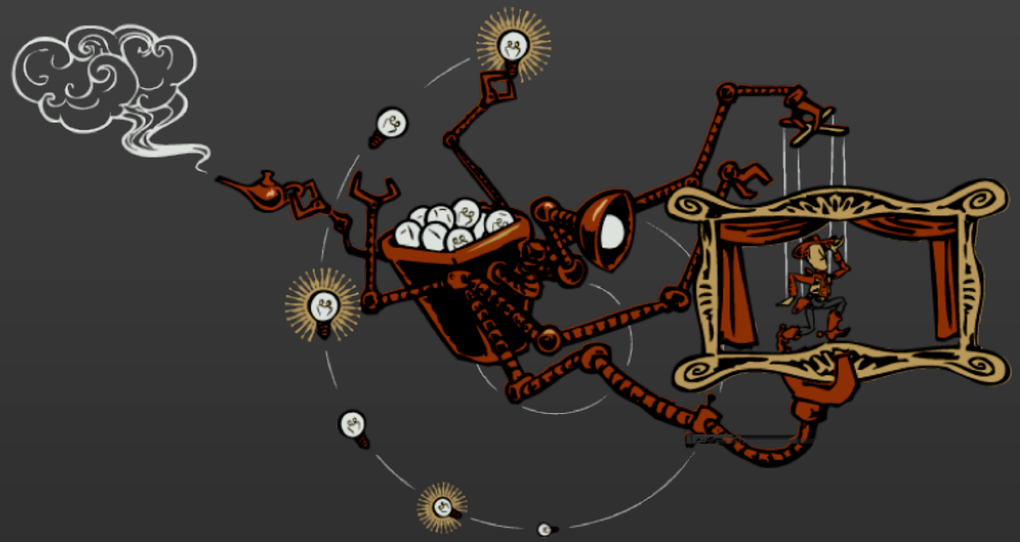
Universal Scene Description

Pixar Animation Studios - July 2013



Universal Scene Description is:

a unified system for representing both **primitives** and **aggregate assets** to enable **concurrent** CG workflows.

We are gauging interest to determine if we want to release USD and its associated IP as an OpenSource project

Interop in the Industry

- Interop between 3D apps is vital in our industry
- Standardization and open-source are key
- Alembic successfully provides this for geometry and materials
- We see the need for a higher level scene description standard

Universal Scene Description (USD)

- Builds on same concepts as, and integrates with, Alembic
- Adds multi-file assembly of assets
- Full composition engine: references with overrides, variants, classes
- Designed for multiple and concurrent department workflows

By Features

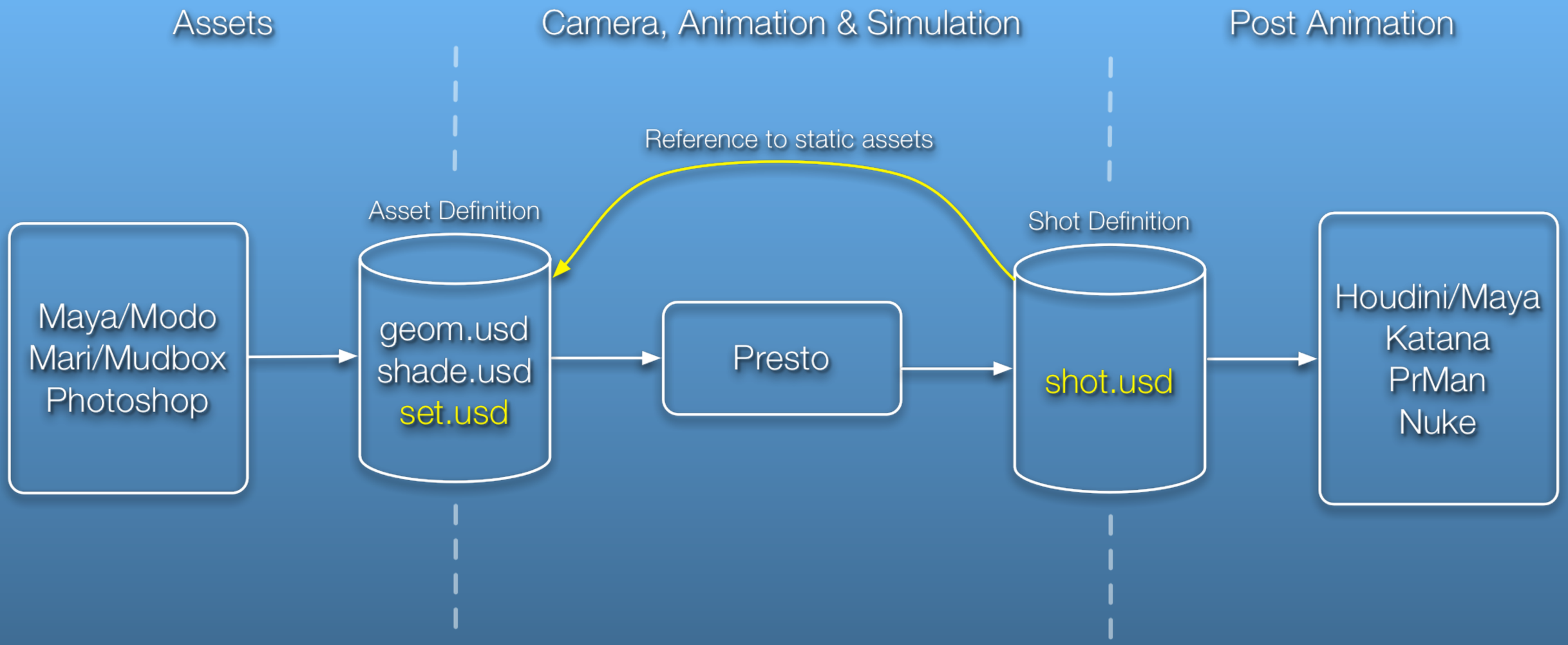
Alembic

- Geom and shading schemas
- High performance streaming
- Time-sampled caching
- Open, vendor supported standard

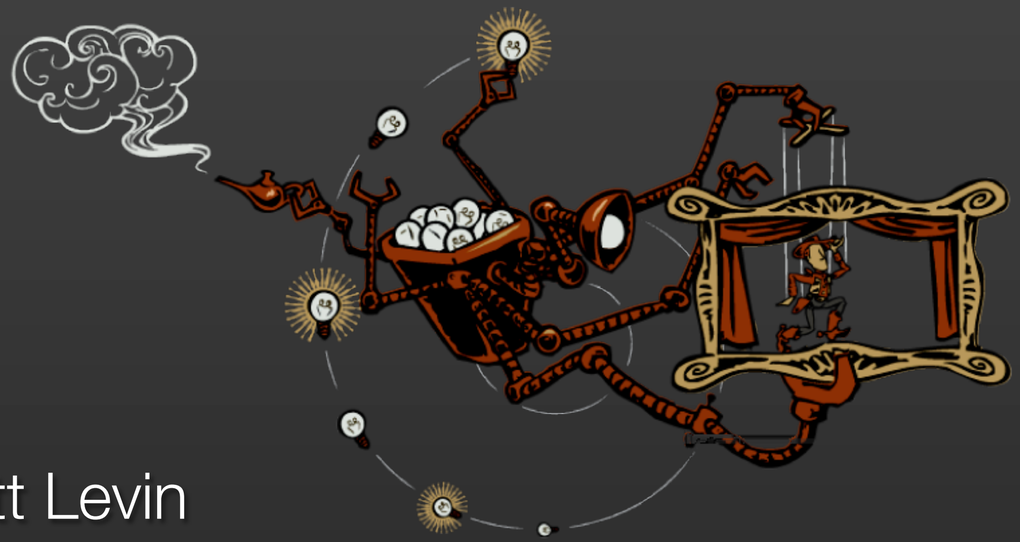
USD

- Geom and shading schemas
- High performance streaming
- Time-sampled caching
- Referencing, composition for scene assembly
- Scenes can be live, editable throughout the pipeline

