Modular Graphics with CLOS

@Shinmera

https://shinmera.com

Modern Graphics

From Old to New

- CPU <-> GPU sync is slow
- Upload as much ahead of time as possible
- When drawing, everything is done GPU-side
- Customise drawing behaviour through shaders

GLSL

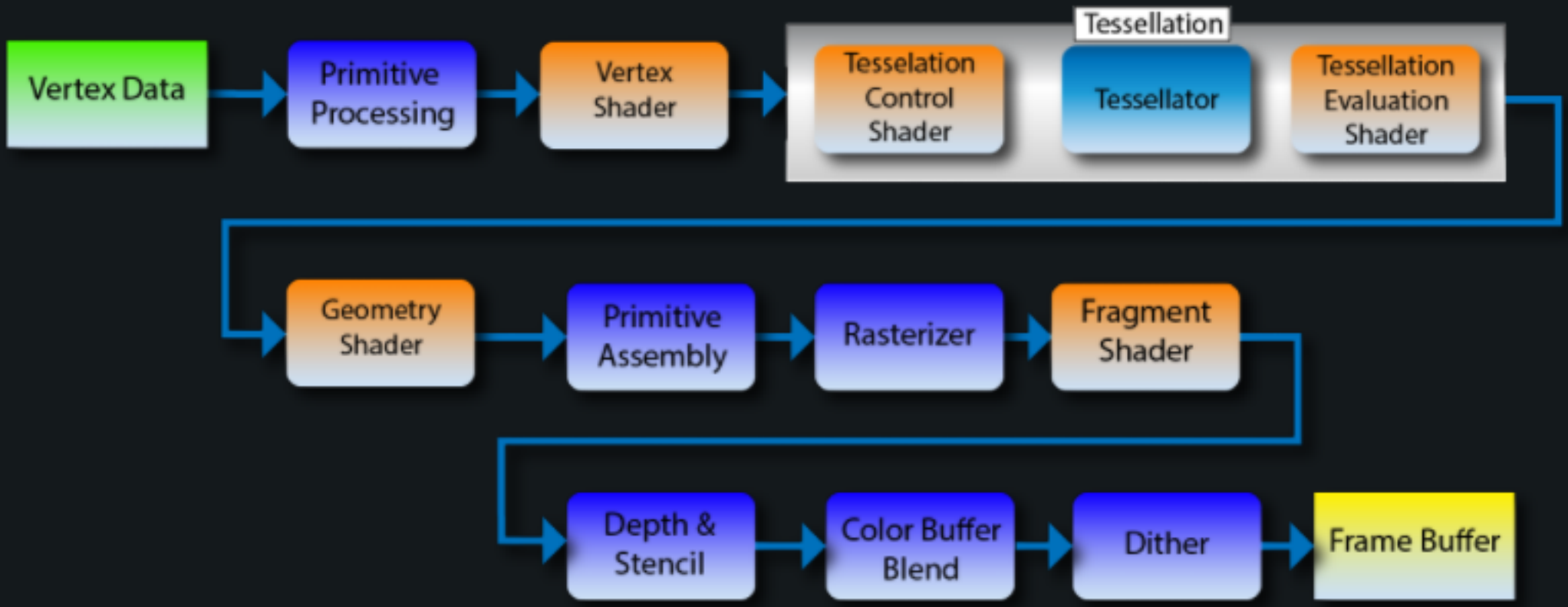
- C-like language for GPU programs
- Specifically about graphics
- Pass strings to driver, compiles to GPU

GLSL Example

```
out vec4 color;

void main() {
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow(tan((p.y+p.x)/50)+1, 20)), 0, 1);
  color.rgb *= -p.z/1.5+0.9;
}
```

OpenGL Render Pipeline



Here's the Problem

- Want to define behaviour modularly
- Want to tie draw behaviour to objects
- But, only one program per stage at a time

Here's a Solution

- Parse GLSL into AST
- Use semantic analysis to merge programs
- Emit single program with combined behaviour
- Use CLOS to attach shaders to classes

GLSL-Toolkit

- Implements a full GLSL 4.1 parser
- GLSL code-walker for semantic analysis
- User just calls MERGE-SHADER-SOURCES

GLSL-Toolkit

```
(glsl-toolkit:parse "out vec4 color;

void main() {
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow(tan((p.y+p.x)/50)+1, 20)), 0, 1);
  color.rgb *= -p.z/1.5+0.9;
}")

;=>

(GLSL-TOOLKIT:SHADER
 (GLSL-TOOLKIT:VARIABLE-DECLARATION (GLSL-TOOLKIT:TYPE-QUALIFIER :
  (GLSL-TOOLKIT:TYPE-SPECIFIER :VEC4) "color" GLSL-TOOLKIT:NO-VALU
 (GLSL-TOOLKIT:FUNCTION-DEFINITION
  (GLSL-TOOLKIT:FUNCTION-PROTOTYPE GLSL-TOOLKIT:NO-VALUE
   (GLSL-TOOLKIT:TYPE-SPECIFIER :VOID) "main")
  (GLSL-TOOLKIT:COMPOUND-STATEMENT
   (GLSL-TOOLKIT:VARIABLE-DECLARATION GLSL-TOOLKIT:NO-VALUE
    (GLSL-TOOLKIT:TYPE-SPECIFIER :VEC4) "p" GLSL-TOOLKIT:NO-VALUE
    "gl_FragCoord")
   (GLSL-TOOLKIT:ASSIGNMENT
    (GLSL-TOOLKIT:MODIFIED-REFERENCE "color"
     (GLSL-TOOLKIT:FIELD-MODIFIER "gb"))
    :*= ...))))
```

Where's the Lisp?

