ShaderLoader

ShaderLoader is a FreeframeGL effect plugin that can be used to load shader files produced from "*GLSL Sandbox*" and "*ShaderToy*".

Files can be created by copy/paste of shader code and saving as text files. The concept is similar to the requirements of the websites. There is a fixed vertex shader and a fixed range of pre-defined uniform variables for the fragment shader.

Not all of the uniform variables are supported, two texture inputs are available and there is no audio input, so many of the shaders on the websites will not work. Nevertheless, a large number can be used either as-is or with minor modification. Mouse interaction is mapped to Freeframe parameters and controls that the host provides. A separate input colour is available for additional control.

Installation

"*ShaderLoader.dll"* must then be copied into the plugin folder for the host application.

For "*Resolume Avenue*" or "*Resolume Arena*", you can copy them to the default plugin "vfx" folder. Or you can define specific folders for effects.

For "*Isadora*" copy to the Program files "..\Common Files\Freeframe" folder. Create this folder if it does not exist.

Magic allows "additional module folders" to be defined so using the "..\Common Files\Freeframe" folder is a good idea.

"*SpoutPanel.exe*" from the *Spout* project, is provided with *ShaderLoader* and should be copied to the same folder as the host executable to enable the file selection and warning dialogs. The latest Version 2.03 (after 9th January 2015) is required.

Shader path

Shaders can be saved in any folder, but if you want to use a simple name entry as described below, they have to be in a known folder.

This can be either the same folder as the *ShaderLoader* plugin or in a sub-folder under that folder called "Shaders".

For example you might have put the *ShaderLoader* plugin in "..\Program files\Common Files\Freeframe" folder. If so, you can create a sub-folder "..\Program files\Common Files\Freeframe\Shaders" and put all the shader files in that. The same applies for any folder where the *ShaderLoader* plugin is located.

Operation

Although *ShaderLoader* is an "effect" plugin, it will work without any texture input if the host allows it. The plugin allows for two texture inputs. Hosts tested : *Magic*, *Isadora*, *Resolume*

For *Magic*, connect one or two texture inputs or none. The textures are used if the shader requires them or are ignored if not.

Resolume will also work with no texture source or with one or two sources and similarly the textures are either used or not depending on whether the shader requires them.

For *Isadora*, the plugin will not work without a texture input. Both texture inputs have to be connected to a texture source.

Loading shader files

A number of *GLSL Sandbox* and *ShaderToy* examples are included with the distribution. These can be loaded using the "Select" button which opens a file selection dialog. After a selected shader is loaded, the file path is saved in the registry so that when the plugin starts again, it will pick up this shader file if there is no entry already in the "Shader Name" field.

Shader entry

If you want different instances of *ShaderLoader* to load different shaders when they start, the "Shader Name" field must be used to enter the name of the file you want to load. This entry is saved by the host and used when the plugin starts instead of the filepath saved in the registry.

When a name is in the entry field, the file selection dialog is disabled and you have to clear the name entry before using it. This way there is no confusion as to which method is being used and whether the file will be retrieved by the host of not. It is not possible for a plugin to write the file name back into the host entry field.

Shader name entry

The entry can either be the full path of the shader file or just the name of the file. A file extension is not required. It is assumed to be ".txt".

If a name is entered, the file has to be in either the same folder as the *ShaderLoader* plugin or in a sub-folder under that folder called "Shaders".

The *ShaderLoader* plugin folder is searched first, then the "Shaders" folder if that exists. If the file is not found it will simply fail to load and the result will be black output.

Shader path entry

The full path of a shader file can also be entered, so that shader files can be loaded from any location.

If you want to copy the full path of a shader file into the clipboard to make entry easier, use either an Explorer window or the File Open dialog. Hold the Shift key down, RH click on the file and "Copy as path". The path can then be pasted into the "Shader Name" entry field. This method puts the path in quotes but ShaderLoader allows for that. You have top click "Update" after entering a name or a path.

Editing shader files

Shaders can be edited by using the "Edit" button. The file is opened with whatever program is associated with the "txt" extension, usually NotePad, or it might be useful to use an editior you prefer such as *NotePad++* which also has GLSL specific highlighting available. After editing and saving the file, use the "Reload" button to load the changed file.

The editor will not open if the File Selection dialog is open, but files can be selected while the editor is open. So you can edit and compare different shader files.

Creating a shader file

Shader files are created by copy and paste from the code shown by *GLSL Sandbox* or *Shadertoy*. Simply paste the contents into Notepad or a similar text editor and save as a text file. Be careful to avoid line wrap which can break lines of code.

A few shortcut hints might be useful. Once you have the web shader running, locate the cursor at the very beginning of the code an click once. CTRL-SHIFT-END should then highlight all the code. Copy this to the clipboard. RH click - Copy or CTRL-C. Now paste to a text editor.

