth = 768
                     kCGImagePropertyHeight = 576
                 }
             kCGImagePropertyWidth = 4032
             kCGImagePropertyHeight = 3024
    }
    ...
}
```

```
// Example Image Source Properties

kCGImagePropertyFileContentsDictionary = {
     kCGImagePropertyImages = {
         kCGImagePropertyAuxiliaryData = {
                     kCGImagePropertyAuxiliaryDataType = kCGImageAuxiliaryDataTypeDisparity
                     kCGImagePropertyWidth = 768
                     kCGImagePropertyHeight = 576
                 }
             kCGImagePropertyWidth = 4032
             kCGImagePropertyHeight = 3024
    }
    ...
}
```

Reading Depth Data

Reading Depth Data

Auxiliary data

Reading Depth Data

Auxiliary data

AVDepthData

Reading Depth Data

Auxiliary data

AVDepthData

CVPixelBuffer

Reading Depth Data

Auxiliary data

AVDepthData

CVPixelBuffer

Single channel

Reading Depth Data

Auxiliary data

AVDepthData

CVPixelBuffer

Single channel

Depth or disparity

Reading Depth Data

Auxiliary data

AVDepthData

CVPixelBuffer

Single channel

Depth or disparity

16-bit or 32-bit float

```
// Reading depth data from an image source

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Read auxiliary data of type disparity
let auxDataType = kCGImageAuxiliaryDataTypeDisparity
let auxDataInfo = CGImageSourceCopyAuxiliaryDataInfoAtIndex(source, 0, auxDataType)

if auxDataInfo == nil {
    return // Image doesn't contain disparity auxiliary data
}
```

```
// Reading depth data from an image source

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Read auxiliary data of type disparity
let auxDataType = kCGImageAuxiliaryDataTypeDisparity
let auxDataInfo = CGImageSourceCopyAuxiliaryDataInfoAtIndex(source, 0, auxDataType)

if auxDataInfo == nil {
    return // Image doesn't contain disparity auxiliary data
}
```

```
// Reading depth data from an image source

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Read auxiliary data of type disparity
let auxDataType = kCGImageAuxiliaryDataTypeDisparity
let auxDataInfo = CGImageSourceCopyAuxiliaryDataInfoAtIndex(source, 0, auxDataType)

if auxDataInfo == nil {
    return // Image doesn't contain disparity auxiliary data
}
```

```swift
// Reading depth data from an image source

import ImageIO

let fileURL: URL
let source = CGImageSourceCreateWithURL(fileURL, nil)

// Read auxiliary data of type disparity
let auxDataType = kCGImageAuxiliaryDataTypeDisparity
let auxDataInfo = CGImageSourceCopyAuxiliaryDataInfoAtIndex(source, 0, auxDataType)

if auxDataInfo == nil {
    return // Image doesn't contain disparity auxiliary data
}
```

```
// Loading depth data into a pixel buffer

import AVFoundation
import CoreVideo

// Create a depth data object from auxiliary data
var depthData = try AVDepthData(fromDictionaryRepresentation: auxDataInfo)

// Check native depth data type
if depthData.depthDataType != kCVPixelFormatType_DisparityFloat16 {
    // Convert to half-float disparity data
    depthData = depthData.converting(toDepthDataType: kCVPixelFormatType_DisparityFloat16)
}

// Get the underlying buffer
let depthMap: CVPixelBuffer = depthData.depthDataMap
```

```
// Loading depth data into a pixel buffer

import AVFoundation
import CoreVideo

// Create a depth data object from auxiliary data
var depthData = try AVDepthData(fromDictionaryRepresentation: auxDataInfo)

// Check native depth data type
if depthData.depthDataType != kCVPixelFormatType_DisparityFloat16 {
    // Convert to half-float disparity data
    depthData = depthData.converting(toDepthDataType: kCVPixelFormatType_DisparityFloat16)
}

// Get the underlying buffer
let depthMap: CVPixelBuffer = depthData.depthDataMap
```

```
// Loading depth data into a pixel buffer

import AVFoundation
import CoreVideo

// Create a depth data object from auxiliary data
var depthData = try AVDepthData(fromDictionaryRepresentation: auxDataInfo)

// Check native depth data type
if depthData.depthDataType != kCVPixelFormatType_DisparityFloat16 {
    // Convert to half-float disparity data
    depthData = depthData.converting(toDepthDataType: kCVPixelFormatType_DisparityFloat16)
}

// Get the underlying buffer
let depthMap: CVPixelBuffer = depthData.depthDataMap
```

```swift
// Loading depth data into a pixel buffer

import AVFoundation
import CoreVideo

// Create a depth data object from auxiliary data
var depthData = try AVDepthData(fromDictionaryRepresentation: auxDataInfo)

// Check native depth data type
if depthData.depthDataType != kCVPixelFormatType_DisparityFloat16 {
    // Convert to half-float disparity data
    depthData = depthData.converting(toDepthDataType: kCVPixelFormatType_DisparityFloat16)
}

// Get the underlying buffer
let depthMap: CVPixelBuffer = depthData.depthDataMap
```

```
// Loading depth data into a pixel buffer

import AVFoundation
import CoreVideo

// Create a depth data object from auxiliary data
var depthData = try AVDepthData(fromDictionaryRepresentation: auxDataInfo)

// Check native depth data type
if depthData.depthDataType != kCVPixelFormatType_DisparityFloat16 {
    // Convert to half-float disparity data
    depthData = depthData.converting(toDepthDataType: kCVPixelFormatType_DisparityFloat16)
}

// Get the underlying buffer
let depthMap: CVPixelBuffer = depthData.depthDataMap
```

Reading Depth Data with Core Image

Reading Depth Data with Core Image

```
// Create depth image from URL
let depthImage = CIImage(contentsOf: imageURL, options: [kCIImageAuxiliaryDepth : true])

// Get the underlying AVDepthData object backing the CIImage
let depthData = depthImage.depthData

// Convert to disparity using dedicated CIFilter
let disparityImage = depthImage.applyingFilter("CIDepthToDisparity", withInputParameters: nil)
```

Reading Depth Data with Core Image

```
// Create depth image from URL
let depthImage = CIImage(contentsOf: imageURL, options: [kCIImageAuxiliaryDepth : true])

// Get the underlying AVDepthData object backing the CIImage
let depthData = depthImage.depthData

// Convert to disparity using dedicated CIFilter
let disparityImage = depthImage.applyingFilter("CIDepthToDisparity", withInputParameters: nil)
```

Reading Depth Data with Core Image

```
// Create depth image from URL
let depthImage = CIImage(contentsOf: imageURL, options: [kCIImageAuxiliaryDepth : true])

// Get the underlying AVDepthData object backing the CIImage
let depthData = depthImage.depthData

