GA Shader Sculptor
User Manual

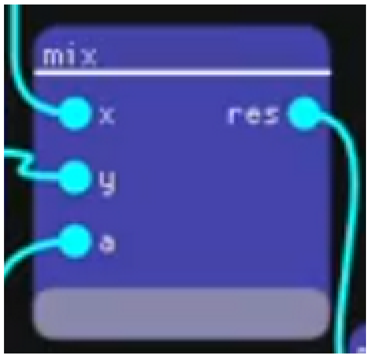
Running:

- Download **Cmake**; See https://cmake.org/download/
- Download **GLFW 3** and ensure it is visible to **CMake**;
  - See https://www.glfw.org/download.html and https://www.glfw.org/docs/3.3/build_guide.html
- In the same directory as **CMakeLists.txt**, create a 'build' directory and navigate into it
- Run **cmake ..**
- Run **cmake --build .**
- Execute compiled binary: **shader_sculptor**

Startup:

- On start up, select a **'Boilerplate'** type from the single visible window.

Nodes And Features:



- A node either represents an operation performed on a set of inputs from other node operations; or it represents a type and values (float, int, VecN, MatN).
- Output pins (on the **right side** of the node) represent the results of the node's operation.
- Input pins (on the **left side** of the node) represent the inputs of the node and connect to output pins of other nodes. Input pins will attempt to create default values when they are not connected to outputs.
- Pins possess **types**, which can be seen by ***hovering over a pin***; default values are not created for all types.
- Pins **will NOT connect** to pins of other types, however if a pin is of a **generic-length type** (VecN or genType), it will connect to any valid vector length of the correct underlying type. This will constrain the type of both input and output pins.
- Pins **will also NOT connect** if such a connection would create a directed cycle in the graph.
- Intermediate Display:
  - Some nodes have a button at the bottom which will ***display the primary output result*** of the node as the final color of a shader.
  - **Shaders for intermediate displays are not compiled automatically**. Any intermediate display, which is no longer representative (due to changes not yet built) of the current graph, will have a **RED background**.

Boilerplate:

The shader graph's **'Boilerplate'** defines the additional code before and after user-generated code. The **Boilerplate** type therefore also defines the parameters and variables which are accessible to the user and defines the pins of the terminal nodes (see **Terminal Nodes**), which are used as input to terminal **Boilerplate** code in order to determine final shader functionality.

Terminal Nodes/Pins:

There are two terminal nodes, which have only input (terminal) pins that are specified by the 'boilerplate' type selected for the graph. Terminal nodes represent the final values which are passed to the boilerplate code to determine the final behavior of the shader.
Each terminal input pin uses a boilerplate-specified default value if an output pin is not connected to it, see **Boilerplate Default Types**.
While non-terminal nodes used for one terminal node can theoretically be used for the other terminal node as well, it is recommended this is avoided.

Boilerplate Default Types:

- **Unlit Boilerplate:**
    - Defines no usable terminal pins for the vertex shader and the final fragment color (no additional modifications performed) for the fragment shader.

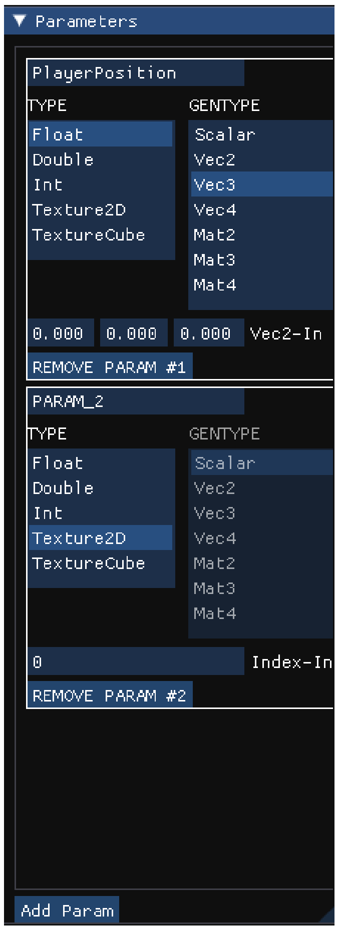
- **PBR-like Boilerplate:**
    - Defines three terminal pins for the vertex shader:
        - Vertex World Position: Add this value to the world space position of the vertex.
            - Default : (0, 0, 0)
        - Vertex Color 1 : Unused by boilerplate, can be used by the user to pass data.
            - Default : (0, 0, 0)
        - Vertex Color 2 : Unused by boilerplate, can be used by the user to pass data.
            - Default : (0, 0, 0)
    - Defines five terminal pins for the fragment shader:
        - Albedo : the base color of the PBR material
            - Default : (1, 1, 1)
        - Metalness : the metalness of the PBR material, should be clamped [0, 1]
            - Default : 0.5
        - Roughness : the roughness of the PBR material, should be clamped [0, 1]
            - Default : 0.5
        - AO : the ambient level of the PBR material, should be clamped [0, 2]
            - Default : 1
        - Normal : the TANGENT normal of the fragment; (0.5, 0.5, 1.0) represents the tangent normal corresponding to the base normal of the fragment
            - Default : (0.5, 0.5, 1.0)

