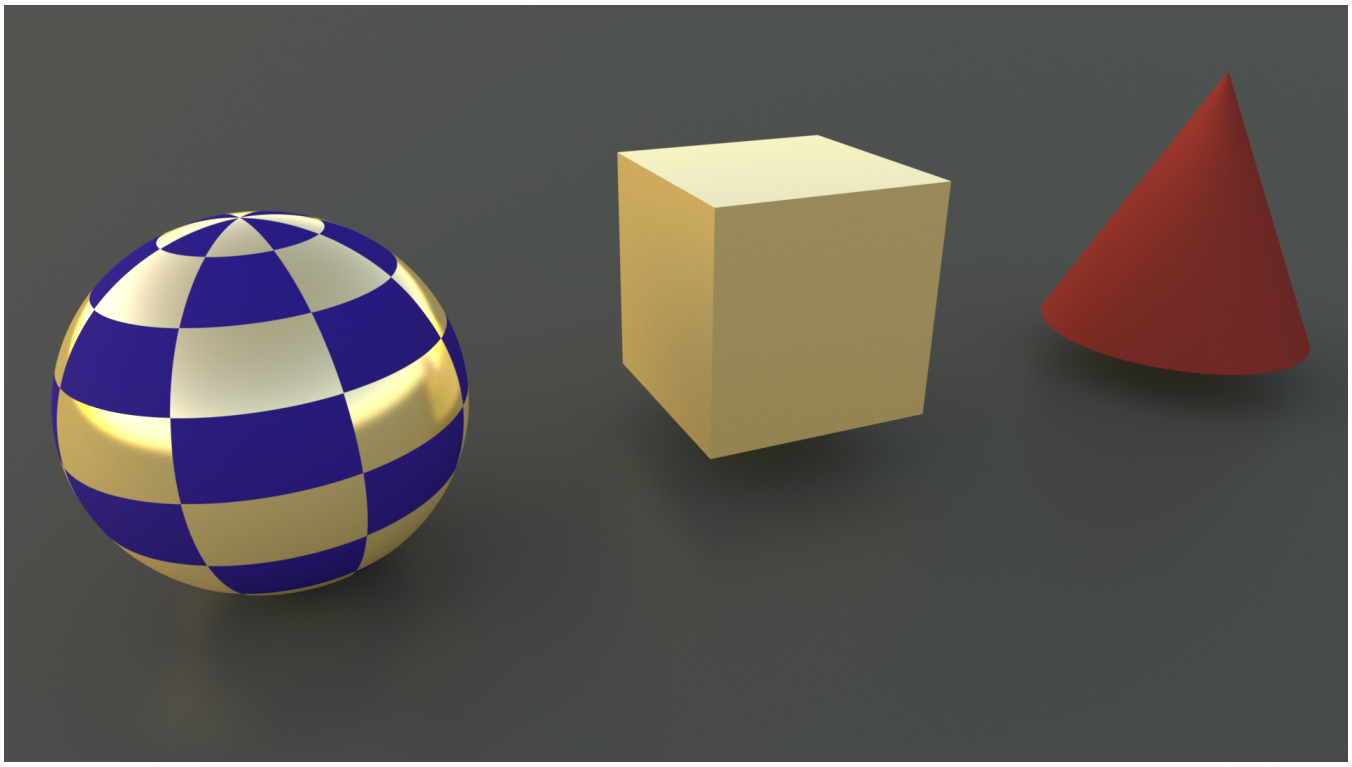
PxrMatteID in Katana

Assigning Matte IDs to rendered objects and outputting them as part of the AOVs can be useful for later compositing because this allows compositors to isolate objects or materials.

In the most basic way, you can do this in a few steps. In this tutorial, we go through the process of setting up Matte IDs to separate objects with different materials for compositing. The scene used in this tutorial is available in the RfK Examples directory - matteid.katana.

Below we have a simple example with four different materials. Each MatteID output allows three mattes since a color output has three channels -- Red, Green, and Blue. Because there are four materials in this scene, we will need to use two MatteID outputs.

