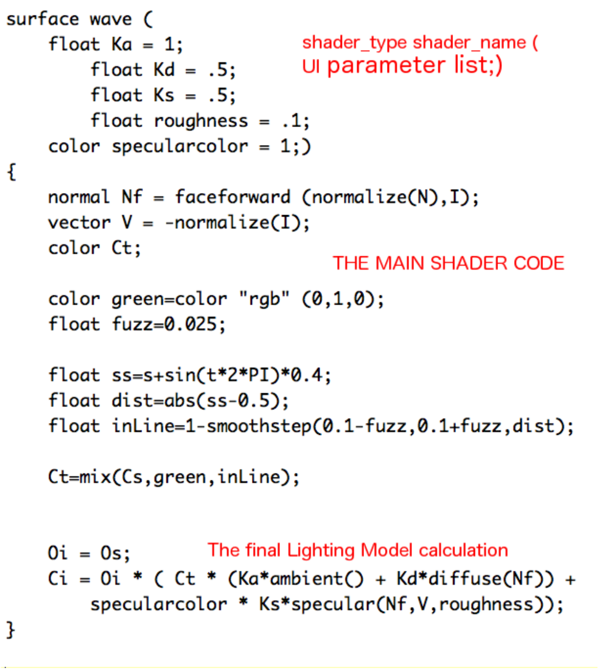
USING RENDERMAN CODE TO GENERATE SHADER PATTERNS

A useful skill is to be able to decipher Renderman code into node based networks. This can be especially helpful when delving into higher-level computer graphics theory books where examples are given in a Renderman compliant language. Inside the directory of SHADER_SAMPLES are a number of Renderman .sl shaders. These can be examined in a text editor to discern their content.



```
surface wave (
    float Ka = 1;
        float Kd = .5;
        float Ks = .5;
        float roughness = .1;
    color specularcolor = 1;)
{
    normal Nf = faceforward (normalize(N),I);
    vector V = -normalize(I);
    color Ct;

    color green=color "rgb" (0,1,0);
    float fuzz=0.025;

    float ss=s+sin(t*2*PI)*0.4;
    float dist=abs(ss-0.5);
    float inLine=1-smoothstep(0.1-fuzz,0.1+fuzz,dist);

    Ct=mix(Cs,green,inLine);


    Oi = Os;
    Ci = Oi * ( Ct * (Ka*ambient() + Kd*diffuse(Nf)) +
        specularcolor * Ks*specular(Nf,V,roughness));
}
```

shader_type shader_name (
UI parameter list;)

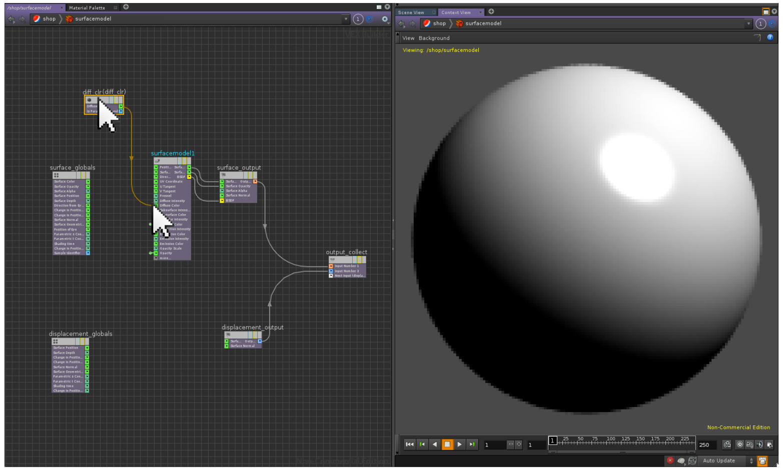
THE MAIN SHADER CODE

The final Lighting Model calculation

There are three main sections to a Renderman Shader. Firstly there is the declaration of the shader_type and the shader_name, with further references inside the brackets to any parameters deemed for End User control. Between the curly braces { }, is the main shader code. This is the engineering of the surface aesthetic before the calculation of scene lighting is factored. The final section is the Lighting Model calculation (ambient + diffuse + specular). When recreating this code as a VOP Network, it is a good idea to start from the bottom and work upwards.