// Convert to disparity using dedicated CIFilter
let disparityImage = depthImage.applyingFilter("CIDepthToDisparity", withInputParameters: nil)
```

Preparing Depth Data

Preparing Depth Data

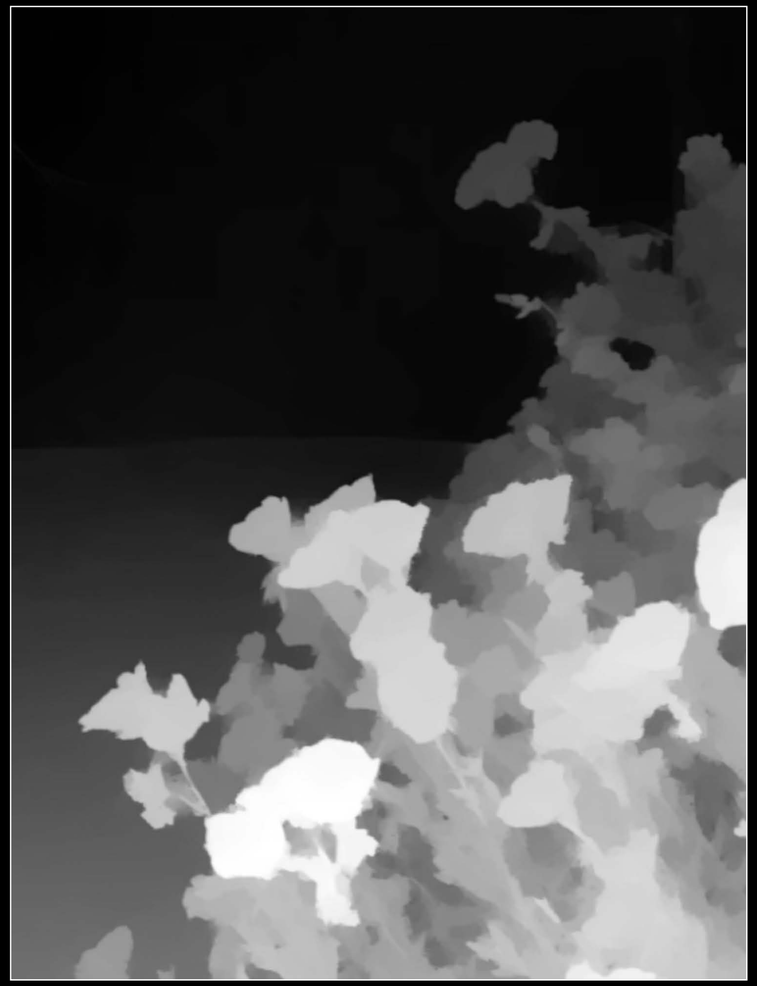
Scaling up

Preparing Depth Data

Scaling up

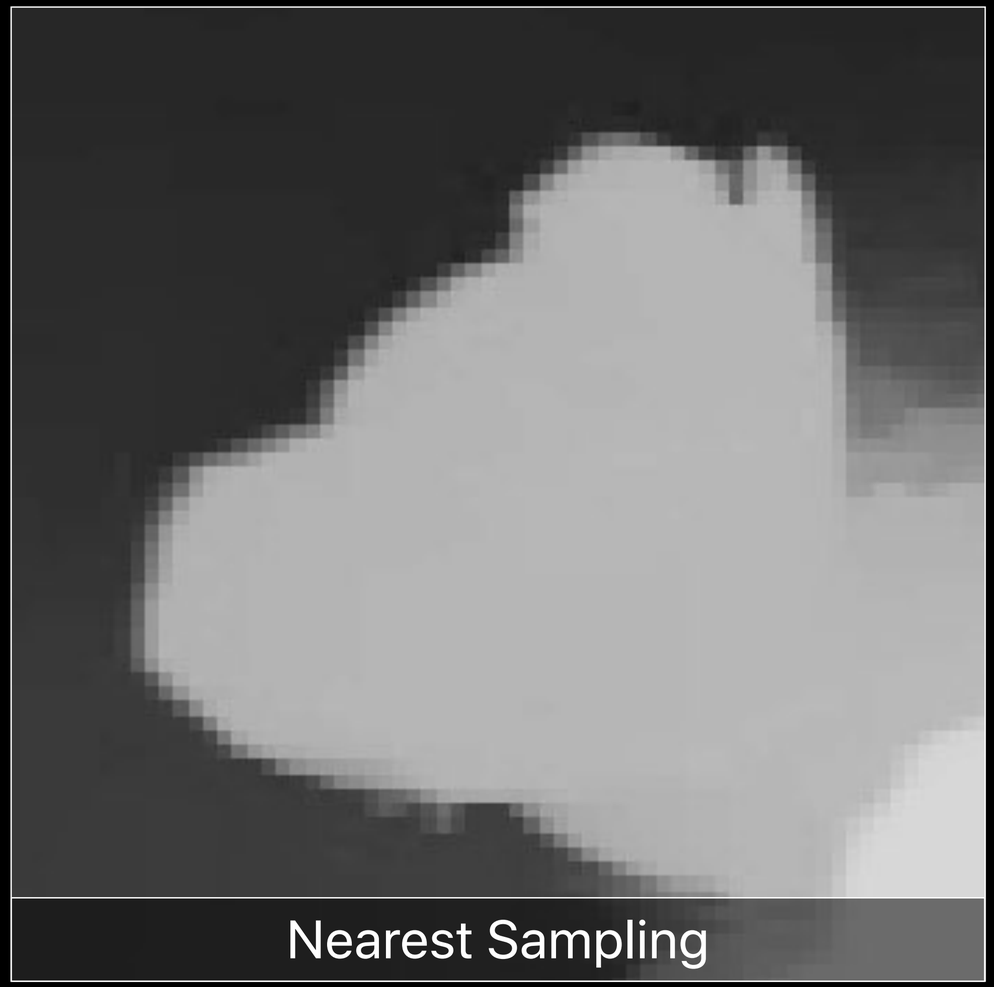
Preparing Depth Data

Scaling up



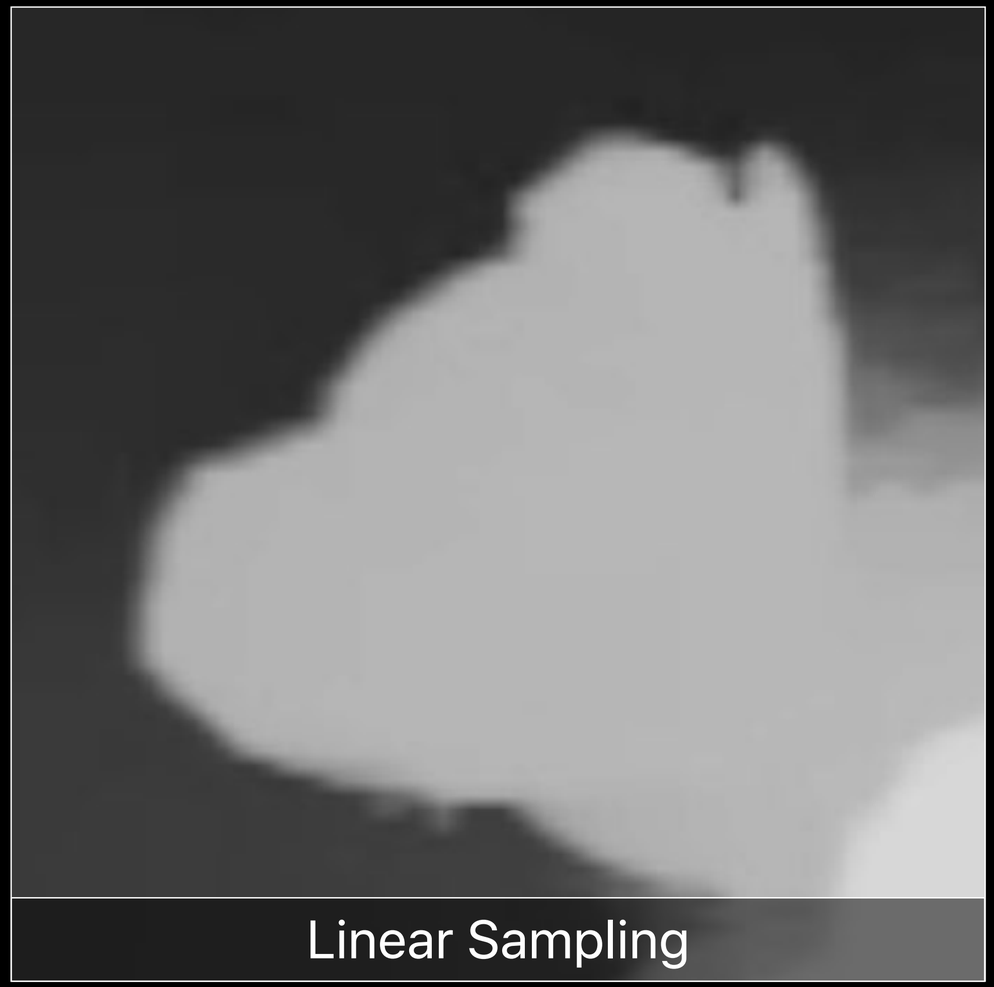
Preparing Depth Data

Scaling up



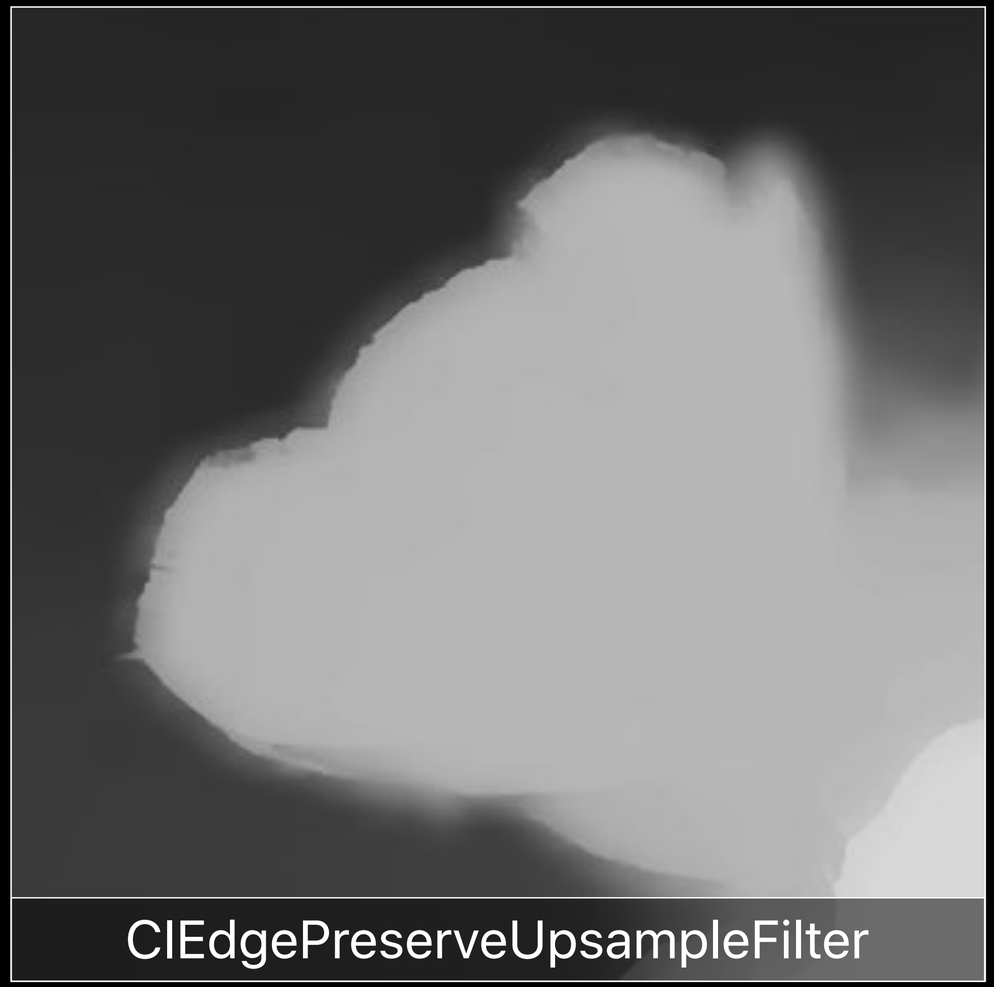
Preparing Depth Data

Scaling up



Preparing Depth Data

Scaling up



Preparing Depth Data

Scaling up

Preparing Depth Data

Scaling up

Computing min/max

Preparing Depth Data

Scaling up

Computing min/max

Normalizing

What is depth?

Loading depth data

Filtering with depth data

Saving depth data

Filtering with Depth Data

Filtering with Depth Data

Simple Background Effects

Filtering with Depth Data

Simple Background Effects

Custom Depth Effect

Filtering with Depth Data

Simple Background Effects

Custom Depth Effect

Depth Blur Effect

Filtering with Depth Data

Simple Background Effects

Custom Depth Effect

Depth Blur Effect

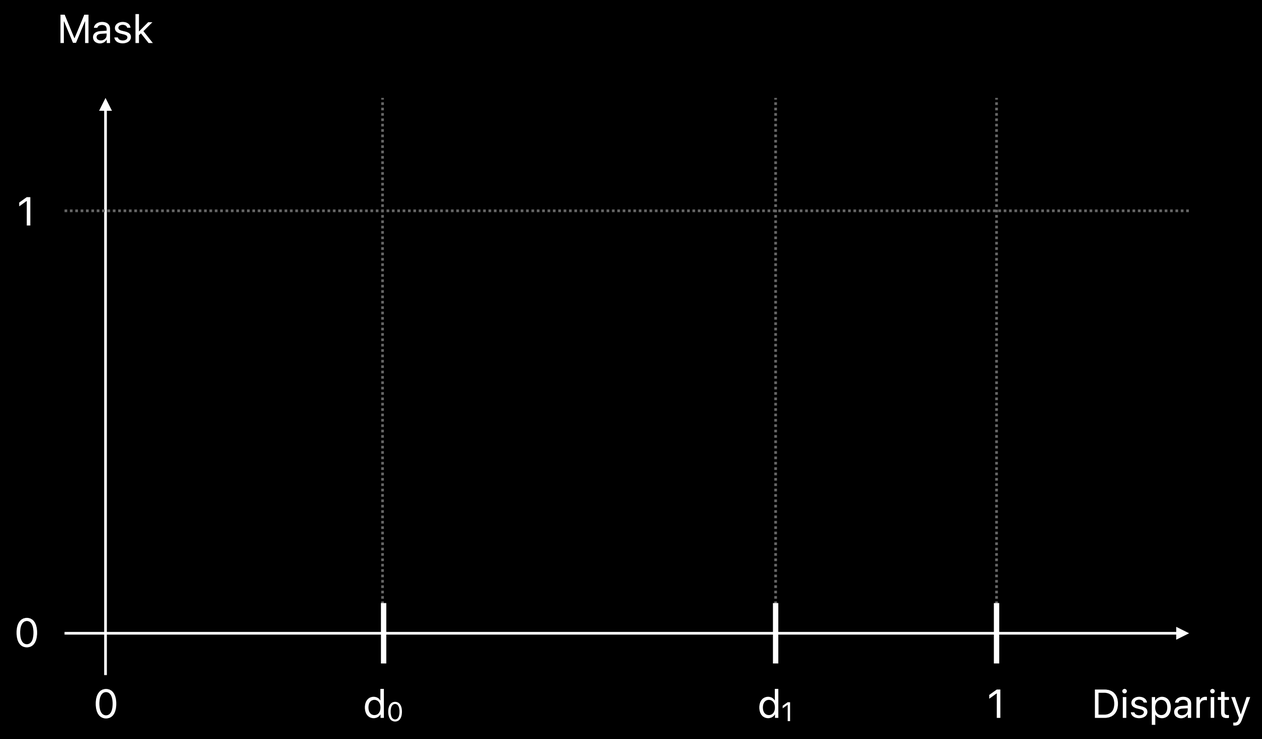
3D Effect

Simple Background Effects

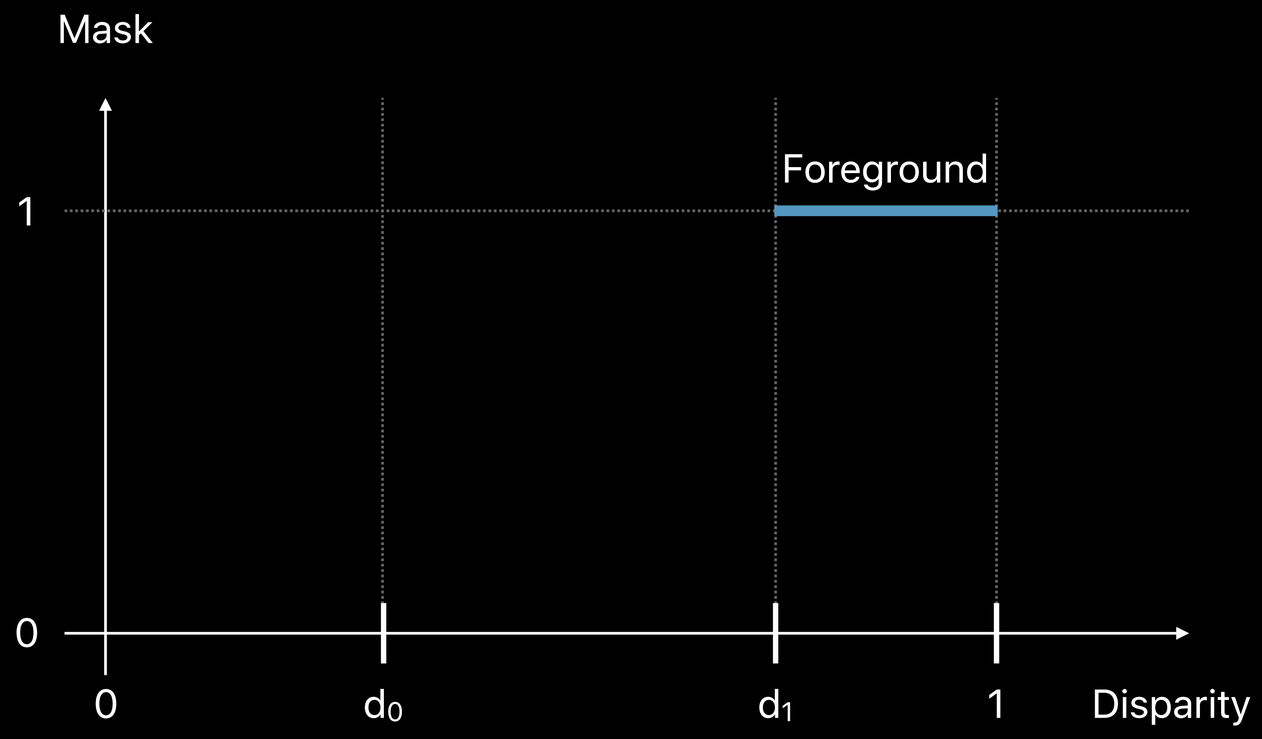
Deep Impact

Stephen Ward, Software Engineer

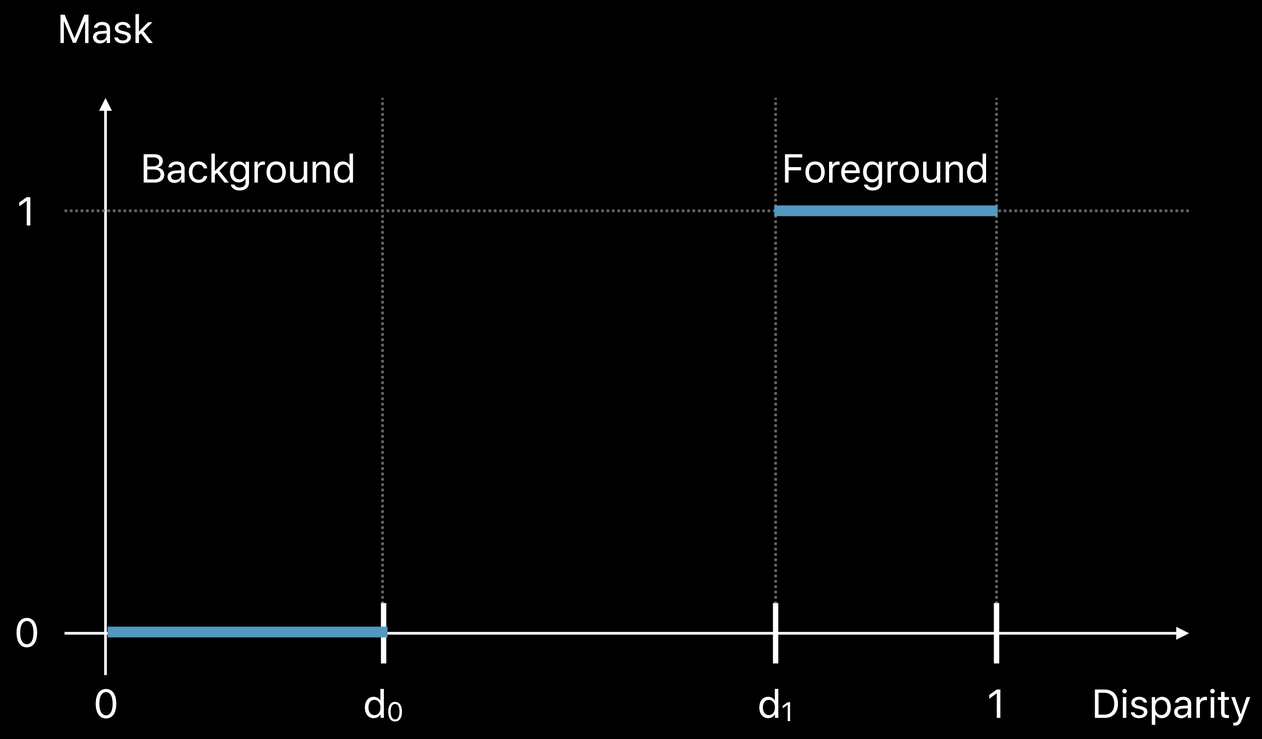
Using Disparity as a Mask



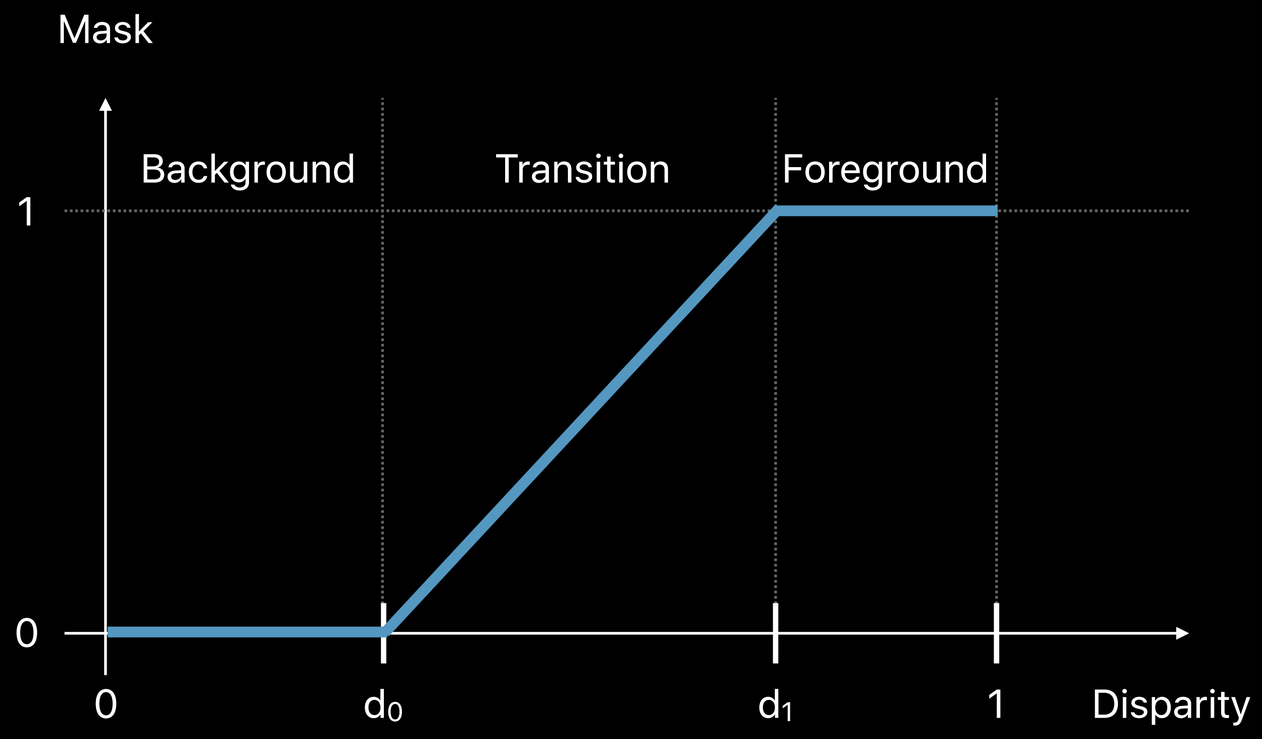
Using Disparity as a Mask



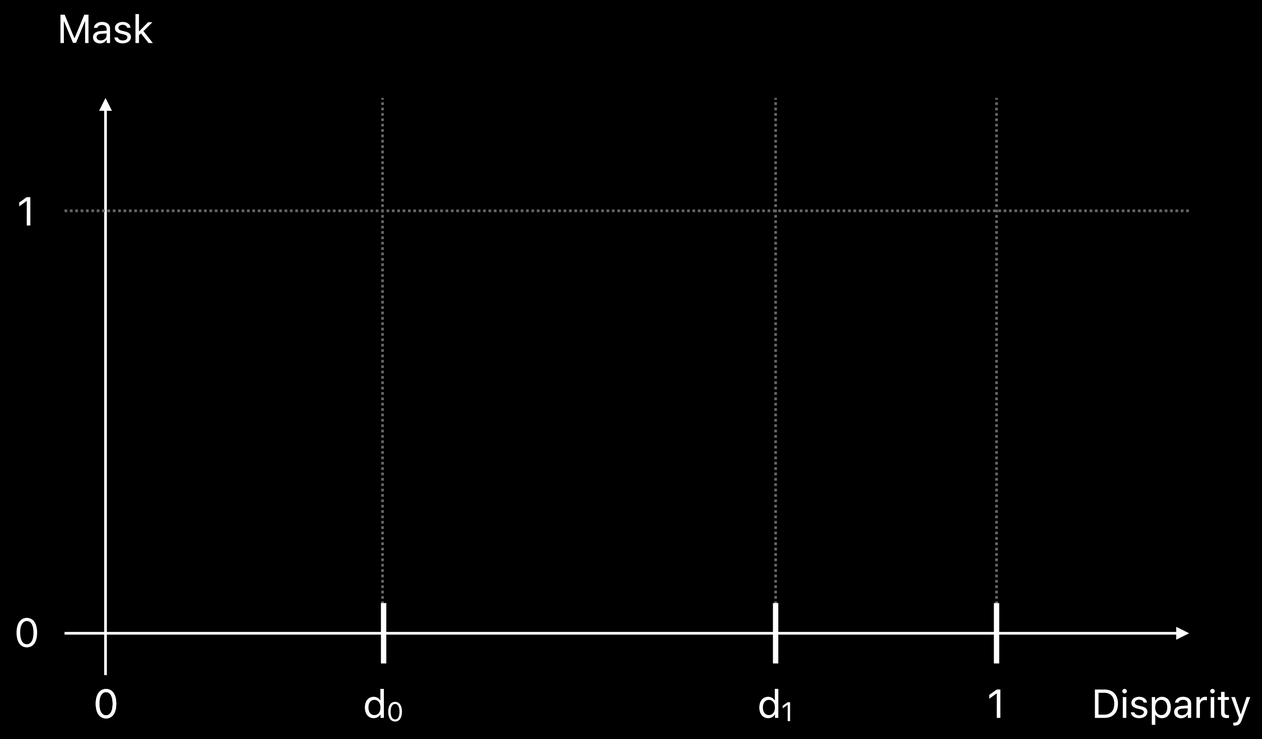
Using Disparity as a Mask



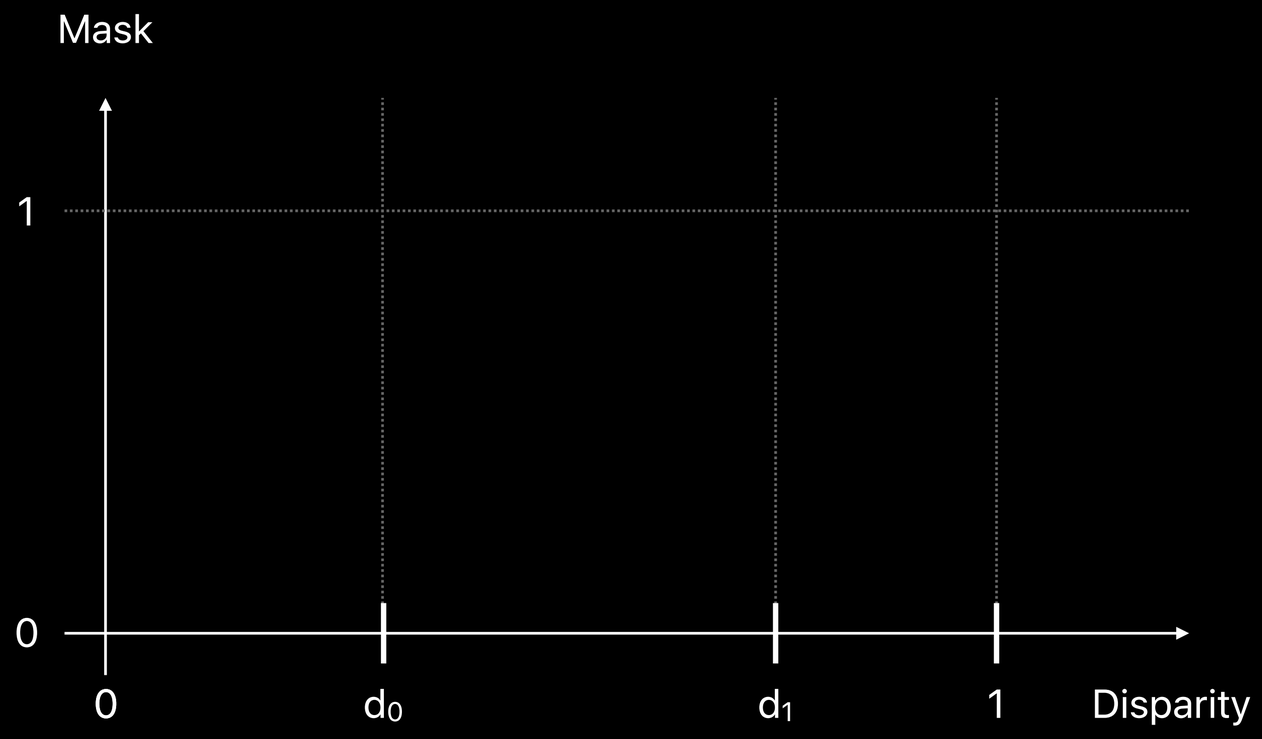
Using Disparity as a Mask



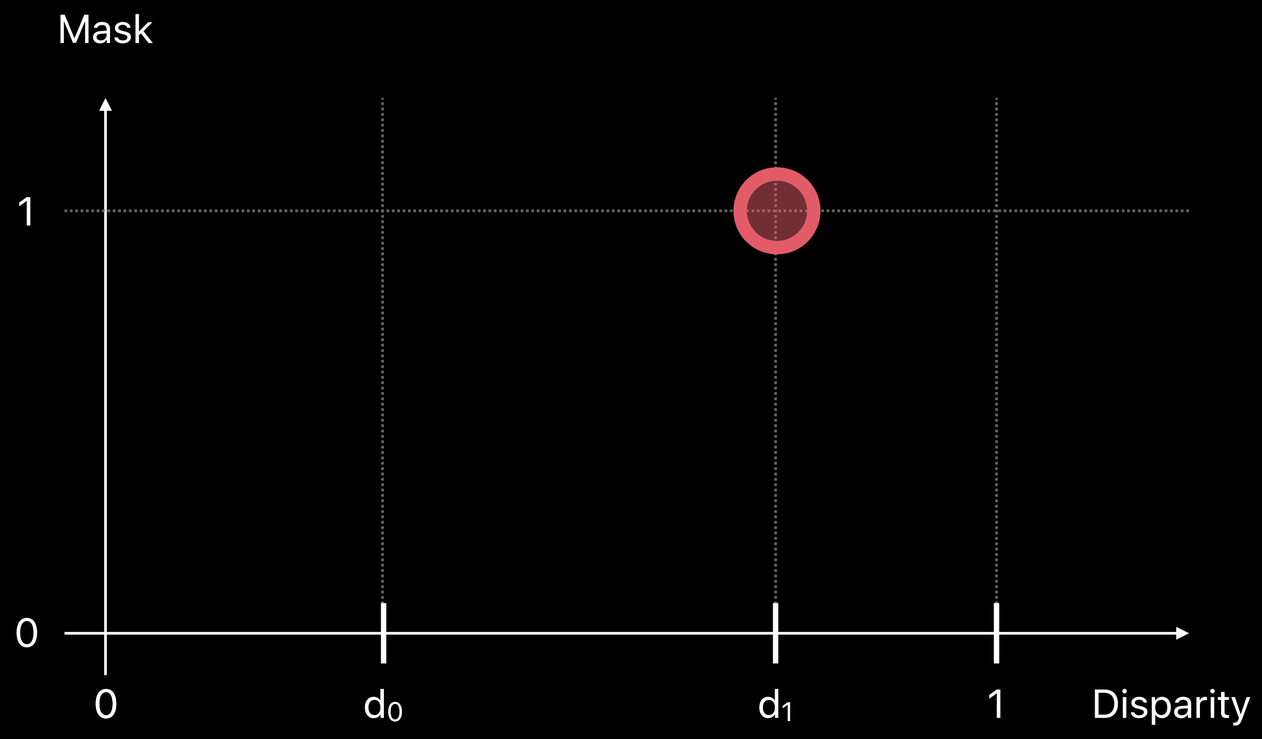
Using Disparity as a Mask



Using Disparity as a Mask



Using Disparity as a Mask



