CS Go
Shaders Without Boilerplate

Jett Andersen,
JeanHeyd Meneide,
David Naveen Dhas Arthur

CS Go

- Simplified GPU programming for game engine developers
- A concise wrapper for OpenGL Compute Shaders
- Averaging two images on the GPU and displaying the result:

```
auto average(csgo::dsl::image2d<glm::vec4> in1, csgo::dsl::image2d<glm::vec4> in2) {
    using namespace csgo::dsl;
    image2d<glm::vec4> x = ( in1 + in2 ) / 2;
    return x;
}

void run_average() {
    GLuint size = 32;
    csgo::program p(average, { {size, size} }, true);
    csgo::image2d_io<glm::vec4> in1(std::vector<glm::vec4>(size * size, glm::vec4(1)), size);
    csgo::image2d_io<glm::vec4> in2(std::vector<glm::vec4>(size * size, glm::vec4(1, 0, 0, 1)), size);

    std::tuple<csgo::image2d_io<glm::vec4>> results = p(in1, in2);
    csgo::display::image(std::get<0>(results));
    while (true);
}
```

OpenGL

Why is it our backend?

- OpenGL is a portable library for 3D graphics
- OpenGL is ubiquitous in game development
- OpenGL has GLSL
  - Powerful language for operations on the GPU
- Lowest level API of *Alien: Isolation* engine
  - Look at those shadows!

OpenGL

But...

- OpenGL has too much boilerplate!
- This code just renders a fullscreen texture
  - Missing fragment and vertex shaders

```
template<typename T>
static void image(const image2d_io<T>& input)
{
    glsl::compiler::make_context();

    GLuint vertexArrayID;
    gl::GenVertexArrays(1, &vertexArrayID);
    gl::BindVertexArray(vertexArrayID);

    GLFWwindow *window = glfwGetCurrentContext();

    GLuint handle = gl::CreateProgram();
    GLuint frag = gl::CreateShader(gl::FRAGMENT_SHADER);
    glsl::compiler::compile(handle, frag, getFragShader());
    GLuint vert = gl::CreateShader(gl::VERTEX_SHADER);
    glsl::compiler::compile(handle, vert, getVertShader());

    gl::Clear(gl::COLOR_BUFFER_BIT | gl::DEPTH_BUFFER_BIT);
    gl::UseProgram(handle);

    gl::EnableVertexAttribArray(0);
    std::vector<GLfloat> quad = getQuad();
    GLuint positions;
    gl::GenBuffers(1, &positions);
    gl::BindBuffer(gl::ARRAY_BUFFER, positions);
    gl::BufferData(gl::ARRAY_BUFFER, (GLint)quad.size() * sizeof(GLfloat), quad.data(), gl::STATIC_DRAW);
    gl::VertexAttribPointer(0, 3, gl::FLOAT, gl::FALSE_, 0, nullptr);

    gl::EnableVertexAttribArray(1);
    std::vector<GLfloat> quad_uvs = getUVs();
    GLuint uvs;
    gl::GenBuffers(1, &uvs);
    gl::BindBuffer(gl::ARRAY_BUFFER, uvs);
    gl::BufferData(gl::ARRAY_BUFFER, (GLint)quad_uvs.size() * sizeof(GLfloat), quad_uvs.data(), gl::STATIC_DRAW);
    gl::VertexAttribPointer(1, 2, gl::FLOAT, gl::FALSE_, 0, nullptr);

    gl::ActiveTexture(gl::TEXTURE0);
    gl::BindTexture(gl::TEXTURE_2D, input.get_texture_ID());
    gl::Uniform1i(gl::GetUniformLocation(handle, "tex"), 0);

    gl::DrawArrays(gl::TRIANGLE_STRIP, 0, (GLint)quad.size());

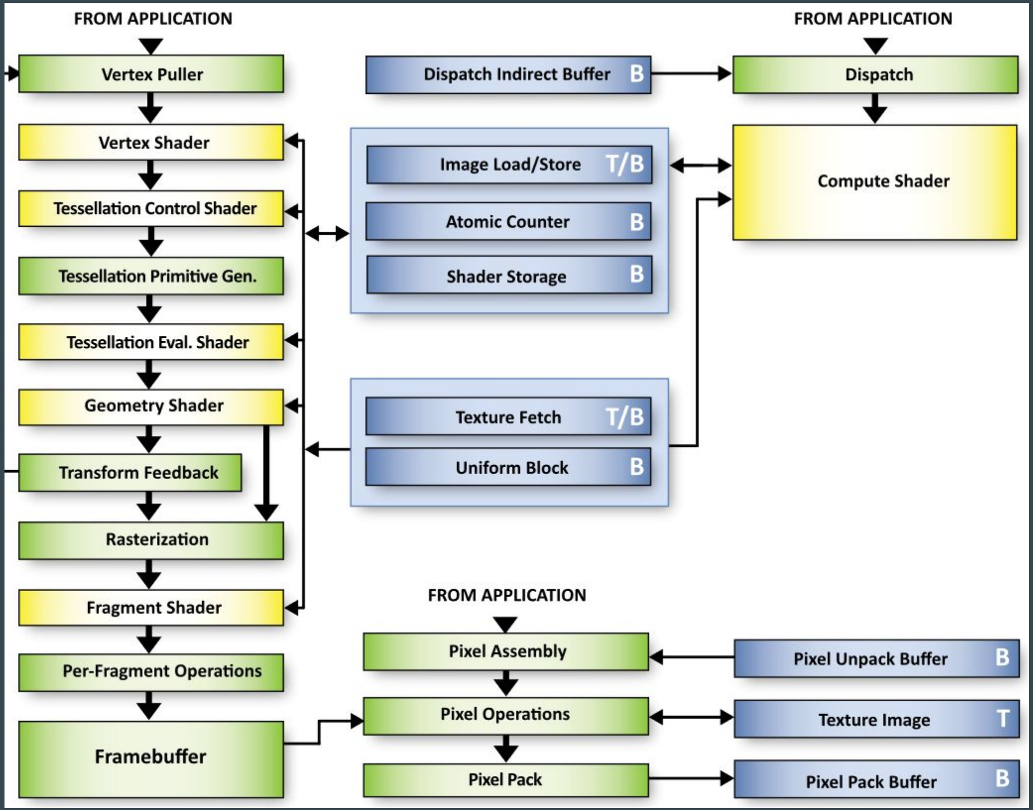
    gl::DisableVertexAttribArray(0);
    gl::DisableVertexAttribArray(1);

    glfwSwapBuffers(window);

    gl::DeleteShader(frag);
    gl::DeleteShader(vert);
    gl::DeleteProgram(handle);
}
```