USD based pipeline



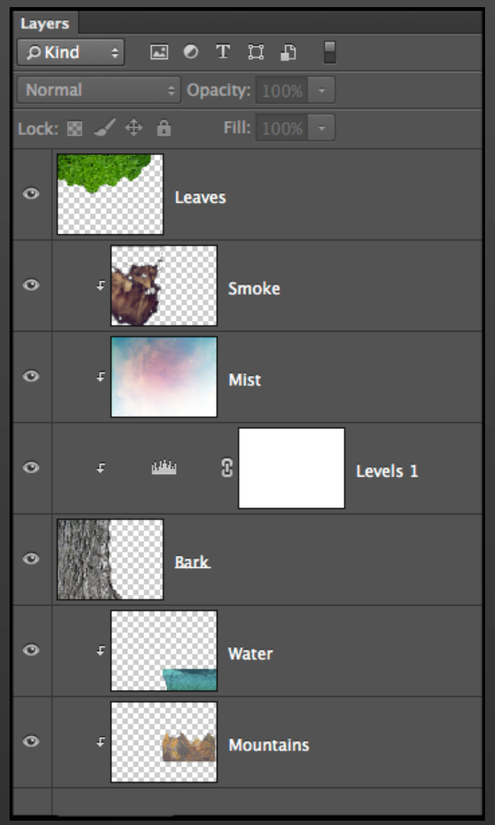
Composition Features in USD



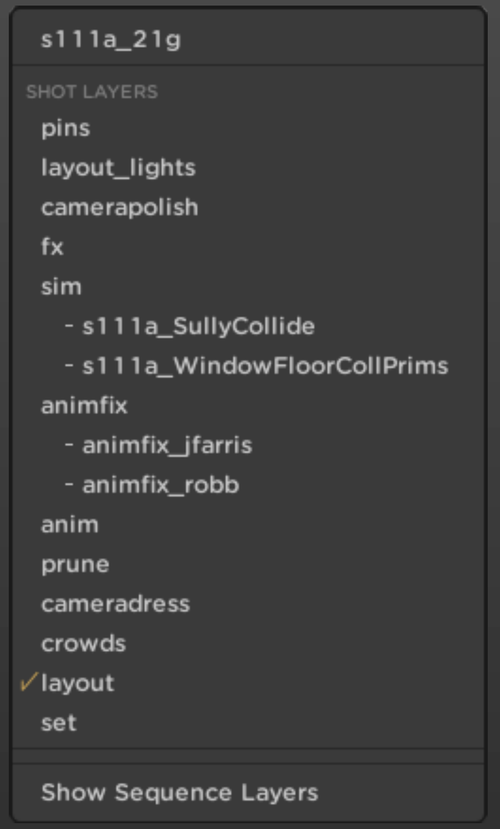
Brett Levin

Layers

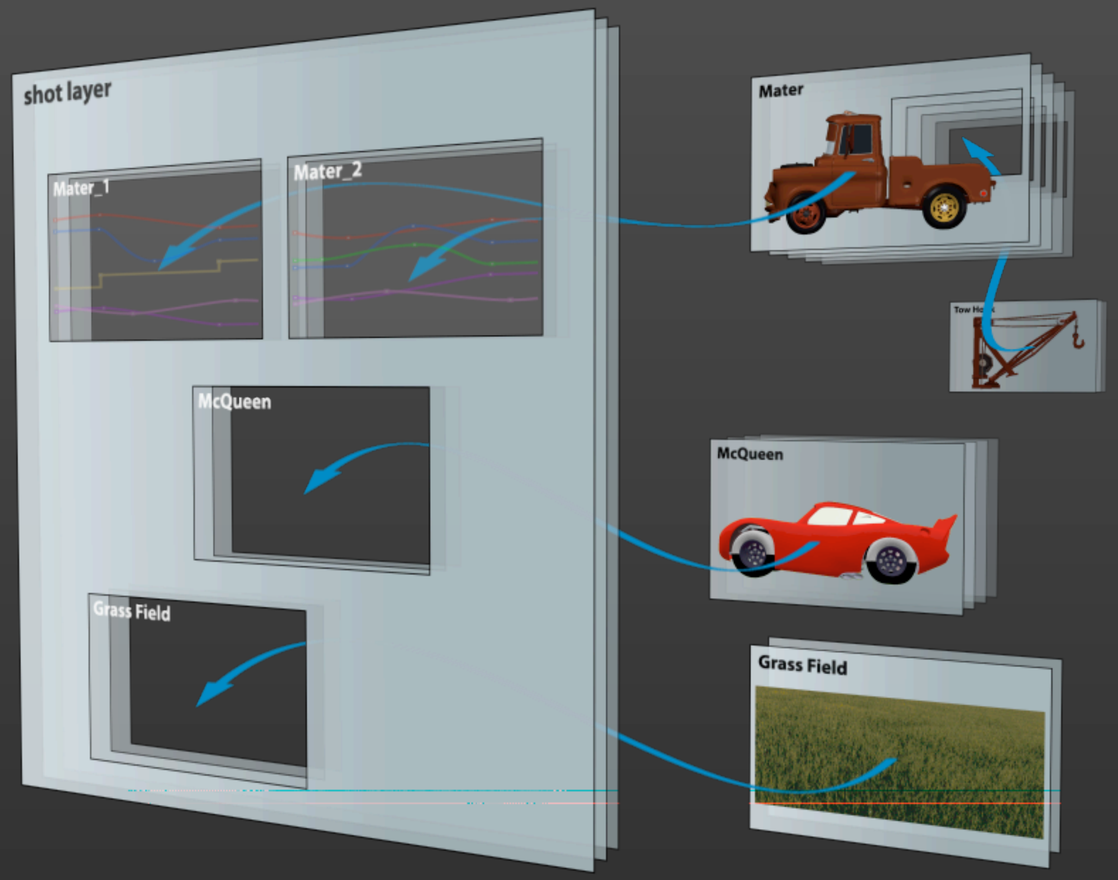
Photoshop



Presto + USD



References



USD Text Format

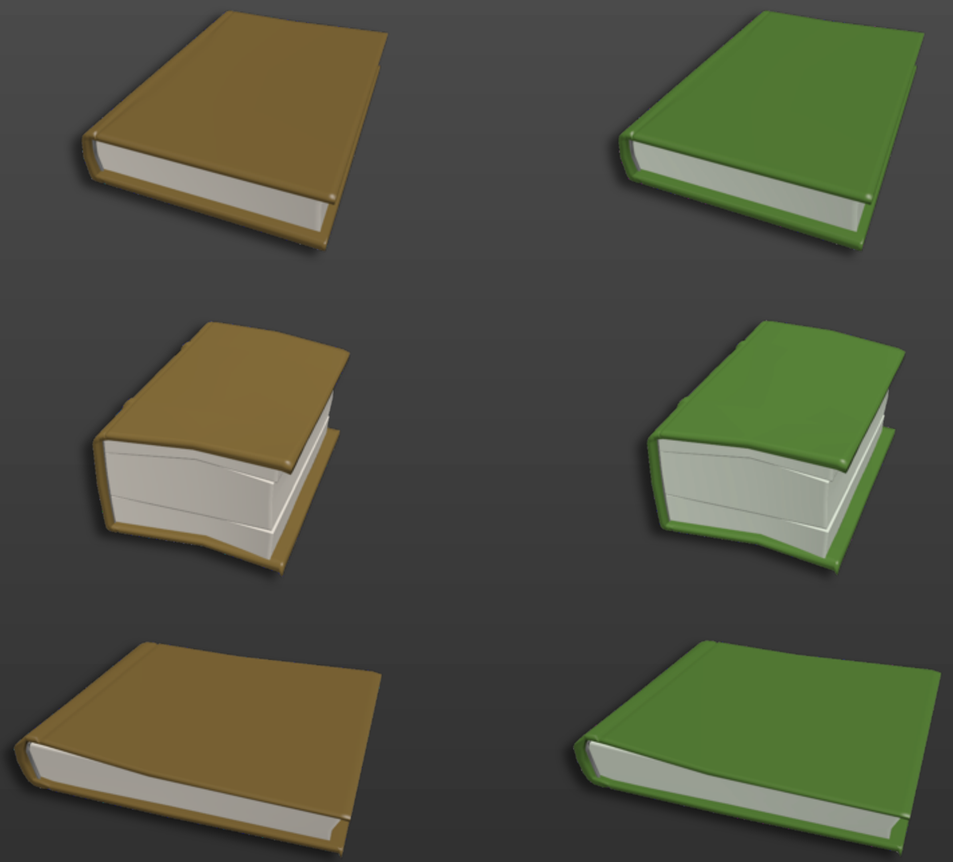
```
over “World”
{
    over “anim”
    {
        over “chars”
        {
            def “Mike”
            (
                add references = @chars/Mike.usd@</Mike>
            )
            {
                double LayTx = 5.770456
            }
        }
    }
}
```

