COLLADA Overview

Mark Barnes
COLLADA Project Lead
Sony Computer Entertainment

COLLADA History

COLLADA was announced at SIGGRAPH 2004

and has become a Khronos standard

- Since then we have
  - Formed the COLLADA work groups
  - Released the 1.4.0 specification
  - Been adopted by thousands of users (about 15% of the market)

What is COLLADA?

- COLLADA is a database schema for 3D assets
  - COLLADA is an *interchange* format.
  - COLLADA is an *intermediate* format.
  - COLLADA is not a delivery format.

- COLLADA is a lossless format.
  - COLLADA enables *interoperability* between DCC tools.
  - COLLADA can be a *source* format that retains all information.
    - Even multiple versions of the same asset.

- COLLADA is a comprehensive format.
  - COLLADA can encode visual scenes *and* physics scenes.
  - COLLADA can contain libraries of assets.
  - COLLADA can contain many types of objects.
  - COLLADA is not a scene graph.

What is COLLADA?

- COLLADA is an open format
  - COLLADA is a fully documented, *open standard* built using standard tools: UTF8, XML, URLs, Base64, XML Schema, etc.
  - COLLADA is not a proprietary format.
- COLLADA is an open standard
  - COLLADA is a standard technology of The Khronos Group industry consortium.
- COLLADA seeks to liberate your 3D assets
  - COLLADA is an open, archive-grade format that retains meta information.
  - When your DCC tool upgrades, you keep your assets.

COLLADA is your data in motion

- COLLADA is designed to transport your content between applications
  - It is the glue that *binds together* your DCC and 3D processing tools into a production pipeline
  - It allows applications to *talk with each other* using the same language
  - It opens up *new opportunities* for developers and tool makers to build on each other's strengths *without reinventing the wheel*

Structure of a COLLADA document

- Every COLLADA document is expressed in XML, with

a *header*,

a set of *libraries*,

and a *scene*.

```
<?xml version="1.0" encoding="UTF-8"?>
<COLLADA>
  <asset> ... </asset>

  <library_geometries>
    <geometry id="Teapot">
      ...
    </geometry>
  </library_geometries>

  <library_visual_scenes>
    <visual_scene id="World">
      <node>
        <instance_geometry url="#Teapot"/>
      </node>
    </visual_scene>
  </library_visual_scenes>

  <scene>
    <instance_visual_scene url="#World"/>
  </scene>

</COLLADA>
```

COLLADA Documents

- We call them "documents" not "files"
  - XML can be more than just files on a file system.
  - XML may be the formatted result of a SQL database query.
  - XML may come from an HTTP request sent to a remote server.
  - XML might be a temporary output fragment from a DCC tool.
  - XML could be a small subset of a large source control system, selected by *"assets most recently altered by David".*
- Using XML to represent structured data enables 3D tools to use the full arsenal of modern database tools

Libraries

- Libraries hold and organize many types of objects

```
<library_animations>     <library_lights>
<library_cameras>        <library_materials>
<library_effects>        <library_nodes>
<library_controllers>    <library_rigid_bodies>
<library_geometries>     <library_physics_models>
```

- COLLADA documents are modular containers of information

Assets

- Every COLLADA document *is an asset* and can be managed using the *meta-information*

```
<?xml version="1.0" encoding="UTF-8"?>
<COLLADA>
  <asset>
    <contributor>
      <author>John Doe</author>
      <authoring_tool>Maya Collada exporter v0.7.4</authoring_tool>
    </contributor>
    <created>2004-12-20T03:14:58Z</created>
    <modified>2004-12-20T22:10:18Z</modified>
    <revision>12.3.257</revision>
    <up_axis>Y_UP</up_axis>
  </asset>
 <library_geometries>
    <geometry id="3f454bb2" name="Teapot>">
      ...
    </geometry>
  </library_geometries>
</COLLADA>
```

Assets

- Every COLLADA document also *contains assets* that can be managed individually

```
<?xml version="1.0" encoding="UTF-8"?>
<COLLADA>
  <asset> ... </asset>
  <library_geometries>
    <geometry id="3f454bb2" name="Teapot>">
      <asset>
        <contributor>
          <author>Jane Doe</author>
          <authoring_tool>XSI COLLADA exporter v5.1</authoring_tool>
        </contributor>
        <created>2005-01-13T18:33:17Z</created>
        <modified>2005-01-13T18:33:17Z</modified>
        <revision>1.04</revision>
      </asset>
    </geometry>
  </library_geometries>
</COLLADA>
```