```
// Code to Sample the Disparity Map with a Tap

// Apply the filter with the sampleRect from the user's tap. Don't forget to clamp!
let minMaxImage = normalizedDisparityImage.clampingToExtent().applyingFilter(
    "CIAreaMinMaxRed", withInputParameters: [kCIInputExtentKey : CIVector(cgRect:sampleRect)])

// A four-byte buffer to store a single pixel value
var pixel = [UInt8](repeating: 0, count: 4)

// Render the image to a 1x1 rect. Be sure to use a nil color space.
context.render(minMaxImage, toBitmap: &pixel, rowBytes: 4,
               bounds: CGRect(x:0, y:0, width:1, height:1), format: kCIFormatRGBA8,
               colorSpace: nil)

// The max is stored in the green channel. Min is in the red.
let disparity = Float(pixel[1]) / 255.0
```

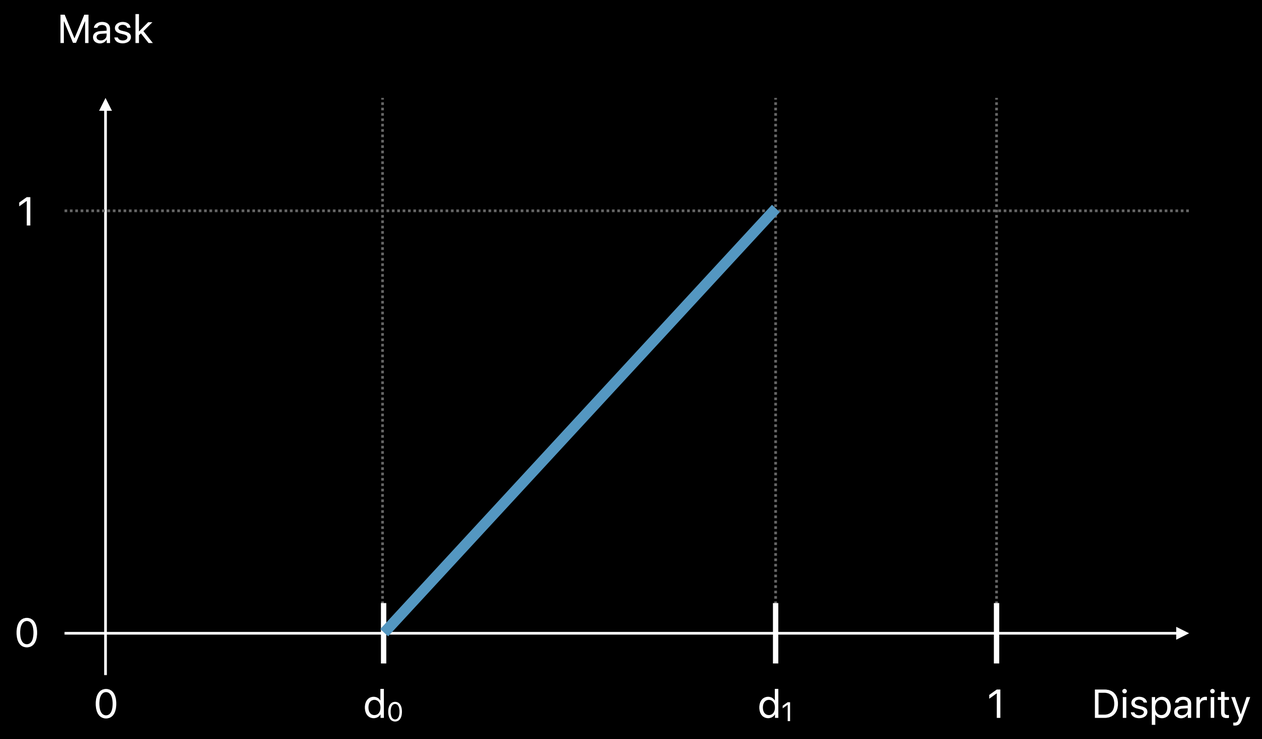
```
// Code to Sample the Disparity Map with a Tap

// Apply the filter with the sampleRect from the user's tap. Don't forget to clamp!
let minMaxImage = normalizedDisparityImage.clampingToExtent().applyingFilter(
    "CIAreaMinMaxRed", withInputParameters: [kCIInputExtentKey : CIVector(cgRect:sampleRect)])

// A four-byte buffer to store a single pixel value
var pixel = [UInt8](repeating: 0, count: 4)

// Render the image to a 1x1 rect. Be sure to use a nil color space.
context.render(minMaxImage, toBitmap: &pixel, rowBytes: 4,
               bounds: CGRect(x:0, y:0, width:1, height:1), format: kCIFormatRGBA8,
               colorSpace: nil)

// The max is stored in the green channel. Min is in the red.
let disparity = Float(pixel[1]) / 255.0
```

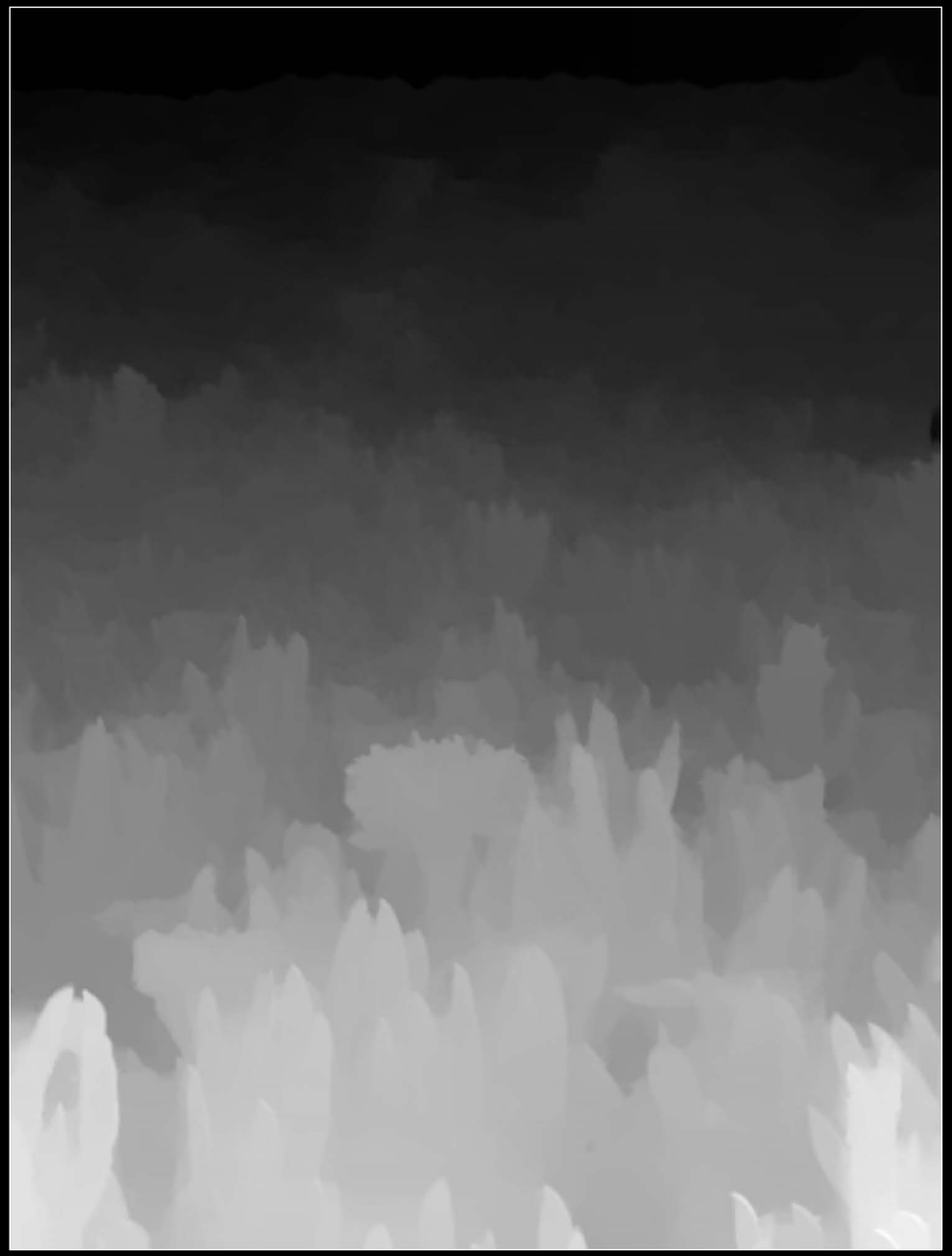
```
// Code to Sample the Disparity Map with a Tap

// Apply the filter with the sampleRect from the user's tap. Don't forget to clamp!
let minMaxImage = normalizedDisparityImage.clampingToExtent().applyingFilter(
    "CIAreaMinMaxRed", withInputParameters: [kCIInputExtentKey : CIVector(cgRect:sampleRect)])

// A four-byte buffer to store a single pixel value
var pixel = [UInt8](repeating: 0, count: 4)

// Render the image to a 1x1 rect. Be sure to use a nil color space.
context.render(minMaxImage, toBitmap: &pixel, rowBytes: 4,
               bounds: CGRect(x:0, y:0, width:1, height:1), format: kCIFormatRGBA8,
               colorSpace: nil)

// The max is stored in the green channel. Min is in the red.
let disparity = Float(pixel[1]) / 255.0
```

```
// Code to Sample the Disparity Map with a Tap

// Apply the filter with the sampleRect from the user's tap. Don't forget to clamp!
let minMaxImage = normalizedDisparityImage.clampingToExtent().applyingFilter(
    "CIAreaMinMaxRed", withInputParameters: [kCIInputExtentKey : CIVector(cgRect:sampleRect)])

// A four-byte buffer to store a single pixel value
var pixel = [UInt8](repeating: 0, count: 4)

// Render the image to a 1x1 rect. Be sure to use a nil color space.
context.render(minMaxImage, toBitmap: &pixel, rowBytes: 4,
               bounds: CGRect(x:0, y:0, width:1, height:1), format: kCIFormatRGBA8,
               colorSpace: nil)

// The max is stored in the green channel. Min is in the red.
let disparity = Float(pixel[1]) / 255.0
```

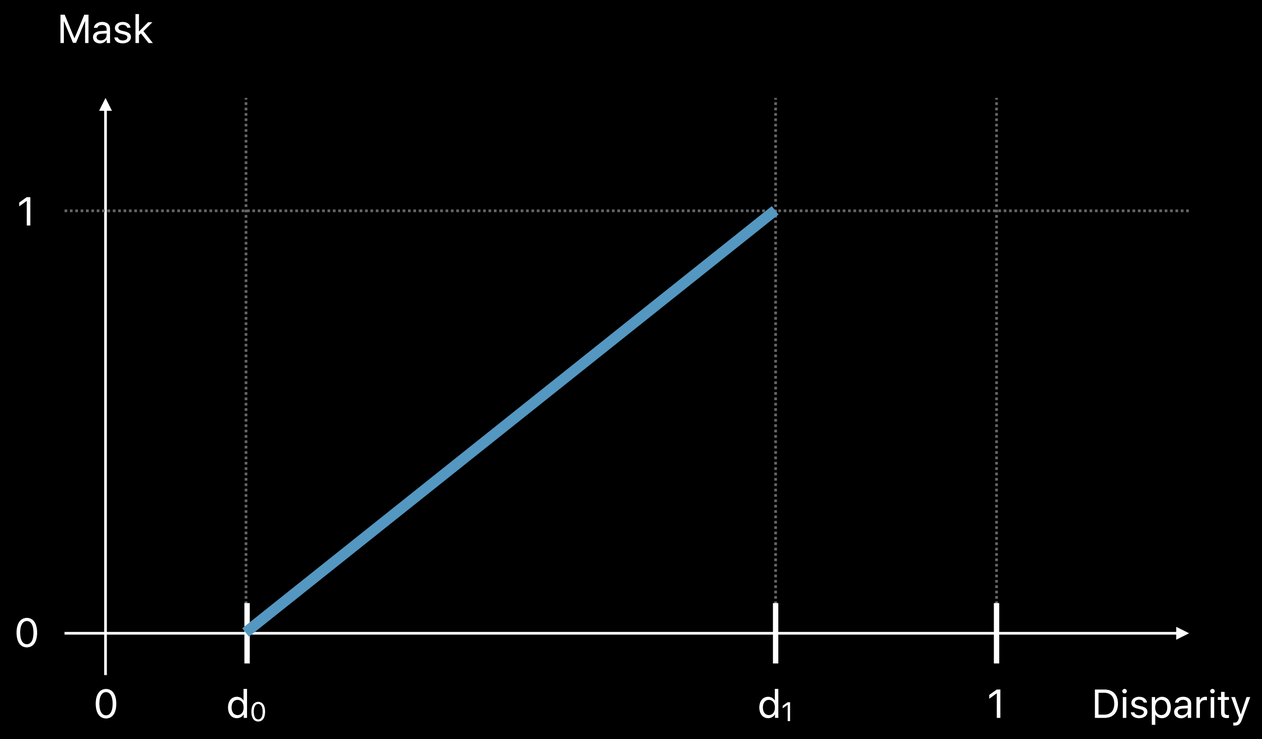
```
// Code to Sample the Disparity Map with a Tap

// Apply the filter with the sampleRect from the user's tap. Don't forget to clamp!
let minMaxImage = normalizedDisparityImage.clampingToExtent().applyingFilter(
    "CIAreaMinMaxRed", withInputParameters: [kCIInputExtentKey : CIVector(cgRect:sampleRect)])

// A four-byte buffer to store a single pixel value
var pixel = [UInt8](repeating: 0, count: 4)

// Render the image to a 1x1 rect. Be sure to use a nil color space.
context.render(minMaxImage, toBitmap: &pixel, rowBytes: 4,
               bounds: CGRect(x:0, y:0, width:1, height:1), format: kCIFormatRGBA8,
               colorSpace: nil)

