Engine-eering a Procedural Pipeline with USD

Michael Rice
michaelr@pixar.com
Pixar Animation Studios

Joshua Jenny
joshj@pixar.com
Pixar Animation Studios

Will Harrower
wharrower@pixar.com
Pixar Animation Studios

ABSTRACT

Typical cache-based non-procedural methods can have downsides within a production studio. Modifications to pre-cached data can introduce an expensive feedback loop between departments (e.g. Layout pushing work back to FX). Proceduralism solves this by allowing artists themselves to make their own modifications, without need for a new cache delivery from another department. Another benefit to procedural generation of data is reduced disk space usage particularly for heavy datasets like volumes. In this case, they no longer need to be cached to disk, but instead created from a procedural 'recipe' on-demand.

Historically, there was a high bar to introducing proceduralism into various parts of our pipeline, requiring custom engineering efforts on a case-by-case basis. Now, we leverage Houdini Engine (HE) [SideFX 2023], a toolkit from SideFX, together with USD [Pixar 2023], to provide a general procedural framework. By removing the engineering bottlenecks of creating procedurals, we can more easily allow artists to create their own, allowing them to iterate more quickly on design within a creative context.

CCS CONCEPTS

• **Computing methodologies → Graphics systems and interfaces**.

ACM Reference Format:
Michael Rice, Joshua Jenny, and Will Harrower. 2023. Engine-eering a Procedural Pipeline with USD. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '23 Talks), August 06–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3587421.3595442

1 PROCEDURAL HOUDINI MODELS

Houdini Digital Assets (HDAs) are represented in USD using a lightweight prim schema: `PxHoudiniEngine`. The schema has a handful of fundamental attributes, including a `SdfAssetPath` to the HDA file on disk, input relationships (for input geometry and cameras), and attributes representing HDA parameter values which will be evaluated when the HDA is cooked.

We define the concept of an "FX Model" as a USD prim hierarchy containing one or more `PxHoudiniEngine` USD primitives. Additionally, this FX Model can include proxy meshes, cached (i.e. non-procedural) data under variants, materials and their bindings (see Figure 2).

SIGGRAPH '23 Talks, August 06–10, 2023, Los Angeles, CA, USA

ACM ISBN 979-8-4007-0143-6/23/08.
https://doi.org/10.1145/3587421.3595442

Figure 1: ©Disney/Pixar. A Houdini Engine and USD pipeline was used to create procedural volumetric characters in Pixar's *Elemental.*

The USD representation of these models, along with the HDAs that they reference, is consistent with non-HE models within the studio and can be published and versioned within the studio's asset management system. Models can be "pinned" to a specific version or left unpinned to automatically pick up the latest.

Since an HDA can produce multiple outputs (for example, multiple meshes, volumes, etc), artists will often want the ability to control both the material bindings, and the visibility of those subcomponents. To this end, a separate USD prim schema – `PxHoudiniEngineMaterialTarget` – represents sub-components of the HDA, which can then have materials bound or visibility set.

We utilize an `SdfFileFormat` plugin for the HDA file path (via a `payload`) in order to dynamically generate these material target prims as children of the `PxHoudiniEngine` prim at USD composition time.

2 MODEL CATALOG

Once created, FX Models are registered in a model catalog, which artists can browse and load models from as cached proxies or as interactive procedurals. HDAs can query which host app they are running in and vary their behavior accordingly (e.g. lower resolution for interactive display).

FX Model instances can be dressed as stand-alone pieces around a set or embedded as parts of other models. Set-dressing artists on Pixar's *Elemental* embedded fire models into buildings and instanced smoke plume models across a city-scape. Variation was achieved by adjusting HDA parameters for plume height, radius, etc., giving artists a higher level of control over look and shape than a standard affine transformation on a non-procedural model.

3 SHOT-CONVERSION

Any FX Models which have been added to a Presto scenegraph will participate in the studio's "shot-conversion" process, where Presto's internal `menva` format is converted to USD for consumption by other departments and for rendering. The resulting USD will include a

Figure 2: An example of a USD FX Model

```
def Xform "FxFire" (
    variants = { string modelVariant = "procedural" }
    append variantSets = "modelVariant"
) {
    def Scope "Looks" {
        def Material "FireVolumeMaterial" { }
    }
    variantSet "modelVariant" = {
        "cached" {
            def Xform "Geom" {
                def Xform "CachedFire" (
                    prepend references = @FxFire.cached.usd@
                ) { }
            }
        }
        "procedural" {
            def Xform "Geom" {
                def PxHoudiniEngine "ProceduralFire" (
                    hdaParams = { token _payloadmode = "materialtargets" }
                    payload = @houdiniengine:ProceduralFire.hda@
                ) {
                    custom string parameters:flameType = "GroundFire"
                    custom double parameters:smokeDensity = 5.2

                    over "VOL_flames" {
                        rel material:binding = </FxFire/Looks/FireVolumeMaterial>
                    }

                    def Mesh "proxy" { ... }
                }
            }
        }
    }
}
```

"references" layer, containing a reference to the original FX Model for each corresponding location in the Presto scenegraph. An additional layer will record any variant selections or HDA parameter changes made in Presto as USD `over` opinions. The layer will also translate input geometry and camera relationships in Presto to USD Relationships which target analogous locations in USD.