Models and Geometry

- Geometry contain multiple streams of data at each vertex, indexed from raw arrays of values

```
<source id="revolve_Pos">
  <float_array count="1824">
    6.44546 2.22100 0.04451 8.94581 ...
  </float_array>
</source>

<vertices id="revolve-Vtx">
  <input semantic="POSITION" source="#revolve-Pos"/>
</vertices>

<polygons count="720" material="Default">
  <input semantic="VERTEX" source="#revolve-Vtx" offset="0"/>
  <input semantic="NORMAL" source="#revolve-0-Normal" offset="1"/>
  <input semantic="TEXCOORD" source="#revolve-0-TexCoord0" offset="2"/>
  <input semantic="TEXCOORD" source="#revolve-0-TexCoord1" offset="3"/>
  <input semantic="TANGENT" source="#revolve-0-Tangent" offset="5"/>
  <p>0 0 0 0 0 0 31 1 1 1 1 1 32 2 2 2 2 2 1 3 3 3 3 3</p>
  ...
</polygons>
```

Scene Descriptions

- Each COLLADA **<visual_scene>** describes a *hierarchy* or *scene graph* of objects in the same space

```
<visual_scene id="World">
  <node id="Camera">
    <lookat> ... </lookat>
    <instance_camera url="#camera"/>
  </node>
  <node id="Light" name="Main_Light">
    <translate>-5.0 10.0 4.0</translate>
    <rotate>0 0 1 0</rotate>
    <instance_light url="#Lt-Light"/>
    <node id="Teapot">
      <instance_node url="#teapot-lod-02"/>
    </node>
  </node>
</visual_scene>
```

