GLSL Abstractions Library for Pd/GEM

pseudonym=glsl12345

Marius Schebella
Salzburg University of Applied Sciences
Urstein Süd 1
5421 Puch bei Hallein
+43.650 808 1612
marius.schebella@gmail.com

ABSTRACT

This paper introduces a Pd library that is built around the GL Shading Language (GLSL[1]). The object classes are a set of Pd abstractions built as a wrapper around basic GEM objects. They allow better and easier handling of GLSL functionality. With the GLSL abstractions it is possible to easily deal with multiple textures and models and chain several shader programs together.

The library also contains a set of example shaders, help files and a step-by-step tutorial that explains the use of GL Shaders within the Pd/GEM environment.

Keywords

Pd, GEM, GLSL, OpenGL Shading Language.

1.INTRODUCTION

GLSL is a high level programming language used to write GPU accelerated graphic programs. GLSL programs consist of a vertex part ("vertex shader") and a texture/pixel part ("fragment shader"). In GEM the object classes [glsl_fragment], [glsl_vertex] and [glsl_program] are used to load and link vertex and fragment shaders to a GLSL program.

Dealing with several textures, and chaining shaders in Pd is possible using [gemframebuffer] and referencing information provided by [pix_texture].

The GLSL abstraction library only deals with the GLSL objects and does not support other Pd shader objects like [fragment_program] and [vertex_program], two object classes which allow to apply ARB fragment shaders.

2.GLSL RELATED OBJECT CLASSES IN GEM

2.1.Glsl Object Classes

[glsl_vertex] loads a vertex program, and [glsl_fragment] loads a fragment program to the graphics card driver. Both programs have to be linked using [glsl_program]. With [glsl_program] it is possible to pass uniform parameters to the shaders, by sending messages to the object's inlet or by mapping different GEM textures to GLSL textures (declared by Sampler2D or Sampler2DRect). This mapping is done by assigning a GEM texunit ID to the name of a Sampler2D variable.

[glsl_program] also allows to print out the list of uniform variables.

2.2[pix_texture] and Multiple Textures

The [pix_texture] object class allows to assign texunit IDs to GEM textures. Texunits are applied by sending the message “texunit” plus a number to [pix_texture]. This is necessary to refer a texture to a shader (via [glsl_program]) that is not directly connected via patch cords. [pix_texture] outputs this ID along with the size, and the color space of the texture. It is also possible to reference a texture ID by sending it to the right inlet of the object. The same method is used to reference the content of a gemframebuffer object to a shader.

[pix_texture] also sets the mode (rectangular or non rectangular) in which the texture is stored.

Multiple textures can be used in a shader by mapping more than one texunit to a shader. [glsl_program] will accept a message like “MyTex 3”, where “MyTex” is the name of the variable in the shader program and “3” is the id number that is passed to [pix_texture] as the argument to the texunit message.[1]

2.3[gemframebuffer] Object Class

[gemframebuffer] is the key object in the process of applying fragment shaders to a whole GEM render chain or for chaining several shaders together in sequence.

Figure 1 shows a schematic layout of a GEM render chain including [gemframebuffer] and a shader. With the latest version of GEM it is possible to set frustrum and other viewing options of [gemframebuffer] via messages.

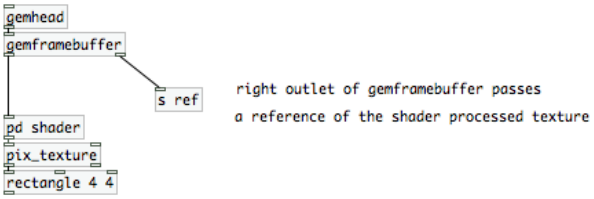

Fig.1: A shader is rendered into a [gemframebuffer].

3.LIBRARY OBJECT CLASSES

The idea of the GLSL abstractions library is to connect objects like other GEM objects but pass texture ID references along

PdCon09, July 19-26, , 2000, São Paulo, SP, Brazil.

[1] As of now, it is not possible to access the correct texture coordinates of textures other than texunit 0. This is a pending bug[

additionally or instead or gem_state pointers. Some of the features were inspired by Max/MSP Jitter objects like [jit.gl.slab].

3.1Glsl Input and Output

There are several ways to feed textures into the glsl chain. [glsl.pix2tex] will take the output of a pix_object and output its texture reference. [glsl.tex2pix] works the other way round it can be connect to the output of a glsl object and reads the framebuffer back to be used with further pix_objects (Fig. 2).

