Lab 2: Hello Shaders!

Publish | Edit | ⋮

GPR-200: Introduction to Modern Graphics Programming
Instructor: Daniel S. Buckstein
Lab 2: Hello Shaders!

Summary:
The OpenGL Shading Language (GLSL) is the shading language of OpenGL (GL). We will explore the modern graphics and shading techniques using GLSL and GLSL ES (GLSL for Embedded Systems) on a variety of platforms. Starting with this "open exploration" introductory assignment using the online platform **Shadertoy** (https://www.shadertoy.com/), we explore color and fragment shaders. Please refer to the latest official specifications for **GLSL 4.6** (https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.pdf) and **GLSL ES 3.2** (https://www.khronos.org/registry/OpenGL/specs/es/3.2/GLSL_ES_Specification_3.20.pdf) as needed.

Submission:
Start your work immediately by ensuring your coursework repository is set up, and public. Create a new main branch for this assignment. ***Please work in pairs (see team sign-up) and submit the following once as a team***:

1. Names of contributors
   e.g. **Dan Buckstein**
2. A link to your public repository online
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework.git**
   (note: this not a real link)
3. The name of the branch that will hold the completed assignment
   e.g. **lab0-main**
4. A link to your video (see below) that can be viewed in a web browser.
   e.g. <insert link to video on YouTube, Google Drive, etc.>

Finally, please submit a ***5-minute max*** demo video of your project. Use the screen and audio capture software of your choice, e.g. Google Meet, to capture a demo of your project as if it were in-

class. This should include at least the following, in enough detail to give a thorough idea of what you have created (hint: this is something you could potentially send to an employer so don't minimize it and show off your professionalism):

- Show the final result of the project and any features implemented, with a voice over explaining what the user is doing.
- Show and explain any relevant contributions implemented in code and explain their purpose in the context of the assignment and course.
- Show and explain any systems source code implemented, i.e. in framework or application, and explain the purpose of the systems; this includes changes to existing source.
- **DO NOT AIM FOR PERFECTION, JUST GET THE POINT ACROSS**. Please mind the assignment rubric to make sure you have demonstrated enough to cover each category.
- **Please submit a link to a video visible in a web browser, e.g. YouTube or Google Drive.**

Objectives:
The purpose of this assignment is to get acquainted with GLSL, specifically fragment shaders, using the Shadertoy platform. The outcome will be a variety of color-based patterns on-screen. Note: efficiency is not the priority just yet... this is your one chance to do whatever you want and get away with inefficient methods before we get into the deeper quirks of GLSL and writing efficient shader code.

Instructions & Requirements:
DO NOT begin programming until you have read through the complete instructions, bonus opportunities and standards, start to finish. Take notes and identify questions during this time. The only exception to this is whatever we do in class.
For this assignment, you will produce a few effects that are based around colors:

1. ***Effect #0***: Output the screen-space or UV coordinate as a static gradient.
    - Produces a familiar pattern of black, red, green and yellow. We'll start this one in class.
2. ***Effect #1***: Implement a static checkerboard pattern with perfectly square checkers (alternating blocks of color on the horizontal and vertical axes; squares should indeed be squares and not rectangles stretched to fit the aspect ratio).
    - Feel free to show the difference between pixel-size checkers and large squares, but the intent of this effect is the latter.
3. ***Effect #2***: Implement a circle whose color and size change over time. The background color should be complementary to that of the circle.
    - If you're feeling daring, color the circle with a radial gradient (from the center to the edge).
4. ***Effect #3***: Implement a unique, full-screen psychedelic effect to morph colors and patterns all across the screen. Use time and other uniforms to animate it.
    - Feel free to research other effects in the Shadertoy community. Provide references to your inspirations and briefly explain why you find them interesting. This is your

first chance to just go nuts and have fun with GLSL without really caring about any particular results or best practices.
- ***Future you, beware: As engineers, you can do better than hobbyists online. Their code is largely undocumented and inefficient. Feel free to research effects for inspiration only. Please refer to the official GLSL specifications for the official meanings of functions and keywords; we will talk about efficiency, good practices and techniques as we move along.***

Set up your code as follows:

1. Open **Shadertoy** (https://www.shadertoy.com/) and make an account. You will always be able to access your existing shaders by clicking on your username on the top right.
   - Note: You may not get the verification email if you use your school-provided address (probably filtered out by the server), so try a personal email instead.
2. Click 'new' in the top right menu. You will be taken to the new shader page with a default shader: an animated gradient. Give your shader a name, tag and brief description and press the save button.
3. Implement the required effects above in the 'Image' tab (which is what you're looking at when you create a new shader).
   - The 'mainImage' function is called by the system ***exactly once per pixel***. Here you are responsible for calculating and passing back the final display color of a single pixel:

```
// mainImage: calculate and pass final display color of current pixel
//  fragColor: final color to be displayed; assigned by function
//    (4D vector representing a color in RGBA format)
//  fragCoord: coordinate of current pixel being processed
//    (2D vector representing the location of the pixel in the image)
//      x: between zero and horizontal display resolution (zero is left side of image)
//      y: between zero and vertical display resolution (zero is bottom of image)
void mainImage(out vec4 fragColor, in vec2 fragCoord)
{
    // CALCULATE FINAL DISPLAY COLOR FOR CURRENT PIXEL
}
```

4. For each of the required effects above, follow this process:
   - Above the 'mainImage' function, add a new function that returns a vec4 (representing the final color in RGBA format) and has at least one parameter, 'vec2 fragCoord'. Add a parameter for any other data that you would like to use in the function. Start off your function by immediately returning a unique solid color. Here is an example that also uses the image resolution as a parameter and returns 'orange' (please use actual names for your functions):

```
vec4 myEffect1(in vec2 fragCoord, in vec2 resolution)
{
    return vec4(1.0, 0.5, 0.0, 1.0);
}
```

- Call the function in 'mainImage', passing any parameters from uniforms or constants. Comment out your other effect functions so to only see the effect that you are testing or demonstrating. The result can be directly assigned to 'fragColor'. I.e. your 'mainImage' function should only consist of your function calls.

```
void mainImage(out vec4 fragColor, in vec2 fragCoord)
{
    //fragColor = myEffect0(fragCoord, iResolution.xy); // previously-tested effect
    fragColor = myEffect1(fragCoord, iResolution.xy); // currently-tested effect
}
```

- Implement the actual effect within your function. Test it frequently by pressing the play button at the bottom left of the code window to execute your image program. Update the function prototype and call within 'mainImage' as needed. A successful compilation will yield a new image in the display area; a green box will flash around the code area. A failed compilation will not show the errors inline with the code in red; a red box will appear around the failed tab.
  - ***Pro tip: Shader debugging is difficult but not impossible. Even if your shader compiles, you will mostly rely on interpreting the color you see as expected or unexpected. That said, it is very important that you press the play button often so you can catch errors early and squash them.***
- Save your work often by pressing the save button.

5. Any function that needs to refer to a uniform should have the value passed as a parameter when called in 'mainImage'.
   - A ***uniform*** is a read-only variable that...
     - ...is passed from the system or engine that invokes the shader (e.g. Shadertoy, a WebGL application running in your browser).
     - ...has the same value for all shader invocations (e.g. all pixels) in a single render.
   - For your reference, here is the meaning of each uniform provided by Shadertoy, with a few more details. They can be viewed by clicking the 'Shader Inputs' dropdown menu (under the 'Image' tab):

```
//  viewport resolution (in pixels)
//    (value shown beneath display image)
//      x: horizontal display (output) image resolution
//      y: vertical display image resolution
//      z: pixel aspect ratio (1.0 if square pixels); NOT display aspect ratio!
uniform vec3 iResolution;

//  shader playback time (in seconds)
//    (value shown beneath display image)
//    (can be reset and paused with buttons beneath image)
uniform float iTime;

//  render time (in seconds)
//    (reciprocal of frame rate; e.g. if fps = 50, then dt = 1/50 = 0.02s)
//    (frame rate shown beneath display image)
uniform float iTimeDelta;

//  shader playback frame
//    (number of frames rendered since loading page)
uniform int iFrame;
```

```
//  channel playback time (in seconds)
//    (playback time of each input channel)
uniform float iChannelTime[4];

//  channel resolution (in pixels)
//    (resolution of each input channel)
//    (same members as iResolution)
uniform vec3 iChannelResolution[4];

//  mouse pixel coords
//    (cursor coordinates for interactivity)
//      x: current horizontal cursor position (if left button down), zero is left side of image
//      y: current vertical cursor position (if left button down), zero is bottom of image
//      z: horizontal cursor position when click occurred
//      w: vertical cursor position when click occurred
uniform vec4 iMouse;

//  input channel
//    (channels are iChannel0, iChannel1, iChannel2 and iChannel3)
//    ('XX' in type changes depending on type of texture)
//      type is 'sampler2D' for 2D textures
//      type is 'samplerCube' for cube maps
uniform samplerXX iChannel0..3;

//  current date
//    (year, month, day, time in seconds)
uniform vec4 iDate;

//  sound sample rate
//    (e.g. 44100hz)
uniform float iSampleRate;
```

Bonus:
You are encouraged to complete one or more of the following bonus opportunities (rewards listed):

- ***Sampler channels (+1)***: Incorporate the sampler channel uniforms into your design. Use one of the 2D textures and the 'texture' function to sample from it.
- ***More effects (+1)***: Implement an additional custom effect (i.e. repeat effect #3 the instructions).