// The max is stored in the green channel. Min is in the red.
let disparity = Float(pixel[1]) / 255.0
```

Using Disparity as a Mask

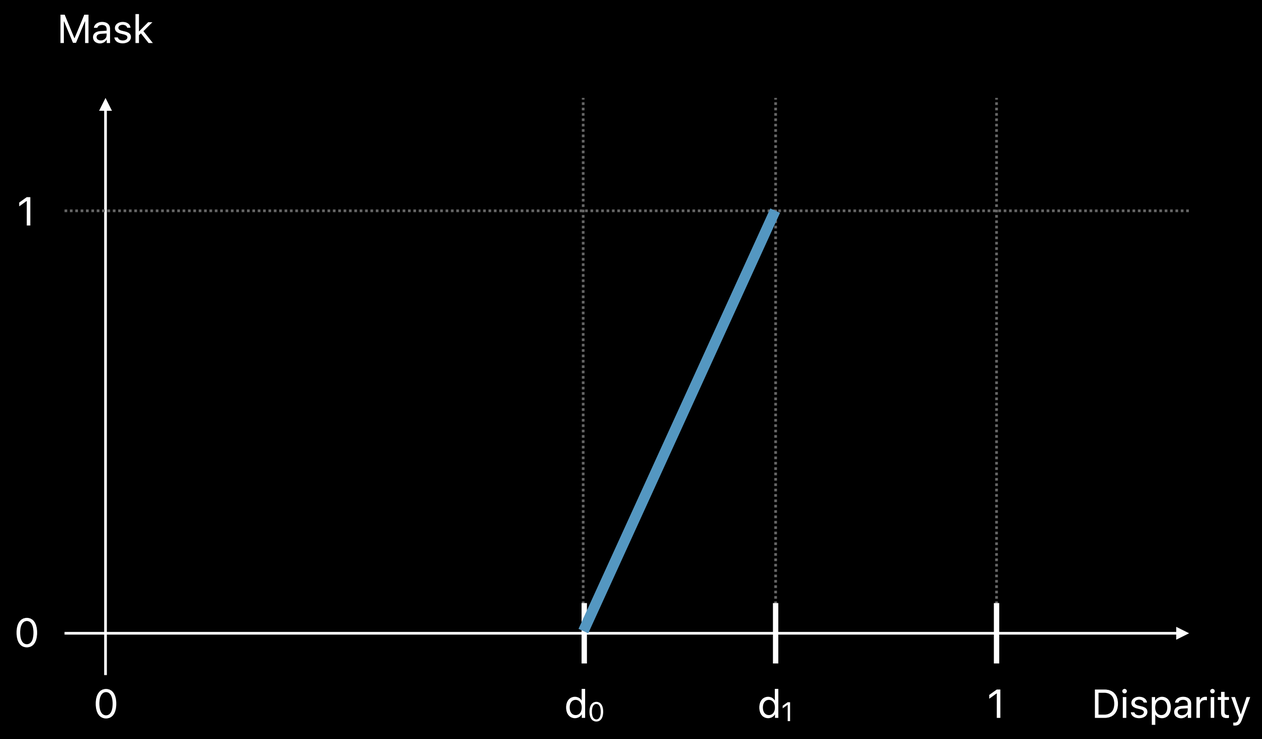




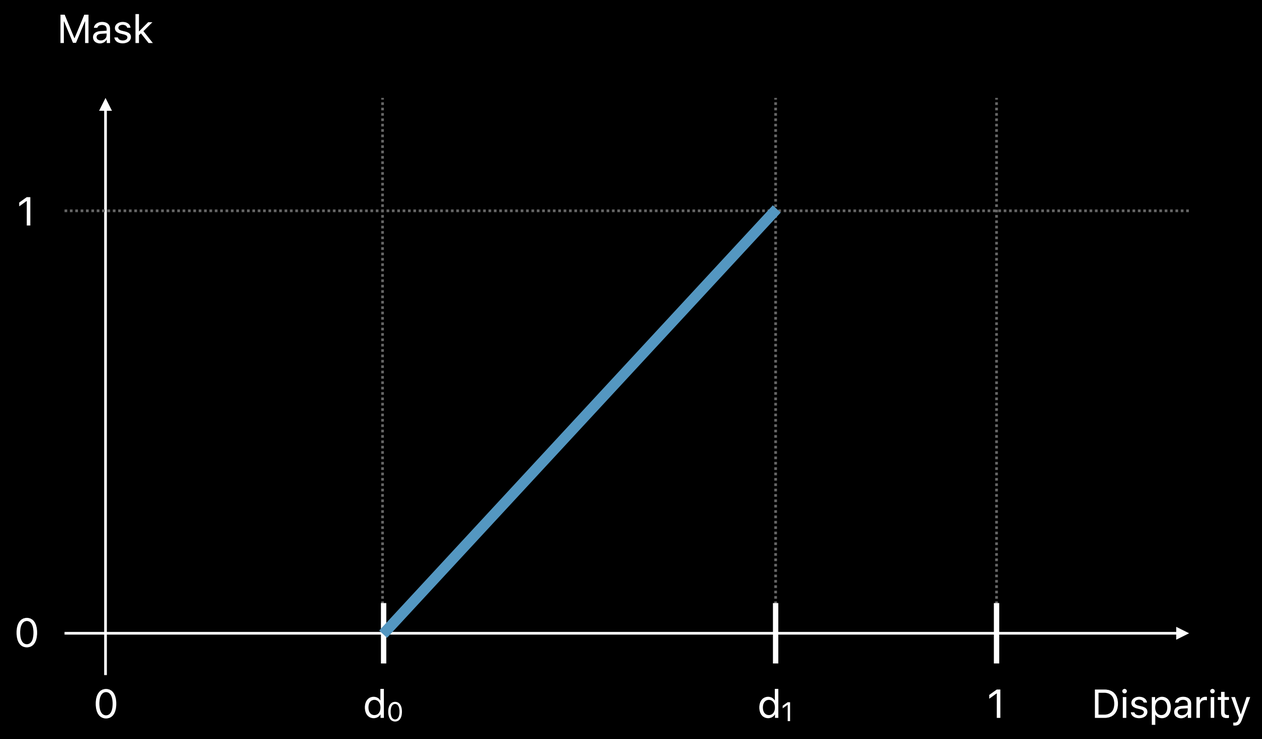
Using Disparity as a Mask



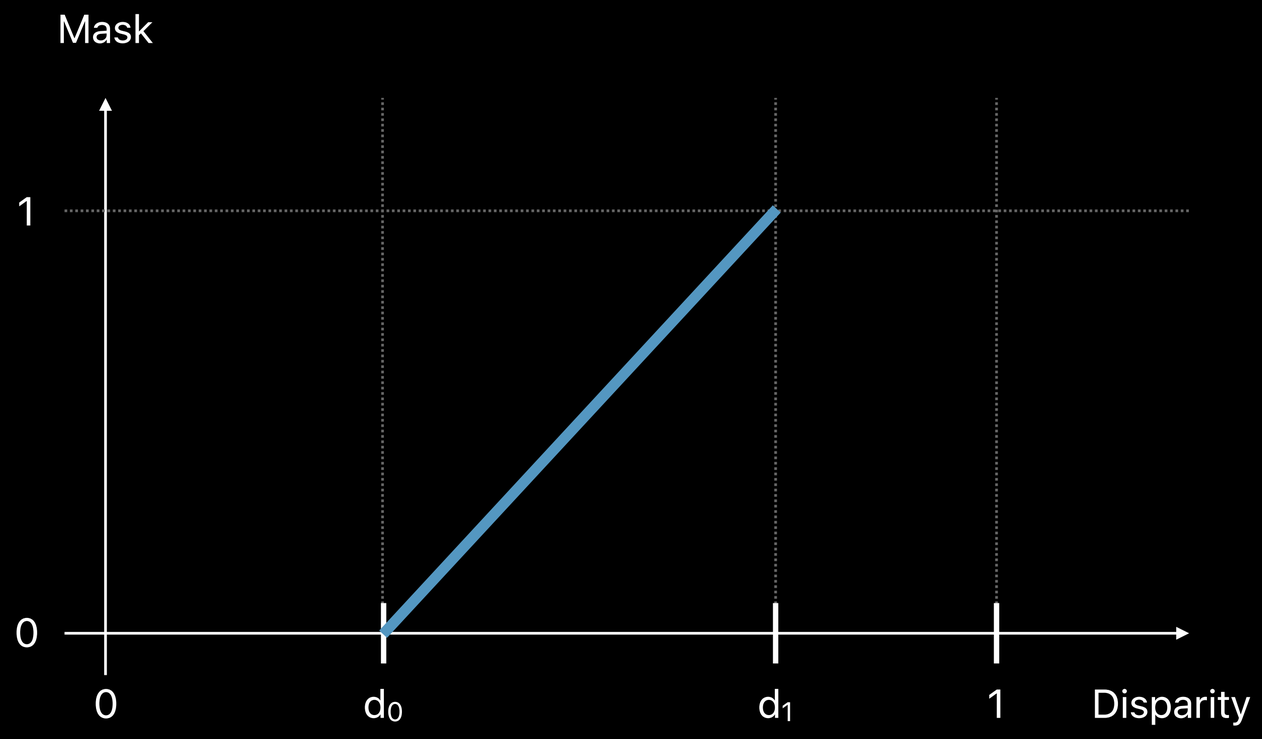
Using Disparity as a Mask



Using Disparity as a Mask

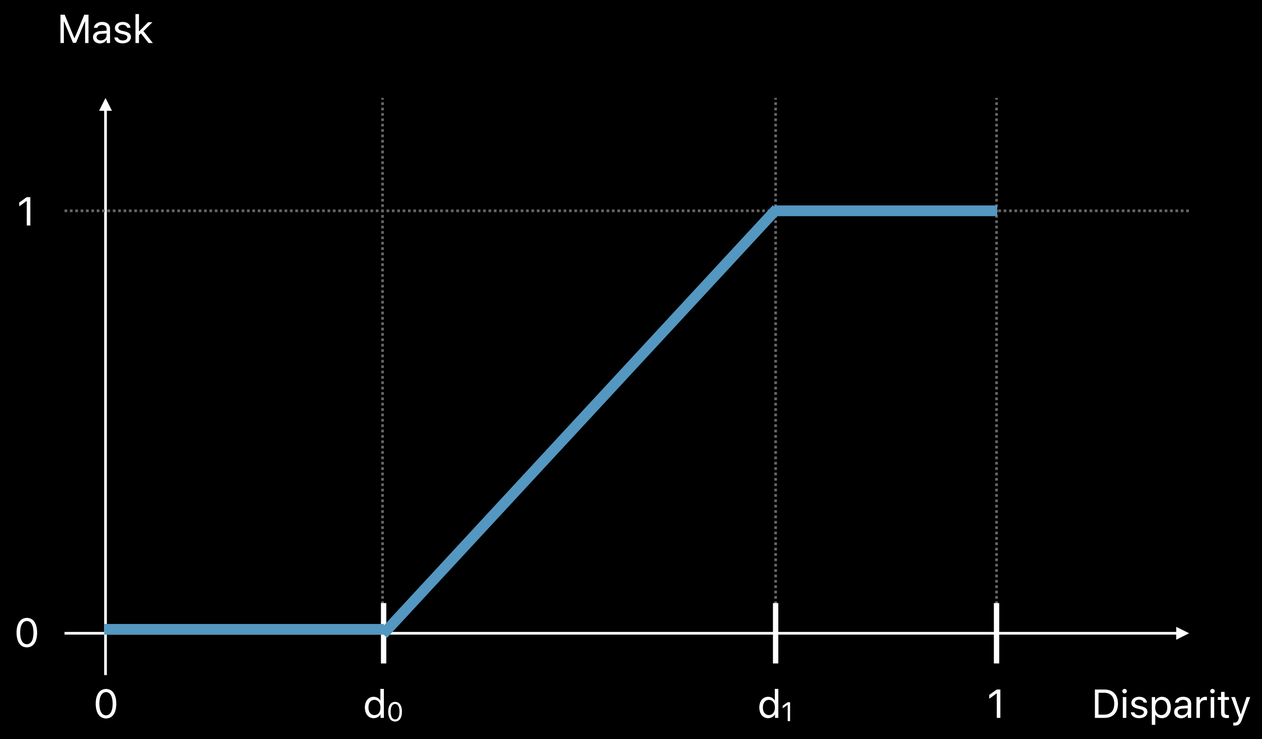


Using Disparity as a Mask



CIColorMatrix

Using Disparity as a Mask



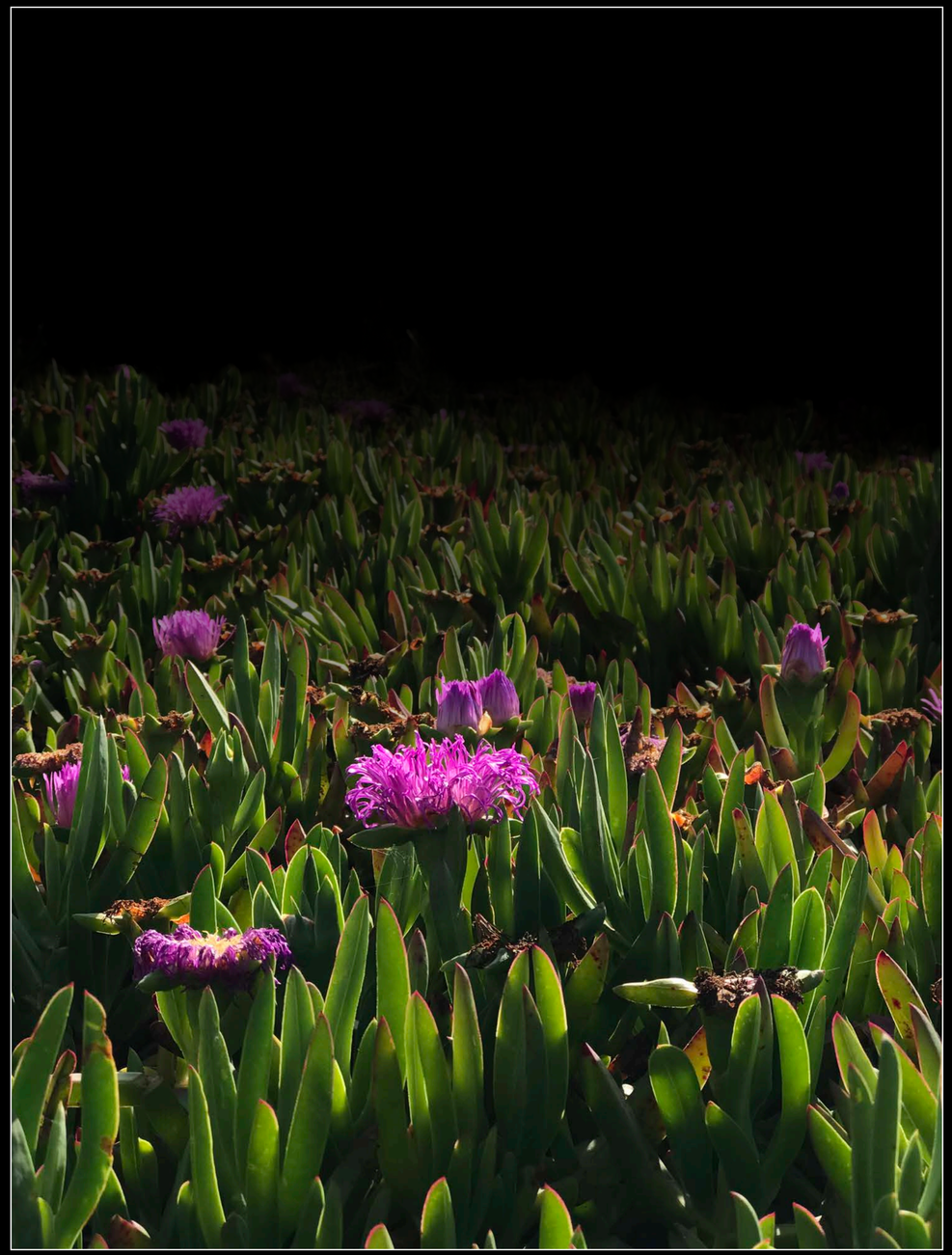
CIColorMatrix

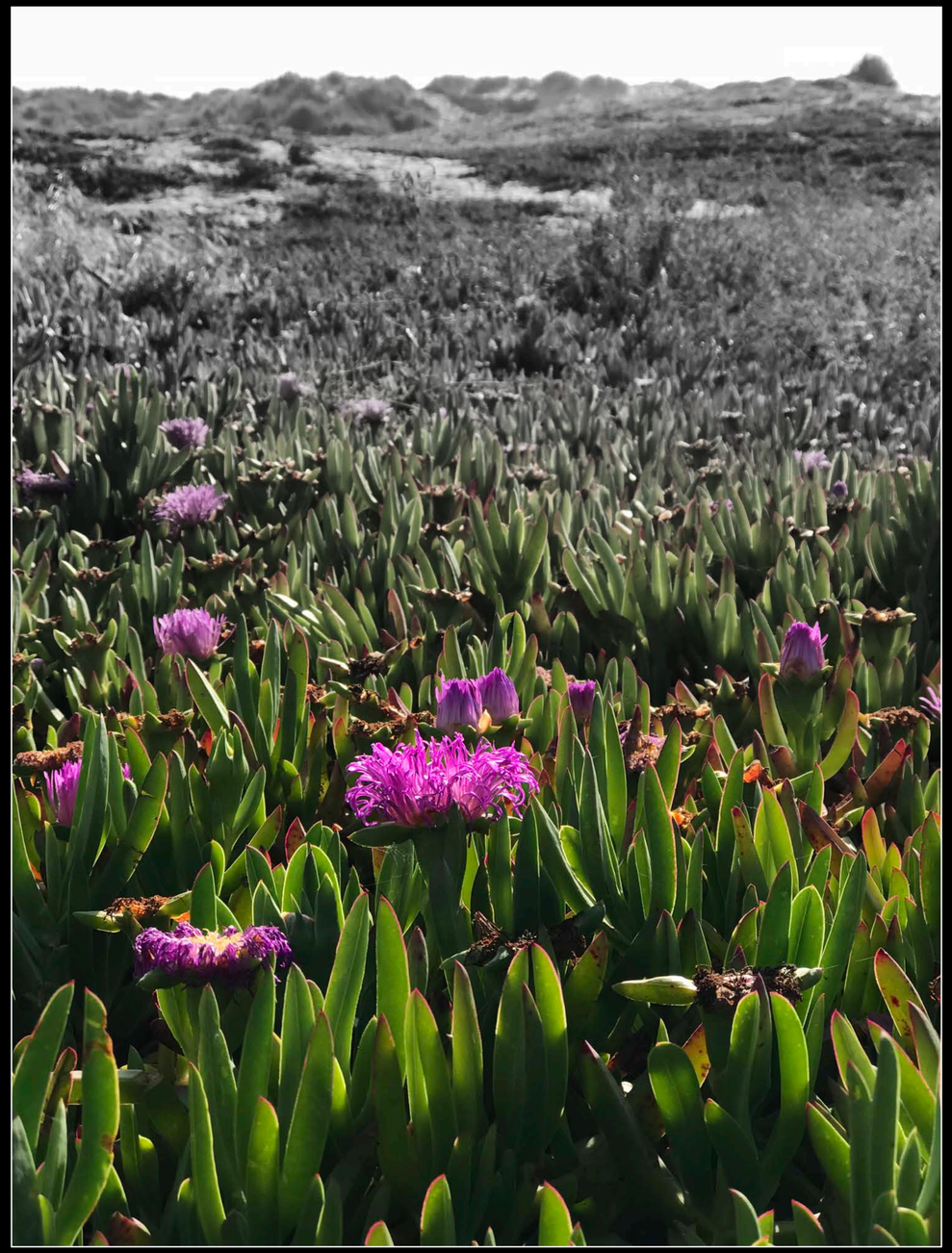
CIColorClamp

```
// Building a blend mask using CoreImage filters

// Scale and offset disparity values according to the slider arguments.
var mask = normalizedDisparityImage.applyingFilter("CIColorMatrix", withInputParameters: [
    "inputRVector"    : CIVector(x: slope, y: 0,     z: 0,     w: 0),
    "inputGVector"    : CIVector(x: 0,     y: slope, z: 0,     w: 0),
    "inputBVector"    : CIVector(x: 0,     y: 0,     z: slope, w: 0),
    "inputBiasVector" : CIVector(x: bias,  y: bias,  z: bias,  w: 0)])

// Clamp the mask values to [0,1]
mask = mask.applyingFilter("CIColorClamp", withInputParameters: nil)
```

```
// Building a blend mask using CoreImage filters

// Scale and offset disparity values according to the slider arguments.
var mask = normalizedDisparityImage.applyingFilter("CIColorMatrix", withInputParameters: [
    "inputRVector"    : CIVector(x: slope, y: 0,     z: 0,     w: 0),
    "inputGVector"    : CIVector(x: 0,     y: slope, z: 0,     w: 0),
    "inputBVector"    : CIVector(x: 0,     y: 0,     z: slope, w: 0),
    "inputBiasVector" : CIVector(x: bias,  y: bias,  z: bias,  w: 0)])

// Clamp the mask values to [0,1]
mask = mask.applyingFilter("CIColorClamp", withInputParameters: nil)
```

```
// Building a blend mask using CoreImage filters

// Scale and offset disparity values according to the slider arguments.
var mask = normalizedDisparityImage.applyingFilter("CIColorMatrix", withInputParameters: [
    "inputRVector"    : CIVector(x: slope, y: 0,     z: 0,     w: 0),
    "inputGVector"    : CIVector(x: 0,     y: slope, z: 0,     w: 0),
    "inputBVector"    : CIVector(x: 0,     y: 0,     z: slope, w: 0),
    "inputBiasVector" : CIVector(x: bias,  y: bias,  z: bias,  w: 0)])

// Clamp the mask values to [0,1]
mask = mask.applyingFilter("CIColorClamp", withInputParameters: nil)
```





```swift
// Code to blend the masked foreground image with the adjusted background image using the
// previously-computed mask

// Apply a filter to the main image to be used as the background
let backgroundImage = image.applyingFilter("CIPhotoEffectMono", withInputParameters: nil)

// Blend the image with the filtered background
let outputImage = image.applyingFilter("CIBlendWithMask", withInputParameters:
    [kCIInputBackgroundImageKey : backgroundImage, kCIInputMaskImageKey : mask])
```

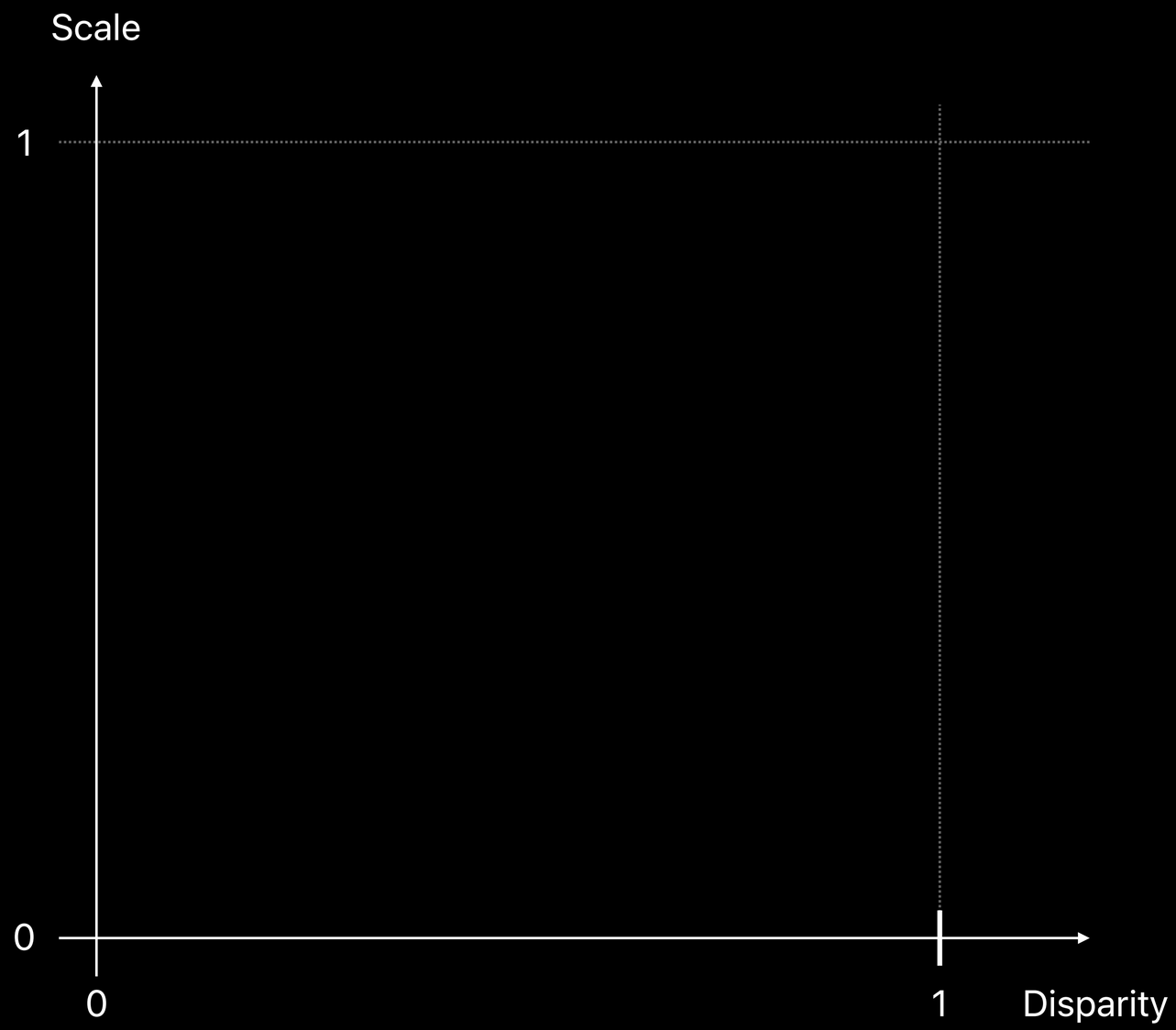
```
// Code to blend the masked foreground image with the adjusted background image using the
// previously-computed mask

// Apply a filter to the main image to be used as the background
let backgroundImage = image.applyingFilter("CIPhotoEffectMono", withInputParameters: nil)

// Blend the image with the filtered background
let outputImage = image.applyingFilter("CIBlendWithMask", withInputParameters:
    [kCIInputBackgroundImageKey : backgroundImage, kCIInputMaskImageKey : mask])
```

```
// Code to blend the masked foreground image with the adjusted background image using the
// previously-computed mask

// Apply a filter to the main image to be used as the background
let backgroundImage = image.applyingFilter("CIPhotoEffectMono", withInputParameters: nil)