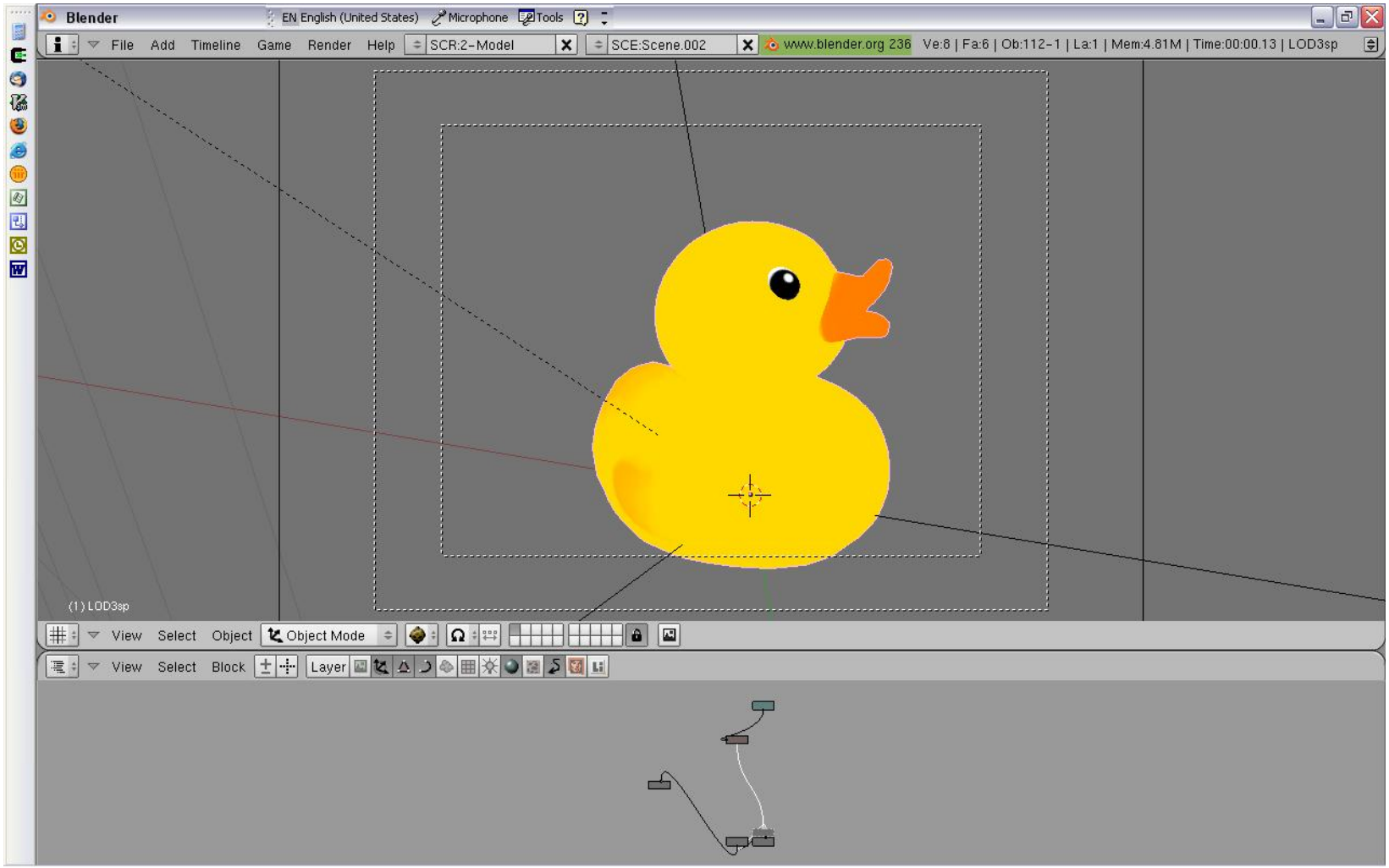
Animations and Skinning

- Animations can be controlled by *curves* or *key frames*
  - Each `<animation>` contains one or more `<source>` arrays of values.
  - `<sampler>` objects take these `<source>` arrays and either *indexes* or *interpolates* them into output variables.
  - Finally, `<channel>` binds these outputs to values in the scene, e.g. "`elbow/rotation.ANGLE`"

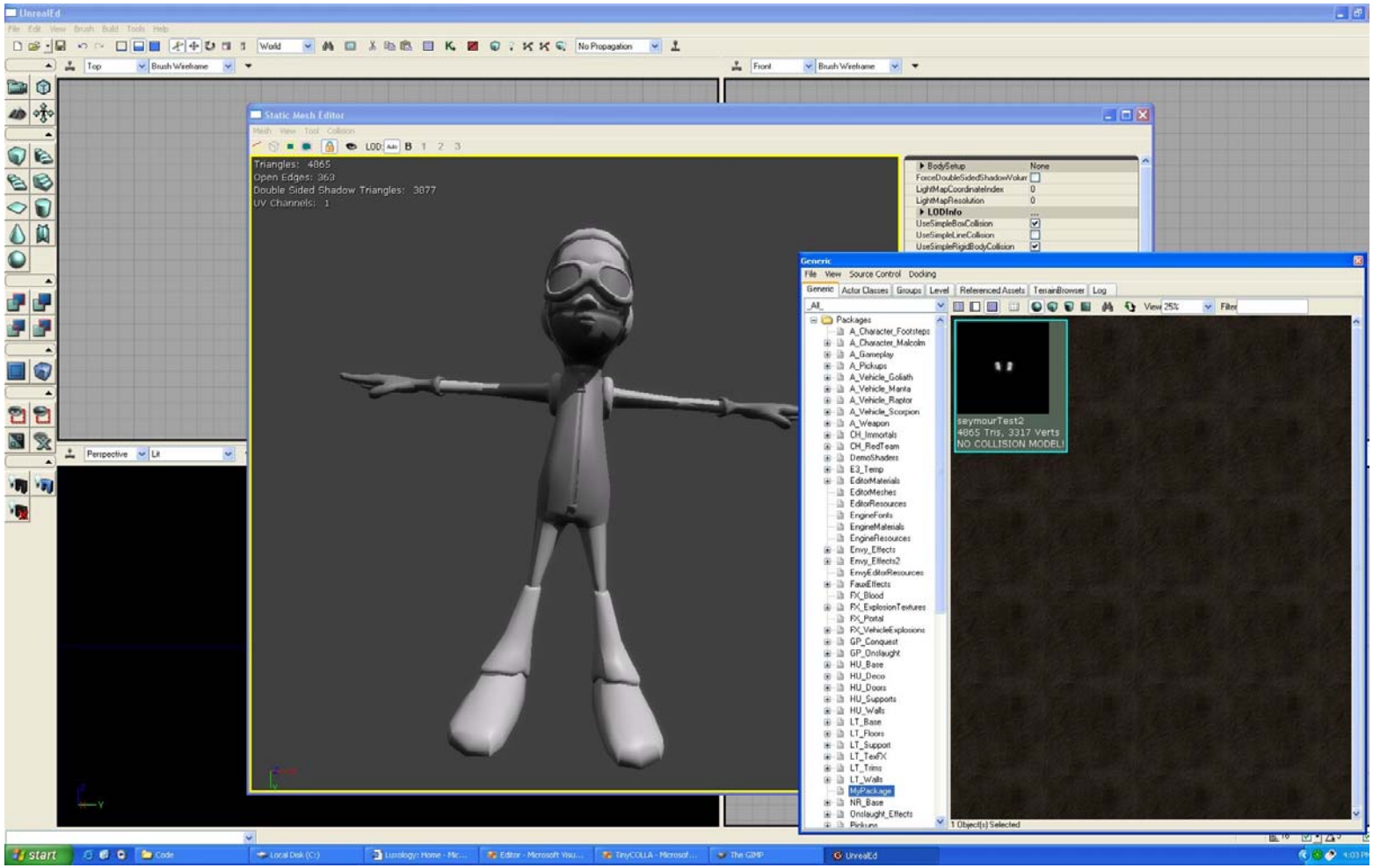
- COLLADA supports *skinned* animations
  - One skeleton can animate *many meshes.*
  - Meshes are bound to a *default pose* by an arbitrary array of weights and an array of *inverse bind matrices*.