> You can find an example file here to follow along: matteid.zip



Part 1: Set the MatteID attribute

First we'll assign a Green MatteID to the sphere. This is done by setting a user attribute on the sphere. Here is the OpScript you can use to set this attribute:

MatteID OpScript

```
gb = GroupBuilder()

gb:set("value", FloatAttribute({0.0, 1.0, 0.0}, 3))

gb:set("type", StringAttribute("color"))

Interface.SetAttr("prmanStatements.attributes.user.MatteID0", gb:build())
```

Create the same user attribute on the cube and cone, but set the value to the two remaining color channels - blue and red. For the ground plane, you will need to set the next available MatteID, MatteID1:

MatteID OpScript

```
Interface.SetAttr("prmanStatements.attributes.user.MatteID1", gb:build())
```

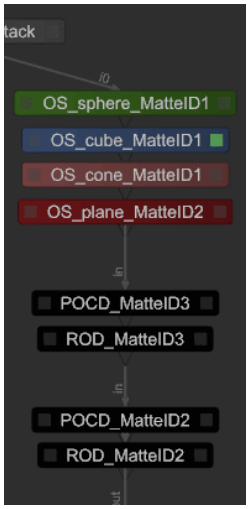
Part 2: Associate the ID in two ways

1. To associate an object or a shader to a MatteIDx, you need to add a special attribute to that object:
   Set the prmanStatements.attributes.user.MatteIDx attribute on each location to assign a color for that object in the MatteIDx AOV.
   Each MatteIDx aov can have 3 sets of objects, one for each RGB channel. In this example, there are 4 objects, so the first three are on the MatteID0 aov, and the last plane is on the MatteID1 aov.

2. If you find it easier to assign MatteID user attributes via the material location rather than on the objects, you can make use of the material. underlayAttrs attribute.
   Set the material.underlayAttrs.prmanStatements.attributes.user.MatteIDx attribute on each location to assign a color for that object in the MatteIDx AOV.

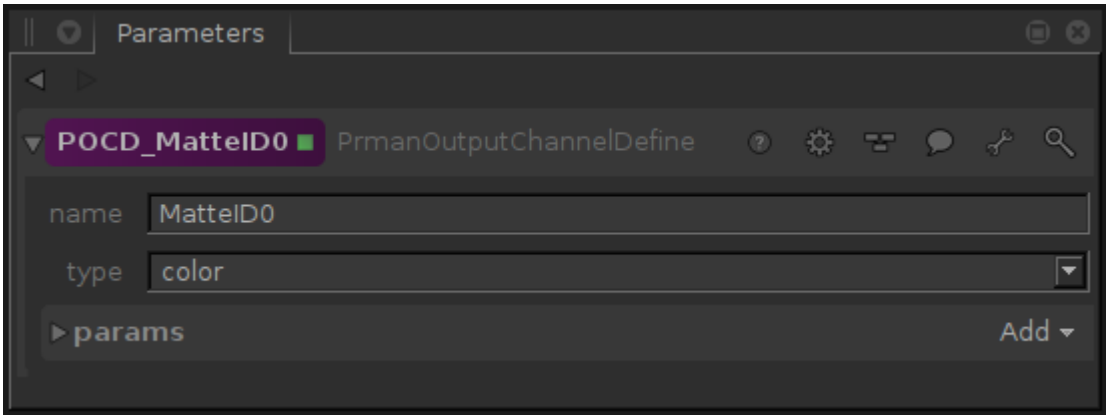
Part 3: Connect PxrMatteID to your material