// Blend the image with the filtered background
let outputImage = image.applyingFilter("CIBlendWithMask", withInputParameters:
    [kCIInputBackgroundImageKey : backgroundImage, kCIInputMaskImageKey : mask])
```

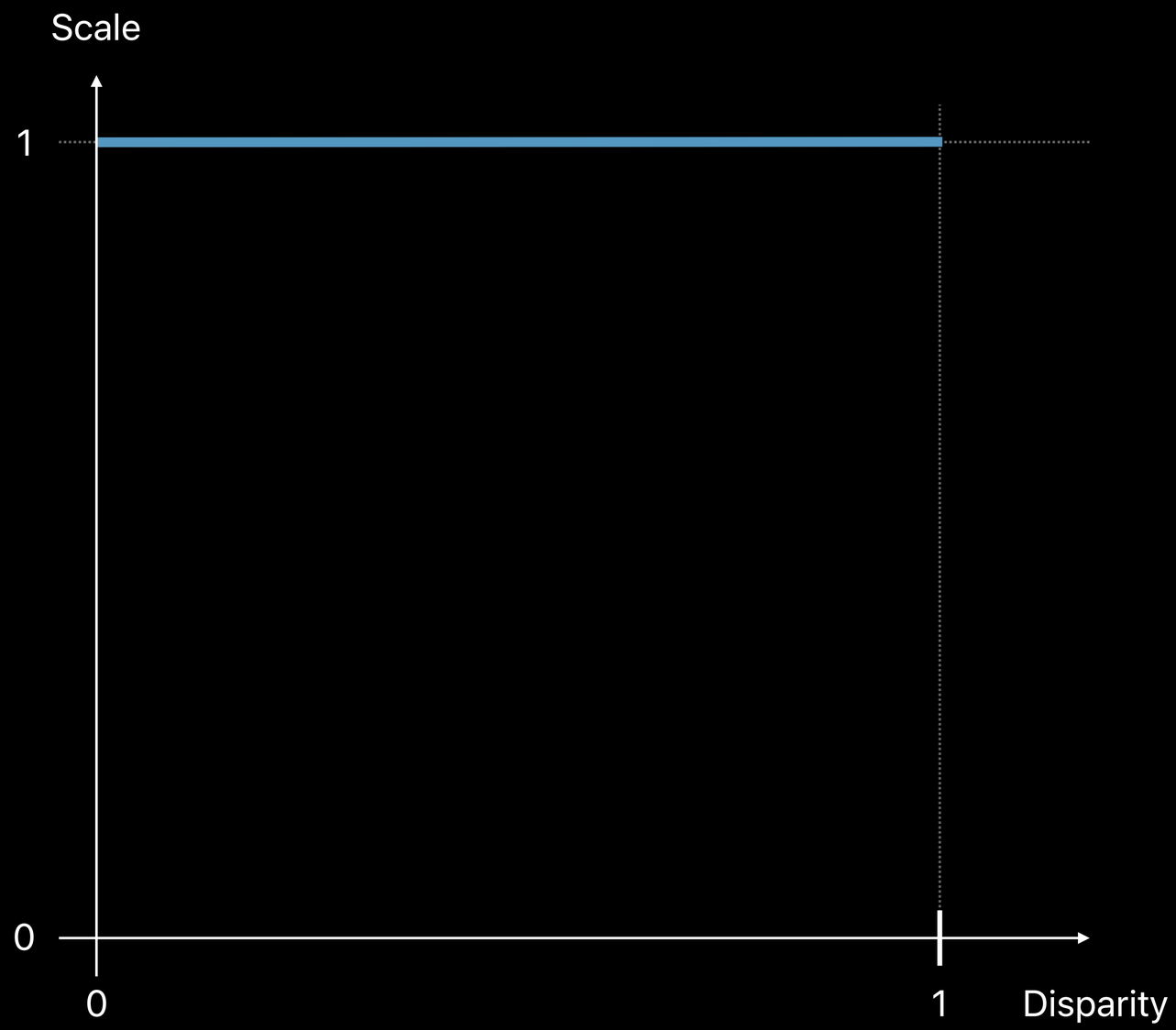
Custom Depth Effect

Into Darkness

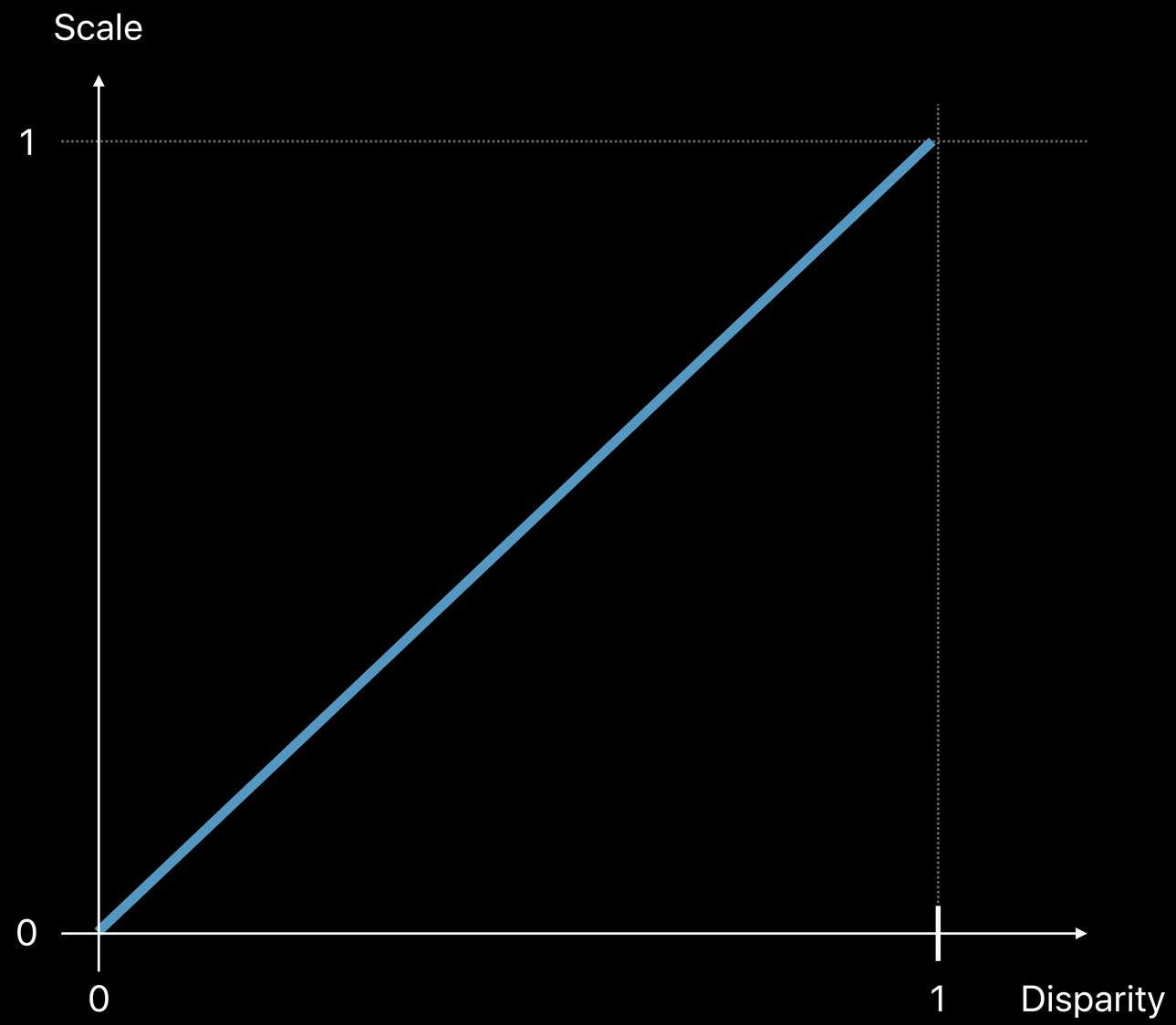
Raising Disparity to a Power



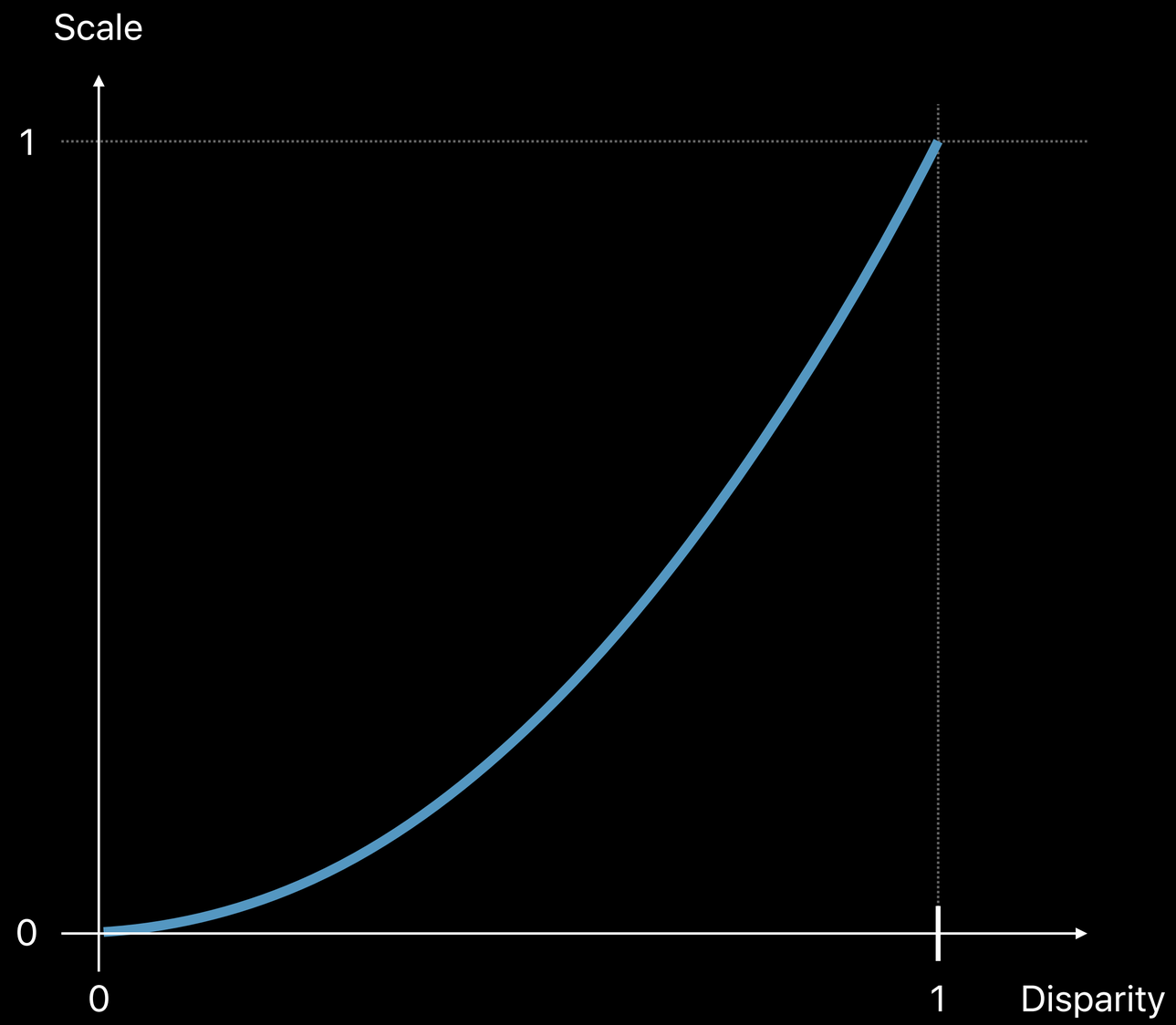
Raising Disparity to a Power



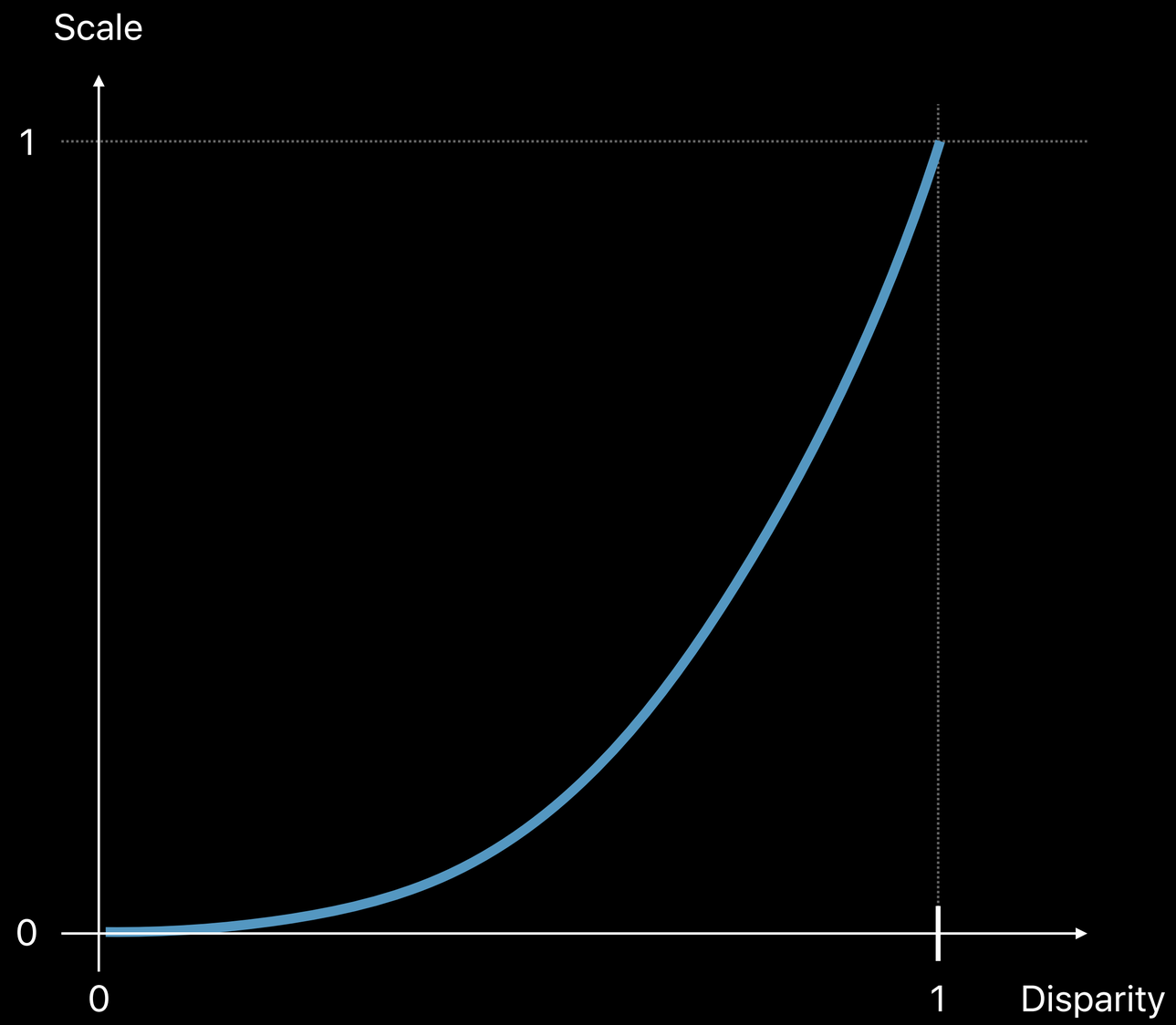
Raising Disparity to a Power



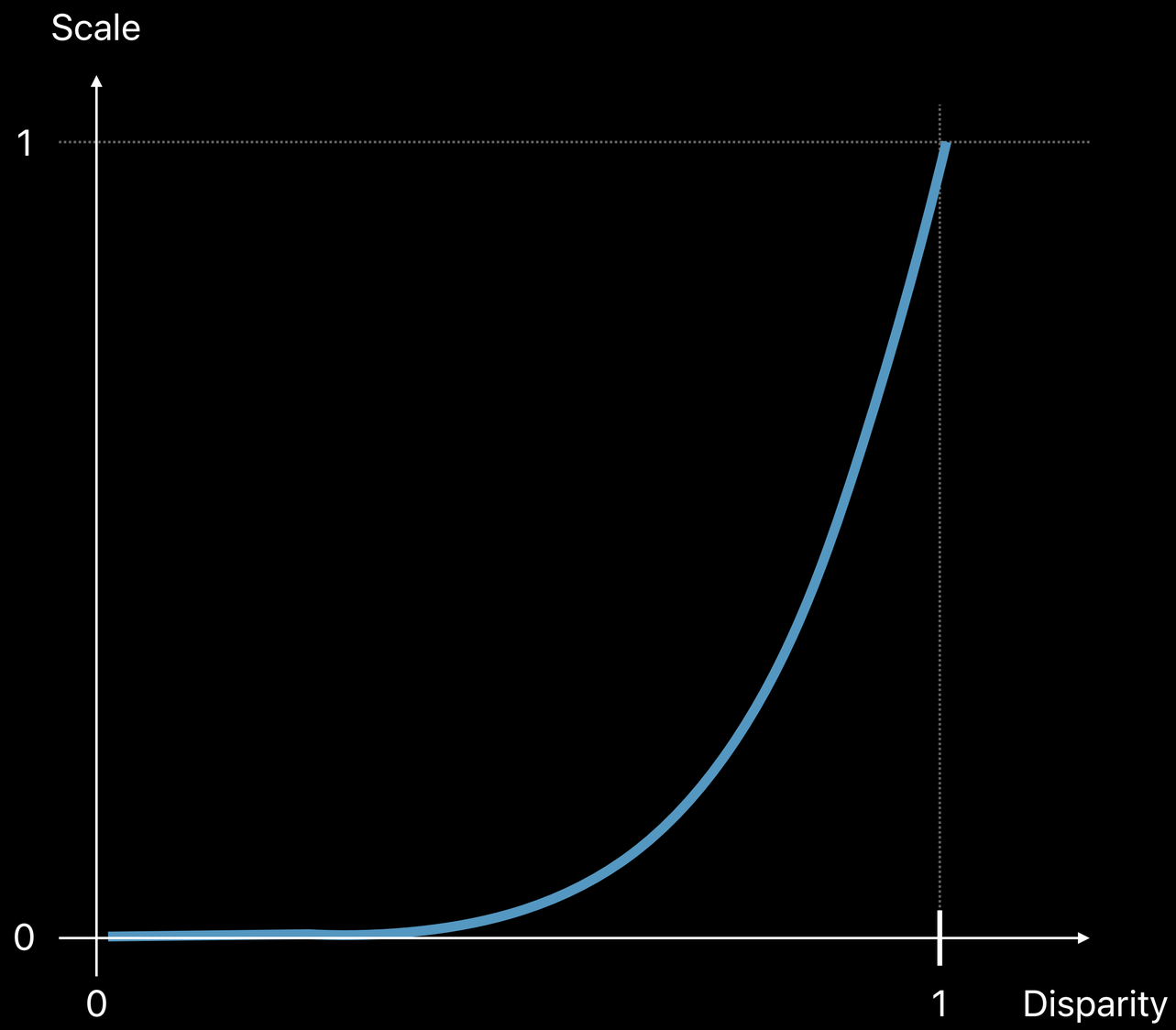
Raising Disparity to a Power



Raising Disparity to a Power



Raising Disparity to a Power



```
// CIColorKernel to dim the image background with disparity

kernel vec4 into_darkness(__sample imageColor, __sample normalizedDisparity, float power)
{
    // Adjust the fall-off intensity with the power slider
    float scaleFactor = pow(normalizedDisparity.r, power);

    // Scale the original color by the computed intensity
    vec4 result = vec4(imageColor.rgb * scaleFactor, imageColor.a);
    return result;
}


// How to apply the color kernel to your image in Swift
let outputImage = kernel.apply(withExtent: image.extent,
                               arguments: [image, normalizedDisparity, power])
```

```
// CIColorKernel to dim the image background with disparity

kernel vec4 into_darkness(__sample imageColor, __sample normalizedDisparity, float power)
{
    // Adjust the fall-off intensity with the power slider
    float scaleFactor = pow(normalizedDisparity.r, power);

    // Scale the original color by the computed intensity
    vec4 result = vec4(imageColor.rgb * scaleFactor, imageColor.a);
    return result;
}


// How to apply the color kernel to your image in Swift
let outputImage = kernel.apply(withExtent: image.extent,
                               arguments: [image, normalizedDisparity, power])
```

```
// CIColorKernel to dim the image background with disparity

kernel vec4 into_darkness(__sample imageColor, __sample normalizedDisparity, float power)
{
    // Adjust the fall-off intensity with the power slider
    float scaleFactor = pow(normalizedDisparity.r, power);