3.1 PxHoudiniEngine Visualization

It can often be useful to visualize the output of procedural FX Models together with the rest of a USD scene, for which we use Pixar's open-source `usdview` application. Our initial approach was to leverage a `SdfFileFormat` plugin to run HE and create child USD primitives. Because the plugin ultimately produces USD primitives, this approach provides a straightforward solution for "baking" procedural FX Models to non-procedural USD. We've abandoned this approach in favor of a Hydra `HdGpGenerativeProcedural` plugin [Cauchois and LaVietes 2022]. Running HE in Hydra provides a more performant and seamless user experience, and we favor a more flexible system for baking procedural data than our C++ `SdfFileFormat` plugin affords.

4 HOUDINI WORKFLOWS

Departments with Houdini workflows (e.g. Crowds, FX, Sim) may need to work with procedural FX Models in Houdini, whether for adjustment or replacement. We provide two workflows to support this. In one workflow, Houdini users can apply USD overrides to the model's `PxHoudiniEngine` primitives, whether for parameter changes or to override the HDA path itself. Another workflow allows users to "capture" the HDA and its parameter state from USD to their interactive Houdini session, allowing them to modify or replace the procedural output and write that back to USD as non-procedural data.

5 LIGHTING AND RENDERING

For lighting and rendering at Pixar, Katana and RenderMan are used. We provide a HE node in Katana for viewing and overriding HDA parameter values. The Katana node does not cook HE, instead, a RenderMan procedural is used to evaluate HE at render time, with the procedural directly producing a RenderMan Rix node graph.

The USD payload mechanism will cause material target children to be created within the Katana scenegraph hierarchy on USD import. These child locations can have materials bound, be targeted as Gaffer mesh lights, have their visibility set, or be pruned. Pruned or invisible children will not have their corresponding Houdini OBJ node cooked at render time, allowing them to be separated efficiently into render passes. They can also accept `PrmanObjectStatements` attributes, meaning custom render settings (such as dicing options) can be set on, and vary between, different outputs of a single HDA.

5.1 Shared Memory Data Transfer

Volumetric (VDB) data is large; transferring it from HE to RenderMan through HAPI is costly and bottlenecks on a single thread. To alleviate this we developed a method of passing data from HE to RenderMan through Linux shared memory. A custom SOP node writes VDB data to a shared memory buffer and adds attributes to the HE output identifying the buffer. These attributes are read via HAPI by the RenderMan host code, and the VDB data is loaded from the shared memory buffer instead of HAPI. In a typical production case, switching from HAPI to shared memory reduces the overall HE cook and data transfer time by a third.

Shared memory transfer is also utilized for passing input geometry from Katana to HE. Katana input locations are serialized into a temporary in-memory USD stage, which is then dumped to a shared memory buffer. The stage ID is passed to HE and loaded with a USD Import SOP. In addition to providing a performance advantage, this approach also causes any USD transformations performed by the USD Import SOP to be applied to input geometry transferred to HE. For example, `creaseweights` on subdivision meshes are stored as detail index array attributes in USD/Katana, but Houdini expects them as edge-pair vertex attributes. Using `HAPI_SetAttributeXYZData()` functions would require the same transform to be performed within the HE host process.

6 HYDRA PROCEDURAL

Our current development efforts are focussed on utilizing Hydra's new "Generative Procedural" system [Cauchois and LaVietes 2022], and executing HE inside of Hydra. Since this can run inside of any Hydra-enabled application, it is a single codebase to maintain (only how parameters are shown in the UI is application-specific).

REFERENCES

Tom Cauchois and Steve LaVietes. 2022. Modular Scene Filtering via the Pixar Hydra 2.0 Architecture. In *ACM SIGGRAPH 2022 Talks* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 44, 2 pages. https://doi.org/10.1145/3532836.3536280

Pixar. 2023. *Universal Scene Description.* graphics.pixar.com/usd

SideFX. 2023. *Houdini Engine.* www.sidefx.com/products/houdini-engine