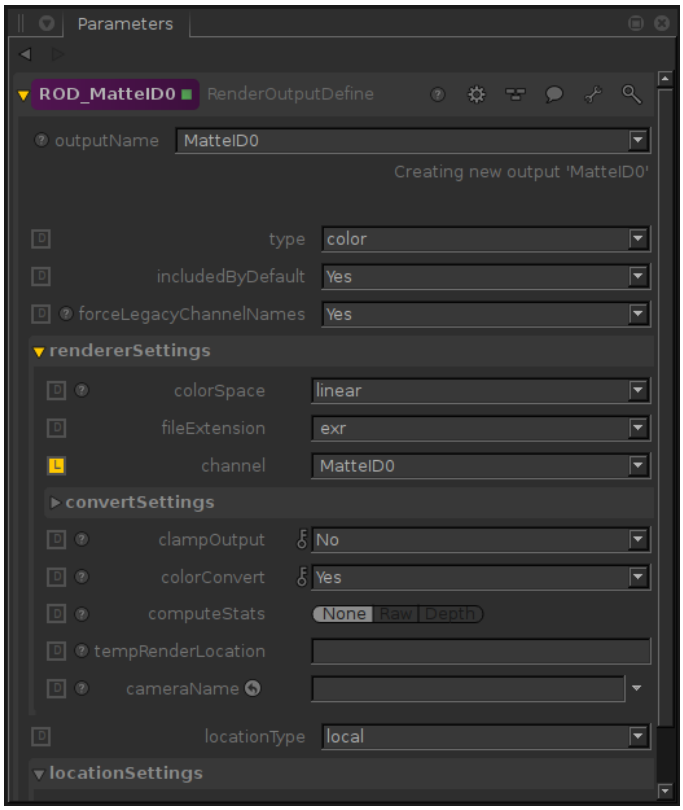
If you were to render at this point, even if you added the correct render pass, it would not work. You need to connect a PxrMatteID pattern to the material. You do this by connecting the resultAOV output of PxrMatteID to the Utility Pattern connection of the PxrSurface material. You can connect the same PxrMatteID pattern to the PxrSurface on each object:



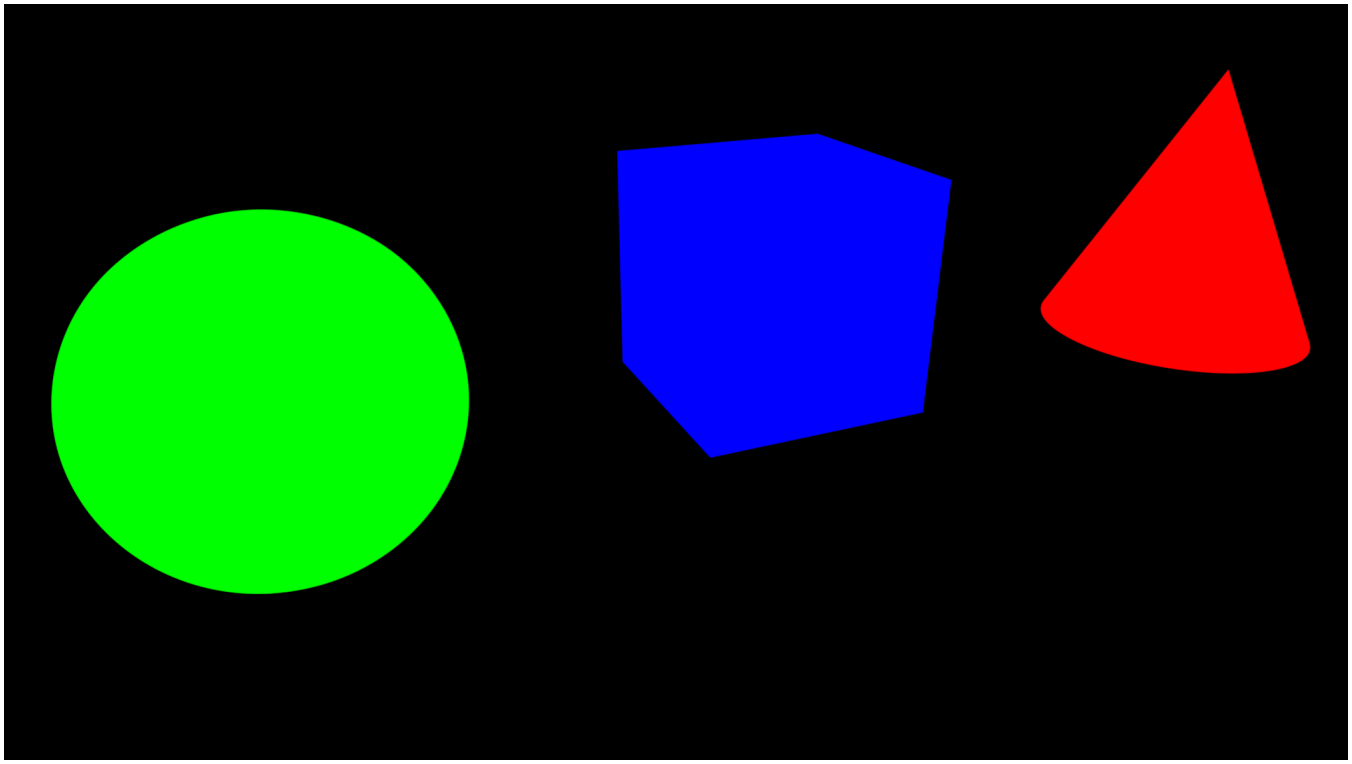
Part 4: Create the MatteID AOVs

Now add the appropriate MatteID AOVs. The name in the PrmanOutputChannelDefine node should match the user attribute name (MatteID0, MatteID1, etc):

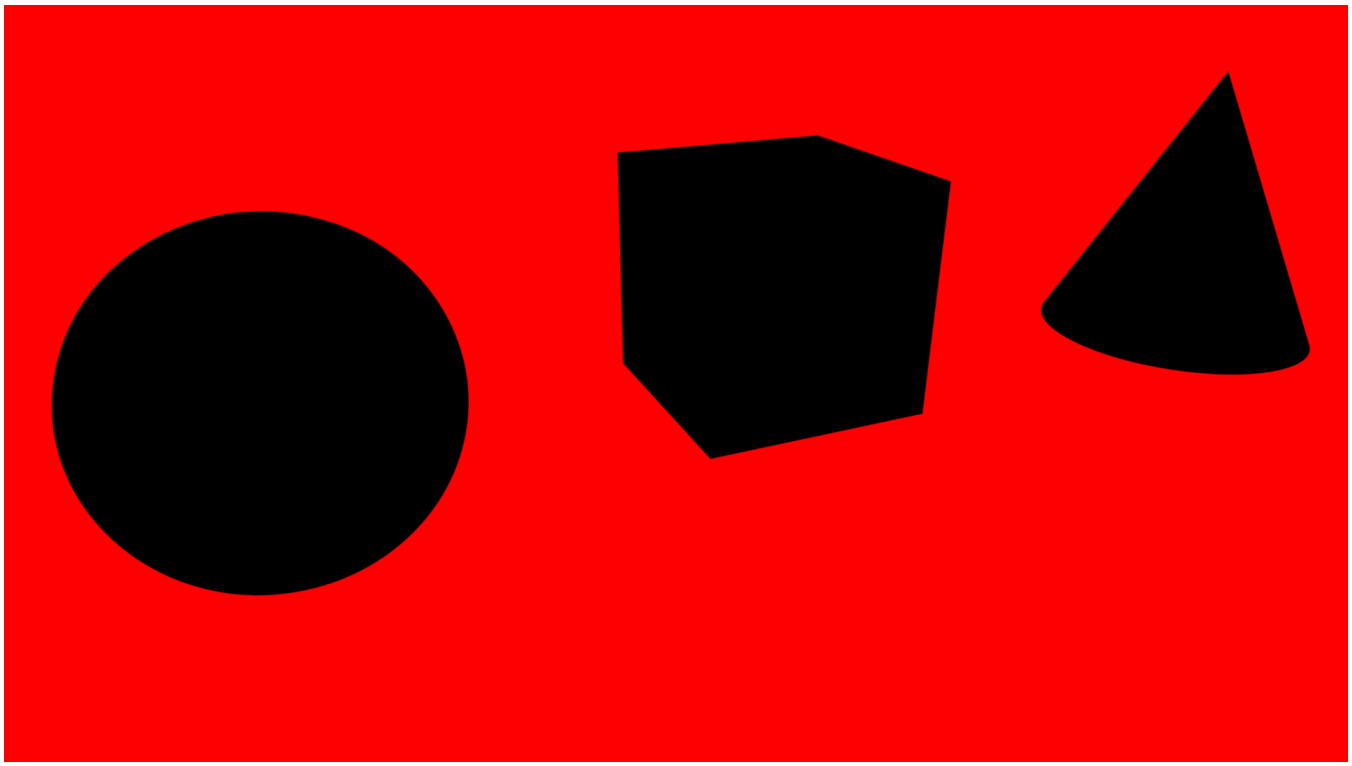




The resulting MatteID0 pass looks like the one below:



And the resulting MatteID1 pass looks like this:

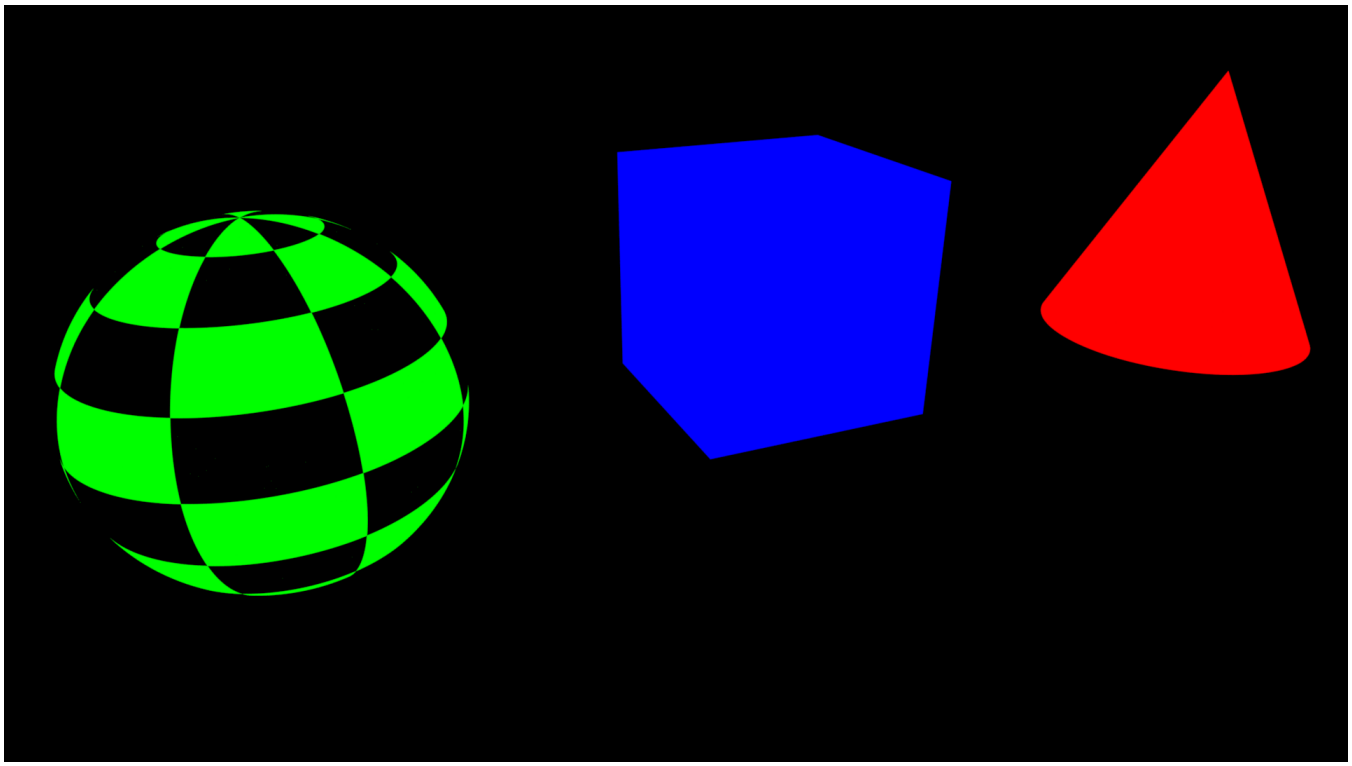


Part 5: Separate materials with MatteID Textures

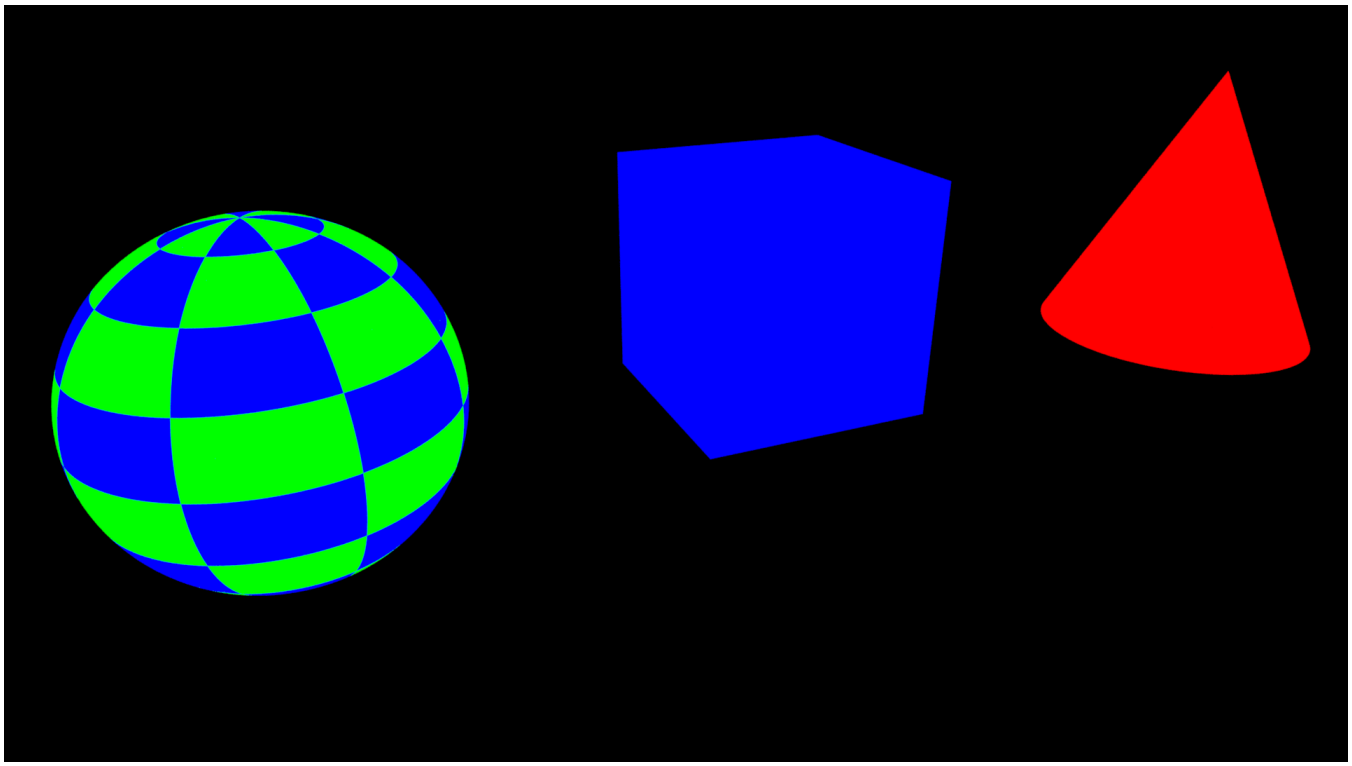
If each of the objects in our scene had a different material, or if we just wanted to isolate each object in the scene, we could stop at Part 3. But since the sphere has two materials, and we are trying to isolate different materials, we need to use the Matte Texture parameter of PxrMatteID. The Matte Texture parameters let you assign colors or patterns as the output for each matte. The goal is to get a matte AOV where the purple parts of the sphere are green, and the gold parts are blue to match the cube's matte.

If you recall, we didn't make any changes to the PxrMatteID node in Part 2. It was left at default. We will leave this one alone but disconnect it from the PxrSurface assigned to the sphere. We're doing this so we can affect just one object and not all three.

Create a new PxrMatteID and connect it to the PxrSurface for the sphere. In this case, the sphere has one unique material and one material that is the same as the cube, so we can still use MatteID0 and Matte Texture 0. Connect a PxrChecker pattern to the Matte Texture 0 parameter of PxrMatteID. Set the checker colors to blue and green.



Ok, that sort-of worked, but the parts of the sphere that should be blue are black. This is because the MatteID color is multiplied against the pattern we connect to the PxrMatteID. To get the correct blue/green checker output, we'll need to change the MatteID0 user attribute that we set on the sphere earlier. Set that value to white, and then render again.



Mission accomplished! Now you can add additional PxrMatteIDs and chain together more patterns for different results in your renders. Your compositor will thank you.