OpenGL Compute Shaders

- Independent from the rendering pipeline
- BUT its input and output can be used for rendering
- Powerful enough to rewrite the entire pipeline with just compute shaders



Revisiting the Average

- The average function contains the CS Go DSL (domain-specific language)
- The run_average function specifies specific inputs and outputs

```
auto average(csgo::dsl::image2d<glm::vec4> in1, csgo::dsl::image2d<glm::vec4> in2) {
    using namespace csgo::dsl;
    image2d<glm::vec4> x = ( in1 + in2 ) / 2;
    return x;
}

void run_average() {
    GLuint size = 32;
    csgo::program p(average, { {size, size} }, true);
    csgo::image2d_io<glm::vec4> in1(std::vector<glm::vec4>(size * size, glm::vec4(1)), size);
    csgo::image2d_io<glm::vec4> in2(std::vector<glm::vec4>(size * size, glm::vec4(1, 0, 0, 1)), size);

    std::tuple<csgo::image2d_io<glm::vec4>> results = p(in1, in2);
    csgo::display::image(std::get<0>(results));
    while (true);
}
```

Types and IO Types

- The image2d type doesn't actually contain data, just supports GLSL operations

```
auto average(csgo::dsl::image2d<glm::vec4> in1, csgo::dsl::image2d<glm::vec4> in2) {
    using namespace csgo::dsl;
    image2d<glm::vec4> x = ( in1 + in2 ) / 2;
    return x;
}
```

- The image2d_io type contains an OpenGL texture, but doesn't support GLSL operations

```
    csgo::image2d_io<glm::vec4> in1(std::vector<glm::vec4>(size * size, glm::vec4(1)), size);
    csgo::image2d_io<glm::vec4> in2(std::vector<glm::vec4>(size * size, glm::vec4(1, 0, 0, 1)), size);

    std::tuple<csgo::image2d_io<glm::vec4>> results = p(in1, in2);
```