Model Hierarchy

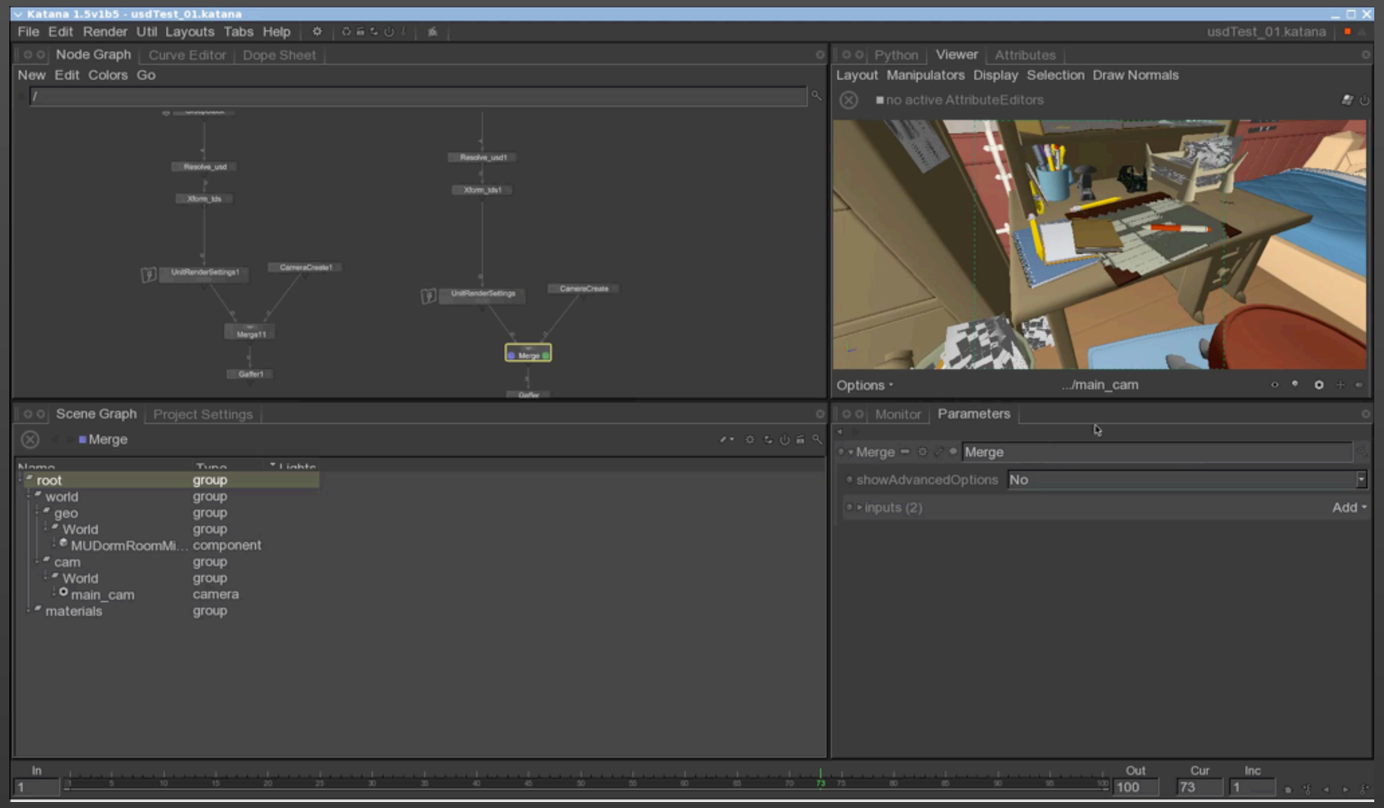
Variants

shadingVariant

modelingVariant

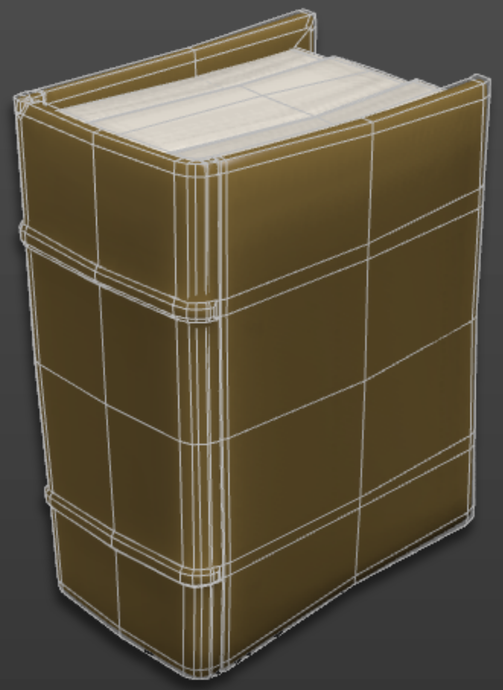


USD Variants in Katana

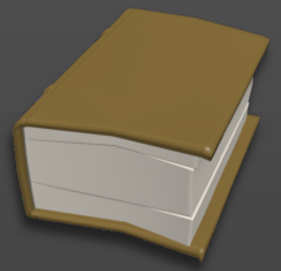


-- video available at: **graphics.pixar.com/usd** --

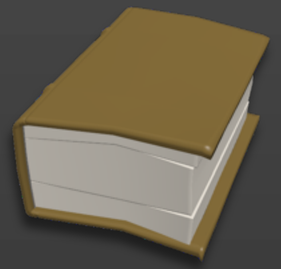
Classes



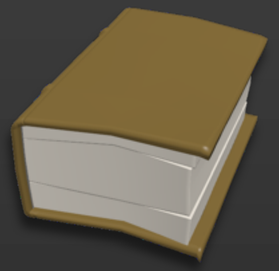
_class_BookCover



Book1

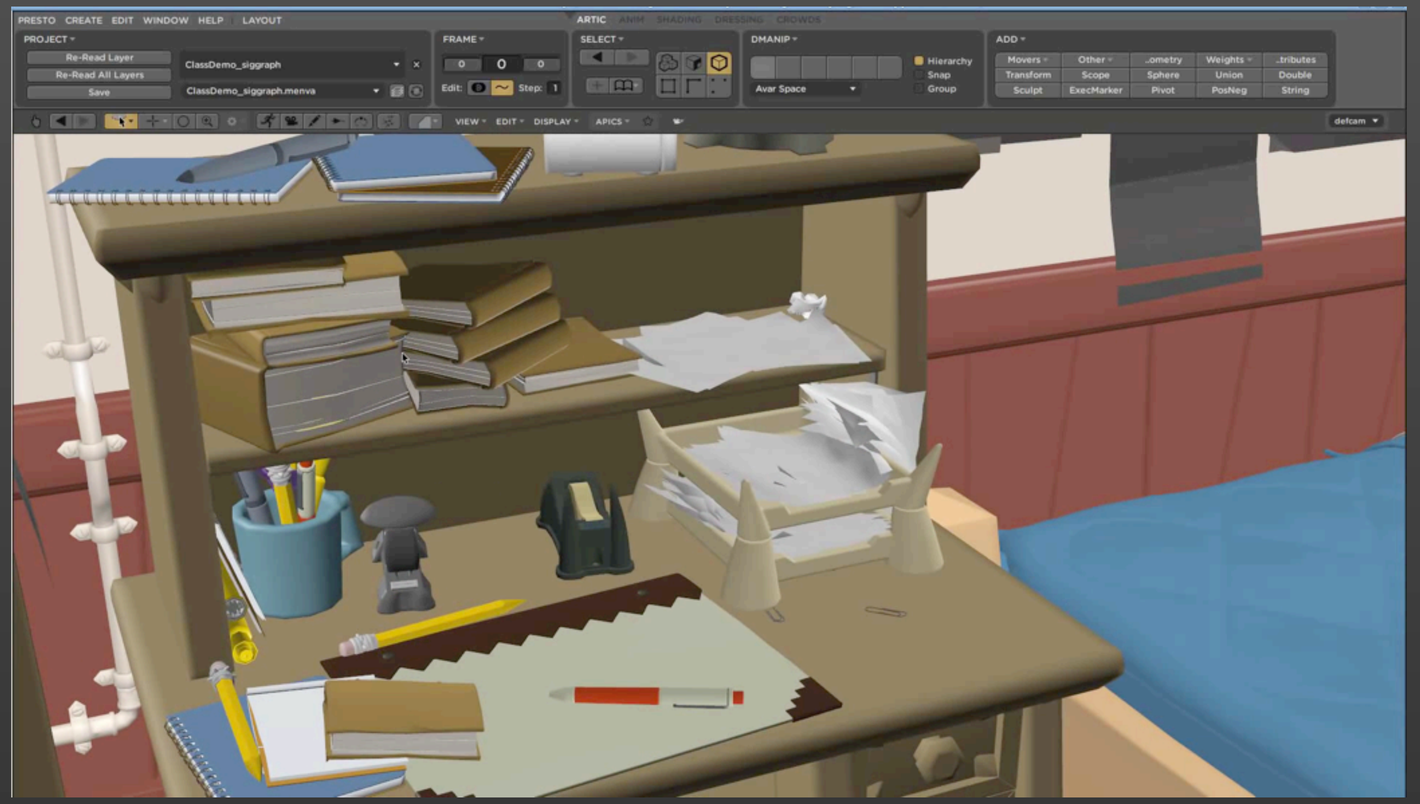


Book2



Book3

USD Classes in Presto

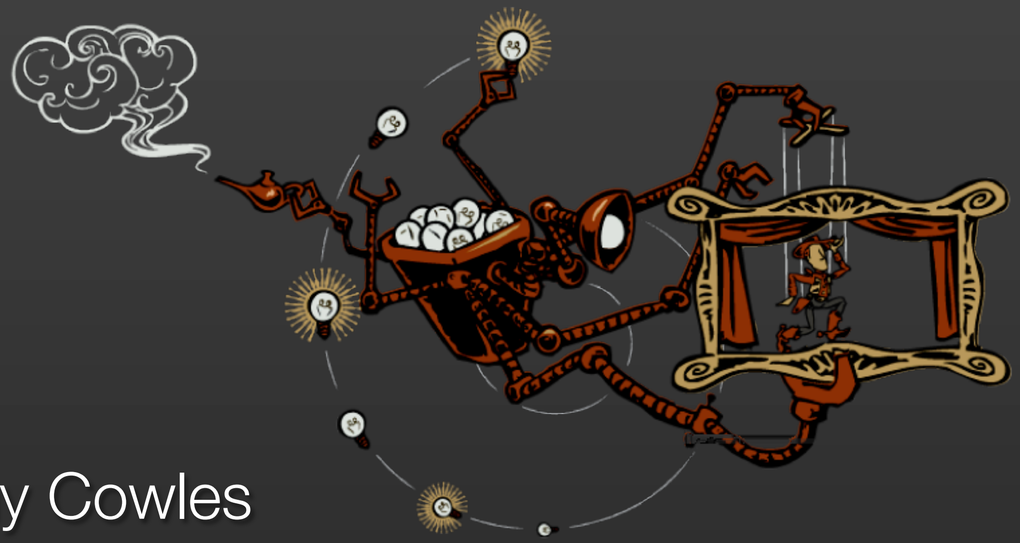


-- video available at: **graphics.pixar.com/usd** --

Summary

- Toolbox for teams to assemble assets from shared pieces, and work concurrently
- Capabilities are opt-in and orthogonal
- Included with USD

