Project 3: Screen-Space Rendering Techniques

Publish | Edit | ⋮

GPR-300 Intermediate Graphics & Animation Programming
Instructor: Daniel S. Buckstein
Project 3: Screen-Space Rendering Techniques

Summary:
Now that we have explored framebuffers and multi-pass rendering, we can now go a bit deeper into applications. In this project we use framebuffers to implement some classic alternatives to traditional forward rendering: deferred rendering.

Objectives:
Upon successful completion of this assignment, you will have accomplished the following:

- Revisit and implement an updated forward shading pipeline.
- Implement a deferred shading pipeline and shaders.
- Implement a deferred lighting pipeline and shaders.

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). Begin your submission immediately, copy the following into the text box, decide on your repository branch name for this assignment and fill in the information below.*** **The submission box locks at the specified deadline; be proactive and don't miss it. Late submissions, even by a minute, will not be accepted.**

Copy, edit and submit the following text once as a team (you do not need the headings, please just provide your info as shown in the examples here):

1. ***Names of contributors***: Write the names of the contributors of this assignment.
   e.g. **Dan Buckstein**
2. ***A link to your public repository online***: Grab the clone link from your working repository, which should end with "*.git*".
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework.git** (note: not a real link)
3. ***The name of the branch that will hold the completed assignment***: Create a new branch for this project, submit only the name of this branch.
   e.g. **project0-main**
4. ***A link to the read-me and user instructions for this project***: Ensure your repository includes a read-me that summarizes how to use your finished product.
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework/blob/project0-main/project0-readme.pdf** (note: not a real link)
5. ***A link to your video (see below) that can be viewed in a web browser***: Ensure your video is public or shared with your instructor.
   e.g. YouTube link: **https://www.youtube.com/watch?v=OqOyxQVs8lY** (note: "Attack on Game Development" by Will Gordon & Connor Breen)

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 definitely show off your professionalism and don't minimize it):

- 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.**
- **Instead of waiting until last-minute for the video to upload, create a folder or links document on Drive that is accessible to your instructor. Submit the link to this folder as part of your official submission, and copy your video file/link there when it is ready.**

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.

Using the provided framework, implement the following:

1. **Setup framework**: Complete the following steps to ready the framework for this project:
   A. ***Project branch***: Check out the project starter branch "*graphics2/proj3*" to begin the project.
   B. ***Implement renderer utilities***: Here are the steps for setting up this demo:
      I. *Explore pre-built example*: Load the pre-built example demo for this project: "*File > Load demo > ...Proj3*". The demo shows the completed scene with procedural and loaded models, and the ability to toggle a variety of pipeline stages and features.
      II. *Initial build and run*: Either build the project in Visual Studio then launch using "*File > DEBUG... > Load without building*", or use "*Quick build*" to hot-build and run the demo directly through the player window.
      III. *Additional setup*: Complete the following steps to make sure the framework is all set:
         - Navigate to the intro demo mode filter: "*Source Files/common/A3_DEMO/a3_DemoMode2_SSFX*"; open the update source for this mode: "*a3_DemoMode2_SSFX-idle-update.c*". Here you must update the uniform buffers and some data so that objects display correctly.
         - Navigate to the render source for this mode: "*a3_DemoMode2_SSFX-idle-render.c*". Here you must uncomment the existing rendering pipeline elements and complete them to have a functional render pipeline.
2. **Implement shaders**: Using the pipeline implemented in part 1:
   A. ***Encode/decode shaders***: Upon successful completion of the render pipeline, using the encoded shaders should allow you to traverse the complete set of passes to see how the bloom effect is achieved. For your actual implementation, remove the "*e/*" from each shader file path.
      - With encoded shaders disabled (required), you will see a full-screen checker texture and/or a solid color.
      - Since the last project, the following decoded shader files have been implemented: *passTexcoord...vs*, *drawTexture_fs*, *utilCommon_fs*. Use these as a reference for the shaders below
   B. ***Implement shader programs***: Implement the following effects by visiting and completing the following GLSL files (navigate to "*Resource Files/A3_DEMO/glsl/4x*"):
      I. *Phong shading with normal mapping and uniform buffers*: We have implemented a couple variants of Phong shading in the past, all of which have been *forward shading* effects. This time, we reimplement it using a uniform buffer to hold the objects' transformation data, and apply *normal mapping*. Normal mapping is a technique in which the surface normals are stored in a texture; we convert the respective sample at each fragment to the target lighting space using a *tangent*

