From Art to Engine with Model I/O

Session 610

Nick Porcino, Game Technologies Engineer
Nicholas Blasingame, Game Technologies Engineer
Model I/O
Apple's toolkit for building pipelines

Import and export 3D assets
Geometry, materials, lighting, cameras, voxels
Data format conversions
Processing tools
Model I/O
What's new?

Improved Importers
Skinned Character Animation
Blend Shapes
Transform Stacks
Model I/O
Intuitive Asset Traversal

Format independent graph
Logical
Consistent
Art Assets

Models, materials, animations

Textures

Scenes composed of many files
From Art to Engine

Art asset is like source code

Compiled for an engine
From Art to Engine
UI based tools

Easy the first few times

Overwhelming during revision
How can the work be scaled?
Introducing the Pipeline

Export the art

Exporter

Artwork → Asset → Engine-Ready Data
Introducing the Pipeline
Transform the asset

Artwork → Asset → Engine-Ready Data

Tool
Introducing the Pipeline
Load engine-ready data

Artwork → Asset → Engine-Ready Data → Engine
Introducing the Pipeline

Export the art

Artwork → Asset → Engine-Ready Data

Exporter
Introducing the Pipeline
Exporter

Maya

• Asset Exporter
• Complex hierarchies of files
• Export script in the sample
Introducing the Pipeline
Exporter

Maya
• Asset Exporter
• Complex hierarchies of files
• Export script in the sample
Universal Scene Description

http://openusd.org
Universal Scene Description
Composition

- Shot Layer
  - Master_1
  - Master_2
  - Pit Building
  - Race Track

- Car
- Wheel
- Tire Wall
- Pit Building
- Race Track
Universal Scene Description
Classes, Variations, and Overrides

shadingVariant

modelingVariant
over "World"
{
  over "anim"
  {
    over "chars"
    {
      def "Car"
      (
        add references = @chars/car.usd@</Car>
      )
    }
    {  // Color display color
      color3f displayColor = (0.9, 0, 0)
    }
  }
}
Introducing the Pipeline
Scriptable command line tool
Introducing the Pipeline
Scriptable command line tool
Introducing the Pipeline
Scriptable command line tool

Repeatable
Introducing the Pipeline
Scriptable command line tool

Repeatable
Consistent
Introducing the Pipeline
Scriptable command line tool

Repeatable
Consistent
Scriptable
Introducing the Pipeline
Scriptable command line tool

Repeatable
Consistent
Scriptable
Scalable
Introducing the Pipeline
Scriptable command line tool

Repeatable
Consistent
Scriptable
Scalable
Composable
Introducing the Pipeline

The sample

Demonstrates principles
- Simplified data
- Uncompressed
- Good jumping-off point

![Diagram showing the flow from Asset to Engine-Ready Data with a cycle back to Asset]
Introducing the Pipeline

Game engine

Artwork → Asset → Engine-Ready Data → Engine
Game Engine
Simple renderer

Single-pass forward renderer
Physically-based shader
Mesh instancing
Skinned and animated meshes
Multiple materials
Render Loop

- Set Transform Buffer
- Set Skinning Data
- Set Vertex Buffer
- Set Pipeline State
- Set Material Uniforms
- Set Fragment Textures
- Draw Indexed Primitive
The Pipeline

Artwork → Asset → Engine-Ready Data

Baker
Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation
Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation
Geometry + Transform

A simple scene graph

Transform Hierarchy

- Tree of nodes
- Meshes and transform objects
Geometry + Transform

Transform hierarchy

Moving the parent will move the children
Geometry + Transform
Compactly encode transform hierarchy
Geometry + Transform
Array of local transforms

Local Transforms
Geometry + Transform
Array of local transforms

Local Transforms
Geometry + Transform
Array of local transforms

Local Transforms

A
B
C
D
Geometry + Transform
Array of local transforms
Geometry + Transform
Assign indices

