Cs-gO Tutorial

Fast, portable GPU programming backed by OpenGL Compute Shaders

Team Members
JeanHeyd Meneide (jm3689@columbia.edu)
Jett Andersen (jca2136@columbia.edu)
David Naveen Dhas Arthur (da2647@columbia.edu)

Environment Settings:

- Linux
    - Linux Makefile
    - `git clone https://github.com/daviddhas/CS-gO.git`
    - A header-only library But you must link against OpenGL 4.3+
    - Navigate to the library directory

- Windows

    - 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+
    - Add the project as includes

Integrating Library:

Incase you have OpenGL already installed

1. Copy the includes folder in the library directory to your project working directory
2. Link the Cs-gO header file

If you don’t have OpenGL already installed

1. Copy the includes folder, the vendor folder and src folder in the library directory to your project working directory
2. Link the Cs-gO header file

Example:

The various library functions, types and programs are illustrated through examples below

1. Average

The average function contains the CS Go DSL (domain-specific language)

```
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 type doesn't actually contain data, just supports GLSL operations.

The image2d_io type contains an OpenGL texture, but doesn't support GLSL operations
Separates the dsl from C++. It also enables different copy constructors

```
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);

    auto result = std::get<0>(results).read();
    std::cout << result.size() << " " << size << std::endl;

    for (glm::vec4 v : result)
        std::cout << v.x << ", " << v.y << ", " << v.z << ", " << v.w << std::endl;

    csgo::display::image(std::get<0>(results));
    while (true);
}
```

The run_average function specifies specific inputs and outputs

Program

A program is constructed from a function that takes image2d's and returns a tuple of image2d's This contains the size of output textures. The program also determines whether or not we need to create an OpenGL context

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

A program is ran by passing image2d_io's for each image2d. It returns a tuple of image2d_io's that can be read from. The program 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));
```

Running the code:

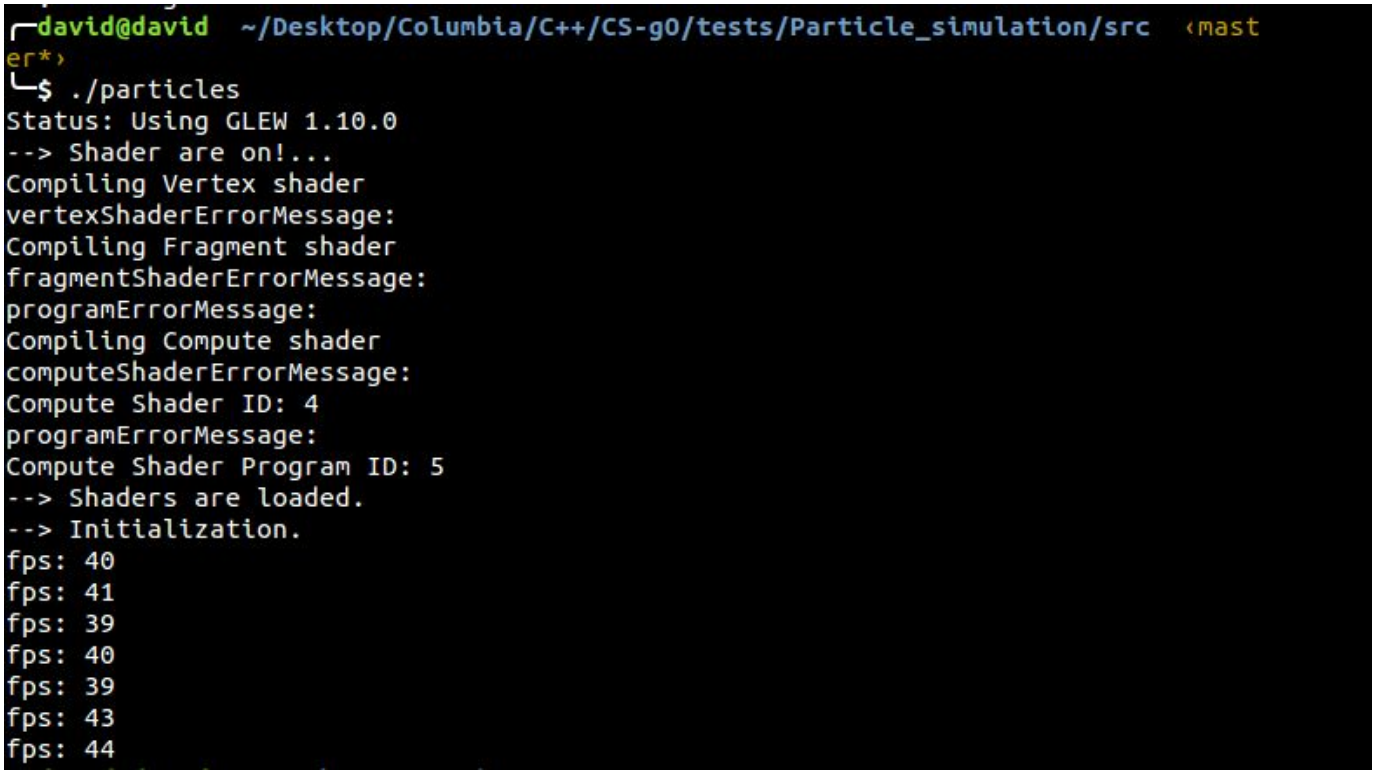
1. In case of Linux, the code can be compiled by using the linux make file provided in the library project directory
2. In case of Windows, the code can be compiled by using pressing Ctrl+F7 in the visual studio window