- Separates the dsl from C++
  - Enables different copy constructors

Constructing and Calling Programs

```
GLuint size = 32;
csgo::program p(average, { {size, size} }, true);
```

- A program is constructed from:
  - A function that takes uniforms and returns (a tuple of) uniforms
  - Size of output textures
  - Whether or not we need to create an OpenGL context

- A program is ran by passing image2d_io's for each image2d
  - Returns a tuple of image2d_io's that can be read from
  - Throws a runtime error if the input or the results tuple type are the wrong sizes

```
csgo::image2d_io<glm::vec4> in1(std::vector<glm::vec4>(size * size, glm::vec4(1)), size);
csgo::image2d_io<glm::vec4> in2(std::vector<glm::vec4>(size * size, glm::vec4(1, 0, 0, 1)), size);

std::tuple<csgo::image2d_io<glm::vec4>> results = p(in1, in2);
csgo::display::image(std::get<0>(results));
```

The Generated Code

- Structure:
  - Inputs
  - Outputs
  - Main function

- Generated by walking an AST constructed from the average function

```
#version 450 core

layout( binding = 0, rgba32f ) readonly uniform image2D _var5;
layout( binding = 1, rgba32f ) readonly uniform image2D _var6;

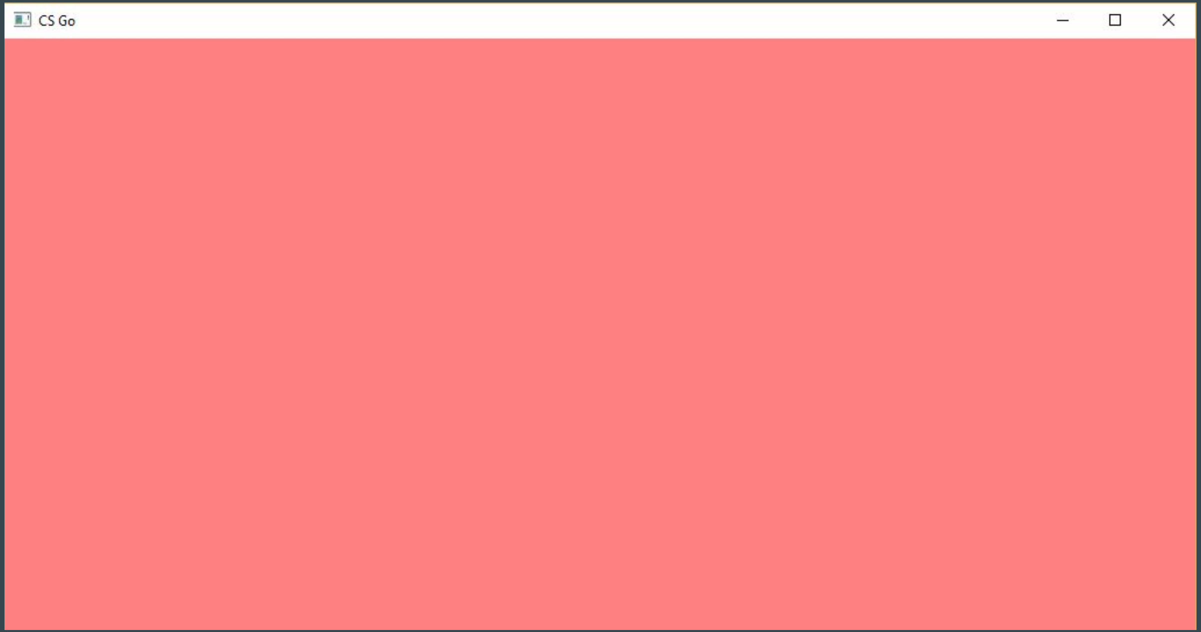
layout( binding = 2, rgba32f ) writeonly uniform image2D _var8;

layout( local_size_x = 16, local_size_y = 16 ) in;

void main() {
        imageStore( _var8, ivec2( gl_GlobalInvocationID.xy ), ( ( imageLoad( _var5, ivec2( gl_GlobalInvocationID.xy ) )
+ imageLoad( _var6, ivec2( gl_GlobalInvocationID.xy ) ) ) / 2 ) );
}
```

Displaying the Result

```
std::tuple<csgo::image2d_io<glm::vec4>> results = p(in1, in2);
csgo::display::image(std::get<0>(results));
while (true);
```



Pink, the average of red and white!

