Framework

Model I/O

Import, export, and manipulate 3D models using a common infrastructure that integrates MetalKit, GLKit, and SceneKit.

iOS 9.0+ | iPadOS 9.0+ | Mac Catalyst 13.0+ | macOS 10.11+ | tvOS 9.0+ | visionOS 1.0+

Overview

The Model I/O framework provides a system-level understanding of 3D model assets and related resources. You can use this framework to import and export assets from and to a variety of industry standard file formats supported by popular authoring tools and game engines. You can also use Model I/O to generate or process model and texture data—for example, to create subdivision surfaces, to bake ambient occlusion textures, or to generate light probes. Model I/O can share data buffers with the MetalKit, GLKit, and SceneKit frameworks to help you load, process, and render 3D assets efficiently.

Model I/O Features

- Importing and exporting 3D assets. A `MDLAsset` object represents a collection of objects that describe elements of a 3D scene—`MDLMesh`, `MDLLight`, and `MDLCamera` objects. Use the `MDLAsset` class to load these objects from a file or to create a collection of 3D objects for export to a file.

- Working with 3D model data. Use the `MDLVertexDescriptor` class to inspect or rearrange a mesh's vertex and index data format. Use classes that adopt the `MDLMeshBuffer` and `MDLMeshBufferAllocator` protocols to minimize the number of times a mesh's vertex and index data is copied and translated between loading, processing, and rendering on a GPU. The MetalKit and GLKit frameworks provide such classes—see MetalKit and GLKit.

- Processing and generating asset data. Use `MDLMesh` methods (for example, the `addNormals(withAttributeNamed:creaseThreshold:)` method) to process a model, generating additional data—such as surface normals, tangent basis vectors, ambient occlusion, or light maps—for use in rendering. Use the `MDLTexture` class and its subclasses to generate procedural textures such as noise, normal maps, and realistic sky boxes. Use the `MDLLightProbe` class to generate light sources whose illumination is based on the contents of a scene. Use the `MDLVoxelArray` class to work with a volumetric description of a model.

- Describing realistic rendering parameters. The `MDLPhysicallyPlausibleScatteringFunction` class—one of many ways to describe the surface appearance for a `MDLMaterial` object associated with a mesh—defines the intended rendering of a surface using the same physically based shading systems seen in popular feature films and high-end game engines. The `MDLPhotometricLight` and `MDLPhysicallyPlausibleLight` classes describe realistic lighting properties for use in rendering, and the `MDLCamera` class also supports physically based rendering parameters.

Topics

3D Asset Basics

Assets are collections of objects representing the elements of a 3D scene, typically loaded from one of several industry standard file formats. Use these classes to load assets and examine or manipulate the 3D objects, or meshes, within.

`class MDLAsset`

An indexed container for 3D objects and associated information, such as transform hierarchies, meshes, cameras, and lights.

`class MDLObject`

The base class for objects that are part of a 3D asset, including meshes, cameras, and lights.

`class MDLTransform`

A description of the local coordinate space transformations for a 3D object.

`class MDLMesh`

A container for vertex buffer data to be used in rendering a 3D object.

`class MDLSubmesh`

A container for index buffer data and material information to be used in rendering all or part of a 3D object.

`class MDLSubmeshTopology`

A description of how a submesh’s index buffer data is arranged and how that arrangement should be used to produce the submesh’s intended 3D shape.

`protocol MDLNamed`

The common interface for Model I/O objects that expose a human-readable name.

Managing Mesh Data

Mesh data—vertex and index buffers—is the main content of a 3D object. These types provide a rich vocabulary for inspecting or rearranging the content and format of asset data, as well as utilities that minimize the number of times a mesh's vertex and index data is copied and translated between loading, processing, and rendering on a GPU with MetalKit or GLKit.

`protocol MDLMeshBuffer`

The general interface for managing storage of vertex and index data used in loading, processing, and rendering meshes.

`protocol MDLMeshBufferAllocator`

The general interface for managing allocation of data buffers to be used in loading, processing, and rendering meshes.

`class MDLMeshBufferData`

A memory buffer that stores vertex or index data for a Model I/O mesh.

`class MDLMeshBufferDataAllocator`

A basic allocator implementation that allocates from main memory using data objects.

`class MDLMeshBufferMap`

An object that manages access to a memory buffer used for the data storage of a Model I/O mesh.

`protocol MDLMeshBufferZone`

The general interface for logical pools of memory used in allocation of related mesh data buffers.

`class MDLMeshBufferZoneDefault`

A standard implementation of the `MDLMeshBufferZone` protocol.

`class MDLVertexAttribute`

A description of the format of per-vertex data for a single vertex attribute in a mesh object.

`class MDLVertexAttributeData`

An object that provides convenience access to vertex data for a specific vertex attribute of a mesh.

`class MDLVertexBufferLayout`

A `MDLVertexBufferLayout` object describes layout information for a vertex buffer in a `MDLMesh` object. A collection of vertex layer objects, vertex attribute objects, and additional information forms a `MDLVertexDescriptor` object, which completely describes the layout of vertex buffers for a mesh.