    // Scale the original color by the computed intensity
    vec4 result = vec4(imageColor.rgb * scaleFactor, imageColor.a);
    return result;
}



// How to apply the color kernel to your image in Swift
let outputImage = kernel.apply(withExtent: image.extent,
                               arguments: [image, normalizedDisparity, power])
```

```
// CIColorKernel to dim the image background with disparity

kernel vec4 into_darkness(__sample imageColor, __sample normalizedDisparity, float power)
{
    // Adjust the fall-off intensity with the power slider
    float scaleFactor = pow(normalizedDisparity.r, power);

    // Scale the original color by the computed intensity
    vec4 result = vec4(imageColor.rgb * scaleFactor, imageColor.a);
    return result;
}



// How to apply the color kernel to your image in Swift
let outputImage = kernel.apply(withExtent: image.extent,
                               arguments: [image, normalizedDisparity, power])
```

```
// CIColorKernel to dim the image background with disparity

kernel vec4 into_darkness(__sample imageColor, __sample normalizedDisparity, float power)
{
    // Adjust the fall-off intensity with the power slider
    float scaleFactor = pow(normalizedDisparity.r, power);

    // Scale the original color by the computed intensity
    vec4 result = vec4(imageColor.rgb * scaleFactor, imageColor.a);
    return result;
}


// How to apply the color kernel to your image in Swift
let outputImage = kernel.apply(withExtent: image.extent,
                               arguments: [image, normalizedDisparity, power])
```

Using CIDepthBlurEffect

Alexandre Naaman, Technical Lead, Core Image

CIDepthBlurEffect

NEW

Input Image

+

Input Disparity Image

=

Result Image

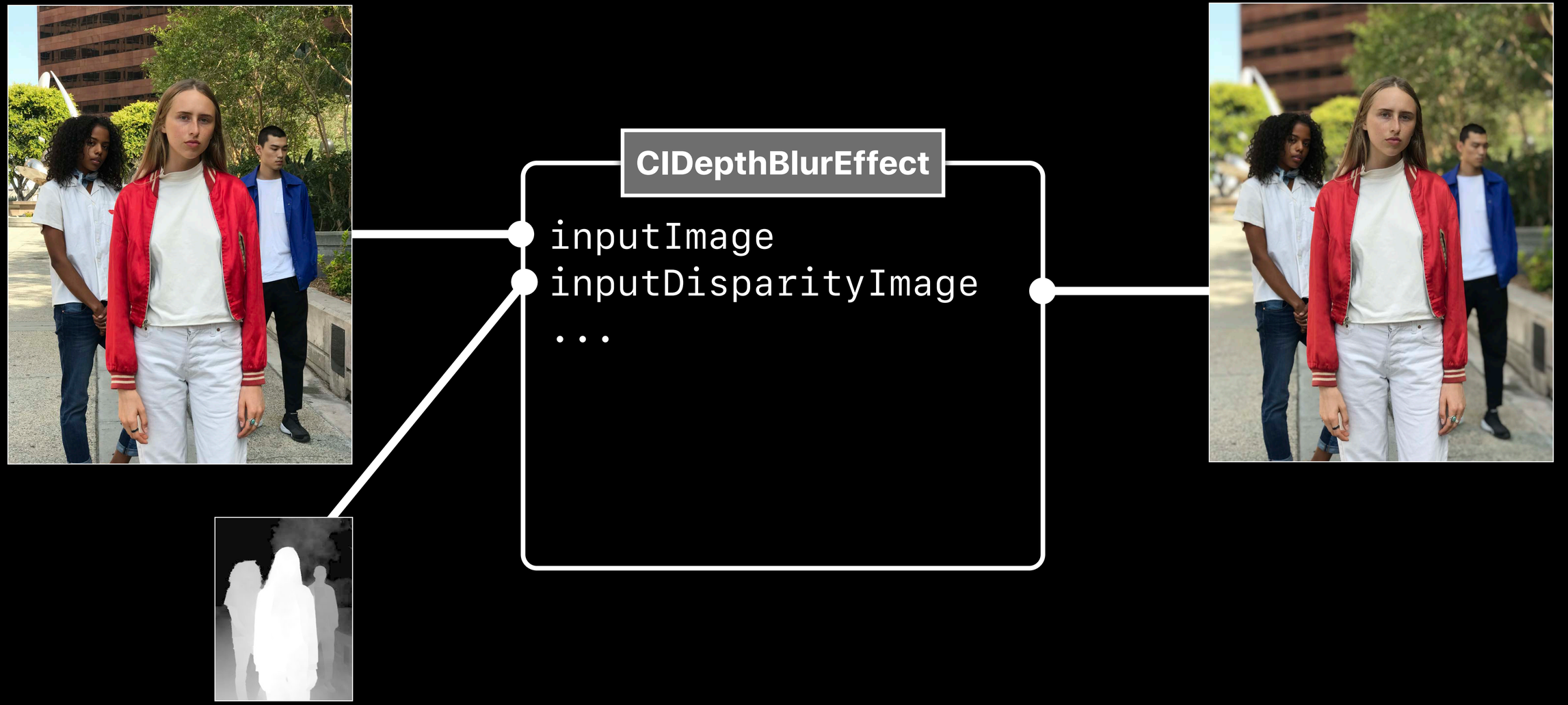
CIDepthBlurEffect: Setting Inputs

NEW

CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
```

CIDepthBlurEffect: Setting Focus Rect and Aperture



CIDepthBlurEffect: Setting Focus Rect and Aperture

CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
inputFocusRect = tap
```

CIDepthBlurEffect: Setting Focus Rect and Aperture

CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
inputFocusRect = tap
inputAperture = pinch
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage

// For each new user action

filter.setValue(aperture, forKey: "inputAperture")
filter.setValue(normalizedFocusRect, forKey: "inputFocusRect")

resultImage = filter.outputImage
```

```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage

// For each new user action

filter.setValue(aperture, forKey: "inputAperture")
filter.setValue(normalizedFocusRect, forKey: "inputFocusRect")

resultImage = filter.outputImage
```

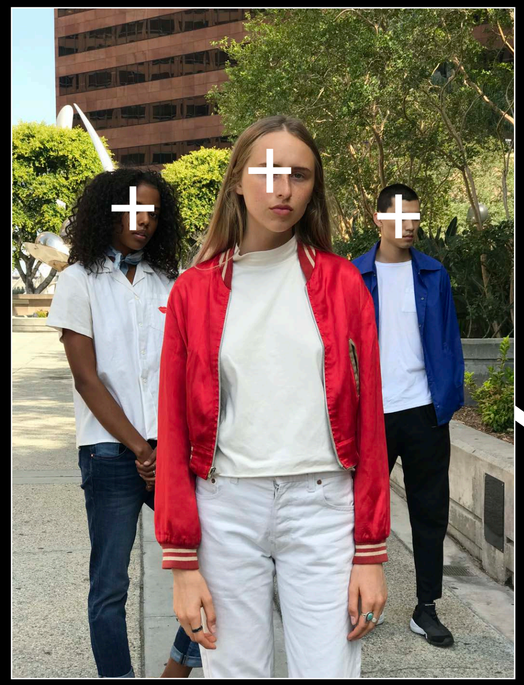
```
let mainImage = CIImage(contentsOf: url)
let disparityImage = CIImage(contentsOf: url, options: [kCIImageAuxiliaryDisparity : true])

let filter = CIFilter(name:"CIDepthBlurEffect",
                      withInputParameters:[kCIInputImageKey : mainImage,
                                           kCIInputDisparityImageKey : disparityImage ])

var resultImage = filter.outputImage

// For each new user action

filter.setValue(aperture, forKey: "inputAperture")
filter.setValue(normalizedFocusRect, forKey: "inputFocusRect")

resultImage = filter.outputImage
```

Use Vision to Specify Points of Faces

CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
inputLeftEyePositions
inputRightEyePositions
inputNosePositions
inputChinPositions
...
```

Use Vision to Specify Points of Faces



CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
inputLeftEyePositions
inputRightEyePositions
inputNosePositions
inputChinPositions
...
```

Use Vision to Specify Points of Faces

CIDepthBlurEffect

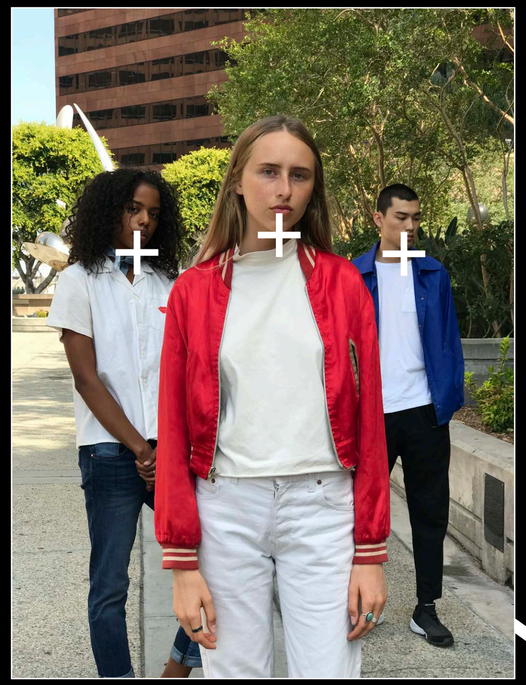
```
inputImage
inputDisparityImage
...
inputLeftEyePositions
inputRightEyePositions
inputNosePositions
inputChinPositions
...
```

Use Vision to Specify Points of Faces

CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
inputLeftEyePositions
inputRightEyePositions
inputNosePositions
inputChinPositions
...
```

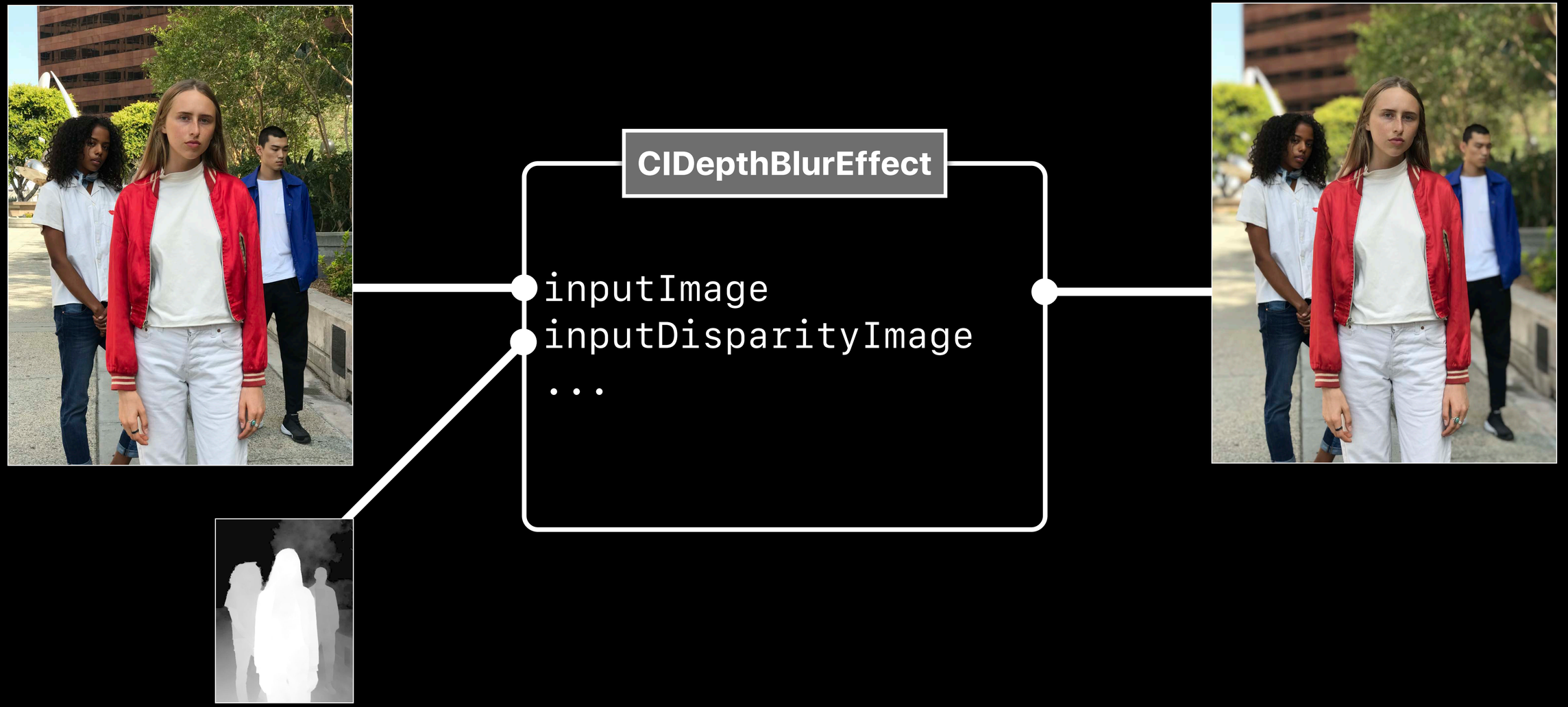
Use Vision to Specify Points of Faces



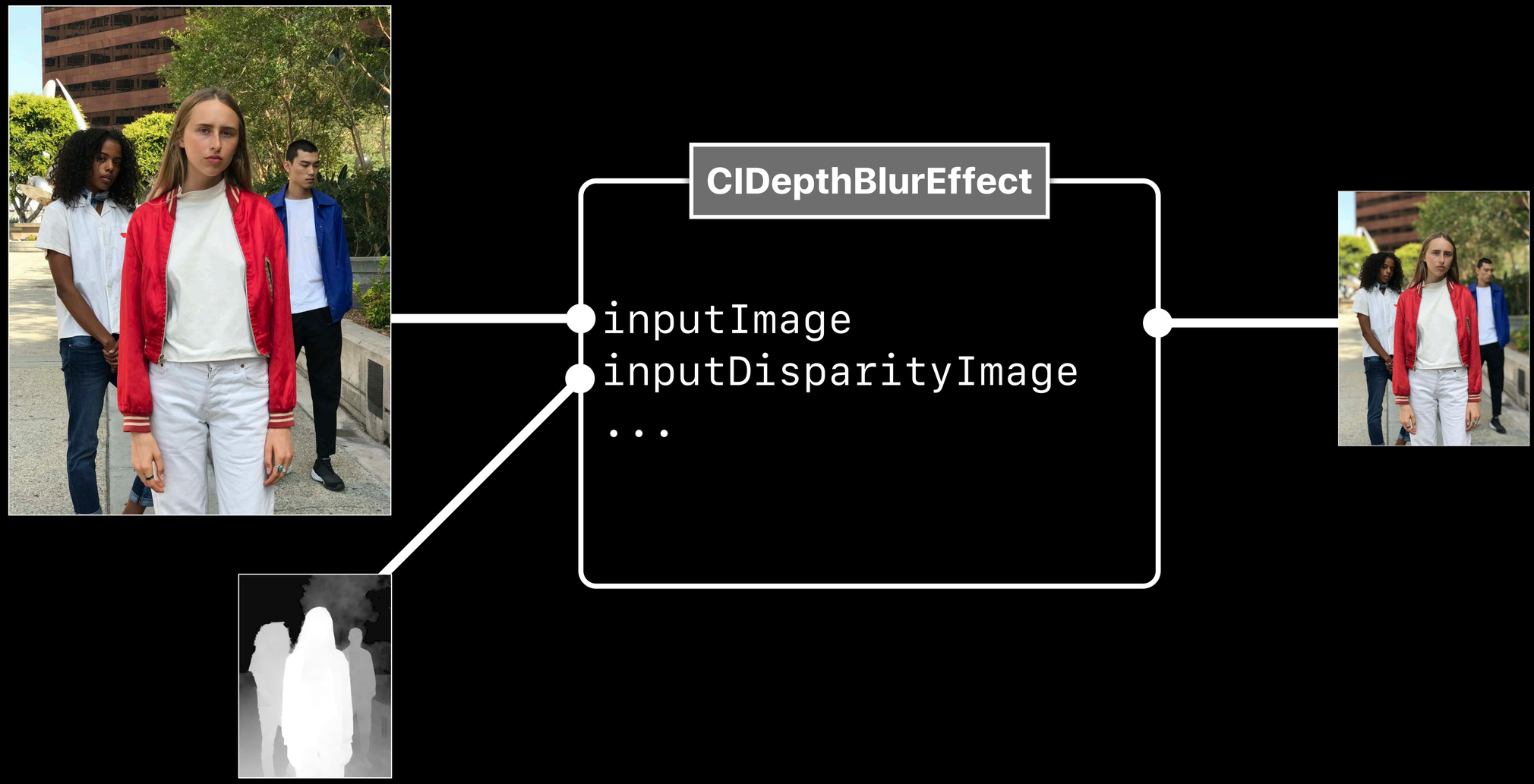
CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
inputLeftEyePositions
inputRightEyePositions
inputNosePositions
inputChinPositions
...
```