ACTIVATING THE MATERIAL

Use the **Material Palette** to create a **Surface Model Material** at SHOP Level. Inside at the shader level of this material, **double LMB** on the **Diffuse input** nodule to expose the associated **Parameter VOP**.

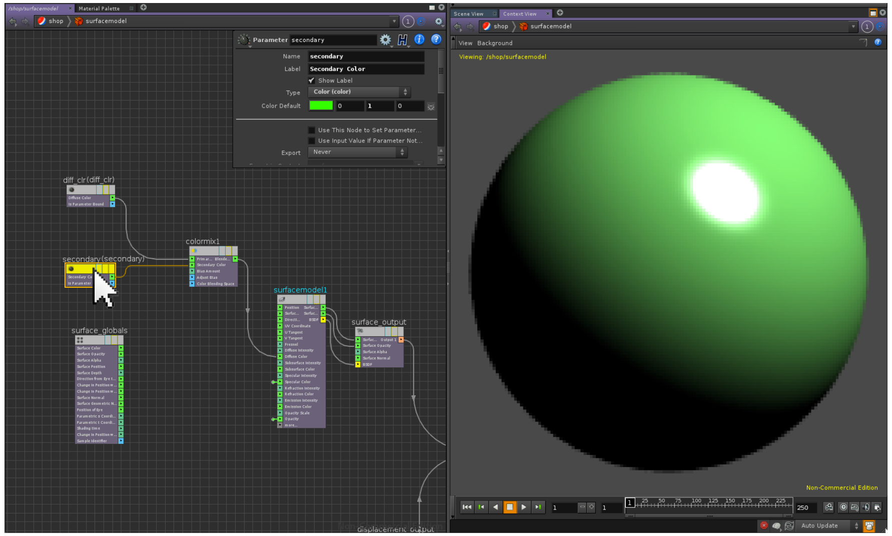


The **Surface Model VOP** can be seen as the **final Lighting Model calculation** of the shader code.

The **final line** of the **main shader code** is:

```
Ct=mix(Cs, green, inLine);
```

In terms of **VOPs** this **mix function** can mean either the **Mix VOP** or the **Color Mix VOP**, where two colours (Cs and Green) are being mixed using a bias amount of a variable called inLine.



Parameter secondary

Name: secondary

Label: Secondary Color

☑ Show Label

Type: Color (color)

Color Default: 🟩 0 1 0

□ Use This Node to Set Parameter...

□ Use Input Value If Parameter Not...

Export: Never

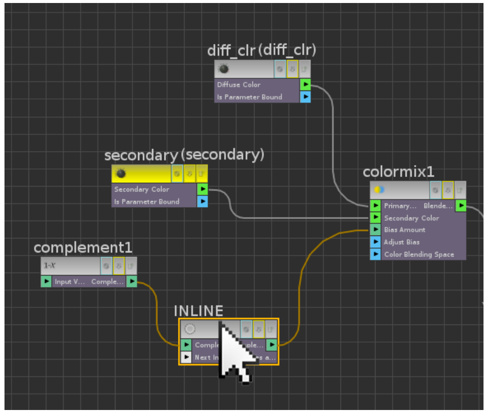
RMB on the **Diffuse Color Parameter VOP** to insert a **Color Mix VOP**. Promote the **Secondary Color input** as a **new parameter**, setting its colour value to **green**.

