GA Shader Sculptor
User Manual

Running:

The program can be run with ./shader_sculptor.exe if run on Windows. The program can also be compiled and run with ./compile_run_ss.bat, which assumes you have the g++ compiler.

Boilerplate:

The shader graph's 'boilerplate' defines the appended 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, which are used as input to terminal boilerplate code in order to determine final shader functionality.

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)

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 above.

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.

Nodes And Features:

A node represents an operation (or multiple operations) performed on a set of output results from other node operations.

A node can have input pins, which connect to output pins of other nodes and which represent the inputs of the operations.

Output pins represent the results of the node's operations.

Input pins will attempt to create default values when they are not connected to valid

Pins each possess types, which can be seen by hovering over a pin, but defaults 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.

Pins will also NOT connect if such a connection would create a directed cycle in the graph.

Finally, some nodes have a button at the bottom which will attempt to display their primary output result as the final color of a shader.

Shaders for intermediate displays and the final shader are not compiled automatically, see below. Any intermediate display, which is no longer representative (due to changes not yet built) of the current graph, will have a RED background.

CONTROLS

- Hold Space and Left Click to drag the camera
- Hold Left Click over the non-pin sections of a node to drag it
- Hold Left Click over a pin to drag a connection line to another pin to connect it to
- Left Click on the intermediate display button of a node to toggle its display
- Right Click to open the ADD NODE MENU to search for and add a node
- Press Delete when hovering over a pin to disconnect it from all pins
- Press Delete when hovering over a node to disconnect its pins and delete it

WINDOW MENU

- Build Vertex Shader : build the vertex shader and relink it to all fragment shaders
    - Also makes an autosave of the current vertex shader's code
- Build Fragment Shader : build the fragment shader and update all intermediate displays
    - Also makes an autosave of the current fragment shader's code
- Controls: Show an abbreviated list of controls
- Credits : Display credits

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 active nodes from all connected nodes.

Changing a parameter's name will cause ALL shaders to become dirty and the terminal shaders to be reset. This is to prevent linker errors from mismatches of uniform name and type between vertex and fragment 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.

Shaders will not be compiled out with these images, but can images easily be bound to/utilized by the shader using the **parameter** created.

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' should be a valid GLSL type and the argument/variable name desired; make sure characters like ',' are not accidently included in the name token.