Scripting & Software Integration



Jeremy Cowles

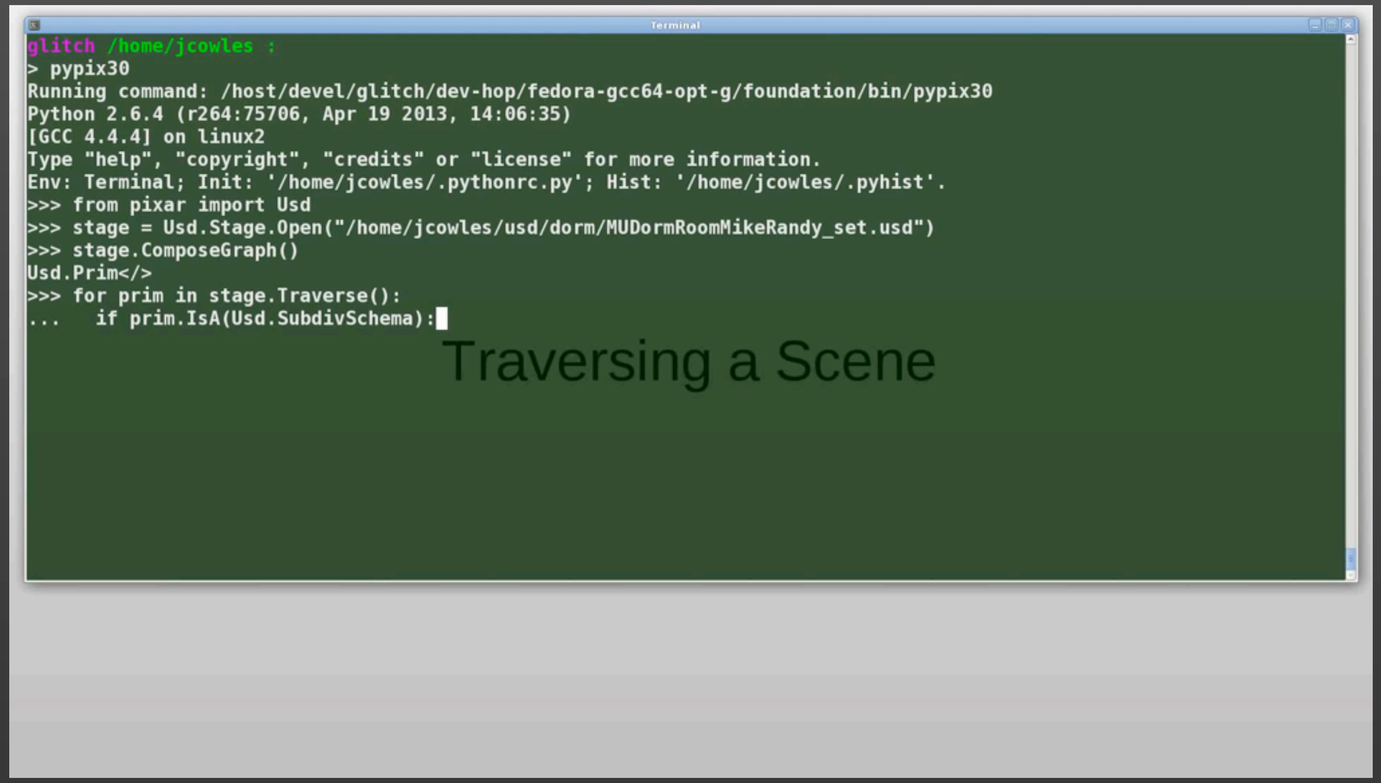
Target Users

- Integration & pipeline software engineers
- Production tech leads:
  - Crowds
  - Sets
  - FX
  - Sim

Performance is critical!

