Documentation > Model I/O

Language: Swift API Changes: Show

Framework

Model I/O

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

Availability

iOS 9.0+

iPadOS 9.0+

macOS 10.11+

Mac Catalyst 13.0+

tvOS 9.0+

On This Page

Overview

Topics

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.