It is often useful to have ShaderLoader running at the same time. Load a shader file and click Edit. Then "File New" and paste with RH click paste or CTRL-V. Then save the new file. Now you can load it straight away with ShaderLoader.

Many of these shaders fail to work. First check that a *ShaderToy* file does not contain uniform variables declared in *GLSL Sandbox* while at the same time using *ShaderToy* uniforms within the code (see further below). Also check that copy/paste has not resulted in line wrap.

Any shader requiring audio input will not work and many require pre-defined textures or more than two textures. Also not all pre-defined uniform variables are supported (see below).

Changing the speed

The "Speed" control allows the animation rate of the shader to be controlled in the same way as for the speed controls in *GLSL Sandbox* or *ShaderToy*. *ShaderLoader* time is calculated internally rather than using FFGL time input from the host.

Position control

ShaderLoader does not react on mouse movement, but parameters can be used in an equivalent way to the Mouse input variables within the shader.

Mouse X & Mouse Y are passed through from the first two parameters of the plugin.

A further two "mouse" coordinates are passed through from the next two parameters of the plugin Mouse Left X and Mouse Left Y. These are equivalent to the left button pressed. But take note of differences and units within the shader as explained below.

Shader uniform variables

Uniform variables link data from the plugin to the shader. *GLSL Sandbox* and *ShaderToy* have different naming conventions and either of these can be used.

For *GLSL Sandbox,* the uniform variables are defined in the fragment shader. For *ShaderToy,* they are not defined in the shader file itself, but are added internally when the file is loaded. To distinguish the file types, the exact string "uniform float time;" has to be present in a *GLSL Sandbox* file. Red highlighting indicates that the uniform variable is not supported by the plugin.

GLSL Sandbox uniforms

```
uniform float time; // Current time in seconds
uniform vec2 mouse; // Normalized mouse position. (between 0.0 and 1.0)
uniform vec2 resolution; // Screen (Viewport) resolution.
uniform vec2 surfaceSize; // size of the visible area of the virtual surface.
uniform vec2 surfacePosition; // position of the current fragment on the virtual surface
uniform sampler2D backbuffer; // sampler for backbuffer
uniform sampler2D texture; // sampler for texture
```

For *GLSL Sandbox*, "mouse" coordinates are normalised 0 - 1 (X & Y mouse). The "surfaceSize" uniform allows a further two "mouse" coordinates to be passed through from the next two parameters of the plugin, but these are in pixels (X & Y mouse left), so have to be divided by the resolution to normalize to 0-1.

GLSL Sandbox uses a rendering process internally, rendering to the backbuffer, then the frontbuffer and then swapping buffers. This process is not supported by the *ShaderLoader* plugin, so the "backbuffer" and "texture" uniforms are not supported for their original function but are mapped to the first texture instead. In order to allow for multiple input textures, the following uniform variable names are used :

```
uniform sampler2D tex0;
uniform sampler2D tex1;
uniform sampler2D tex2;
uniform sampler2D tex3;
```

tex0 is the first input texture and tex1, the second. Presently only two input textures are supported. These are equivalent to the ShaderToy "iChannel" uniforms.

ShaderToy uniforms

```
uniform float iGlobalTime; // current time (in seconds)
uniform vec4 iMouse; // xy are pixel coords (Left button down). zw are the click pixel.
uniform vec3 iResolution; // the rendering resolution (in pixels)
uniform vec4 iDate; // (year, month, day, time in seconds)
uniform float iChannelTime[4]; // channel playback time (in seconds)
uniform vec3 iChannelResolution[4]; // channel resolution (in pixels)
uniform sampler2D iChannel0; // sampler for input texture 0.
uniform sampler2D iChannel1; // sampler for input texture 1.
uniform sampler2D iChannel2; // sampler for input texture 2.
uniform sampler2D iChannel3; // sampler for input texture 3.
```

Normally *ShaderToy* acts on mouse movement with the left button down, but in this case the first two (xy) are mapped to the plugin parameters (X & Y mouse). The next two (zw) are usually the click coordinates but here they are mapped to the next two controls (X & Y mouse left). *ShaderToy* mouse coordinates are all in pixels, so have to be divided by the resolution to normalize to 0-1.

ShaderLoader uniforms

Additional uniform variables are possible, but each has to be mapped to the plugin user controls. Currently one additional uniform allows a colour to be sent to the shader. Naturally the values do not need to be used for colour, so the four floats are available for other purposes.

uniform vec4 inputColour; // (red, green, blue, alpha)

This uniform is added internally for a ShaderToy shader file, but for a GLSL Sandbox file it has to be entered into the file itself.

Examples

A number of *GLSL Sandbox* and *ShaderToy* examples are included with reference in the source to the link and credit to the source and author where possible. Others with "_example" are testing shaders which might be useful. Those with "_filter" take a texture input and operate on that texture.

Contact and License

Contact us at spout.zeal.co