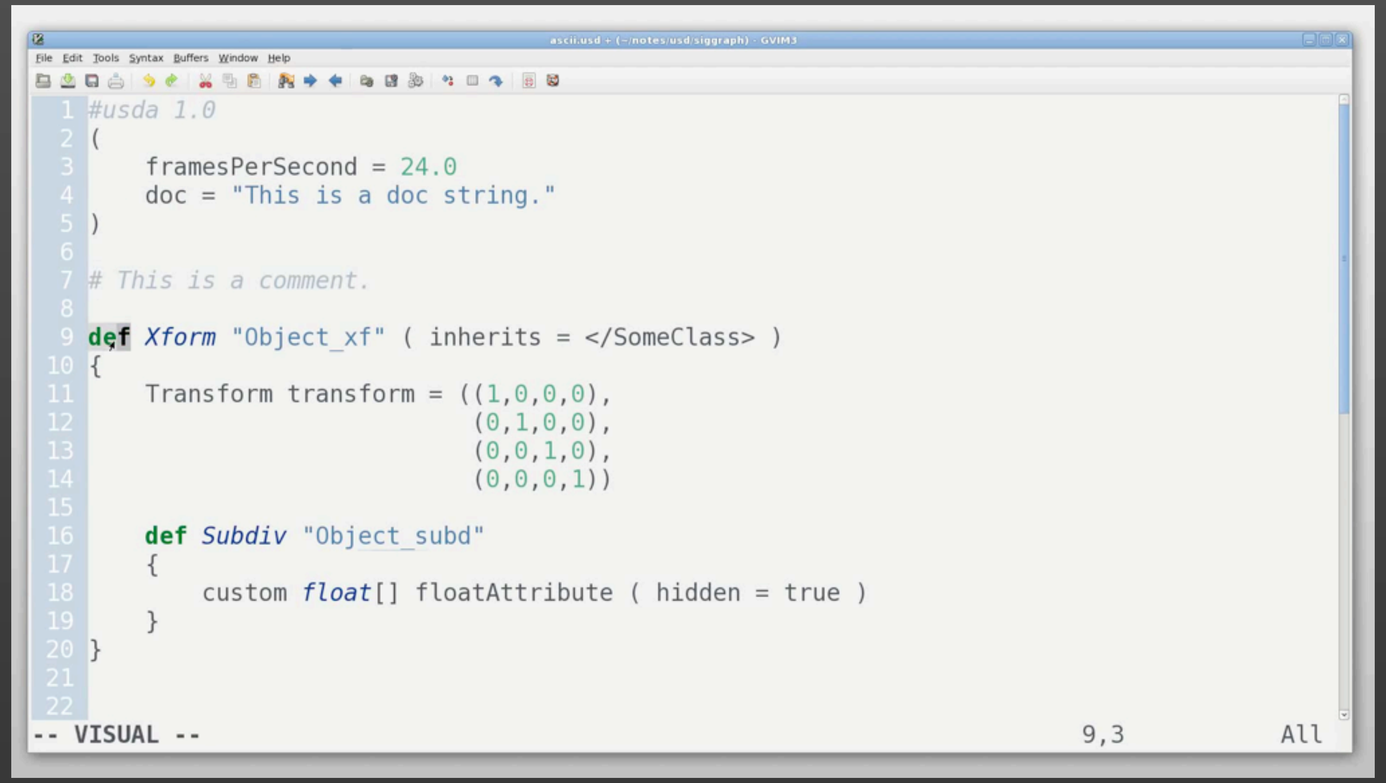
Common tasks must be **easy!**

Demo: Traversing a Scene



-- video available at: **graphics.pixar.com/usd** --

ASCII File Format

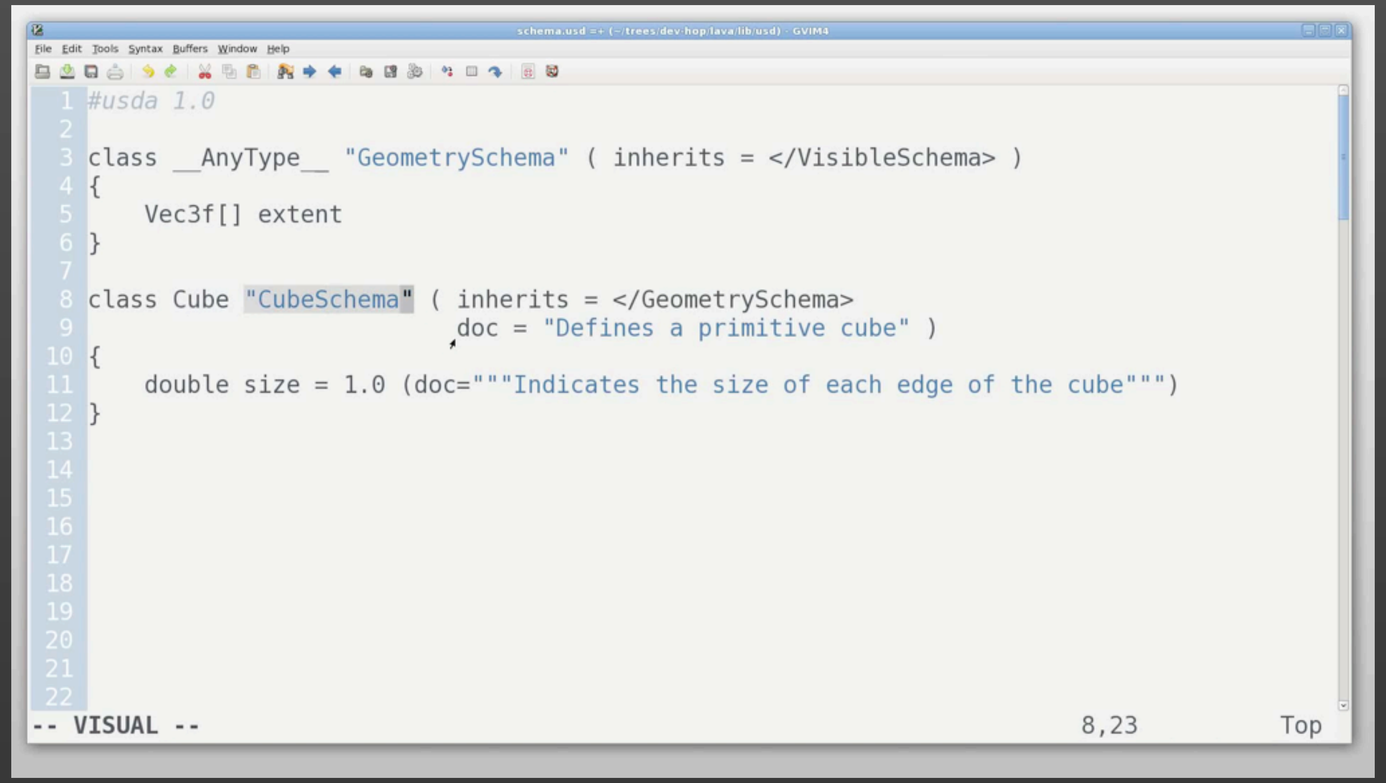


-- video available at: **graphics.pixar.com/usd** --

Core USD Schema

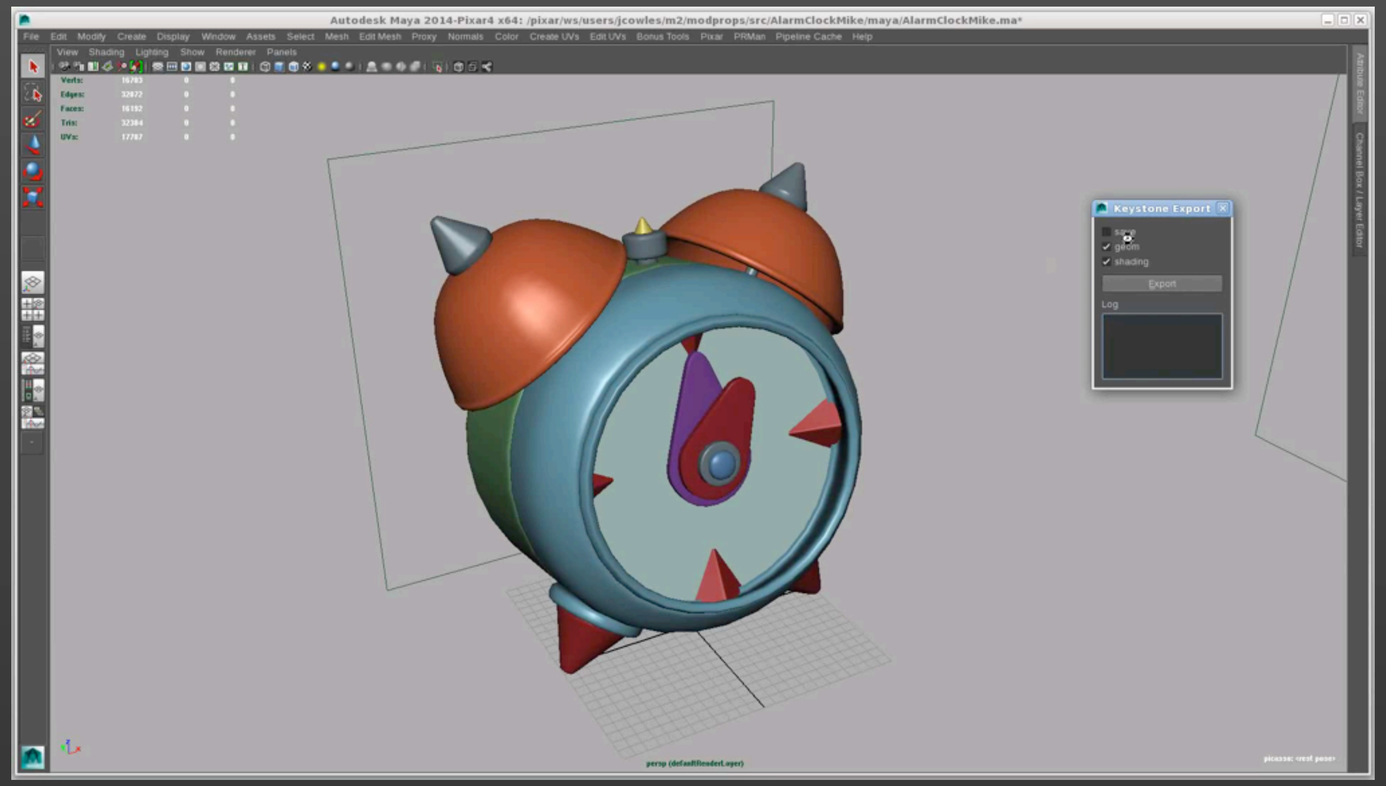
- Pull from Renderman, Alembic, Katana, also Presto
- **Geometry:** Xform, Subdiv, Curve, etc
- **Shaders:** Shader, bindings, params
- **Models:** Model hierarchy, model kind such as prop, set, etc.
- **Custom:** create new schema or extend existing definitions

Schema Definition



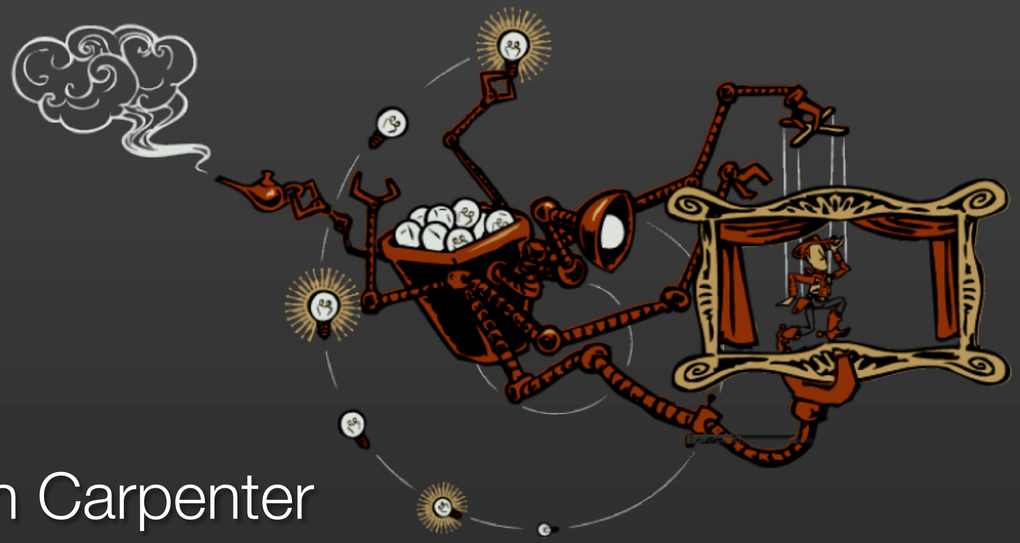
-- video available at: **graphics.pixar.com/usd** --