CLOS & MOP Primer

- Classes allow multiple inheritance
- Class behaviour is defined by metaclasses
- MOP can attach new information to classes

Connecting Shaders and Classes

- Metaclass that holds shader sources
- On inheritance, sources are merged
- CLOS' class-precedence defines merge order

Let's Make Some Tea

```
(define-shader-subject teapot (slide-subject)
  ())

(define-handler (teapot tick) (ev dt tt)
  (incf (vz (rotation teapot)) dt))
```

A Fragment Shader

```
(define-shader-subject teapot (slide-subject)
  ())

(define-class-shader (teapot :fragment-shader)
  "out vec4 color;

void main() {
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow(tan((p.y+p.x)/50)+1, 20)), 0, 1);
  color.rgb *= -p.z/1.5+0.9;
}")
```

A New Mixin

```
(define-shader-entity striped-entity () ())

(define-class-shader (striped-entity :fragment-shader)
  "out vec4 color;

void main() {
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow(tan((p.y+p.x)/50)+1, 20)), 0, 1);
  color.rgb *= -p.z/1.5+0.9;
}")
```

Lisp Source:

```
(define-shader-subject teapot (slide-subject
;;                             striped-entity
;;                             textured-entity
;;                             colored-entity
                               )
  ())
```

Fragment Shader:

```
#version 330 core
out vec4 color;

void _GLSLTK_main_1(){
  color = vec4(1.0, 1.0, 1.0, 1.0);
}

void main(){
  _GLSLTK_main_1();
}
```

Lisp Source:

```
(define-shader-subject teapot (slide-subject
                               striped-entity
;;                               textured-entity
;;                               colored-entity
                               )
  ())
```

Fragment Shader:

```
#version 330 core
out vec4 color;

void _GLSLTK_main_1(){
  color = vec4(1.0, 1.0, 1.0, 1.0);
}

void _GLSLTK_main_2(){
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow((tan(((p.y + p.x) / 50)
  color.rgb *= ((-p.z / 1.5) + 0.90000004);
}

void main(){
  _GLSLTK_main_1();
  _GLSLTK_main_2();
}
```

Lisp Source:

```
(define-shader-subject teapot (slide-subject
                               striped-entity
                               textured-entity
;;                               colored-entity
                               )
  ())
```

Fragment Shader:

```
#version 330 core
out vec4 color;

void _GLSLTK_main_1(){
  color = vec4(1.0, 1.0, 1.0, 1.0);
}
in vec2 texcoord;
uniform sampler2D texture_image;

void _GLSLTK_main_2(){
  color *= texture(texture_image, texcoord);
}

void _GLSLTK_main_3(){
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow((tan(((p.y + p.x)
  color.rgb *= ((-p.z / 1.5) + 0.90000004);
}

void main(){
  _GLSLTK_main_1();
  _GLSLTK_main_2();
  _GLSLTK_main_3();
```

Lisp Source:

```
(define-shader-subject teapot (slide-subject
                               striped-entity
                               textured-entity
                               colored-entity
                                 )

  ())
```

Fragment Shader:

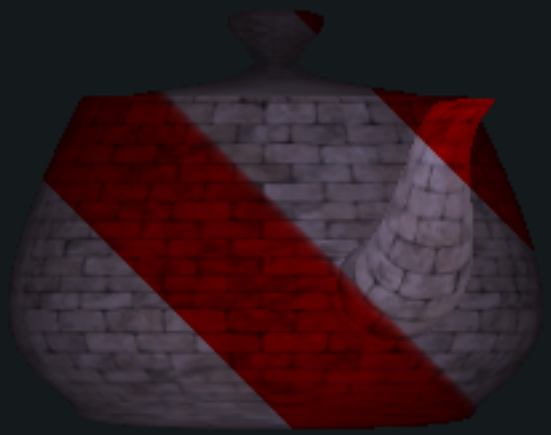
```
#version 330 core
out vec4 color;

void _GLSLTK_main_1(){
  color = vec4(1.0, 1.0, 1.0, 1.0);
}
uniform vec4 objectcolor;

void _GLSLTK_main_2(){
  color *= objectcolor;
}
in vec2 texcoord;
uniform sampler2D texture_image;

void _GLSLTK_main_3(){
  color *= texture(texture_image, texcoord);
}

void _GLSLTK_main_4(){
  vec4 p = gl_FragCoord;
  color.gb *= clamp(abs(pow((tan(((p.y + p.x) / 50)) + 1), 20)), 0, 1);
  color.rgb *= ((-p.z / 1.5) + 0.90000004);
}
```



Lisp Source:

```
(define-shader-subject teapot (slide-subject
;;                             striped-entity
                              textured-entity
;;                             colored-entity
                              )
  ())
```

Fragment Shader:

```
#version 330 core
out vec4 color;

void _GLSLTK_main_1(){
  color = vec4(1.0, 1.0, 1.0, 1.0);
}
in vec2 texcoord;
uniform sampler2D texture_image;

void _GLSLTK_main_2(){
  color *= texture(texture_image, texcoord);
}

void main(){
  _GLSLTK_main_1();
  _GLSLTK_main_2();
}
```

Great, What's the Catch?

- Some shaders not automatically mergeable
- Currently very primitive strategy
- How effects are combined can be unintuitive
- Shaders might need to be adapted

Future Ideas

- Verify correctness of GLSL code
- Optimisation passes
- Offer automatic correction for problems
- More exploration of composition strategies

Questions