CREATING THE INLINE CALCULATION

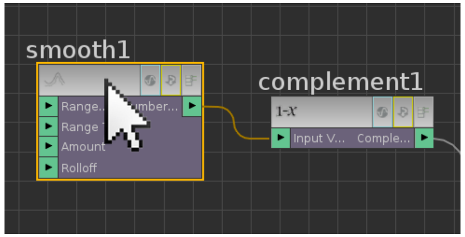
The inline variable section of the code reads:

```
float inline = 1 – smoothstep(0.1-fuzz, 0.1+fuzz, dist);
```

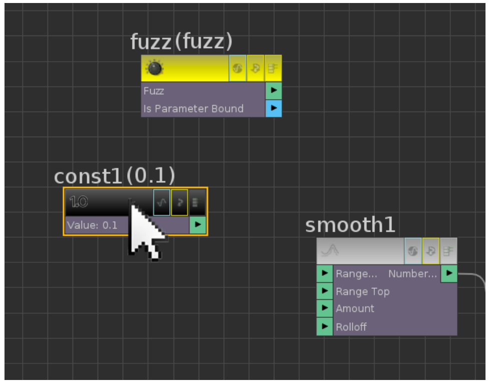
In the **Network Editor**, create a **Compliment VOP** (**1-**), and append to it as **Null VOP**. **Rename** the Null VOP to **INLINE**.



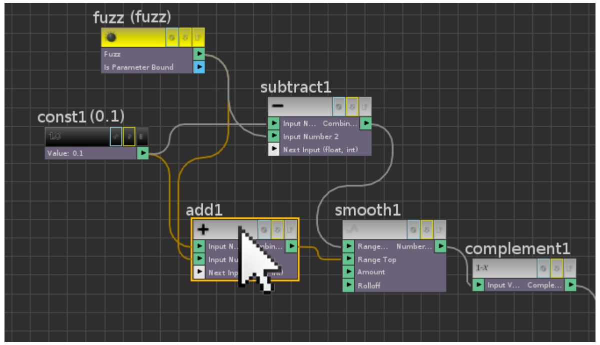
RMB on the **input** of the **Compliment VOP** to create a **Smooth VOP**.



Examination of the main shader code reveals that the **variable fuzz** is a constant float value of **0.025**. This can be created using a **Parameter VOP**.



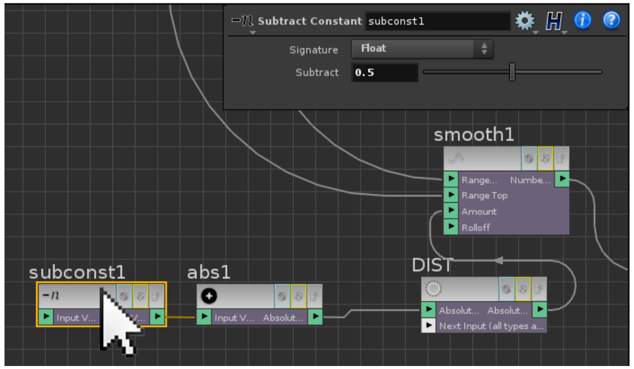
Similarly the value of **0.1** + or – **fuzz** can be created with a **Constant VOP**. With both of these nodes activated; they can be wired into the Smooth SOP as per the shader code.



The **third input** required for the **Smooth VOP** is a **variable** called **dist**. In terms of its shader code, dist is described as:

dist =abs(ss-0.5);

Where a secondary variable called **ss** is being turned positive by an absolute function, and has 0.5 subtracted from it.


Subtract Constant subconst1

Signature Float

Subtract 0.5

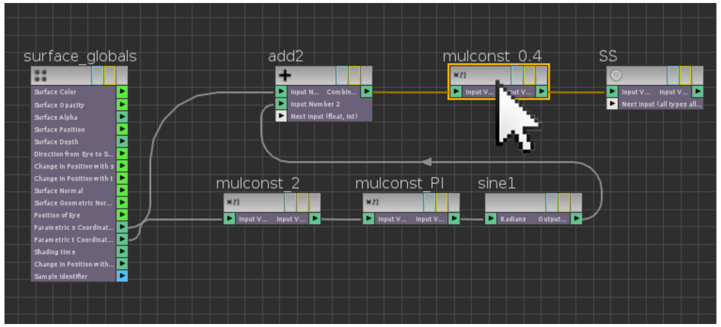
RMB on the **third input** of the **Smooth VOP** to create an **Absolute VOP**. Insert into it a **Subtract Constant VOP** with a value of **0.5**.