Single Asset Structure

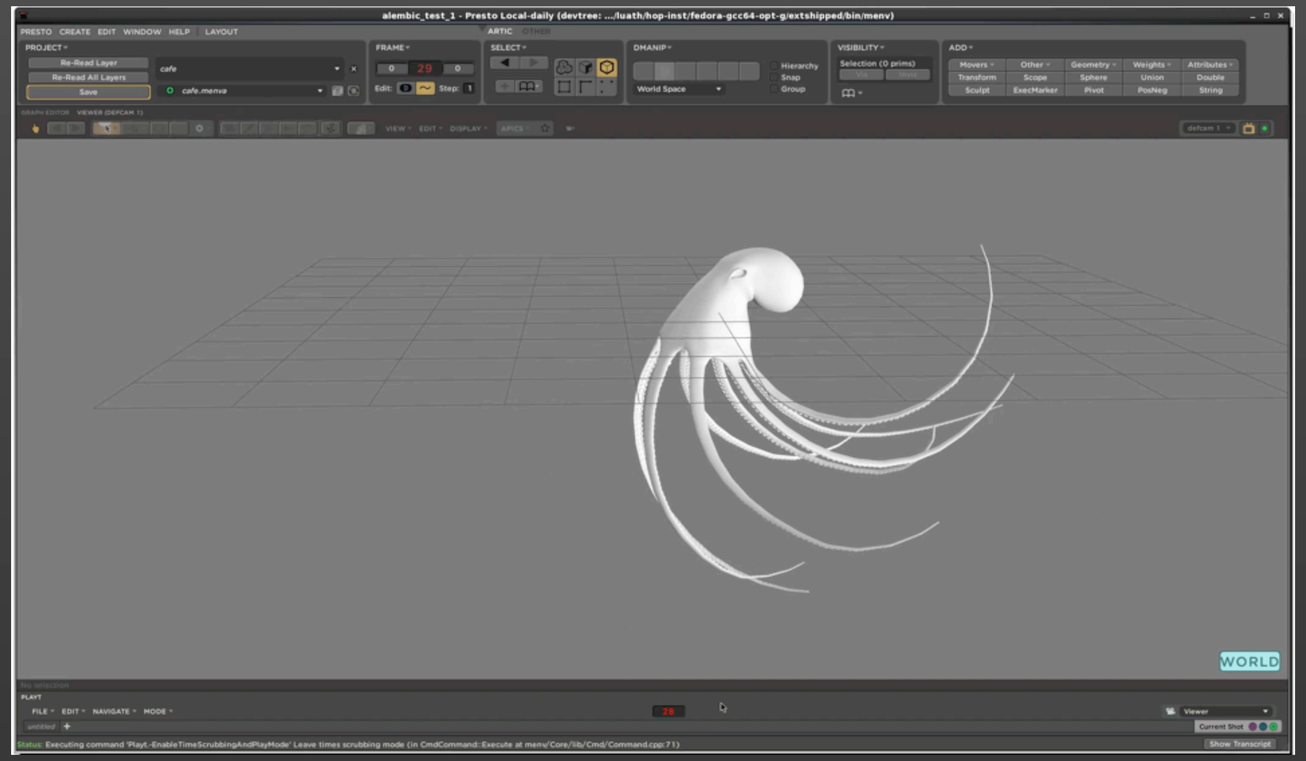


-- video available at: **graphics.pixar.com/usd** --

Alembic USD Integration

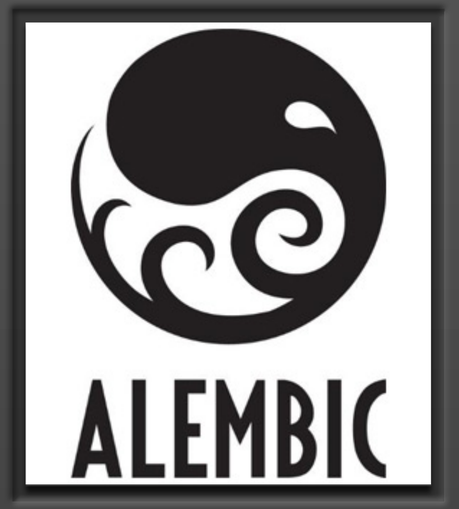


Jeremy Cowles, Loren Carpenter



-- video available at: **graphics.pixar.com/usd** --

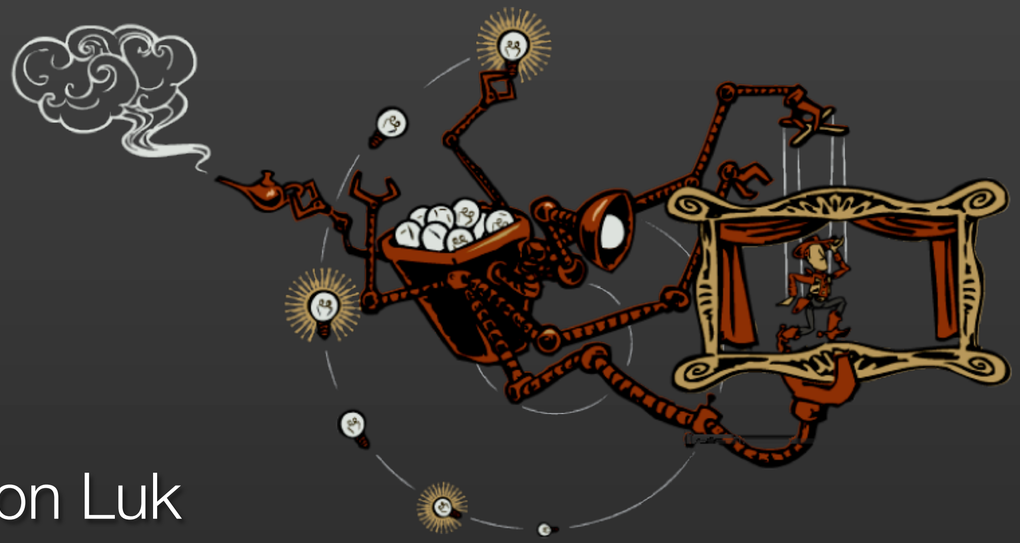
Key Ideas



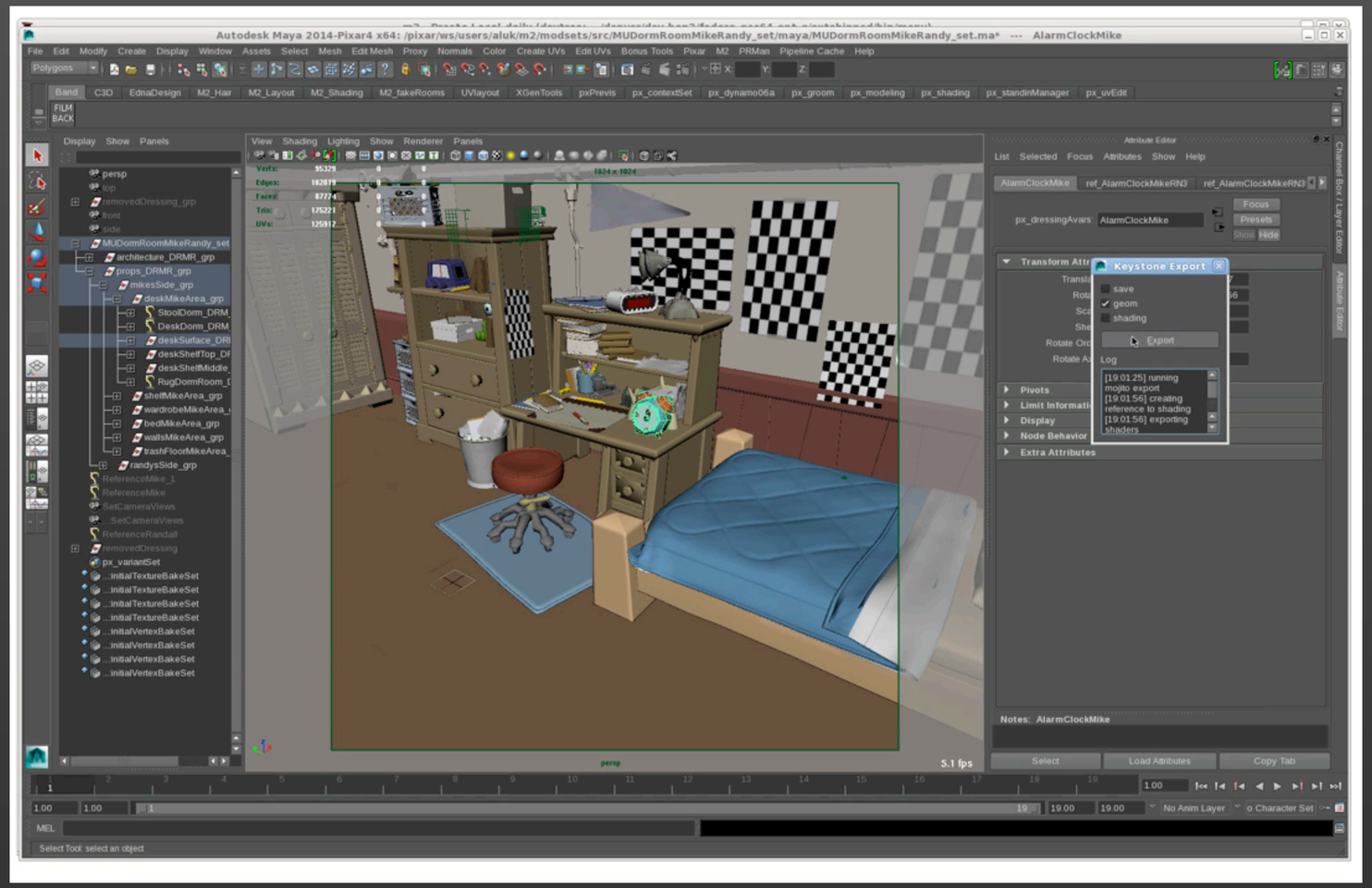
- Leverage Alembic archives as they are produced today
- Lazy loading of Alembic data is preserved
- Once in USD, all composition rules apply