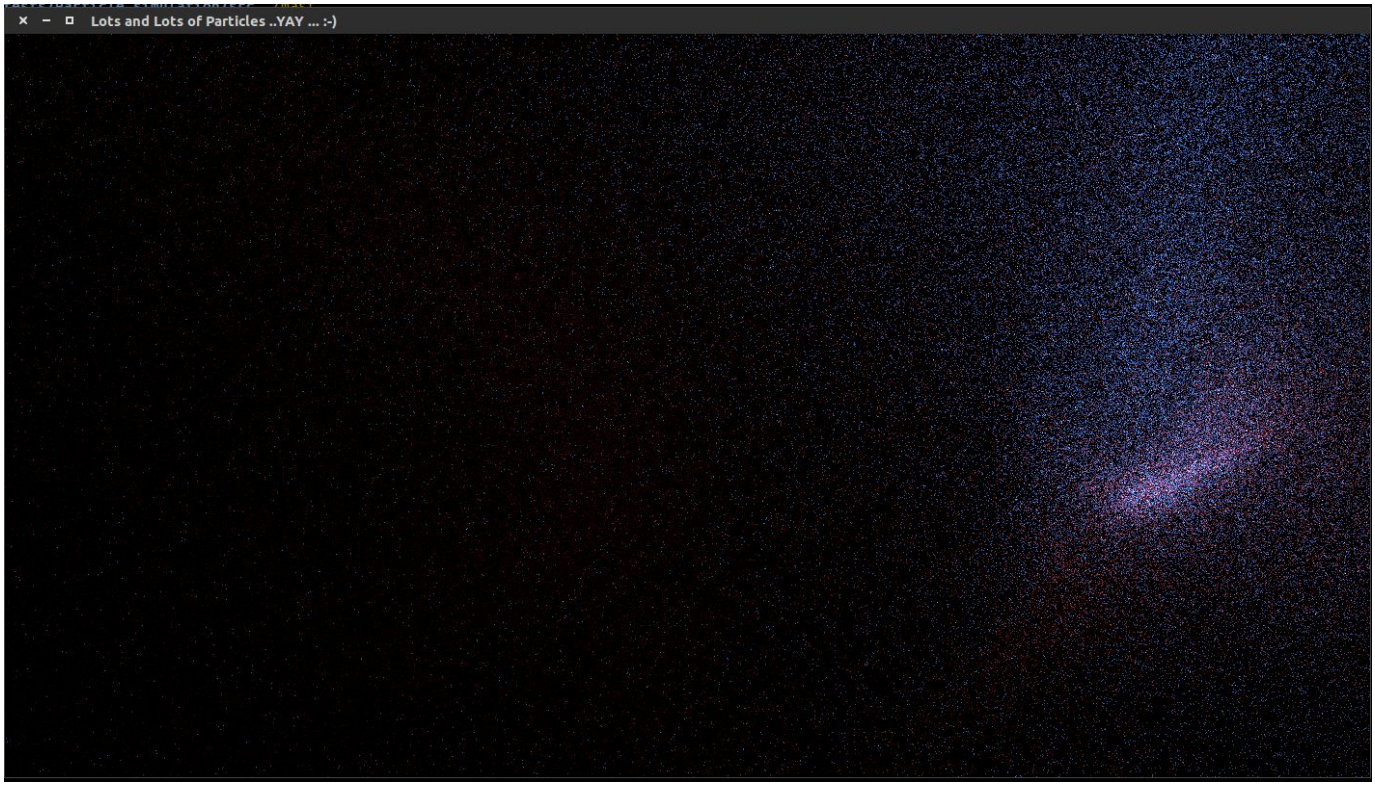
2. Particle Simulation

Each particle has a life span in which it fades from red (new born) to blue (about dying). When a particle dies, it respawns in its negated (x,y,z) position.

The movement itself is computed using an Euler integration method for each particle.

There are several global randomly moving attraction points. Particles are attracted and accelerated by these points.




Lots and Lots of Particles ..YAY ... :-)