Final Project

Publish | Edit | ⋮

GPR-200 Introduction to Modern Graphics Programming
Instructor: Daniel S. Buckstein
Final Project

Summary:
The primary takeaways of this course are, in no particular order or rank, ***portfolio*** and ***engineering***. The final project is for you to demonstrate all you have learned in this course from a game programmer's lens and produce a portfolio-worthy project. All components of the project must be functional and impressive; this is something you would want to show off at an interview; hence, this project is tied with a technical interview! Find a way to demonstrate the requirements as part of the technical interview (which involves a presentation). Implementing the bare minimum will earn you up to 80% on the project; go above and beyond for the remaining 20%.

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 ***10-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:
This project is geared towards developing your portfolio and thinking critically about new real-time effects, applying your skills throughout.

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.
For this assignment, you will choose and design a project of your own liking that integrates everything we have talked about in the course. You must integrate all of the following topics in one demo:

- ***GLSL vertex and fragment shaders***: The project will be designed and implemented using multiple GLSL programs, incorporating both vertex and fragment shaders, using SHADERed.
  - Each program must make effective use of attributes, varyings, uniforms and render targets.
- ***Data structures and functions***: The project must make use of data structures for organization and utility functions to help use them in code.
  - In C++ we have classes with member functions, constructors, operators; in GLSL we have structs and global functions. Use input and output parameters where appropriate.
- ***Lighting and shading***: At minimum, the Phong reflectance model must be used in some creative way.
  - This can be done using the simplified simulated method or ray-tracing.
- ***Textures***: Use textures and texture coordinates in some interesting and creative way to help color your scene.
  - We have explored 2D and cube samplers, use one or both of these creatively and appropriately.
- ***Multi-pass***: At minimum, your shader will use one pass to draw a scene and another to display the final image. You must have at least one intermediate post-processing pass,

designed and integrated appropriately.
    - Note: For organization, each shader should have a unique text file in your repository (e.g. final_scene_vs.glsl; final_scene_fs.glsl; etc.). Commit frequently as you update these files.
- ***Interactivity***: The use must be able to interact with your scene in some way.
    - Use the input features of SHADERed to help with this.
- ***UML diagram***: Implement a detailed UML diagram to map out your code before beginning the implementation. This will be an opportunity for you to visualize your data structures and algorithms before jumping into them. Use an online tool such as LucidChart to help build your diagram. Be as specific as possible, including interface diagrams for your data structures and functions, and data flow diagrams for the general flow of the program. Remember that it is not actual code but is meant for design; the purpose is defeated by doing this after you have code. Submit this in your repository, make sure the time stamp shows it was done well before programming.

There are three general categories that your project falls into. Do some research into effects or topics that interest you, but here are some examples. ***You are by no means limited to these***, be creative and make something you like. The categories are:

- ***Architecture***: This category focuses more on the design, data structures and algorithms side of shaders and programs. Projects in this category could involve re-implementing some existing some cross-language frameworks in SHADERed (e.g. move Peter Shirley's Ray Tracing in One Weekend into one of his other books), or even developing your own shader program architecture from scratch (a real challenge, learn some actual OpenGL/WebGL programming).
- ***Visuals***: The goal of this category is to produce a new, interesting and complex visual effect that integrates all of the features. Do some research into some effect that interests you. Remember: existing effects are generally not optimized, so other shaders online are good for reference only. Be creative!
- ***Midterm continuation***: One avenue for the project is to continue your midterm project, this time more in-depth with vertex shaders. Determine how your needs have changed and how you will adapt your original idea to the new vertex and fragment combo program architecture.

Remember: this project is worth 10% of your grade, labs are 5%. The outcome of the project must be unique a higher caliber than anything produced in the labs. There is also a higher standard for project maintenance and documentation; make sure that every function is fully documented and as optimized as possible, justified by describing the development process in your video (10 minutes this time). While your video serves the purpose of allowing you to privately explain your project to your instructor, there is also the technical interview which takes place during exam week and is a presentation among peers... so prepare your best work! Have fun, learn lots, make something cool!

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 10

Submitting a text entry box or a website url

Due	For	Available from	Until
-	Everyone	-	-

GraphicsAnimation-Master-Range-x2

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**2 to >1.0 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.	**1 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.	2 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.).	**2 to >1.0 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.	**1 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.	2 pts
DEMONSTRATION: Presentation & Walkthrough Live presentation and walkthrough of code, implementation, contributions, etc.	**2 to >1.0 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.	**1 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.	2 pts
DEMONSTRATION: Product & Output Live showing and explanation of final working implementation, product and/or outputs.	**2 to >1.0 pts** **Full points** Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable.	**1 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.	2 pts

Criteria	Ratings			Pts
ORGANIZATION: Documentation & Management Overall developer communication practices, such as thorough documentation and use of version control.	**2 to >1.0 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.	**1 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.	2 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: 10