Lab 8: Shader Pipelines

Publish | Edit | ⋮

GPR-200: Introduction to Modern Graphics Programming
Instructor: Daniel S. Buckstein
Lab 8: Shader Pipelines

Summary:
In lab 7, we were introduced to vertex shaders, related vocabulary, transformations, spaces and GLSL programs with both vertex and fragment shaders. Given our experience with GLSL shaders so far, it is time to wrap up the labs with this final exercise of putting it all together. In this lab, we revisit some display techniques explored in previous labs.

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.**

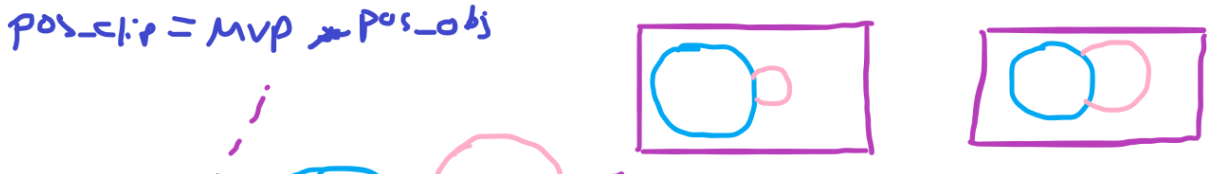
Objectives:
The purpose of this assignment is to apply techniques that use both vertex and fragment shaders. You will revisit some texturing and post-processing principles, as seen in lab 6, and implement a multi-pass pipeline using different vertex and fragment shaders. This may also require additional research.

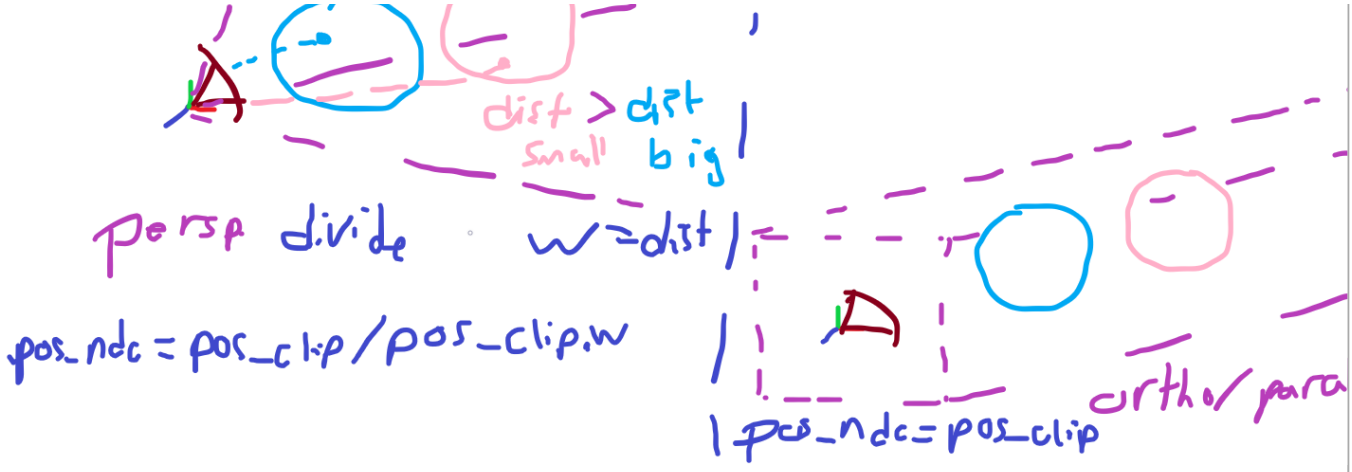
Instructions & Requirements:
DO NOT begin programming until you have read through the complete instructions, bonus opportunities and standards, start to finish. Take notes and identify questions during this time. The only exception to this is whatever we do in class.
Review the following, then implement the specified shader programs below (the instructions are deliberately vague because you have experience with everything that you need to do, and also there is room for creativity):