CONTROLS:

- **ADDING A NODE:**
  - Press Right Click to open the **ADD NODE MENU** to *search* for and add a node using a text input field *search bar.*
- **CONSTRUCTION:**
  - Hold Space and Left Click to drag the view.
  - Hold Left Click over the non-pin sections of a node to drag and move that node.
    - Release Left Click to stop dragging.
  - Hold Left Click over a pin to drag a connection line to another pin to connect it to.
    - Release Left Click over a compatible pin to connect the two pins. If successful, a blue curve will be drawn between the two.
- **DELETION:**
  - Press Delete when hovering over a pin to disconnect it from all connected pins.
  - Press Delete when hovering over a node to disconnect its pins and delete it.
- **DISPLAY:**
  - Press Left Click on the intermediate display button of a node to toggle its display.

WINDOW MENU

- Build Shaders: Build the vertex shader and the fragment shader and update all intermediate displays. Also makes an autosave of the current generated text code.
- Save: Save text source code of vertex and fragment shaders.
  - Make sure the destination directory exists before saving.
- Controls: Show an abbreviated list of controls (or see **Controls**).
- Credits : Display Credits and Special Thanks.



PARAM PANEL DESCRIPTION

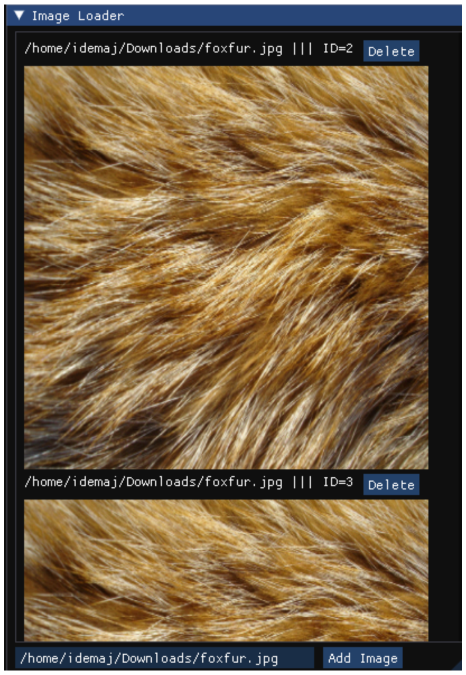
The parameter panel allows the specification of parameter values; these values can be changed in graph mode and in any engine without recompiling the shaders. Parameters are implemented as uniforms (in GLSL).

- Creating a parameter with the **Add Param** button will create a default Vec3 parameter and add an option to the **ADD NODE MENU** to add that parameter as a node to the graph.
- Deleting a parameter will disconnect and remove all nodes from the graph which correspond to that node. Changing a parameter's type will disconnect its pins.
- Changing a parameter's name will cause ALL shaders to become dirty and require a recompile. This is to prevent linker errors from mismatches of uniform name and type between vertex and fragment shaders.
- Changing a parameters default value using **the input fields just above the REMOVE button (*-In)** will NOT cause shaders to become dirty and will propagate immediately to the shaders.

IMAGE PANEL

Stb_image compatible images can be added as **OpenGL** textures using this panel.
Specify a file path to an image in the input field and press the ADD IMAGE button to add and bind the image to an OpenGL texture. If successful, the image and a texture ID will be displayed in a scroll group within the panel.

- It is recommended you access these images by creating a textureSampler **parameter** and setting its value to the texture ID specified by the image panel.
    - For the default value of **image parameters**, use the ID specified in the **IMAGE PANEL** as the **'Index-In'.**
- **WARNING**: Shaders will not be compiled out with these images. They must be bound and linked to the shader using the **image parameter** created (in the **PARAM PANEL**).


▼ Image Loader
/home/idemaj/Downloads/foxfur.jpg ||| ID=2
Delete
/home/idemaj/Downloads/foxfur.jpg ||| ID=3
Delete
/home/idemaj/Downloads/foxfur.jpg
Add Image

Advanced Use:

CUSTOM BUILTIN NODES

An advanced user can add, remove, or change existing 'built-in' functions/nodes available to the user by modifying the plaintext file "builtin_glsl_funcs.txt."
The format for a valid node is as follows:
out <output type> <output name> = code in <input type> <input name> , … **;** <NODE NAME>
The keywords in and out, and their 2 postceding tokens will be replaced by the appropriate variable or constant at build time.
Any tokens which are not immediately preceded by an in or out keyword are considered an immutable part of the code and will not be replaced. ';' indicates a conclusion of the node's code. Unfortunately, this means that only one line of code may be utilized to describe a custom node.
Take care that the two next tokens after 'in' or 'out' are a valid GLSL type and the argument/variable name desired; make sure characters like ',' are not accidently included in the name token.