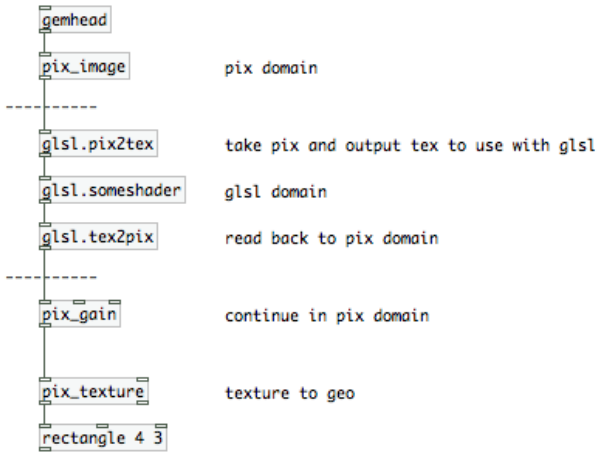

Fig.2: Showing the use of [glsl.pix2tex] and [glsl.tex2pix]

It is also possible to connect the output of a glsl object directly to a pix_object by sending an "*output pix*" message to the glsl object (Fig.3).

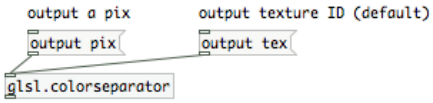

Fig.3: Send output message to switch between pix and tex output.

A group of GLSL library objects also provides wrappers for images, films and video.

[glsl.image], [glsl.video], [glsl.movie] and [glsl.film] load files or live video and directly map them to shaders. They all render their textures to a framebuffer or optionally take an argument to specify the texunit. It it also possible to set the dimensions of the framebuffer according to the image size or adapt it to the settings of the parent glsl object.

3.2Glsl Effects

There are a group of glsl object classes that do predefined computations like glsl.colorseparator, glsl.duotone, glsl.lighttunnel. Some of them duplicate the functionality of pix_objects. Some of them are ported from the v001 shader library by vade[2]. The v001 library contains a basic set of shaders originally developed for Max/MSP Jitter.

The effects part of the library is a work in progress, but since glsl shaders are easy to write and are compiled at runtime which makes it easy to deploy I also expect other people to add on to the effects side of the glsl library.

3.3[glsl.shader] and [glsl.shader2]

[glsl.shader] is a generic shader object. It laods vertex and/or fragment shaders. If you apply only one of the shaders the other one will be bypassed. [glsl.shader2] works similar but allows to feed two textures directly into the object.

3.4Geometry and Vertex Shaders

The library abstractions introduced so far tried to go with the Pd (aka dataflow) way of doing things. With geometry[2] and vertex shaders this is difficult. It would mean to first create the vertices (geometry shader), then transform them in a vertex shader and then apply texture to it. For now the vertex shaders can be applied at any time in the render chain.

3.5Additional Object Classes

[glsl.videoplane] maps a texture onto a rectangle.

[glsl.gemwin] is an optional gemwin wrapper that deploys the window size and easy fullscreen swithing.

[glsl.handle] is a mouse handle to rotate a model or GEM scene

[glsl.gemhead] is a gemhead wrapper that automatically renders to gemframebuffer plus optionally also to the gemwin rendercontext.

3.6General Features of GLSL objects

The abstractions of the glsl library can easily be chained together (Fig.4).

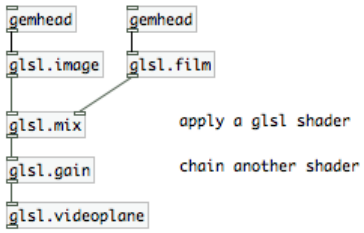

Fig.4:Chaining two shaders.

Most GLSL object classes use optional @attributes to pass arguments. Settings can also be sent via messages to the left inlet. There is a special storage mechanism for attributes that need to be passed to the shader, because this cannot be done on loadbang, but only after the gemwin is created and the shader running.

Attributes are optional and can be given in any order. They support settings like dimensions or dimensions of the texture and gemframebuffer, respectively. Common attributes are @dim and @quality for the texture dimensions and quality settings, @txu to define a texunit and @rect for rectangular vs non rectangular texture mode. The @adapt 1 message will use the parent's object dimension settings.

In [glsl.shader] and [glsl.shader2] @vert is used to load vertex programs, @frag for fragment programs, @param to send uniform parameters to the GLSL program.

4.ACKNOWLEDGMENTS

My thanks go to Cyrille Henry, vade, Chris Clepper and IOhannes Zmölnig..

[2] The [glsl_geometry] object class in GEM is still under development.

5.REFERENCES

[1] OpenGL Shading Language. Gold Standard Group. Mar 15, 2009. http://www.opengl.org/documentation/glsl.

[2] v001, by Vade. Mar 15, 2009. http://001.vade.info.