COLLADA in Lightwave™

COLLADA in Blender™



COLLADA in Unreal Engine™



What's in COLLADA 1.4?

COLLADA FX

COLLADA Physics

Conformance and Standardization

Reconfigurable Production Pipelines

What are Shader Effects?

- Shaders are programs that run on a GPU and produce graphical effects

- Shader Effects additionally describe the environment that a shader is designed to run within:
  - Set render states (culling, blending, depth tests, etc.)
  - Which shader to use in which pipeline stage.
  - Which parameters are automatically set by the runtime.
  - Which parameters are tweakable by users.
  - How many passes are required to finish the effect.
  - Which textures are used during which pass.

The Shader Effects Problem

- Each platform has its own *proprietary file format* (or none at all) that expresses multi-pass shader effects.

Cg	*CgFX effect format*
HLSL	*.FX effect format*
GLSL	*No effect format yet*
OpenGL|ES	*No standard effect format*

- If you are designing a multi-platform game, you have a real resource management problem.

COLLADA FX

- COLLADA FX is designed to:
  - Allow *interchange* of complex, multi-pass shader effects.
  - Encapsulate *multiple descriptions* of an effect.
    - E.g. levels of detail, daytime/nighttime versions, etc.
  - Encode shaders for *multiple platforms* in one file.
    - e.g. CG and GLSL versions of your effect in one document.
  - Supports both *fixed-function* and *programmable-pipeline* effects in the same format.

COLLADA FX

- Additional features:
  - COLLADA FX can be coded as *stand-alone* COLLADA documents.
    - Each document can contain all the elements of an effect inline.
  - COLLADA FX supports *advanced shader language* techniques.
    - Use *Interface objects* and declare *Unsized Arrays*.
    - *Multiple Render Targets* and off-screen buffers.
  - Share simple parameters *across all platforms*
    - Control multiple platforms or techniques with a *single parameter*.
  - *Share source code* between multiple techniques.
  - All elements can be *annotated* with meta-information

How COLLADA FX works

- COLLADA FX does not attempt to create a single set of render states or an intermediate shader language
  - *Platform specific elements* allow access to the full set of data types and render states for that platform
  - COLLADA FX assumes the use of a *Run-time API* that holds state
  - COLLADA FX does *not* require that a run-time supports a scripting interpreter

```
<effect id="MossyRockEffect">
  <profile_GLSL id=“GLSL_medium_LOD”>
      ...
  </profile_GLSL>
  <profile_CG id="CG_with_lighting">
      ...
  </profile_CG>
</effect>
```

Techniques and Passes

- COLLADA FX files are structured into techniques and passes.

```
<effect id="Level3-spaceship-damaged ">
  <profile_GLSL id="high-LOD-shadows">
    <technique sid="Multipass1">
      <code> ... </code>
      <pass sid="shadow-pass">
        ...
      </pass>
      <pass sid="specular-reflection">
        ...
      </pass>
    </technique>
  </profile_GLSL>
</effect>
```