Workflow Demos

USD Throughout the Pipeline at Pixar



Aaron Luk



-- video available at: **graphics.pixar.com/usd** --

Export of Animation Overrides

-- video available at: **graphics.pixar.com/usd** --

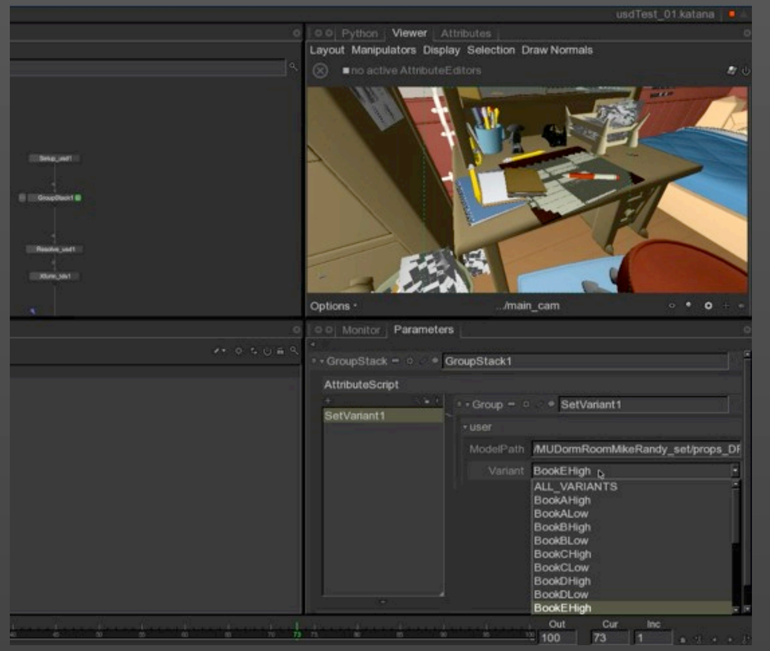
Structure of a Referenced Pose-Cache

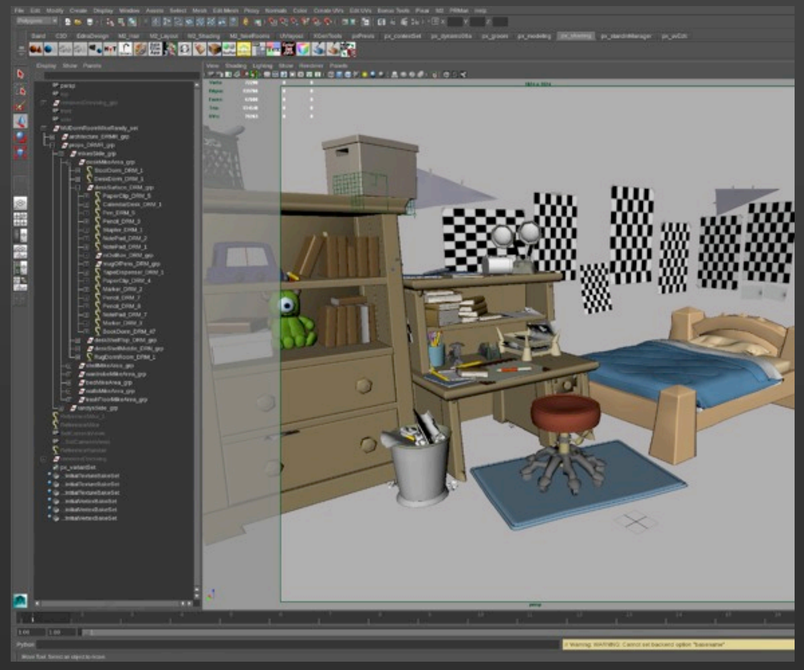
```
#usda 1.0

def Xform “World”
{
    def Xform “anim” (kind = “group”)
    {
        def Xform “chars” (kind = “group”)
        {
            def “Mike” (
              kind = “char”
              add references = [ @chars/Mike.usd@</Mike> ]
            )
            {
                Transform transform.timeSamples = {
                  19: ((-0.75, 0.65, 0, 0), (-0.64, -0.75, -0.12, 0), (-0.08, -0.09, 0.99, 0), (-381.7, -252.3, 337.6, 1)),
                  20: ((-0.75, 0.65, 0, 0), (-0.64, -0.74, -0.12, 0), (-0.08, -0.1, 1, 0), (-381.8, -252.3, 337.6, 1) )
                }
                over “Geom”
                {
                  over “Body”
                  {
                    PointFloat[] points.timeSamples = {
                      19: [(64.8254, -37.7543, 90.7112), (64.9756, -37.8067, 89.8514), ... ]
                      20: [(64.8327, -37.7363, 90.739), (64.9843, -37.788, 89.8794), ... ]
                    }
                  }
```


Proxy Drawing

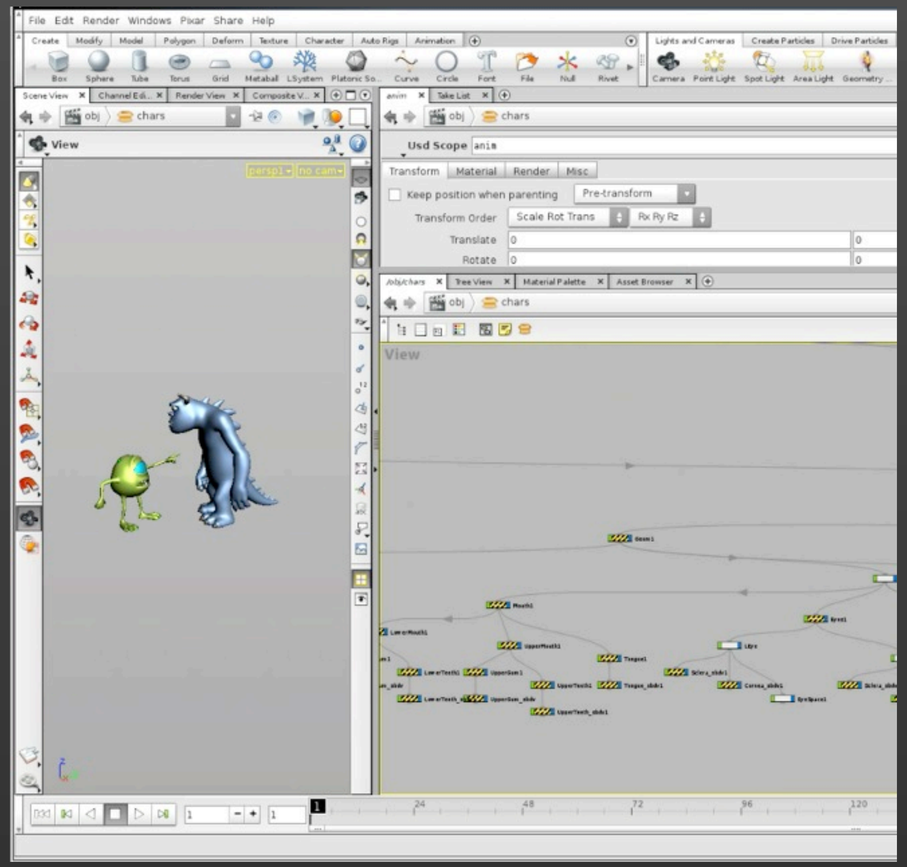
- All departments use proxy renderer as “gpu-cache” direct from USD in...
- Proxy drawing also used for crowd construction and preview





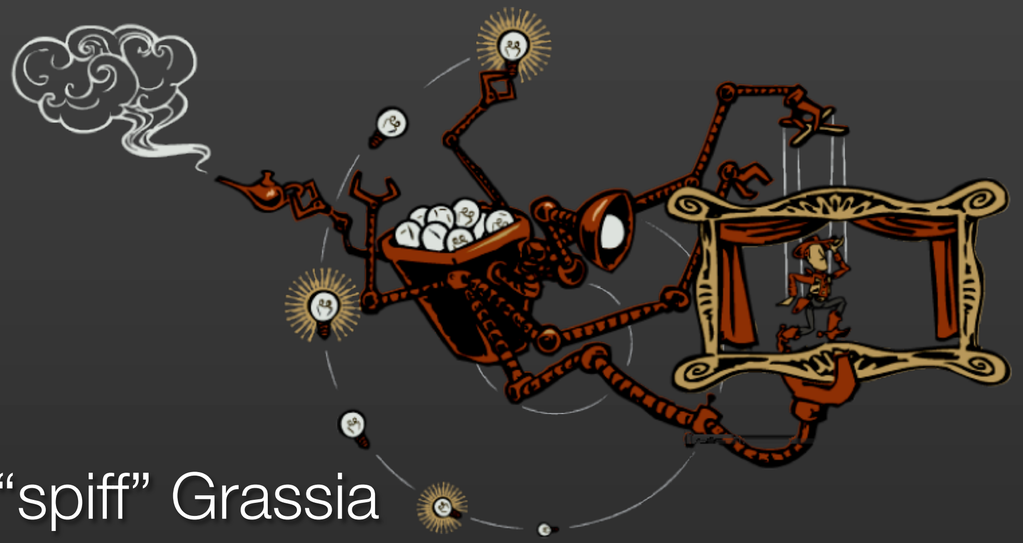
FX

- Primarily Houdini, import USD
- Several types of USD export:
  - Override/redefine imported geo
  - Generate new asset for shot
  - Single frame of large sim



Software Architecture

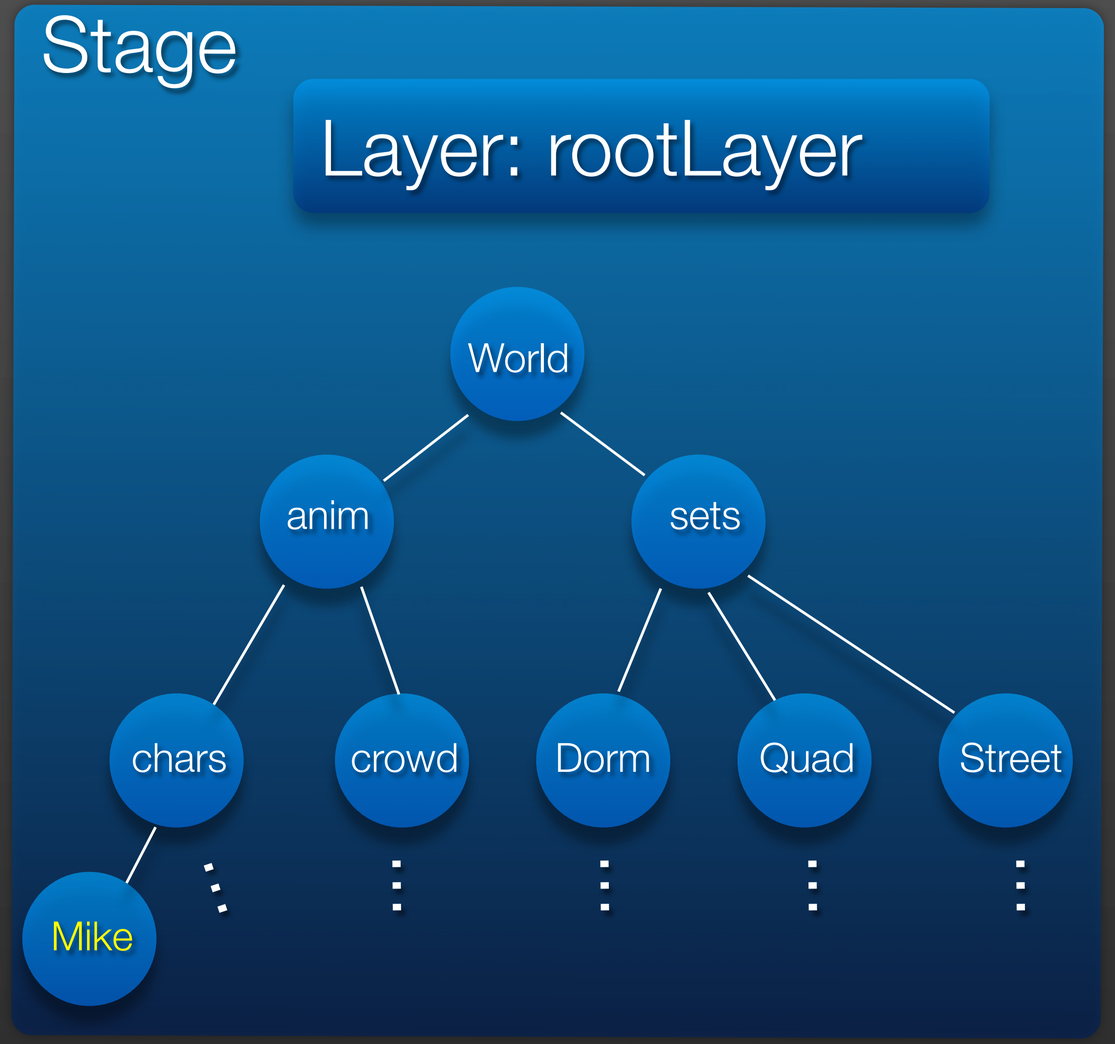
Object Model, Stack, and Key Elements



Sebastian “spiff” Grassia

USD Object Model

- Scene presented on **Stage**
- Stage owns root **Layer**
- Stage populated with **Prims**
  - forms a scene-graph