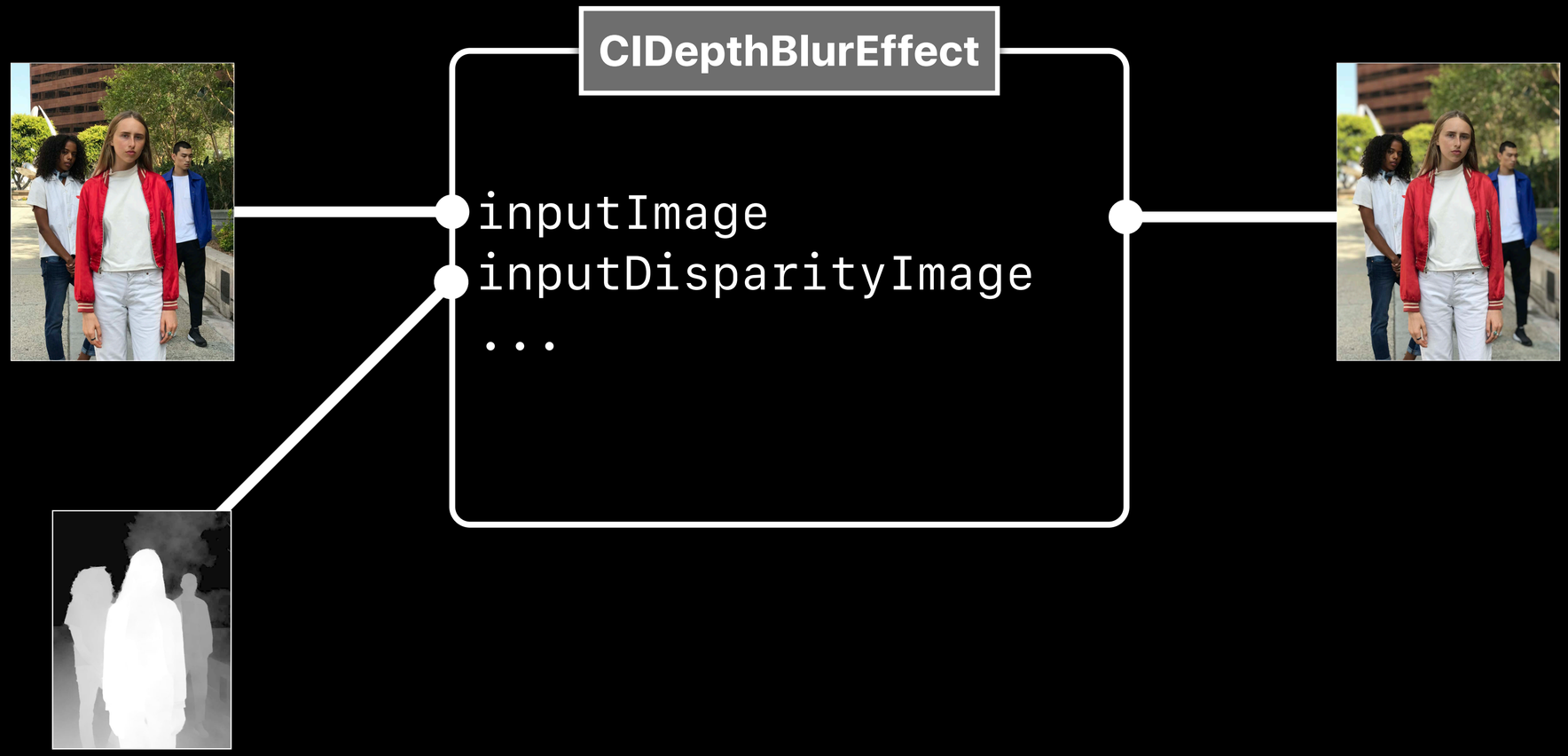
Scaling the Output



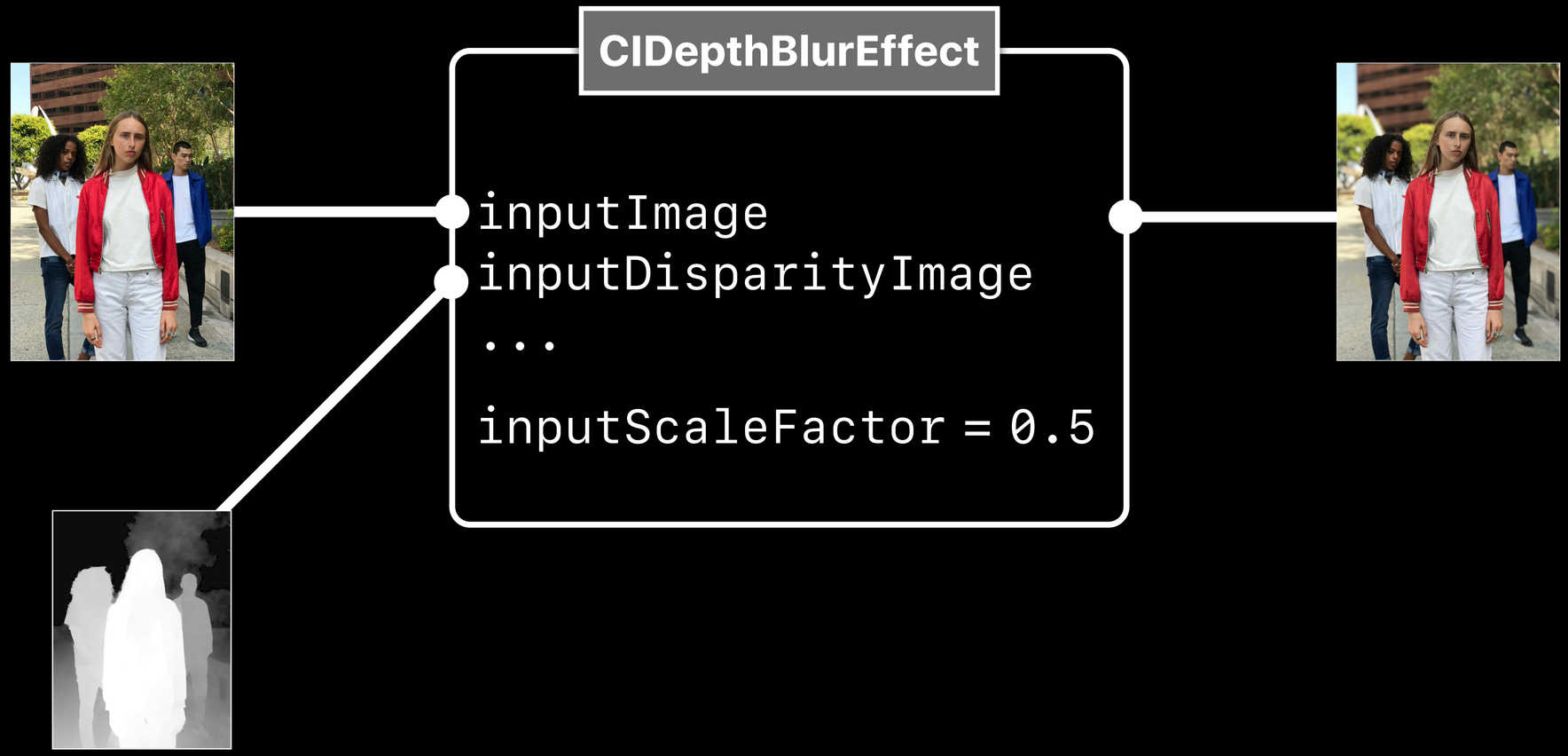
Scaling the Output



Scaling the Output



Scaling the Output



Additional Reminders

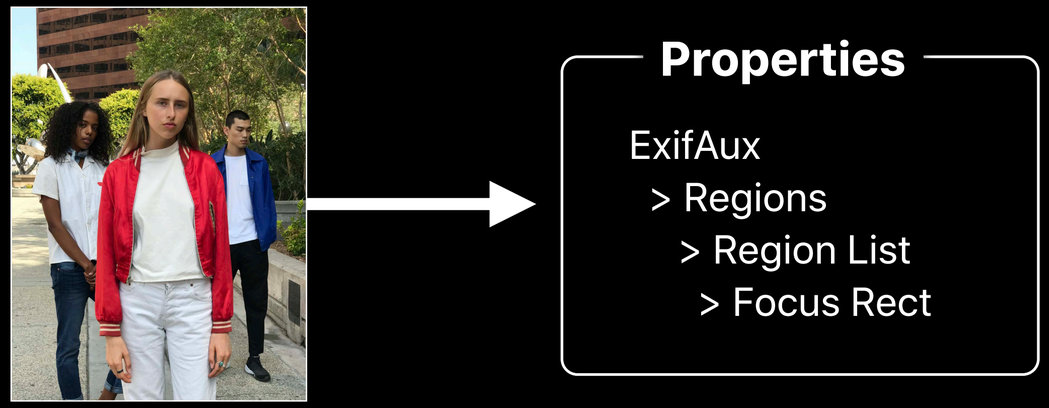
Create the CIContext with half-float intermediates

```
let ctx = CIContext(options: [kCIContextWorkingFormat : kCIFormatRGBAh])
```

Additional Reminders

Create the CIContext with half-float intermediates

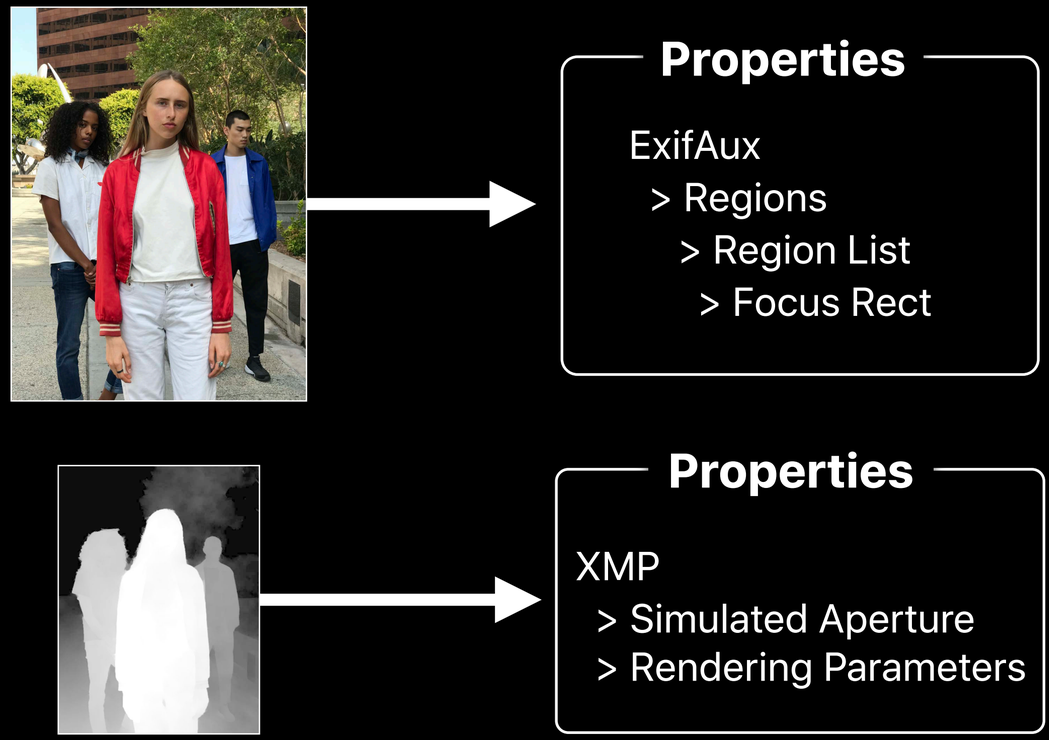
```
let ctx = CIContext(options: [kCIContextWorkingFormat : kCIFormatRGBAh])
```



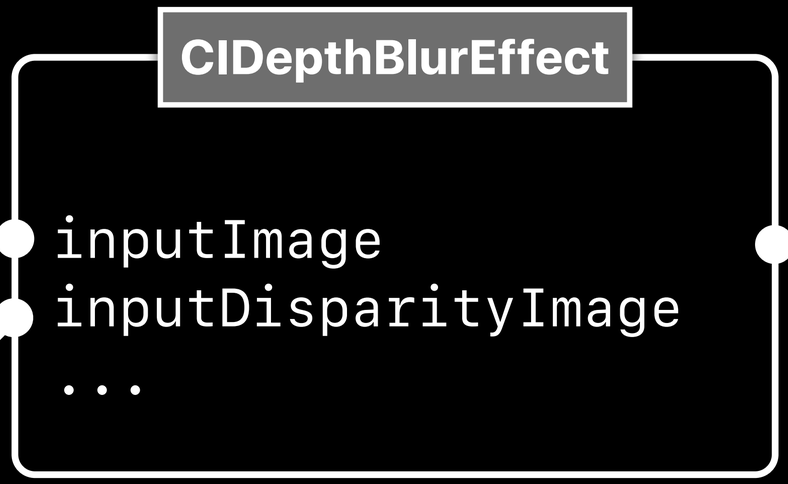
Additional Reminders

Create the CIContext with half-float intermediates

```
let ctx = CIContext(options: [kCIContextWorkingFormat : kCIFormatRGBAh])
```



CIDepthBlurEffect: Custom Disparity Upsample



CIDepthBlurEffect: Custom Disparity Upsample

CIDepthBlurEffect

```
inputImage
inputDisparityImage
...
```

Filtering with Depth Data

Simple Background Effects

Custom Depth Effect

Depth Blur Effect

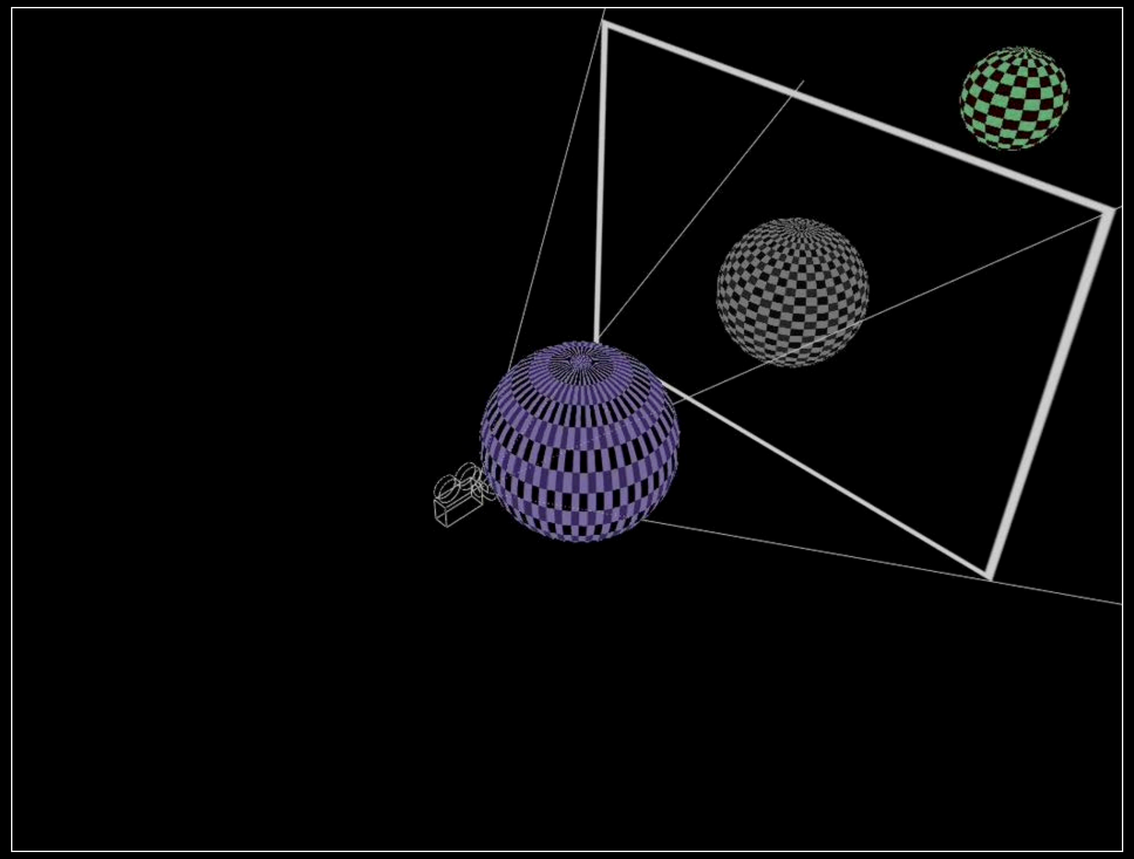
3D Effect

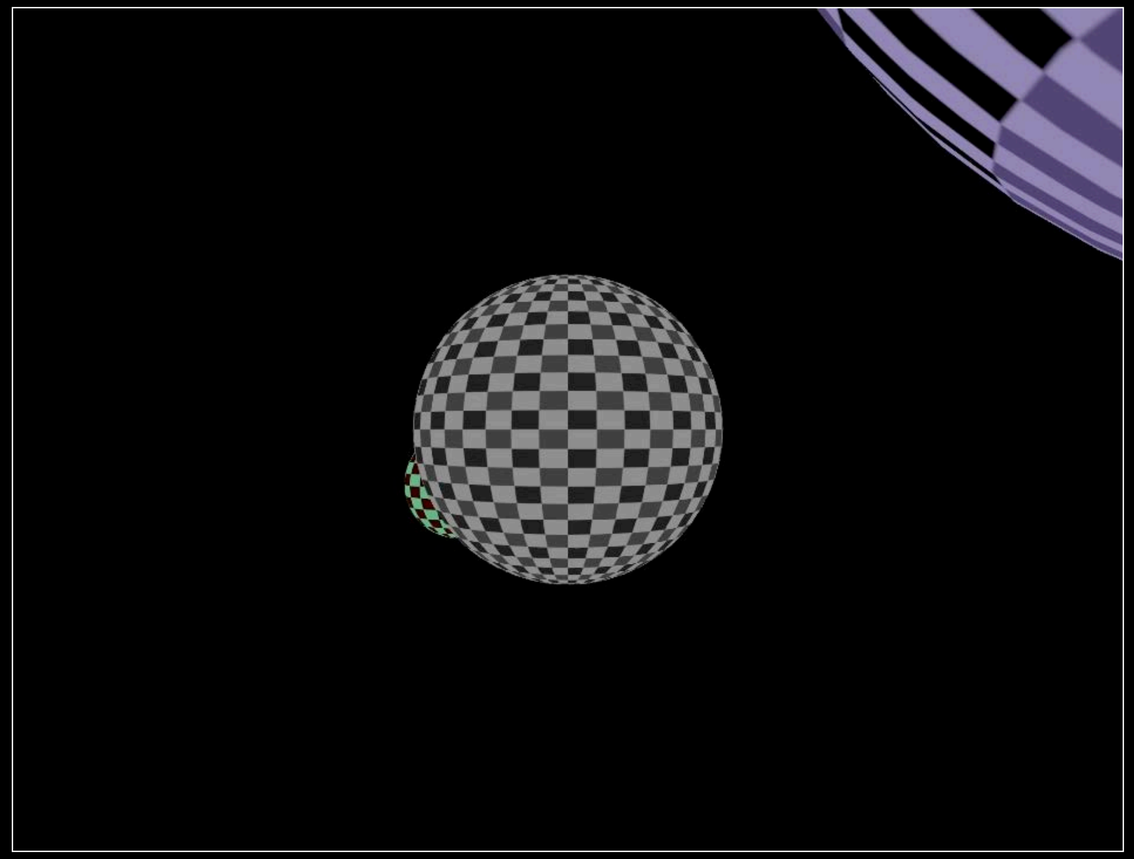
3D Effect

Dolly Zoom

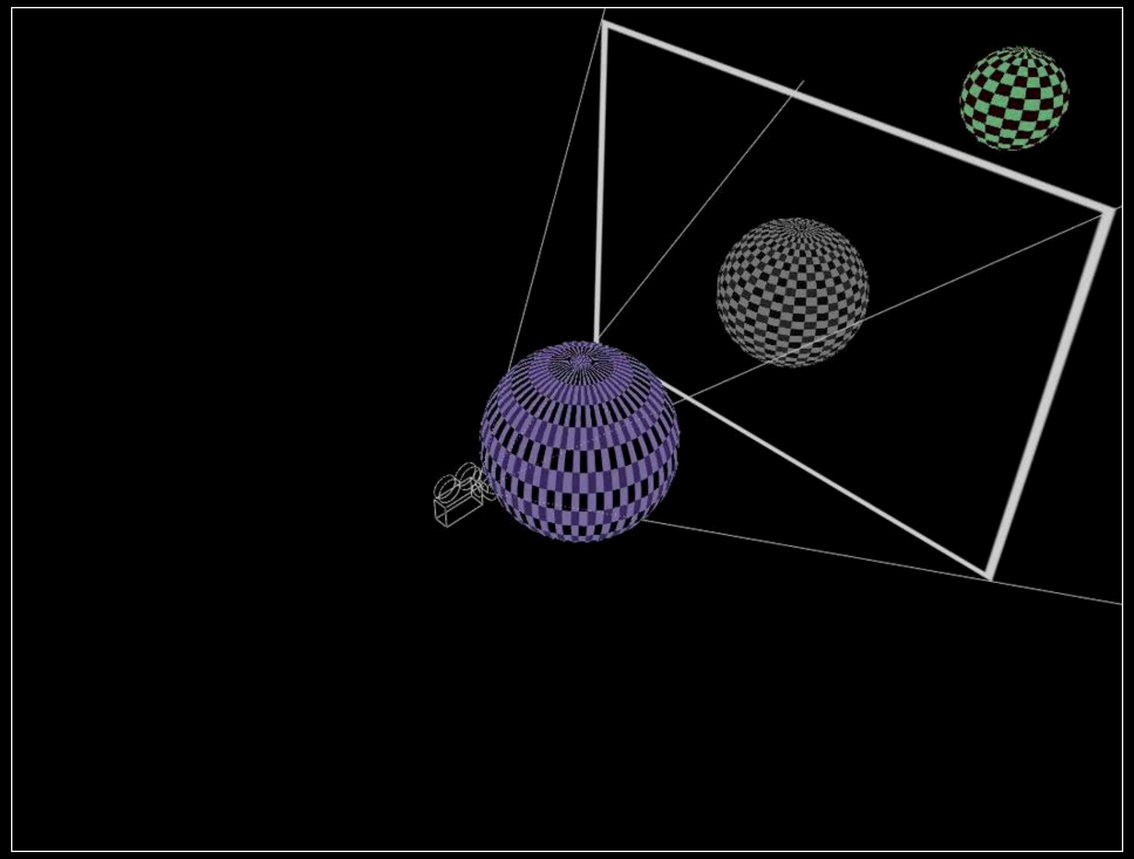
Stephen Ward, Software Engineer Extraordinaire