Passes

- Each pass sets *render states* and declares as many *programmable pipeline stages* as your platform supports
- Uniform shader inputs can be bound to *variables* or *literal values*
- Profiles *without programmable shaders* can only set render states
- Render states can be set by literal values or by reading a parameter

```
<pass sid="pass0">

  <shader stage="VERTEX">
    <compiler_target>vs_2_x</compiler_target>
    <name>vertex_function</name>
  </shader>

  <depth_test_enable> TRUE </depth_test_enable>
  <depth_func> LEQUAL </depth_func>
  <cull_face> BACK </cull_face>

  <shader stage="FRAGMENT">
    <compiler_target>ps_2_x</compiler_target>
    <name>pixel_function</name>
    <bind symbol="damage">
      <param ref="damage_amount"/>
    </bind>
  </shader>

</pass>
```

Parameters

```
<effect id="ThinFilm2">
  <profile_CG>

    <include sid=“thin1” url="#thinfilm_code"/>

    <technique sid=“Thin1”>

      <annotate name="UIName">
        <string>Light Pos 1</string>
      </annotate>
      <annotate name="Object">
        <string>PointLight</string>
      </annotate>
      <annotate name="Space">
        <string>World</string>
      </annotate>

      <setparam ref="LightPosP1">
        <float3> 10.0 10.0 10.0 </float3>
      </setparam>

      ...
```

- Parameters declared within source code can be *given values* using **<setparam>**
- Parameters can have multiple *annotations*. COLLADA FX does not interpret these annotations in any way
- Parameters can be *created* from data types defined in source code using **<newparam>**

Render to Texture

- COLLADA FX supports Multiple Render Targets using indexed output <surface> objects that can persist between passes in an effect.

```
<pass sid="pass0">

   <color_target index="0"> color_surface </color_target>
   <color_target index="1"> normal_surface </color_target>
   <depth_target> depth_stencil_surface </depth_target>
   <stencil_target> depth_stencil_surface </stencil_target>


   <clear_color index="0"> 0 0 0 </clear_color>
   <clear_color index="1"> 0 0 0 </clear_color>
   <clear_depth> 1.0 </clear_depth>
   <clear_stencil> 128 </clear_stencil>

   <draw>SCENE</draw>

   ...
</pass>
```

