Image Plane Schema Proposal (draft)

Introduction

This proposal aims to add native support for Image Planes to USD. An Image Plane is a camera constrained image that can be seen in "screen space" when looking through that camera.

The majority of work in VFX is centered around augmenting existing footage, so it is important to view animation in the context of that footage (it doesn't matter if the animation looks good if it doesn't look **integrated**). Without this ability, the utility of tools like `usdview` is quite restricted and requires compositing before work can be evaluated in context.

High Level Anatomy of an Image Plane

Conceptually, an Image plane must provide the following information:

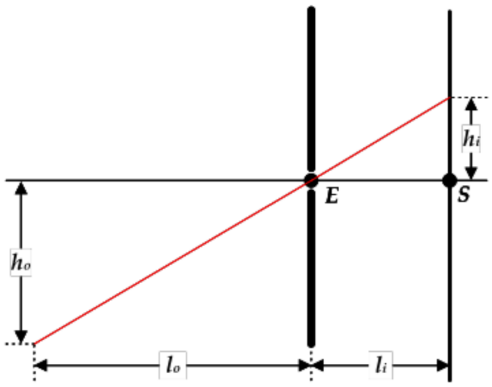
- associated camera
- image (can be animated per frame)
- fit to filmback (how image is positioned in relation to cameras filmback)
- visibility (whether image plane is visible or not, because it should be hide-able for different workflows)

And should probably also provide:

- depth (how far away from pinhole camera this plane should exist in space)
- alpha gain or other "image attributes"
- more attributes to fine tune fit

Pinhole Camera Diagram

This proposal will follow the same nomenclature from the Cooke site. Here is a reproduced CG Camera diagram. (from https://cookeoptics.com/i-technology/ > "Cooke Camera Lens Definitions for VFX" > CG Camera diagram 2.1.)



where:

- h_i is the image height,
- h_o is the object height,
- l_i is the distance between the pinhole and the image plane (i.e. the pinhole focal length, often just called the focal length in VFX software),
- l_o is distance from the pinhole to the point on the optical axis closest to the object point,
- E is the pinhole location, and
- S is the intersection of the optical axis and the image plane.

UsdGeomImagePlaneAPI Schema

The `UsdGeomImagePlaneAPI` schema defines basic properties for rendering an image plane. The `UsdGeomImagePlaneAPI` schema derives from `UsdSchemaBase` and will be a Multi-Apply Schema that would be restricted to `UsdGeomCamera` prim types.

Properties

- asset image = @@
  - Path to image file
    - can be time sampled per frame
    - This will have to be per frame, as USD/Hydra do not at this time support a $F substitution for image/texture asset paths.
- double depth = 0
  - distance, in scene units, from the pinhole to the point on the optical axis where the image plane sits
    - l_o in above pinhole camera diagram
    - alternative name: "distance" (Autodesk Maya used "depth")
    - A nice default could be -1 or "infinite" so that it would always be behind CG. That might be hard to coexist with a physically placed in scene camera depth that exists in front of and behind cg elements.
- uniform token fit = "vertical"
  - How the image plane is fit to the filmback. Possible values:
    - "horizontal" - fit image width to filmback and keep image aspect ratio
    - "vertical"- fit image height to filmback and keep image aspect ratio
    - "best" - from Maya, but maybe this behavior is too ambiguous
    - "fill" - stretched to filmback
    - "to size" - constant size, centered on filmback, and requiring more data to define "image size"
- float[2] offset
  - in same units as camera filmback - tenth of scene unit
- bool enabled
  - Control image plane visibility.
    - alternate name: "visible"
    - regardless of this value, if the (camera) prim itself is invisible, the image plane will not be visible in the viewport/render

Why a Multi-Apply API Schema?

If we implement a Multi-Apply API Schema for Image Planes we can:

- author attributes directly to camera
- author multiple image planes to camera, since multi-apply schemas can be applied several times to the same prim
  - multiple image planes could be used to view CG elements in context with foreground and background plates

API Methods

TODO:

- CreateImagePlane(imagePlaneName)
- SetImagePlane*Attr(imagePlaneName, value)
- GetImagePlane*Attr(imagePlaneName, attribute)
- SetImagePlanes([])
- GetImagePlanes()

A Minimum Usage Example:

```
camera = stage.GetPrimAtPath('/world/cam')
imagePlaneApi = UsdGeomImagePlaneAPI(camera)
imagePlaneApi.CreateImagePlane("imagePlane1")
framesToImages = [(f, fileSequence.frameAt(i) for f in fileSequence.frames()]
imagePlaneApi.SetImagePlaneImageAttr(framesToImages, "imagePlane1")
```

Which would generate this .usda:

```
def Xform "world" {
    def Camera "cam" (
        apiSchemas = ["UsdGeomImagePlaneAPI:imagePlane1"]
    ){
        ...
        string[] imagePlanes = ["imagePlane1"]
        asset imagePlane1:image = {
            1001: @/path/to/image.1001.exr@,
            1002:...
            }
        float imagePlane1:depth = 1000.0
    }
}
```

Hydra Implementation

TODO: Flesh out and discuss with Hydra team.

- Use UsdPreviewSurface textures in a way similar to GeomModel texture cards.
- Do the compositing in an Hdx post task

Open Questions

Should you have the ability to define multiple Image planes per camera? YES

Should the image plane be a separate prim or part of the camera prim? CAMERA

Should the "image" be implemented as a UsdTexture? Or any other part of Image Plane defined as its component part?

Alternative Implementation Strategies

As a concrete prim type

Here is a working implementation that was done before Multi-Apply schemas were added to USD. Its a fully featured solution that is based around Autodesk Maya's "underworld" image plane nodes where many planes can live under a camera. This specific implementation is just a quick attempt to move maya image planes across packages and the hydra implementation is just a workaround that generates an HdMesh rprim (instead of creating a new image plane prim type).

Reference PR from Luma**:** https://github.com/LumaPictures/USD/pull/1

Pros:

- Prim attributes like [visibility, purpose] can be leveraged to provide intuitive functionality
- Easy to see Image Planes in a hierarchy

Shortfalls:

- Requires relationships between camera + plane prims
- Requires an extra API schema to encode the relationship **on the Camera.**
- There's the possibility that someone targets something other than an ImagePlane with the relationship, so there are more ways to "get stuff wrong".

Using existing USD types (Plane + material + camera + expressions)

It is possible to create a UsdPlane with a material and accomplish the same thing as an image plane.

TODO: Get example usda file.

Shortfalls:

- Requires users to implement complex authoring code.
- No explicit identity as an "Image Plane". This will make it hard to successfully roundtrip from DCCs to USD.

Example Use Cases

Hydra / Usdview

It would be helpful to be able to view CG elements against a backdrop of the production plate when checking exported usd caches.

DCC Interchange

Having a "UsdLux" style standard for Image Planes would be very useful when implementing importers and exporters for DCCs like Maya, Blender, Nuke, etc.