Local Transforms

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Geometry + Transform

Array of parent indices
Geometry + Transform
Array of parent indices

Local Transforms

Parent Indices

A
B
C
D

0
1
2
3

0
Geometry + Transform

Array of parent indices

Local Transforms

A
B
C
D

Parent Indices

- 0
1 1
Geometry + Transform

Array of parent indices

Local Transforms

Parent Indices

A
0

B
1

C
2

D
3

A
0

B
1

C
2

D
3

-
Geometry + Transform

Array of mesh indices

- Local Transforms:
  - A: 0
  - B: 1
  - C: 2
  - D: 3

- Parent Indices:
  - A: -
  - B: 0
  - C: 1
  - D: 1

- Mesh Indices:
  - 2
Geometry + Transform

Array of mesh indices

Local Transforms:
- A
- B
- C
- D

Parent Indices:
- A 0
- B 1
- C 2
- D 3

Mesh Indices:
- 2
- 3
Geometry + Transform

Mesh data

Descriptor | Vertex Buffers | Index Buffer
---|---|---
Descriptor | Vertex Buffers | Index Buffer
Geometry + Transform
Vertex buffers

```swift
//for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
    vertexDescriptors.append(mesh.vertexDescriptor)
    for vertexBuffer in mesh.vertexBuffers {
        let vertexBufferData = Data(bytes: vertexBuffer.map().bytes, count: vertexBuffer.length)
        ...
    }
    for submesh in mesh.submeshes! {
        if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
            let indexBufferData = Data(bytes: indexBuffer.map().bytes, count: indexBuffer.length)
            ...
        }
    }
}
```
// for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
    vertexDescriptors.append(mesh.vertexDescriptor)
    for vertexBuffer in mesh.vertexBuffers {
        let vertexBufferData = Data(bytes: vertexBuffer.map().bytes,
                                     count: vertexBuffer.length)
    ...
    }

    for submesh in mesh.submeshes! {
        if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
            let indexBufferData = Data(bytes: indexBuffer.map().bytes,
                                         count: indexBuffer.length)
        ...
        }
    }
}
//for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
    vertexDescriptors.append(mesh.vertexDescriptor)
    for vertexBuffer in mesh.vertexBuffers {
        let vertexBufferData = Data(bytes: vertexBuffer.map().bytes, count: vertexBuffer.length)
        ...
    }
    for submesh in mesh.submeshes! {
        if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
            let indexBufferData = Data(bytes: indexBuffer.map().bytes, count: indexBuffer.length)
            ...
        }
    }
}
/for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
    vertexDescriptors.append(mesh.vertexDescriptor)
    for vertexBuffer in mesh.vertexBuffers {
        let vertexBufferData = Data(bytes: vertexBuffer.map().bytes,
                                     count: vertexBuffer.length)

        ...
    }
}
for submesh in mesh.submeshes! {
    if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
        let indexBufferData = Data(bytes: indexBuffer.map().bytes,
                                     count: indexBuffer.length)

        ...
    }
}
var localTransforms: [matrix_float4x4] = []

// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
    localTransforms.append(transform.matrix)
}
Geometry + Transform

Local transform

```swift
var localTransforms: [matrix_float4x4] = []

// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
    localTransforms.append(transform.matrix)
}
```
Geometry + Transform

Mesh Data
- Descriptors
  - Vertex + Index Buffers

Scene Composition Data
- Parent Indices, Mesh Indices

Transform Data
- Local Transforms
Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation
Texture Paths + Materials
Materials stored on MDLSubmesh

Fetch properties referenced by shader

Record texture paths and values
// for every submesh:
if let material = submesh.material {

    for property in material.properties(with:<MDLMaterialSemantic>) {
        if property.type == .string || property.type == .URL {
            // texture
        }
        else if property.type == <MDLMaterialPropertyType> {
            // uniform value
        }
    }
}
// for every submesh:
if let material = submesh.material {

    for property in material.properties(with:<MDLMaterialSemantic>) {
        if property.type == .string || property.type == .URL {
            // texture
        }
        else if property.type == <MDLMaterialPropertyType> {
            // uniform value
        }
    }
}
Texture Paths + Materials

- Mesh Data
  - Descriptors
  - Vertex + Index Buffers

- Scene Composition Data
  - Parent Indices, Mesh Indices

- Transform Data
  - Local Transforms
Texture Paths + Materials

Mesh Data
- Descriptors
- Vertex + Index Buffers

Scene Composition Data
- Parent Indices, Mesh Indices

Transform Data
- Local Transforms

Material Data
- Material Uniforms
- Texture Paths
Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation
Instancing