Coding Standards:
You are required to mind the following standards (penalties listed):

- ***Reminder: You may be referencing others' ideas and borrowing their code. Credit and provide a link to source materials (books, websites, forums, etc.) in your work wherever code from there is borrowed, and credit your instructor for the starter framework, even if it is adapted, modified or reworked (failure to cite sources and claiming source materials as one's own results in an instant final grade of F in the course)***. Recall that borrowed material, even when cited, is not your own and will not be counted for grades; therefore you must ensure that your assignment includes some of your own contributions and substantial modification from what is provided in the book. ***This principle applies to all evaluations***.
- ***Reminder: You must use version control consistently (zero on organization)***. Even though you are using Shadertoy, a platform which allows you to save your work

online, you must frequently commit and back up your work using version control. In your repository, create a new text file for each tab used in Shadertoy (e.g. 'Image') and copy your code there. This will also help you track your progress developing your shaders. Remember to work on a new branch for each assignment.

- ***The assignment must be completed using Shadertoy (https://www.shadertoy.com/) (zero on assignment)***. No other platforms will be permitted for this assignment.
- ***Do not reference uniforms in functions other than 'mainImage' (-1 per instance)***: Use the 'mainImage' function provided to call your effect functions. Any uniform value to be used in a function must be passed as a parameter when the function is called from 'mainImage'.
- ***Every section, block and line of code must be commented (-1 per ambiguous section/block/line)***. Clearly state the intent, the 'why' behind each section, block and even line (or every few related lines) of code. This is to demonstrate that you can relate what you are doing to the subject matter.
- ***Add author information to the top of each code file (-1 for each omission)***. If you have a license, include the boiler plate template (fill it in with your own info) and add a separate block with: 1) the name and purpose of the file; and 2) a list of contributors and what they did.

Points 5

Submitting a text entry box or a website url

Due	For	Available from	Until
-	Everyone	-	-

GraphicsAnimation-Master

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**1 pts** **Full points** Strong evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely both efficient and functional.	**0.5 pts** **Half points** Mild evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely either efficient or functional.	**0 pts** **Zero points** Weak evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely neither efficient nor functional.	1 pts
IMPLEMENTATION: Content & Material Practical knowledge of content relevant to the discipline and course (e.g. shaders and effects for graphics, animation algorithms and techniques, etc.).	**1 pts** **Full points** Strong evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely both efficient and functional.	**0.5 pts** **Half points** Mild evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely either efficient or functional.	**0 pts** **Zero points** Weak evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely neither efficient nor functional.	1 pts
DEMONSTRATION: Presentation & Walkthrough Live presentation and walkthrough of code, implementation, contributions, etc.	**1 pts** **Full points** Strong evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely both accurate and confident.	**0.5 pts** **Half points** Mild evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely either accurate or confident.	**0 pts** **Zero points** Weak evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely neither accurate nor confident.	1 pts
DEMONSTRATION: Product & Output Live showing and explanation of final working implementation, product and/or outputs.	**1 pts** **Full points** Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable.	**0.5 pts** **Half points** Mild evidence of correct and stable final product that runs as expected; end result is largely either correct or stable.	**0 pts** **Zero points** Weak evidence of correct and stable final product that runs as expected; end result is largely neither correct nor stable.	1 pts

Criteria	Ratings			Pts
ORGANIZATION: Documentation & Management Overall developer communication practices, such as thorough documentation and use of version control.	**1 pts** **Full points** Strong evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely both documented and organized.	**0.5 pts** **Half points** Mild evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely either documented or organized.	**0 pts** **Zero points** Weak evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely neither documented nor organized.	1 pts
BONUSES Bonus points may be awarded for extra credit contributions.	**0 pts** **Points awarded** If score is positive, points were awarded for extra credit contributions (see comments).		**0 pts** **Zero points**	0 pts
PENALTIES Penalty points may be deducted for coding standard violations.	**0 pts** **Points deducted** If score is negative, points were deducted for coding standard violations (see comments).		**0 pts** **Zero points**	0 pts
				Total Points: 5