USD Scenegraph

Composition is **uniform** across Layers

- Stage topology is computed *composition* result
  - Scenegraph caches topology for performance
  - Manages *internal* Layer/file and composition caches
  - Provides *external* notification of changes to clients
- Clean, uniform API for I/O
  - Hides details of file referencing (unless you care)

Prims

- **Prims** are typed, scoped, namespace containers
- Prims can contain:
  - Prim children
  - Meta-data
  - Attributes
  - Relationships

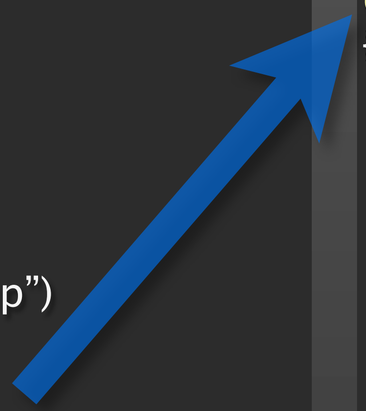
```
def Xform “Mike” ( kind = “char” )
{
    def Xform “Geom”
    {
      def Subdiv “Body”
      {
            PointFloat[] points = [ ... ] (
              detail = “vertex” )
            rel surface = </Mike/Shaders/Skin_Surface> (
              type = “binding” )
        }
    }

    def Scope “Shaders”
    {
        def Shader “Skin_Surface”
        {
        }
    }
...
}
```

Relationships and Path Translation

```
#usda 1.0

def Xform “World”
{
    def Xform “anim” (kind = “group”)
    {
        def Xform “chars” (kind = “group”)
        {
            def “DiscoMike” (
              kind = “char”
              add references = [ @chars/
              Mike.usd@</Mike> ]
            )
            { ... }
...
}
```



```
def Xform “Mike” ( kind = “char” )
{
    def Xform “Geom”
    {
      def Subdiv “Body”
      {
            PointFloat[] points = [ ... ] (
              detail = “vertex” )
            rel surface = </Mike/Shaders/Skin_Surface> (
              type = “binding” )
        }
    }

    def Scope “Shaders”
    {
        def Shader “Skin_Surface”
        {
        }
    }
...
}
```

surface.GetTarget() : </World/anim/chars/DiscoMike/Shaders/Skin_Surface>

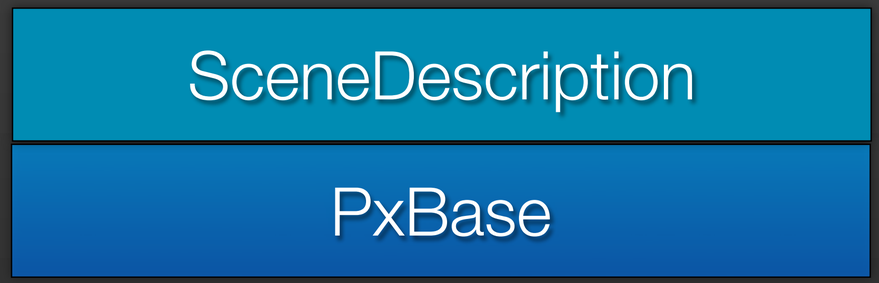
USD Software Stack

- **Pixar Base** provides:
  - Perf/Memory tracking
  - Enhanced containers
  - Type management system
  - imath-like vec/matrix pkg

PxBase

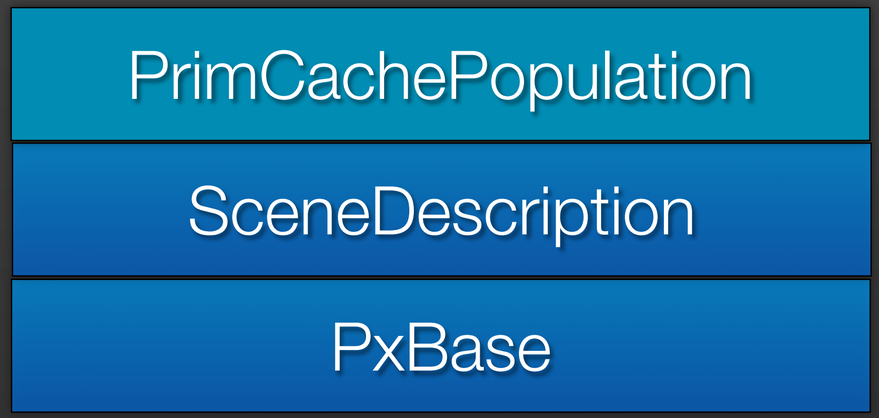
USD Software Stack

- “**Sd**” provides:
  - Core data model
  - Layer/file abstraction
  - File Format plugin
  - “Asset Resolution” plugin
  - Ascii file format



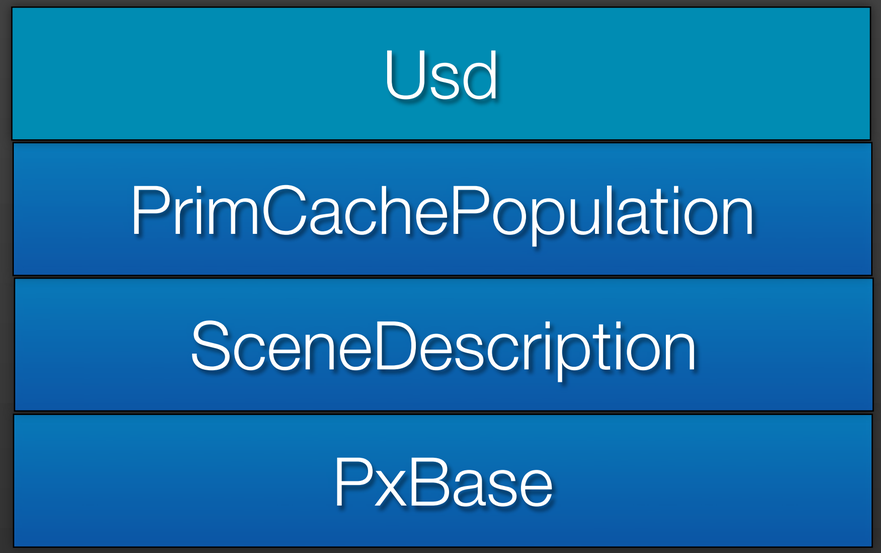
USD Software Stack

- “**Pcp**” provides:
  - Composition rules
  - Computes an *Index*, per-prim
  - An **Index** is a roadmap for which layers might contribute opinions to each attribute



USD Software Stack

- **Usd** provides:
  - Primary client API's for I/O
  - *Stage* scenegraph
  - Geom and shading schemas
  - File-Format plugins (BDB, abc)
  - Iterator, visitor for scenegraph
  - *Live* update, with client notices



USD Binary: To BDB and Beyond!

- USD will always provide a native, optimized (for Sd), binary encoding
- Currently built on Open Berkeley Database (BDB)
  - High performance
  - Files are editable
  - Possible licensing issues
- Investigating alternatives, including Ogawa

Alembic Integration

- Available as a USD format plugin (IP)
- Compatible with existing archives
- Does not support references or other composition features

Format Plugins

- Not just about abc, caf, obj integration...
- E.g.: can reference .slo's directly to create interface
  - Plugin extracts xml-schema from slo
  - Creates scene description for shader prims, properties
  - Overrides in native USD layers compose seamlessly

Building Blocks

- Leverage Pixar code base
- External Dependencies:
  - Boost
  - numpy
  - Berkeley Database (for now)
  - Facebook’s jemalloc (preferred, not required)

USD Is Editable

- Not any particular file format, as much as...
- *Scenegraph/Stage* must be mutable
  - Enables non-destructive editing while seeing results live
  - Enables multiple departments to work on same scene in parallel
  - “Memory only” layers enable use as intra-process interchange

Performance

- Has scaled to Pixar's needs for three films
- Target scenes of millions of prims, thousands of files
- Will leverage multi-cores for graph construction and data streaming

Performance Test

- Blue Umbrella CitySet.usd
  - 113 unique assets
  - 340 unique files
  - 5807 instanced models
  - 490,000 composed prims

Performance Test

- Single-threaded results
  - **1.0 s** : compose model hierarchy
    - time to first bucket
    - ~ time to browse CitySet in katana
  - **14.5 s** : compose entire set
  - **8.2 s** : compute bounds
  - **429** MB memory

Next Steps

- Deploy in Pixar's pipeline, EOY
- Multi-threaded Stage population
- Sparse-export standardization
- Proxy drawing using **OpenSubdiv Batching**

Universal Scene Description is:

a unified system for representing both **primitives** and **aggregate assets** to enable **concurrent** CG workflows.

We are gauging interest to determine if we want to release USD and its associated IP as an OpenSource project

Where Credit is Due

- Dozens of engineers and artists over 20 years refined these concepts and designs
- Thanks to Pixar leadership for encouraging us to share, and making it possible
- Special thanks to Davide Pesare, Victor Mateevitsi, and Loren Carpenter for fearlessly jumping in to help with this
- Thanks to our friends at ILM and DFA for valuable early feedback

Thank you for coming!

Questions?