Examination of the shader code reveals the **ss variable** as:

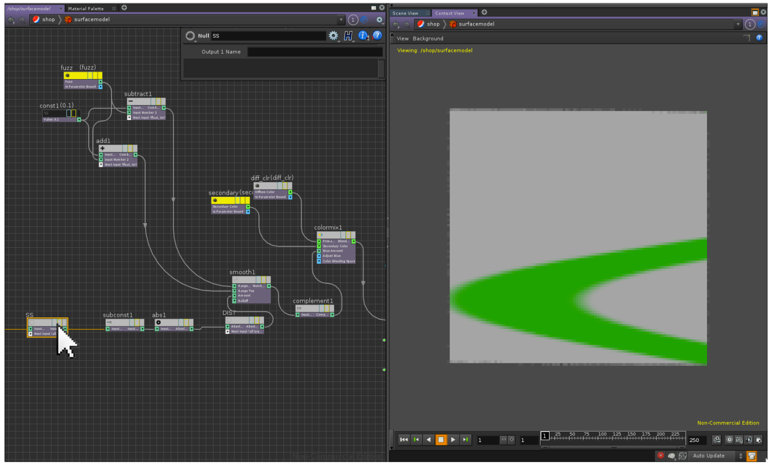
ss=s+sin(t*2*PI)*0.4;

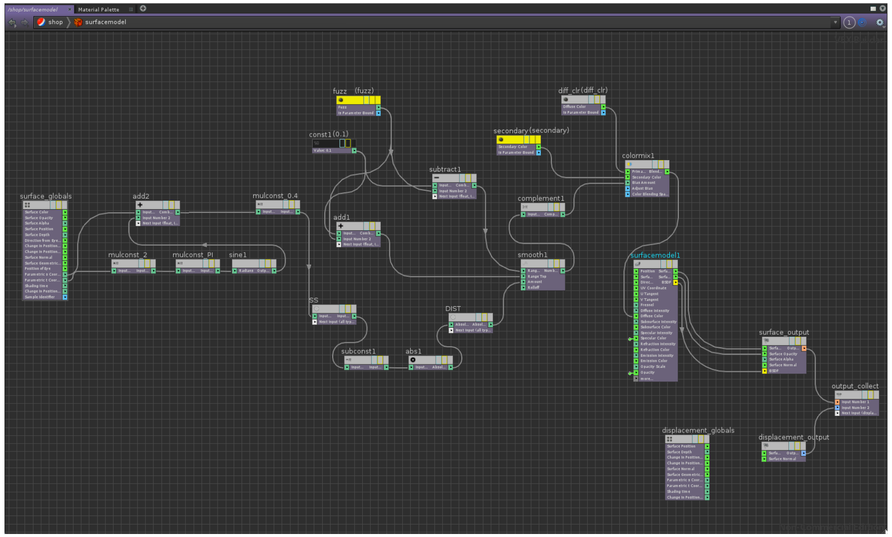
The use of **s+sin** denotes the **global variable s** added to a **sine function**.

This line of code can be recreated as nodes by starting with the **Parametric s and t Coordinate outputs** of the **Global Variables VOP**.



Once the ss variable network is completed, it can be used to finish off the recreation of the shader code.





The Final Network…

Other .sl files in **SHADER_SAMPLES** include:

- **band.sl**
- **disk.sl**
- **random.sl**
- **stripe.sl**
- **wave.sl** (this is the shader replicated above)

Have a go at recreating some of these other shader .sl files to gain better understanding of how to configure simple patterns.