1. ***Setup***: Set up a new project using **SHADERed** **(https://shadered.org/)** .
2. ***Shaders and programs***: The main part of this assignment is to revisit multi-pass rendering, this time using vertex-based models and geometry. Implement a minimum of four passes, each requiring their own vertex and fragment shader. In SHADERed, the "output" of a pass is stored in a ***render texture*** which can be sampled by future passes as a regular texture.
   - **Pipeline**: Implement ***one*** pipeline with a minimum of four passes. The passes must be arranged with some greater purpose in mind. Here are a few pipeline examples:
     - *Bloom*: Implement the bloom algorithm from lab 6.
     - *Non-photorealism*: Implement a pipeline that highlights some sort of non-photorealistic rendering style, such as a pencil sketch or watercolor.
     - *Shadow-mapping*: A basic shadow technique that requires an additional scene "pre-pass" before the actual scene pass.
     - *Be creative!* Choose your desired effect/pipeline and draw a flow diagram to visualize it. Use the programs and shaders below as your building blocks (make the diagram without worrying too much about the implementation specifics because it will help you get there).
   - **Programs (passes)**: Your pipeline must have at least four passes. Two are required, the other two facilitate post-processing algorithms up to your creative interests:

    - *Scene pass*: Display at least four geometric models to make an interesting scene. Allow the user to toggle between different shader algorithms applied to the models (e.g. solid color, Phong shading, just a texture, etc.).
    - *Display pass*: Display the end result of the pipeline on a *full-screen quad*. This is just a plane drawn directly to NDC with a texture applied (e.g. the output of the scene pass would be good to test).
  - **Shaders**: At minimum, you must implement and use shaders that meet the following specifications and perform the following tasks:
    - *Vertex shader for scene objects*: Implement a vertex shader that transforms vertices and other attributes appropriately for the selected shading algorithm. This is used in the scene pass.
    - *Fragment shader for scene objects*: Implement a fragment shader that performs your desired per-fragment shading algorithm for the scene pass.
    - *Vertex shader for full-screen quad, pass texture coordinate*: For at least one of the post-processing/display passes, have a vertex shader that passes a varying for the texture coordinate attribute only, and outputs the position in NDC.
    - *Fragment shader for post-processing, receive texture coordinate*: Paired with the above vertex shader is a fragment shader that receives the texture coordinate attribute and uses it to sample a full-screen texture.
    - *Vertex shader for full-screen quad or scene objects, pass clip-space position*: For at least one of the passes, have a vertex shader that passes the clip-space coordinate as a varying.
    - *Fragment shader for post-processing or scene objects, calculate screen-space UV*: Paired with the above vertex shader is a fragment shader that procedurally calculates the screen-space UV by first manually performing the *perspective divide* on the inbound clip-space coordinate, then converting the resulting NDC point (-1 to +1 range on each dimension) to a screen-space point (0 to 1 range on each dimension, therefore usable as a UV).
3. ***Optimization***: Avoid expensive code wherever possible. In this case, you may rewrite functions where appropriate as there is no concept of some 'common' tab like in Shadertoy; the shaders (vertex and fragment) operate independently from each other. That said, still try to write and reuse flexible code wherever possible. In your demo video, showcase the differences between the four modes and discuss any opportunities for optimization that you can identify.
4. ***Gallery***: Here are some images:

  - Perspective divide:





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

- ***Normal mapping (+1)***: In the scene pass, implement *normal mapping* to give the geometry a bumpy look. This requires use of the tangent attribute.
- ***Ray-tracing (+2)***: Coming full-circle, incorporate a pass that adds ray-traced procedural geometry into the scene.

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 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 your selected tool (e.g. "vertex.glsl" and "fragment.glsl") 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 should be completed using SHADERed (https://shadered.org/) or some other interface that allows easy editing of both GLSL vertex and fragment shaders (zero on assignment)***. You must use a platform that gives you the easy ability to edit both ***vertex and fragment shaders*** with real-time feedback. The modern language we are using is ***GLSL*** and you must be able to edit shaders using either GLSL

4.50 (standard) or GLSL ES 3.00 (WebGL 2.0). Other viable interfaces that can include KickJS and ShaderFrog; if you can find another one that satisfies this requirement, please propose it to your instructor before using (also good luck).

- ***Do not reference uniforms in functions other than 'main' (-1 per instance)***: Use the required 'main' function in each shader to call your related functions. Any uniform value to be used in a function must be passed as a parameter when the function is called from 'main'.
- ***Use the most efficient methods (-1 per inefficient/unnecessary practices)***: Your goal is to implement optimized and thoughtful shader code instead of just implementing the demo as-is. Use inefficient functions and methods sparingly and out of necessity (including but not limited to conditionals, square roots, etc.). Put some thought into everything you do and figure out if there are more efficient ways to do it.
- ***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-Range

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**1 to >0.5 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 to >0.0 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 to >0.5 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 to >0.0 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 to >0.5 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 to >0.0 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 to >0.5 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 to >0.0 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 to >0.5 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 to >0.0 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