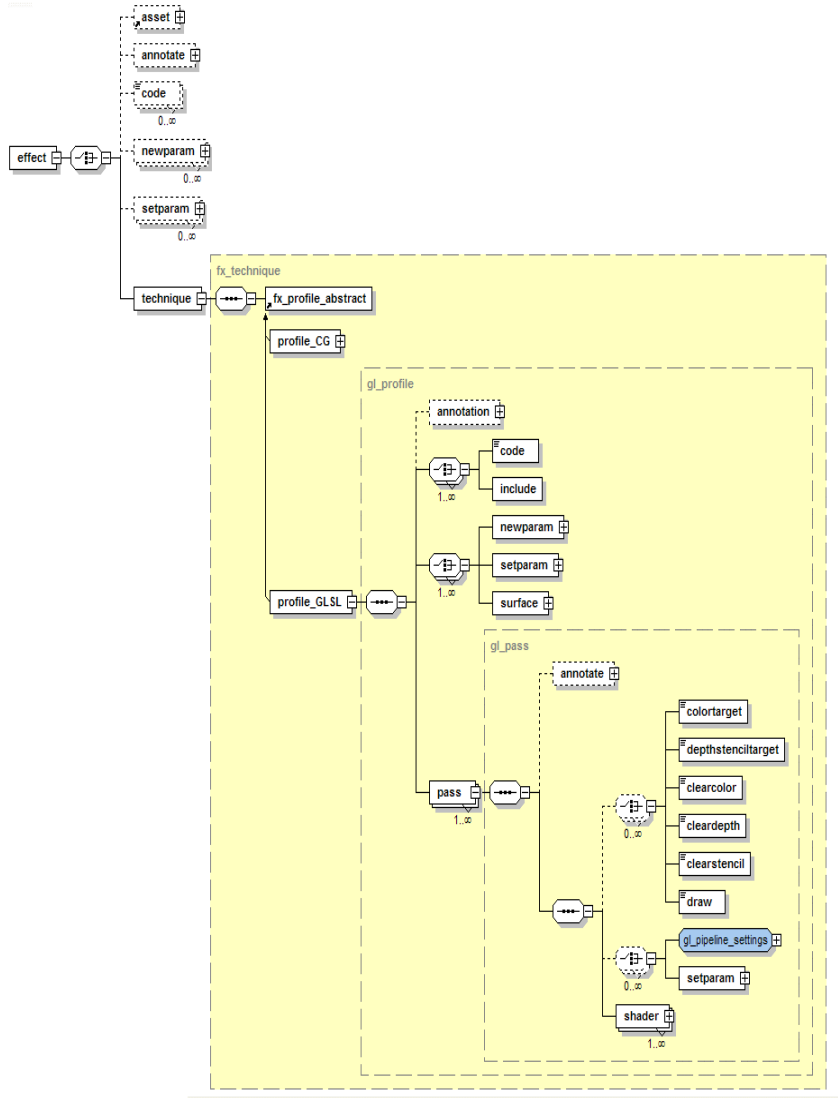
Advanced Shaders

- Cg style v*ariable-length arrays* and i*nterface objects* are supported by COLLADA FX
  - Structures and Interfaces are instanced as parameters of type `<usertype>`
  - Structures can be initialized by *leaf-node* order or by *element name*.
  - *Array types* can be given a size at creation time or by setting the array length using `<setparam>`.
  - After creation and setting up, these dynamically created parameters can be *bound to uniform inputs*, as usual, using `<bind>.`

The COLLADA FX Schema

The schema is designed to be *flexible and expressive* – with clear ways to declare and implement effects.

There are clear, *one-to-one mappings* between a validating XML document and a sequence of calls in a each shader runtime.



Physical Simulation File Formats

- Until now, no standard format existed for physical properties and physically simulated animation
  - Every 3D API has their *own file format*
  - Every 3D API has to provide their own *export plugin* for each DCC platform
- There is *no standard way* to interchange the same physics settings across different physics engines
- In many cases it is even impossible to author physical simulations without *direct programming*

COLLADA Physics

- An API independent description of basic rigid body and articulated models
  - Center of mass, initial velocity, moment of inertia, coefficient of friction, damping, coefficient of gravity, etc.
- Declaration of simplified geometric or convex collision bounds for rigid bodies
- Declare constrained joints and degrees-of-freedom and for compound models
- Scene partitioning with static, dynamic and nonparticipating geometry

How COLLADA Physics Works

- Properties are described as libraries of *physical materials* that can be applied to geometries

```
<library_physics_materials>

  <physics_material id="WoodPhysMtl">
     <technique_common>
        <dynamic_friction sid=“X”> 0.12 </dynamic_friction>
        <restitution> 0.05 </restitution>
        <static_friction sid=“Y”> 0.23 </static_friction>
     </technique_common>
  </physics_material>
  ...
</library_physics_materials>
```

How COLLADA Physics Works

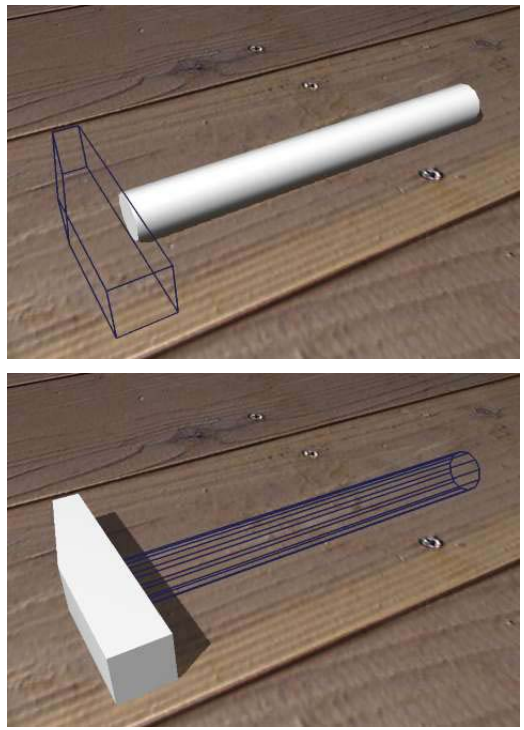
- Physics Models are described by rigid bodies and constraints

```
<library_physics_models>

  <physics_model id="Model1">

    <rigid_body sid="Hammer_Rigid_Body">
    </rigid_body>

    <rigid_constraint sid="Rigid_Hinge_Constraint">
    </rigid_constraint>

  </physics_model>

  <physics_model id="Model2">
    ...
  </physics_model>

</library_physics_models>
```

How COLLADA Physics works

- Rigid Bodies are created from analytical shapes or convex hulls, combined with physical materials