A mesh can be used multiple times
Instancing

A mesh can be used multiple times
Instancing

A mesh can be used multiple times

Why store it multiple times?
Instancing
Masters

MDLAsset has a masters array
Instancing
Model I/O support

MDLObject instance pointers refer to masters
Instancing

Masters

Local Transforms
- A
- B
- C
- D
- E

Parent Indices
- 0
- 1
- 1
- 1
- 0

Mesh Indices
- 2
- 3
- 4
Instancing

Masters

A

B

C

D

E

Local Transforms

A 0
B 1
C 2
D 3
E 4

Parent Indices

A 0
B 1
C 1
D 1
E 0

Mesh Indices

2
3
4
Instancing

Masters

Local Transforms | Parent Indices | Mesh Indices

A | 0 | - | 2
B | 1 | 0 | 4
C | 2 | 1 | 4
D | 3 | 1 | 4
E | 4 | 0 | 3

A� B� C� D� E�
Instancing

Masters

A
B
C
D
E

Local Transforms

A
B
C
D
E

0
1
2
3
4

Parent Indices

A
B
C
D
E

- 0
1 1
1

Mesh Indices

2
4
3

Instance Count

2
1
Instancing

Mesh Data
- Descriptors
- Vertex + Index Buffers

Scene Composition Data
- Parent Indices, Mesh Indices
- Instance Count

Transform Data
- Local Transforms

Material Data
- Material Uniforms
- Texture Paths
Demo
Geometry, Materials, and Instancing

Nicholas Blasingame, Game Technologies Engineer
Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation
Transform Animation
Transforms that vary over time
Transform Animation
Sample the animations

Animated Local Transforms

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

t0, t1, ..., tn...
var localTransforms: [matrix_float4x4] = []

// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
    localTransforms.append(transform.matrix)
}
var localTransforms: [matrix_float4x4] = []

// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
    localTransforms.append(transform.matrix)
}

if (transform.keyTimes.count > 1) {
    let sampledXM = sampleTimes.map{ transform.localTransform!(atTime: $0) }
    animatedLocalTransforms.append(sampledXM)
    ...
}
}
var localTransforms: [matrix_float4x4] = []

// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
    localTransforms.append(transform.matrix)

    if (transform.keyTimes.count > 1) {
        let sampledXM = sampleTimes.map{ transform.localTransform!(atTime: $0) }
        animatedLocalTransforms.append(sampledXM)
    }
}
Transform Animation

Mesh Data
- Descriptors
- Vertex + Index Buffers

Scene Composition Data
- Parent Indices, Mesh Indices
- Instance Count

Transform Data
- Local Transforms

Material Data
- Material Uniforms
- Texture Paths
Transform Animation

- **Mesh Data**
  - Descriptors
  - Vertex + Index Buffers

- **Scene Composition Data**
  - Parent Indices, Mesh Indices
  - Instance Count

- **Transform Data**
  - Local Transforms
  - Animated Local Transforms

- **Material Data**
  - Material Uniforms
  - Texture Paths
Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation
Skinned Character Animation
Skinned Character Animation
Skinned Character Animation

Mesh has geometry
Skinned Character Animation
Embedded skeleton
Skinned Character Animation
Vertex weighting to joints
Skinned Character Animation

Vertex weighting to joints
Skinned Character Animation
Vertex blending in shader

```cpp
{
  ...
  float4 position = vertex.pos;
  packed_uchar4 jIdx = vertex.jointIndices;
  packed_float4 w = vertex.jointWeights;

  float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                           w[1] * (palette[jIdx[1]] * modelPosition) +
                           w[3] * (palette[jIdx[3]] * modelPosition);

  ...
}
```
Skinned Character Animation

Input vertex attributes

```plaintext
{
  ...
  float4 position = vertex.pos;
  packed_uchar4 jIdx = vertex.jointIndices;
  packed_float4 w = vertex.jointWeights;

  float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                         w[1] * (palette[jIdx[1]] * modelPosition) +
                         w[3] * (palette[jIdx[3]] * modelPosition);

  ...
}
```
Skinned Character Animation
Per vertex joint indices

```c
{
    ...
    float4 position = vertex.pos;
    packed uchar4 jIdx = vertex.jointIndices;
    packed float4 w = vertex.jointWeights;

    float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                            w[1] * (palette[jIdx[1]] * modelPosition) +
                            w[3] * (palette[jIdx[3]] * modelPosition);

    ...
}
```
Skinned Character Animation
Indexed bones for weight each vertex

```cpp
{
    ...  
    float4 position = vertex.pos;
    packed_uchar4 jIdx = vertex.jointIndices;
    packed_float4 w = vertex.jointWeights;

    float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                             w[1] * (palette[jIdx[1]] * modelPosition) +
                             w[3] * (palette[jIdx[3]] * modelPosition);

    ...  
}
```
Skinned Character Animation