Behind the Scenes Pt. 1.1: The AST

- Operators on types in the dsl namespace generate expressions

```
template <typename L, typename R, meta::enable<meta::any<dsl::is_expression<L>, dsl::is_expression<R>>> = meta::enabler>
inline addition operator + (L&& l, R&& r) {
    dsl::addition op(
        dsl::make_unique_expression(std::forward<L>(l)),
        dsl::make_unique_expression(std::forward<R>(r))
    );
    return op;
}
```

- Also constructs a symbol table

```
struct symbol_table {
    std::unordered_map<id, std::size_t> variable_id_indices;
    std::vector<std::string> names;
    std::vector<std::reference_wrapper<const variable>> variables;

    // ...
```

Behind the Scenes Pt. 1.2: The AST and Assignment

- Assignment ruins everything
  - If new type return, violates user expectations and breaks syntax
  - Made everything really ugly

```
auto in1 = std::make_shared<ReadTexture<Float>>(vec1);
auto in2 = std::make_shared<ReadTexture<Float>>(vec2);

auto out = std::make_shared<WriteTexture<Float>>(size);

Program p({ in1, in2 }, { out });

p.set(out, (in1 + in2) + in1);
```

- “Blackhole”
  - Literally sucks up ‘statement’/‘terminating expression’ types
  - Prevents having to chain / propagate an expression through an assignment operator
  - consume(…) - eats expression into blackhole

```
template <typename T>
image_variable& operator= (T&& right) {
	consume(assignment(
		dsl::make_unique_expression(*this),
		dsl::make_unique_expression(std::forward<T>(right))
	));
	return *this;
}
```

Behind the Scenes Pt. 2: Code Generation

- Pure Agony
  - GLSL is C-like: C++ is... well, C++
    - There's a reason LLVM -> C printer was created, and then ***abandoned***, and then created...

- Example: imageStore, imageLoad
  - ```
    image2d<glm::vec4> x;
    x[gl_LocalInvocationID.xy] = glm::vec4(1.0, 0.0, 1.0, 1.0);
    ```
  - Becomes: `imageStore( x, gl_LocalInvocationID.xy, vec4( 1.0, 0.0, 1.0, 1.0 ) );`

- Expression Tree generated by C++: eww
  - Ordered in C++ as: gl_InvocationID -> .xy access -> x [...] -> = vec4 (...)
    ```

Behind the Scenes Pt. 3: Outputs

- Can have many outputs of different types from one program
  - image2d_io<float>, image2d_io<vec4>, etc.
- We return a generic type that is automatically castable

```cpp
template<typename... Args>
struct converter<std::tuple<Args...>> {
    template <std::size_t... Indices>
    static std::tuple<Args...> convert(std::index_sequence<Indices...>, const std::vector<texture_data>& data) {
        return std:make_tuple(converter<Ts>::convert(outputs[Indices])...);
    }

    static std::tuple<Args...> convert(const std::vector<texture_data>& data) {
        return convert(std::make_index_sequence<sizeof...(Args)>(), data);
    }
};
```

C++14!!

```cpp
std::tuple<csgo::image2d_io<glm::vec4>> results = p(in1, in2);
```

Cross Platform Deployment & Build

- Linux Makefile
- Visual Studio Project
- Available on GitHub!
  - `git clone` https://github.com/daviddhas/CS-gO.git
- A header-only library
  - But you must link against OpenGL 4.3+

The /* TODO */ List

- Add CS Go conditionals
  - Very tricky! We cannot use built in C++ if statement
  - Macros?!
- Add CS Go loops
  - Only necessary when number of iterations isn't an input to the program
- Support operations on integers and uniform buffers
- Better error-reporting and hand-holding
  - GLSL and OpenGL currently report errors as thrown errors

What We Learned

- Making DSLs in C++ is great for simple things
  - Real painful later on (conditionals?! ternary?! for loops, while loops, initializer lists oh my...)
  - Compiler differences (two-phase lookup, VC++ vs. g++/clang++)
- GLSL is really painful
  - operator[ ] doesn’t exist, everything written in terms of imageLoad / imageStore
- Compute Shaders are pretty powerful
  - We’ve only scratched the surface!
- Compiling for All Platforms is :(
  - Scrambling to find right libraries, GLFW installation problems, Windows OS file handles case fold text by default, Linux does not care

Thank you !!