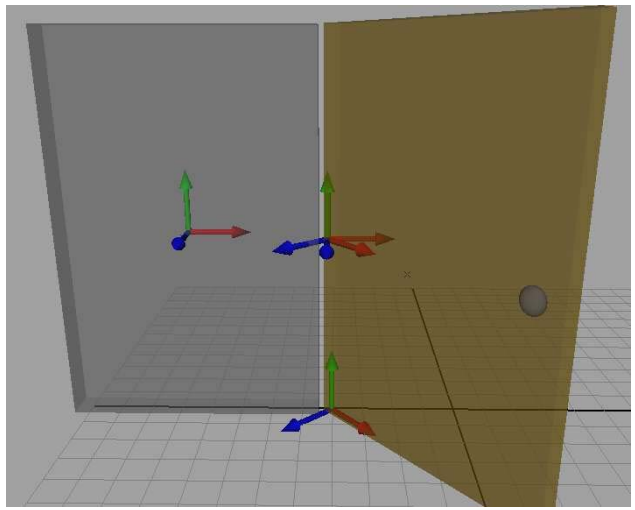
```
<rigid_body sid="Hammer_Rigid_Body">
  <technique_common>

    <mass> 1.25 </mass>

    <shape>
      <mass> 0.25 </mass>
      <instance_physics_material url="#WoodPhysMtl"/>
      <instance_geometry url="#hammerHandleForPhysics"/>
    </shape>

    <shape>
      <mass> 1.0 </mass>
      <instance_physics_material url="#SteelPhysMtl"/>
      <instance_geometry url="#hammerHeadForPhysics"/>
      <translate> 0.0 8.0 0.0 </translate>
    </shape>

  </technique_common>

</rigid_body>
```



How COLLADA Physics works

- *Joint constraints* can be applied to compound objects

```
<rigid_constraint id="rigidHingeConstraint">

    <ref_attachment rigid_body="#wallRigidBody">
      <translate sid="translate">5 0 0</translate>
    </ref_attachment>

    <attachment rigid_body="#doorRigidBody">
      <translate sid="translate">0 8 0</translate>
      <rotate sid="rotateX">0 1 0 -45.0</rotate>
    </attachment>

    <limits>
      <swing_cone_and_twist>
        <min>0 -90 0</min>
        <max>0 +90 0</max>
      </swing_cone_and_twist>
    </limits>

</rigid_constraint>
```



Conformance Framework

- COLLADA is a standard of The Khronos Group
  - As part of the standardization effort, Khronos is creating a *conformance test suite*
- In order to rely on a DCC plug-in, we need a lot of test cases to verify that the correct behaviour is observed.
  - For example, COLLADA exporters must obey *import and export rules*
    - If it's imported it must be exported, even if it's not recognized.
    - Unrecognised data does not get transformed or discarded.
- We want to help users and developers to have *reliable*, production ready exporters and importers for all DCC applications
- To reach this goal:
  - Khronos *Adopters* will use the conformance test suite to qualify their products against hundreds of test cases.
  - Passing the conformance test earns *Adopters* the use of the COLLADA logo on their products.
  - Customers can purchase tools with the COLLADA logo.

The COLLADA Conformance Suite

- Currently over *500 feature tests* and growing.

200 *Maya* tests	150 *XSI* tests
150 *3DS Max* tests	30 *XML parsing* tests

- The COLLADA Conformance Test is available to Khronos *Adopters* for a fee
  - To register as a Khronos *Adopter* visit http://www.khronos.org/members/join

COLLADA and The Khronos Group

- Khronos is an independent organization with a strong representation in the 3D world, home to OpenGL ES, OpenML and other standards

- COLLADA continues to develop under a strong Intellectual Property framework encompassing all Khronos member companies.
  - Sony Computer Entertainment is a Khronos *Promoter*, to more actively participate in the development of open standards.
  - Other Khronos members actively supporting COLLADA include:
    - NVIDIA, AVID/Softimage, Autodesk M&E, Feeling Software, Nokia, and Emdigo

Tools for Asset Wrangling

- Our ultimate goal is to provide a framework that allows users to reconfigure their own *production pipelines* using standard tools, with COLLADA as the interchange format between them

- Our toolset consists of:

    1. *The COLLADA Digital Asset Schema*
    2. *<asset> elements and Multi-Representation*
    3. *Standards based XML technologies*
    4. *Commercial tools for authoring, versioning, and database management*

