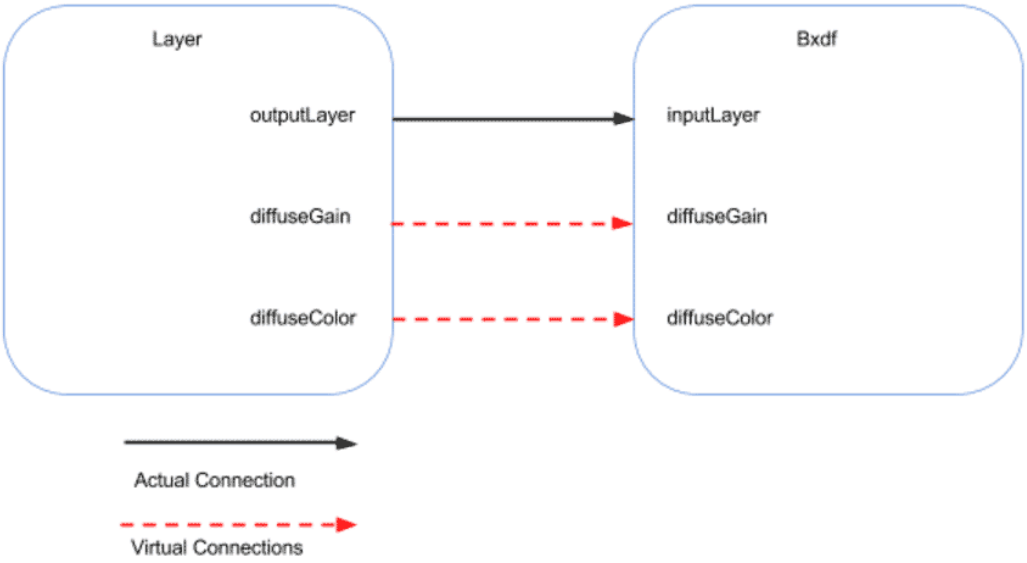
Vstructs

Introduction

This tech note is for shader plugin developers, as well as RenderMan bridge developers. It is meant to help in using vstructs in args files as well as handling the rib generation of vstructs.

What is a Vstruct?

Vstructs or Virtual Structs are a way to connect a set of parameters from one bxdf/pattern plugin or OSL shader to another set, while representing this connection as one virtual struct. Simply put, the artist connects a vstruct in a shader network and behind the scenes, many connections of existing parameters are made. For example:



PxrSurface as an example

In this example we have a simple vstruct connection that connects diffuse gain and color virtually. When the artist connects the material layer in their shader network to the bxdf layer input, behind the scene the diffuse and gain values from the layer are connected to the bxdf respective inputs.

This fact is enough for artists, and the rest of this note will deal with the implementation details for developers. (Nerd firewall activate!)

Vstruct details

Like all parameters, vstructs and their connections are defined in the args files for a plugin. Or in OSL shaders in the metadata of the parameters. As the name Vstruct (virtual struct) implies, they act as structs made up of a set of virtual parameters. Connecting a vstruct will allow a set of virtual connections to be made. First we need a vstruct parameter that declares the connection that the vstruct will be made through.

Declaring a vstruct parameter:

```
<param name="inputLayer"
        label="Input Layer"
        type="float">
   <tags>
        <tag value="vstruct"/>
   </tags>
   <help>
        This parameter represents all the connectable patterns.
        Use as input connection to a material.
   </help>
</param>
```

This is an example of an input vstruct parameter. This can be hooked up to another vstruct from an output parameter. Let's take a look at an example of that:

```
<output name="outputLayer" type="float">
    <tags>
         <tag value ="vstruct"/>
    </tags>
</output>
```

Fairly straightforward. Output parameters with the vstruct tag can connect to inputs with vstruct tags just like any other connection between pattern parameters. However the virtual members of the struct must also be hooked up.

Vstruct members

It's also fairly simple to declare that a parameter gets its value from a vstruct. On the input side here's an input parameter getting its value from the member of a vstruct (if inputLayer is connected)

```
<param name="diffuseColor"
           label="Color"
           type="color" default="0.18 0.18 0.18"
           vstructmember="inputLayer.diffuseColor">
        <tags>
             <tag value="color"/>
        </tags>
        <help>
            Diffuse color.
        </help>
</param>
```