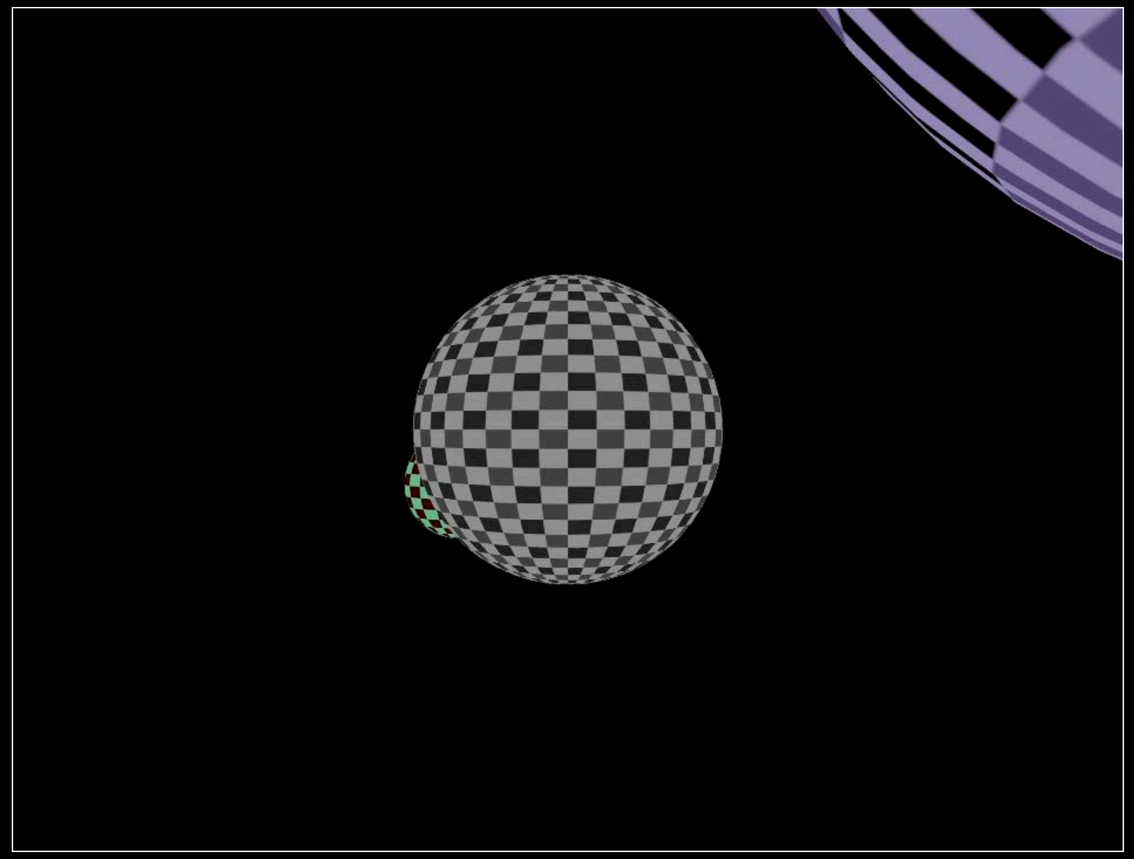
Dolly Zoom





Dolly Zoom





Dolly Zoom Demo

Dolly Zoom

Dolly Zoom

Metal

Dolly Zoom

Metal

Mesh

Dolly Zoom

Metal

Mesh

Vertex shader

Dolly Zoom

Metal

Mesh

Vertex shader

Fragment shader

Dolly Zoom

Metal

Mesh

Vertex shader

Fragment shader

CIImageProcessorKernel

```
// Dolly Zoom Effect Vertex Shader

vertex OutVertex vertexShader(vertices, constants, depthTexture, texSampler, ...) {
    // Sample the vertex's depth.
    float z = depthTexture.sample(texSampler, vertices[vid].texCoord).r;

    // Compute the scale and new vertex position to achieve the 3D effect
    float zt = constants.threshold; // [zmin..zmax]
    float dz = constants.strength; //zoom value [-zmin^2..zmin/2]
    float scale = (z/zt) * ((zt + dz)/(z + dz)); // Projection transformation
    float2 newPosition = vertices[vid].position.xy * scale;

    OutVertex vertex; // Output a transformed vertex
    vertex.position = float4(newPosition, z, 1.0);
    vertex.texCoord = vertices[vid].texCoord;
    return vertex;
}
```

```
// Dolly Zoom Effect Vertex Shader

vertex OutVertex vertexShader(vertices, constants, depthTexture, texSampler, ...) {
    // Sample the vertex's depth.
    float z = depthTexture.sample(texSampler, vertices[vid].texCoord).r;

    // Compute the scale and new vertex position to achieve the 3D effect
    float zt = constants.threshold; // [zmin..zmax]
    float dz = constants.strength; //zoom value [-zmin^2..zmin/2]
    float scale = (z/zt) * ((zt + dz)/(z + dz)); // Projection transformation
    float2 newPosition = vertices[vid].position.xy * scale;

    OutVertex vertex; // Output a transformed vertex
    vertex.position = float4(newPosition, z, 1.0);
    vertex.texCoord = vertices[vid].texCoord;
    return vertex;
}
```

```
// Dolly Zoom Effect Vertex Shader

vertex OutVertex vertexShader(vertices, constants, depthTexture, texSampler, ...) {
    // Sample the vertex's depth.
    float z = depthTexture.sample(texSampler, vertices[vid].texCoord).r;

    // Compute the scale and new vertex position to achieve the 3D effect
    float zt = constants.threshold; // [zmin..zmax]
    float dz = constants.strength; //zoom value [-zmin^2..zmin/2]
    float scale = (z/zt) * ((zt + dz)/(z + dz)); // Projection transformation
    float2 newPosition = vertices[vid].position.xy * scale;

    OutVertex vertex; // Output a transformed vertex
    vertex.position = float4(newPosition, z, 1.0);
    vertex.texCoord = vertices[vid].texCoord;
    return vertex;
}
```

```
// Dolly Zoom Effect Vertex Shader

vertex OutVertex vertexShader(vertices, constants, depthTexture, texSampler, ...) {
    // Sample the vertex's depth.
    float z = depthTexture.sample(texSampler, vertices[vid].texCoord).r;

    // Compute the scale and new vertex position to achieve the 3D effect
    float zt = constants.threshold; // [zmin..zmax]
    float dz = constants.strength; //zoom value [-zmin^2..zmin/2]
    float scale = (z/zt) * ((zt + dz)/(z + dz)); // Projection transformation
    float2 newPosition = vertices[vid].position.xy * scale;

    OutVertex vertex; // Output a transformed vertex
    vertex.position = float4(newPosition, z, 1.0);
    vertex.texCoord = vertices[vid].texCoord;
    return vertex;
}
```

```
// Dolly Zoom Effect Vertex Shader

vertex OutVertex vertexShader(vertices, constants, depthTexture, texSampler, ...) {
    // Sample the vertex's depth.
    float z = depthTexture.sample(texSampler, vertices[vid].texCoord).r;

    // Compute the scale and new vertex position to achieve the 3D effect
    float zt = constants.threshold; // [zmin..zmax]
    float dz = constants.strength; //zoom value [-zmin^2..zmin/2]
    float scale = (z/zt) * ((zt + dz)/(z + dz)); // Projection transformation
    float2 newPosition = vertices[vid].position.xy * scale;

    OutVertex vertex; // Output a transformed vertex
    vertex.position = float4(newPosition, z, 1.0);
    vertex.texCoord = vertices[vid].texCoord;
    return vertex;
}
```

```
// Dolly Zoom Effect Vertex Shader

vertex OutVertex vertexShader(vertices, constants, depthTexture, texSampler, ...) {
    // Sample the vertex's depth.
    float z = depthTexture.sample(texSampler, vertices[vid].texCoord).r;

    // Compute the scale and new vertex position to achieve the 3D effect
    float zt = constants.threshold; // [zmin..zmax]
    float dz = constants.strength; //zoom value [-zmin^2..zmin/2]
    float scale = (z/zt) * ((zt + dz)/(z + dz)); // Projection transformation
    float2 newPosition = vertices[vid].position.xy * scale;

    OutVertex vertex; // Output a transformed vertex
    vertex.position = float4(newPosition, z, 1.0);
    vertex.texCoord = vertices[vid].texCoord;
    return vertex;
}
```

```
// Dolly Zoom Effect Fragment Shader

fragment float4 fragmentShader(OutVertex vertex [[stage_in]],
                               texture2d<float, access::sample> imageTexture [[texture(0)]],
                               sampler texSampler [[sampler(0)]])
{
    return imageTexture.sample(texSampler, vertex.texCoord);
}
```

```
// Dolly Zoom Effect Fragment Shader

fragment float4 fragmentShader(OutVertex vertex [[stage_in]],
                               texture2d<float, access::sample> imageTexture [[texture(0)]],
                               sampler texSampler [[sampler(0)]])
{
    return imageTexture.sample(texSampler, vertex.texCoord);
}
```

What is depth?

Loading depth data

Filtering with depth data

Saving depth data

Saving Depth Data

Saving Depth Data

Preserve depth data

Saving Depth Data

Preserve depth data

Apply geometry

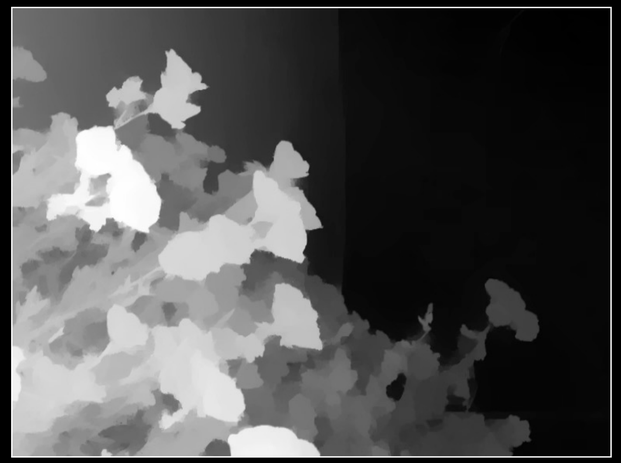
Transforming Depth Data

Transforming Depth Data

Orientation

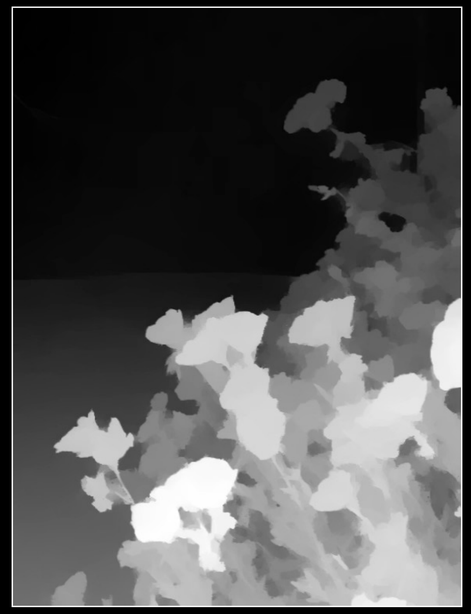
Transforming Depth Data

Orientation



Transforming Depth Data

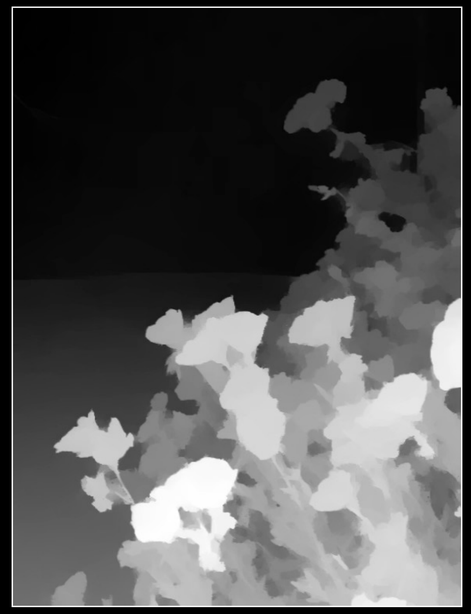
Orientation



Transforming Depth Data

Orientation

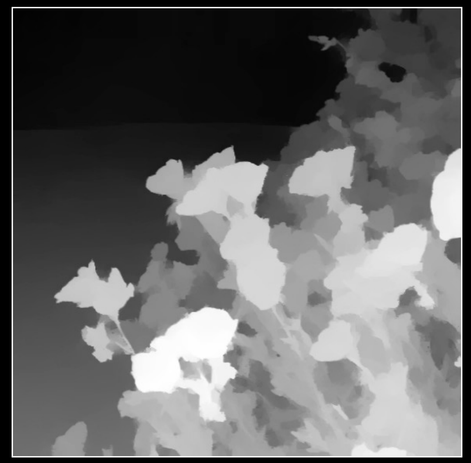
Crop



Transforming Depth Data

Orientation

Crop

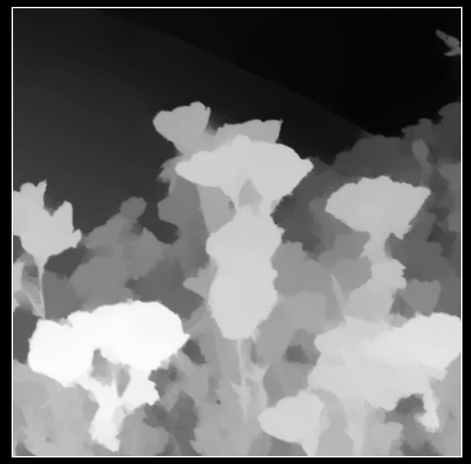


Transforming Depth Data

Orientation

Crop

Transform



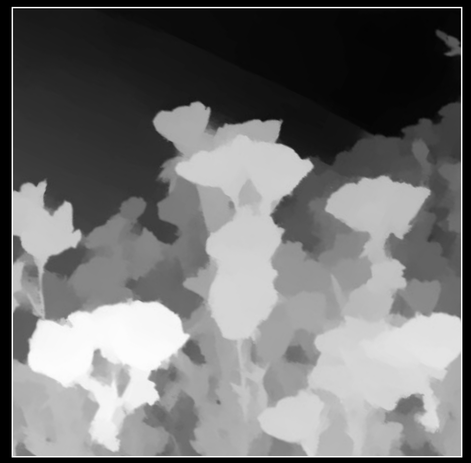
Transforming Depth Data

Orientation

Crop

Transform

Native resolution



Transforming Depth Data

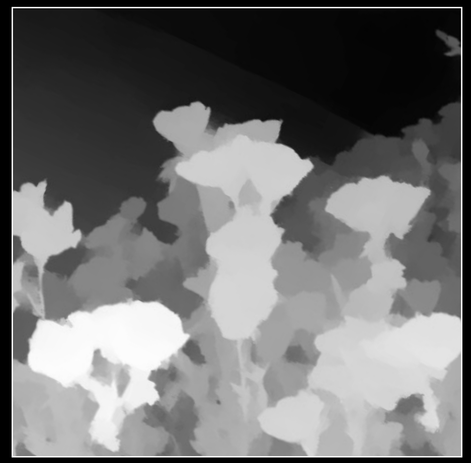
Orientation

Crop

Transform

Native resolution

No color matching



Transforming Depth Data

Transforming Depth Data

CVPixelBuffer

Transforming Depth Data

CVPixelBuffer

AVDepthData

Transforming Depth Data

CVPixelBuffer

AVDepthData

```
let renderedDepthBuffer: CVPixelBuffer
let originalDepthData: AVDepthData

// Create new AVDepthData object from rendered buffer
let outputDepthData = try originalDepthData.replacingDepthDataMap(with: renderedDepthBuffer)
```

Writing Depth Data with ImageIO

```
// Saving Original Depth Data Using ImageIO

let outputURL: URL // output image file
let outputImage: CGImage // output image
let destination = CGImageDestinationCreateWithURL(outputURL, kUTTypeJPEG, 1, nil)
CGImageDestinationAddImage(destination, outputImage, nil) // Add image first

// Use AVDepthData to get auxiliary data dictionary
let depthData: AVDepthData // output depth data
var auxDataType: NSString?
let auxData = depthData.dictionaryRepresentation(forAuxiliaryDataType: &auxDataType)

// Add auxiliary data to image destination
CGImageDestinationAddAuxiliaryDataInfo(destination, auxDataType, auxData)

// Write image file
let success = CGImageDestinationFinalize(imageDestination)
```

```
// Saving Original Depth Data Using ImageIO

let outputURL: URL // output image file
let outputImage: CGImage // output image
let destination = CGImageDestinationCreateWithURL(outputURL, kUTTypeJPEG, 1, nil)
CGImageDestinationAddImage(destination, outputImage, nil) // Add image first

// Use AVDepthData to get auxiliary data dictionary
let depthData: AVDepthData // output depth data
var auxDataType: NSString?
let auxData = depthData.dictionaryRepresentation(forAuxiliaryDataType: &auxDataType)

// Add auxiliary data to image destination
CGImageDestinationAddAuxiliaryDataInfo(destination, auxDataType, auxData)

// Write image file
let success = CGImageDestinationFinalize(imageDestination)
```

```
// Saving Original Depth Data Using ImageIO

let outputURL: URL // output image file
let outputImage: CGImage // output image
let destination = CGImageDestinationCreateWithURL(outputURL, kUTTypeJPEG, 1, nil)
CGImageDestinationAddImage(destination, outputImage, nil) // Add image first

// Use AVDepthData to get auxiliary data dictionary
let depthData: AVDepthData // output depth data
var auxDataType: NSString?
let auxData = depthData.dictionaryRepresentation(forAuxiliaryDataType: &auxDataType)

// Add auxiliary data to image destination
CGImageDestinationAddAuxiliaryDataInfo(destination, auxDataType, auxData)

// Write image file
let success = CGImageDestinationFinalize(imageDestination)
```

```
// Saving Original Depth Data Using ImageIO

let outputURL: URL // output image file
let outputImage: CGImage // output image
let destination = CGImageDestinationCreateWithURL(outputURL, kUTTypeJPEG, 1, nil)
CGImageDestinationAddImage(destination, outputImage, nil) // Add image first

// Use AVDepthData to get auxiliary data dictionary
let depthData: AVDepthData // output depth data
var auxDataType: NSString?
let auxData = depthData.dictionaryRepresentation(forAuxiliaryDataType: &auxDataType)

// Add auxiliary data to image destination
CGImageDestinationAddAuxiliaryDataInfo(destination, auxDataType, auxData)

// Write image file
let success = CGImageDestinationFinalize(imageDestination)
```

```
// Saving Original Depth Data Using ImageIO

let outputURL: URL // output image file
let outputImage: CGImage // output image
let destination = CGImageDestinationCreateWithURL(outputURL, kUTTypeJPEG, 1, nil)
CGImageDestinationAddImage(destination, outputImage, nil) // Add image first

// Use AVDepthData to get auxiliary data dictionary
let depthData: AVDepthData // output depth data
var auxDataType: NSString?
let auxData = depthData.dictionaryRepresentation(forAuxiliaryDataType: &auxDataType)

// Add auxiliary data to image destination
CGImageDestinationAddAuxiliaryDataInfo(destination, auxDataType, auxData)

// Write image file
let success = CGImageDestinationFinalize(imageDestination)
```

```
// Saving Original Depth Data Using ImageIO

let outputURL: URL // output image file
let outputImage: CGImage // output image
let destination = CGImageDestinationCreateWithURL(outputURL, kUTTypeJPEG, 1, nil)
CGImageDestinationAddImage(destination, outputImage, nil) // Add image first

// Use AVDepthData to get auxiliary data dictionary
let depthData: AVDepthData // output depth data
var auxDataType: NSString?
let auxData = depthData.dictionaryRepresentation(forAuxiliaryDataType: &auxDataType)

// Add auxiliary data to image destination
CGImageDestinationAddAuxiliaryDataInfo(destination, auxDataType, auxData)

// Write image file
let success = CGImageDestinationFinalize(imageDestination)
```

Writing Depth Data with Core Image

Writing Depth Data with Core Image

```
// Write image and depth data to a JPEG file
let context: CIContext
try context.writeJPEGRepresentation(of: image, to: outputURL, colorSpace: outputSpace,
    options: [kCIImageRepresentationAVDepthData : depthData])
```

Writing Depth Data with Core Image

```
// Write image and depth data to a JPEG file
let context: CIContext
try context.writeJPEGRepresentation(of: image, to: outputURL, colorSpace: outputSpace,
    options: [kCIImageRepresentationAVDepthData : depthData])
```

Writing Depth Data with Core Image

```
// Write image and depth data to a JPEG file
let context: CIContext
try context.writeJPEGRepresentation(of: image, to: outputURL, colorSpace: outputSpace,
    options: [kCIImageRepresentationAVDepthData : depthData])
```

```
// Write image and depth image to a JPEG file
let disparityImage: CIImage // output disparity image
try context.writeJPEGRepresentation(of: image, to: outputURL, colorSpace: outputSpace,
    options: [kCIImageRepresentationDisparityImage : disparityImage])
```

Summary

Depth and disparity

Load and prepare depth data

Filter using depth data

Save depth data

More Information

https://developer.apple.com/wwdc17/508

Related Sessions

Advances in Core Image: Filters, Metal, Vision, and More	Executive Ballroom	Thursday 1:50PM
What's New in Photos APIs		WWDC 2017
Capturing Depth in iPhone Photography		WWDC 2017

Labs

Photos Editing and Core Image Lab	Technology Lab F	Thursday 3:10PM–6:00PM
Photos Depth and Capture Lab	Technology Lab A	Thursday 3:10PM–6:00PM
Photos Depth and Capture Lab	Technology Lab F	Friday 1:50PM–4:00PM


WWDC17