Asset Management

- Asset elements are *meta-data* used to describe the history, meaning and uses of an asset.

```
<asset>
  <contributor>
    <author>Joe Schmoe</author>
    <authoring_tool>Maya Collada exporter v0.7.4</authoring_tool>
  </contributor>
  <created>2004-12-20T22:10:18Z</created>
  <keywords>Blink animation level_04</keywords>
  <modified>2004-12-20T22:10:18Z</modified>
  <revision>1.04</revision>
  <unit name="millimeter" meter="0.001"/>
  <up_axis>Y_UP</up_axis>
</asset>
```

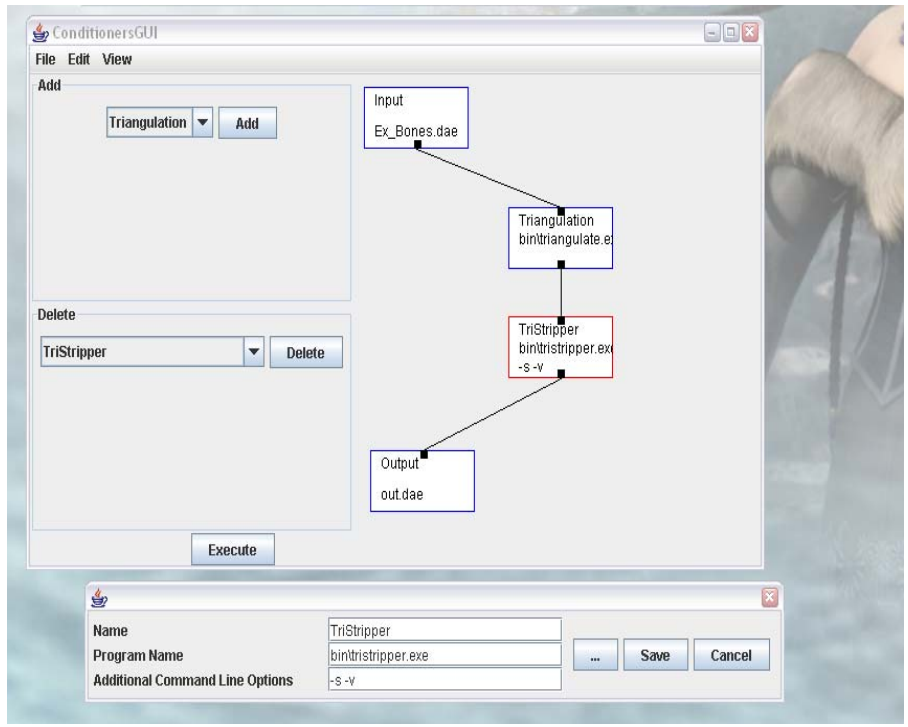
  - Most objects in a COLLADA document can have asset meta-data.
  - Assets can have *multiple representations*.
  - <asset> elements enable *tracking* of asset creation and ownership over time and across databases.

Towards a Makefile for 3D Assets

- Asset elements can make pipeline processing more efficient:
  - Parent-child relationships can be used to rebuild or re-export only those assets affected by an edit.
  - <asset> elements can be used to guide automatic generation scripts by recording command lines, parameters, tools and settings.
  - <asset> elements allow integration with Versioning Systems at a finer granularity, so assets no longer need to be fragmented into thousands of individual files.
  - Asset management is an active area of research, join the COLLADA working group!

Pipeline Editors

- Joining tools called “asset conditioners” together should be *simple*, like connecting nodes in a graph
- Designing your data pipeline out of *modular components* enables experimentation and optimisation
- Asset conditioners can be small efficient programs dedicated to a single task or algorithm



Java-based pipeline generator

Summary

COLLADA is an open standard for the interactive entertainment industry

- Academic Researchers, please investigate COLLADA!
  - Collaborative, many open source projects including:
    - COLLADA plug-in for Maya - http://sourceforge.net/projects/colladamaya
    - COLLADA DOM - http://sourceforge.net/projects/collada-dom
    - COLLADA plug-in for Blender - http://sourceforge.net/projects/colladablender
    - COLLADA plug-in for OpenSceneGraph - http://sourceforge.net/projects/osgdb-collada
  - Make tools and extend the standard.
- Read the *documentation* and visit the Khronos website at:

http://khronos.org/collada

- Join Khronos, join a working group and *help define the standard*.