`class MDLVertexDescriptor`

A description of the structure, format, and layout for vertex data buffers associated with a mesh.

Materials

These classes provide several different ways to describe the intended surface appearance for rendering a 3D object.

`class MDLMaterial`

A collection of material properties that together describe the intended surface appearance for rendering a 3D object.

`class MDLMaterialProperty`

A definition for one specific aspect of the rendering parameters for a material.

`class MDLMaterialPropertyConnection`

`class MDLMaterialPropertyGraph`

`class MDLMaterialPropertyNode`

`class MDLScatteringFunction`

A set of material properties that describes a basic shading model for materials, and the superclass for more complex shading models.

`class MDLPhysicallyPlausibleScatteringFunction`

A set of material properties that describes a physically realistic shading model for materials.

Textures

Use these classes to access the textures associated with an asset or to procedurally generate texture content.

`class MDLTexture`

A source of texel data to be used in rendering material surface appearances.

`class MDLCheckerboardTexture`

A generator of texel data that creates a checkerboard pattern with two specified colors.

`class MDLColorSwatchTexture`

A generator of texel data that creates a gradient between two specified colors.

`class MDLNoiseTexture`

A generator of texel data that creates a field of random noise.

`class MDLNormalMapTexture`

A generator of texel data that computes a normal map from a supplied texture.

`class MDLSkyCubeTexture`

A generator of texel data that creates cube textures using a physically realistic simulation of the sunlit sky.

`class MDLURLTexture`

A lightweight reference to a URL from which to load texture data.

`class MDLTextureFilter`

A description of filtering modes for a renderer to use when sampling from a texture.

`class MDLTextureSampler`

An object that pairs a source of texture data with sampling parameters to be used in rendering the texture.

Lights

These classes provide several different ways to describe light sources for use in rendering 3D scenes.

`class MDLLight`

The abstract superclass for objects that describe light sources in a scene.

`class MDLAreaLight`

A light source that illuminates a 3D scene from an area with a specific shape.

`class MDLLightProbe`

A light source described in terms of the variations in color and intensity of its illumination in all directions.

`protocol MDLLightProbeIrradianceDataSource`

Adopt this protocol to provide information for use in automatic placement of light probes around a scene.

`class MDLPhotometricLight`

A light source whose shape, direction, and intensity of illumination are determined by a photometric profile.

`class MDLPhysicallyPlausibleLight`

A light source for use in shading models based on real-world physics.

Cameras

Use these classes to describe, or to access in imported assets, information about viewpoints for 3D scene rendering.

`class MDLCamera`

A point of view for rendering a 3D scene, along with a set of parameters describing an intended appearance for rendering.

`class MDLStereoscopicCamera`

A point of view for rendering a stereoscopic display of a 3D scene.

Extensible Asset Format Support

These classes and protocols enable both Model I/O support of widely differing standard asset file formats and the creation of new asset formats that include both standard and custom elements.

`protocol MDLComponent`

The base protocol for extensible file format support in Model I/O.

`class MDLObjectContainer`

A default implementation for handling object hierarchy relationships in a 3D asset.

`protocol MDLObjectContainerComponent`

The general interface for classes that can act as containers in an object hierarchy.

`protocol MDLTransformComponent`

The general interface for classes that manage local coordinate space transforms for 3D objects

Volumetric Representations

Voxels provide an alternate way of working with 3D objects, which can be useful for applications like modeling special effects, preparing designs for physical manufacturing, and performing constructive solid geometry operations.

`class MDLVoxelArray`

A model of a 3D object’s solid volume as a collection of *voxels*, or cubic units.

Reference

- Model I/O Data Types
- Model I/O Structures
- Model I/O Enumerations
- Model I/O Constants

Classes

`class MDLAnimatedMatrix4x4`

`class MDLAnimatedQuaternion`

`class MDLAnimatedQuaternionArray`

`class MDLAnimatedScalar`

`class MDLAnimatedScalarArray`

`class MDLAnimatedValue`

`class MDLAnimatedVector2`

`class MDLAnimatedVector3`

`class MDLAnimatedVector3Array`

`class MDLAnimatedVector4`

`class MDLAnimationBindComponent`

`class MDLBundleAssetResolver`

`class MDLMatrix4x4Array`

`class MDLPackedJointAnimation`

`class MDLPathAssetResolver`

`class MDLRelativeAssetResolver`

`class MDLSkeleton`

`class MDLTransformMatrixOp`

`class MDLTransformOrientOp`

`class MDLTransformRotateOp`

`class MDLTransformRotateXOp`

`class MDLTransformRotateYOp`

`class MDLTransformRotateZOp`

`class MDLTransformScaleOp`

`class MDLTransformStack`

`class MDLTransformTranslateOp`

`class MDLUtility`

Protocols

`protocol MDLAssetResolver`

`protocol MDLJointAnimation`

`protocol MDLTransformOp`