

Indigo Shader Language Beginner Tutorial

This tutorial covers:

- Whats ISL? And why should I use it?
- Whats a functional language?
- How to define functions
- Which channel expects what function parameters and return value?
- What data-types are there?
- Functional, huh? Does that mean I have to use functions instead of operators?
- How to use a shader in a material?
- Going further

Whats ISL? And why should I use it?

ISL stands for **Indigo Shader Language**. It's a functional language that allows you to write shaders for every channel in Indigo materials.

With shaders you're not tied to the restrictions of textures anymore because they are procedurally computed for every point on the surface (or in the volume, as of Indigo 2.4's ability to have volumetric shaders).

Whats a functional language?

In functional language there are only functions that are used to compute something, no variables no loops (unless you write them as a function). By functions it means functions in the mathematical sense, it calculates something and returns the result.

Let's have a look at this shader function:

```
def doSomething(vec3 pos) real: noise(fbm(pos * 3.0, 8))
```

This is a function called 'doSomething', it has one parameter, a vector called 'pos' and it returns a real value. It uses two other functions, fbm() and noise().

Now what's happening here?

Like in a mathematical function you calculate the innermost parenthesis first. That means the vector 'pos' is multiplied by 3.0, then passed to the fbm() function which calculates a value of the type 'real', which is then passed to the function noise(), which calculates a 'real' value, which is the return value of the function 'def'.

Pretty simple, isn't it?

How to define functions

Now, let's see how to define a function.

A ISL function definition always starts with the keyword 'def', needs at least a name and a return value type, and it can also have function parameters:

```
def name(parameters) return_value_type: [...actual function...]
```

Although you can give your functions an any name you want the main function in a channel always has to have the name 'eval'. Which takes us directly to the next topic: different channels expect different parameters and return values!

Which channel expects what function parameters an return value?

There are three different channel types, wavelength dependent, wavelength independent and displacement.

Wavelength dependent expects a vec3 as a return value, wavelength independent expects a real and displacement expects real and cannot use the position in world-space (vec3 pos) as a function parameter.

Here's a table that illustrates all that:

Channel	Channel type	Eval function expected
Diffuse	Wavelength dependent	def eval(vec3 pos) vec3:
Emission	Wavelength dependent	def eval(vec3 pos) vec3:
Base Emission	Wavelength dependent	def eval(vec3 pos) vec3:
Specular Reflectivity	Wavelength dependent	def eval(vec3 pos) vec3:
Absorption Layer	Wavelength dependent	def eval(vec3 pos) vec3:
Exponent	Wavelength independent	def eval(vec3 pos) real:
Sigma	Wavelength independent	def eval(vec3 pos) real:
Bump	Displacement	def eval() real:
Displacement	Displacement	def eval() real:
Blend	Displacement	def eval() real:

What data-types are there?

First of all, its very important to know that ISL does not convert values implicitly, so if a function expects an integer, you have to make sure you give pass an integer value instead of a real.

real – floating point number

A real value always has to be written as a floating point number, for example 214.0.

int – whole numbers

Only whole numbers, like 20 or -1545.

vec3 – 3 component vector

There are two constructors for a vec3, vec3(real x) and vec3(real x, real y, real z). The first one applies the number passed to any of the 3 components and the second one sets each component separately.

You can access the three components separately with the functions doti(), dotj() and dotk().

vec2 – 2 component vector

Like vec3, only just 2 components.

bool

A boolean value, true or false.

mat2x2 and mat3x3

2x2 and 3x3 matrix, I'm not going to talk about these right now.

Functional, huh? Does that mean I have to use functions instead of operators?

No, you don't have to use functions as operators, but you can, if you like to, are a hardcore mofo or just a little insane :)

Operators are available for multiplication, division, addition and subtraction (*, /, + and -, would you believe it?) for every data-type that supports them, but the order of operands is important sometimes, for example:

0.333 * vec3(5.2)

will not work since it expects the vec3 first.

Vec3(5.2) * 0.333

works.

The equivalent functions are called mul(), div(), add() and sub().

How to use a shader in a material?

Your exporter most likely has an option to use ISL shaders, also, you can use ISL in the Indigo Material Editor it allows you to use ISL shaders.

And if you're hardcore, you can edit the .igs file generated by you exporter and insert your shaders manually, here's how you do it:

Open the .igs file and look for a material. I'll just assume we found a phong material:

```
<material>
  <name>phong_mat</name>
  <phong>
    <ior>1.466</ior>
    <exponent>
      <constant>800</constant>
    </exponent>
    <diffuse_albedo>
      <constant>
        <rgb>0.588235 0.588235 0.588235</rgb>
        <gamma>2.2</gamma>
      </constant>
    </diffuse_albedo>
  </phong>
</material>
```

```
        </rgb>
    </constant>
</diffuse_albedo>
</phong>
</material>
```

What you have to do is, is to replace the <constant>...</constant> XML elements (and anything in between) in the diffuse_albedo channel with this:

```
<shader>
  <shader>
    <![CDATA[

        #paste your shader in here, oh by the way: every line starting with # is a comment

    ]]>
  </shader>
</shader>
```

Then paste your shader in between '<![CDATA[' and ')]>'.

Going further

For a complete list of all available functions, have a look at the ISL section in the 'Indigo Technical Reference.pdf' and the 'ISL_stdlib.txt' in the Indigo folder.

Also, more ISL tutorials are coming!