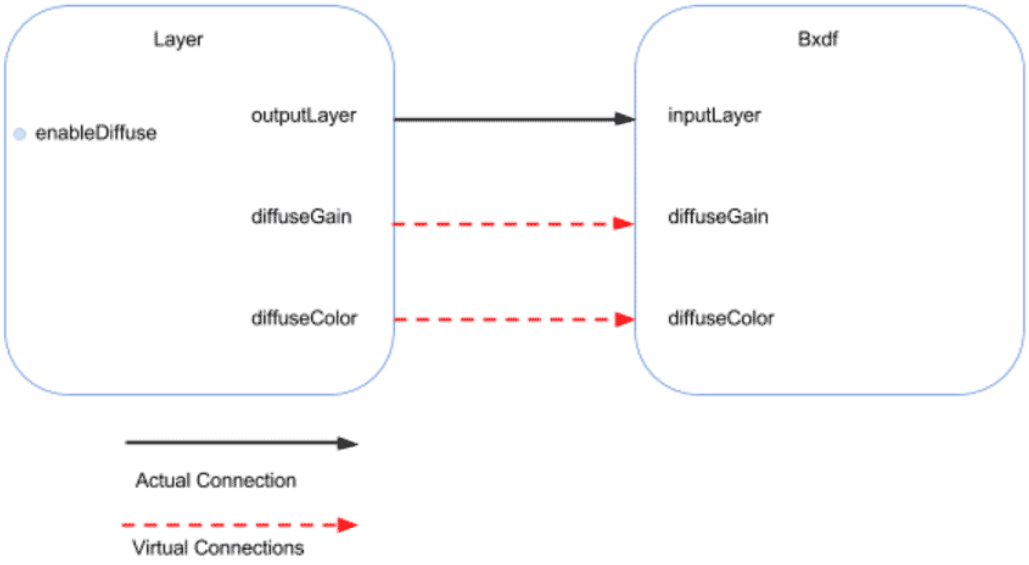
And similarly on the output side a parameter can declare itself to be a member of a vstruct:

```
<output name="outputLayer_diffuseColor" vstructmember="outputLayer.diffuseColor">
    <tags>
        <tag value ="color"/>
    </tags>
</output>
```

Now, when a vstruct connection is made between outputLayer and inputLayer, a connection will be made automatically between inputLayer.diffuseColor and outputLayer.diffuseColor, which in this case would go to outputLayer_diffuseColor. If the corresponding name on the vstruct input side is not present, no connection is made.

Vstruct conditionals

To take this idea a bit further we can also make connections only if a criteria is met. Let's say for example there is a parameter called "enableDiffuse" on our Layer pattern in the above example:



Vstructs Conditionals

We only want to make the diffuseColor connection if enable diffuse is turned on of course. So using a second piece of metadata "vstructConditionalExpr", we can define when to make that connection.

For example:

```
<output name="outputLayer_diffuseColor" vstructmember="outputLayer.diffuseColor"
vstructConditionalExpr="connect if enableDiffuse == 1">
   <tags>
       <tag value ="color"/>
   </tags>
</output>
```

These conditionals can of course be more complex than this. Examples:

```
"connect if enableRoughSpecular == 1 and roughSpecularBumpNormal is connected"
"set 1.0 if enableRR == 1 else set 0.0"
```

Shader plugin developers

Vstructs are fairly simple to set up with parameter metadata. Vstructs are defined as float parameters with the vstruct as above. Or in OSL: "string tag=" vstruct" ". On input parameter the vstructmember meta data should match IN_VSTRUCT_NAME.OUT_PARAM_NAME which will define the parameter to make the connection to. On the output parameter side the parameters just need to define vstructmember similarly:

OUT_VSTRUCT_NAME.OUT_PARAM_NAME And during rib gen connections will be made between the two.

Vstruct conditional grammar

Here are some more examples of vstruct grammar. The meaning of the code should be fairly self-explanatory:

"connect if underMaterial_singlescatterK > 0 " "or (enableSinglescatter == 1 and (singlescatterK > 0" "or singlescatterK is connected or singlescatterDirectGain > 0 " "or singlescatterDirectGain is connected)) " "connect if ((rrReflectionK is connected or rrReflectionK > 0) " "and enableRR == 1) or underMaterial_walterReflectionK is connected" "else set 0" "connect if enableClearcoat == 1 "

Bridge Product developers

Implementing vstructs in a bridge product can be somewhat difficult, mainly in that the vstruct conditional needs to be parsed. Furthermore, it can involve some walking of a shader network and evaluation of parameters to determine when or if to make connections.

Node connections. Vstruct parameters should be treated like structs for Node Tree UI's where shader networks are connected. When a vstruct connection is made, special steps should be followed when generating rib for that node.

- If an input parameter has a vstructmember tag, check if the corresponding vstruct is hooked up.
- If the vstuct is hooked up and has an output member corresponding to the vstructmember by name, try to make a connection between the input connection and output connection.
- Evaluate the vstructcondional on the output connection to see if a connection should be made or the value on the input set.

This may involve checking parameters further up the shader tree as a vstructconditional may depend on whether another input parameter is connected, which may in term depend on a vstructconditional. For example if in the above, enableDiffuse is also hooked up to another incoming vstruct which needed to be evaluated.