Skeleton

Root

Body

Wheel 1

Wheel 2
Skinned Character Animation

Skeleton

E

Root

Body

Wheel 1

Wheel 2
Skinned Character Animation

Skeleton

- Root
  - Body
    - Wheel 1
    - Wheel 2
Skinned Character Animation

Skeleton

- Root
- Body
- Wheel 1
- Wheel 2
Skinned Character Animation

Skeleton

- E
- Root
- Body
- Wheel 1
- Wheel 2

Joint Parent Indices

0
-1
1
1
Skinned Character Animation
Bind Poses + Joint Indices

Skeleton:
- Root
- Body
- Wheel 1
- Wheel 2

Joint Parent Indices:
- Root: -
- Body: 0
- Wheel 1: 1
- Wheel 2: 1

Bound Joint Indices:
- Root: 1
- Body: 2
- Wheel 1: 3
- Wheel 2: 3

Inverse Bind Pose:
- B1⁻¹
- B2⁻¹
- B3⁻¹
Skinned Character Animation
Bind Poses + Joint Indices

Skeleton
- E
- Root
- Body
- Wheel 1
- Wheel 2

Joint Parent Indices:
- 0
- 1
- 2
- 3

Bound Joint Indices:
- 1
- 2
- 3

Inverse Bind Pose:
- B₁⁻¹
- B₂⁻¹
- B₃⁻¹
Skinned Character Animation
Bind Poses + Joint Indices

Skeleton

- E
- Root
- Body
- Wheel 1
- Wheel 2

Joint Parent Indices:
0: -
1: 0
2: 1
3: 1

Bound Joint Indices:
1: 1
2: 2
3: 3

Inverse Bind Pose:
B_1^{-1}
B_2^{-1}
B_3^{-1}
Skinned Character Animation

Time sample animation

Skeleton

Joint Parent Indices

Bound Joint Indices

Inverse Bind Pose

Animation Clip

Root $t_0...t_N$

Body $t_0...t_N$

Wheel 1 $t_0...t_N$

Wheel 2 $t_0...t_N$
// for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
    if let skin =
        object.componentConforming(to:MDLSkinDeformerComponent.self) as? MDLSkinDeformerComponent {
        let inverseBindTransforms = skin.jointBindTransforms().map{simd_inverse($0)}
        ...
    }
}
// for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
    if let skin = object.componentConforming(to: MDLSkinDeformerComponent.self) as? MDLSkinDeformerComponent {
        let inverseBindTransforms = skin.jointBindTransforms().map { simd_inverse($0) }
        ...
    }
}
Skinned Character Animation

- **Mesh Data**
  - Descriptors
  - Vertex + Index Buffers

- **Scene Composition Data**
  - Parent Indices, Mesh Indices
  - Instance Count

- **Transform Data**
  - Local Transforms
  - Animated Local Transforms

- **Material Data**
  - Material Uniforms
  - Texture Paths
Skinned Character Animation

- **Mesh Data**
  - Descriptors
  - Vertex + Index Buffers

- **Scene Composition Data**
  - Parent Indices, Mesh Indices
  - Instance Count

- **Transform Data**
  - Local Transforms
  - Animated Local Transforms
  - Animation Clips

- **Material Data**
  - Material Uniforms
  - Texture Paths

- **Skinning Data**
  - Inverse Bind Transforms
  - Joint to Palette Mapping
  - Skeleton Parent Indices
Demo
Instancing and Characters
Recap

Artwork → Asset → Engine Ready Data

USD
Enhancements

Light Mapping
Enhancements

Light Mapping

UV Unwrapping
Enhancements

- Light Mapping
- UV Unwrapping
- Ambient Occlusion
Enhancements

Image-based lighting
More Information

https://developer.apple.com/wwdc17/610
## Related Information

<table>
<thead>
<tr>
<th>Topic</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing 3D Assets with Model I/O</td>
<td>WWDC 2015</td>
</tr>
<tr>
<td>Introducing Metal 2</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>What’s New in SceneKit</td>
<td>WWDC 2017</td>
</tr>
</tbody>
</table>