basis matrix, comprised of the geometric tangent, bitangent and normal vectors (initially read as attributes, passed to the fragment shader for application). See Blue Book for examples.
    - Vertex shader: Open "*vs/02-pipeline-deferred/passTangentBasis_ubo_transform_vs4x.glsl*" and follow the to-do prompts.
    - Fragment shader: Open "*fs/02-pipeline-deferred/drawPhongNM_fs4x.glsl*" and follow the to-do prompts.

II. *G-buffer output*: Here we enter ***deferred rendering***. It works like this: instead of calculating lighting all of the fragments in all of the models, as is done with the previous effect, we split lighting into at least two passes; this describes the first pass in any deferred rendering pipeline. We want to save the expensive lighting calculations for later (hence, "***deferred***") and only visualize the information about the scene that we need to perform lighting in a subsequent pass. This is theoretically more optimal since we will later only do lighting where there are areas to be lit on-screen. The "***g-buffers***" (geometry buffers) are the outputs of this pass, and they store geometric data as color. We will use our framebuffer's MRT capabilities to output multiple images at the same time. The depth buffer is handled automatically, but we need to produce images for at least the scene normals and texture coordinates. Some people like to output the diffuse and specular texture samples as well, but this is optional since that doesn't really describe the geometry itself.
    - Vertex shader: This program uses the same vertex shader as Phong with normal mapping.
    - Fragment shader: Open "*fs/02-pipeline-deferred/drawGBuffers_fs4x.glsl*" and follow the to-do prompts.

III. *Deferred Phong shading*: In a ***deferred shading*** pipeline, we use the final image of the scene's geometric data (the g-buffers) to perform lighting in screen-space, on a full-screen quad. This ultimately achieves the same effect as forward shading, but the fashion in which we attain the scene composite is different: since the geometry is stored in textures, we don't need varyings to describe them; instead, we only use the screen-space texture coordinate to grab the attribute data from the g-buffers.
    - Code setup: In "*a3_DemoMode2...render.c*", make sure all of the required code is uncommented up to and including the deferred shading switch case in the composite pass (two passes after scene). Complete the missing activation steps.
    - Vertex shader: This program uses the pass texture coordinate vertex shader (implemented) because it is a screen-space effect.
    - Fragment shader: Open "*fs/02.../postDeferredShading_fs4x.glsl*" and follow the to-do prompts.

IV. *Deferred lighting, light volume pre-pass*: In a ***deferred lighting*** pipeline, we use the final image of the scene's geometric data (the g-buffers) to perform lighting only in areas contained by a light volume. Once per light, we draw an actual model representative of the volume (sphere for a point light), which triggers fragment generation only where there is potential lighting to calculate. With each fragment, we determine the screen-space coordinate and use this to sample the g-buffers, whose data we use to calculate the diffuse and specular components for the current light. This is particularly efficient when dealing with many small light volumes.
   - Code setup: In "*a3_DemoMode2...render.c*", make sure all of the required code is uncommented up to and including the deferred light volume pre-pass (pass immediately after scene). Complete the remaining setup tasks described here.
   - Vertex shader: Open "*vs/02.../passClipBiased_transform_instanced_vs4x.glsl*" and follow the to-do prompts.
   - Fragment shader: Open "*fs/02.../drawPhongPointLight_fs4x.glsl*" and follow the to-do prompts.

V. *Deferred lighting composite*: The deferred lighting pipeline has a third pass, which is to composite the lighting information acquired in the previous pass with the surface color data. This program simply samples the textures with light data (from the previous pass, the lighting pre-pass) and the textures with surface color (diffuse and specular maps), and uses all of this to calculate the Phong sum: (diffuse light)(diffuse color) + (specular light)(specular color) + (dim ambient constant color).
   - Code setup: In "*a3_DemoMode2...render.c*", make sure all of the required code is uncommented up to and including the deferred lighting switch case in the composite pass (two passes after scene). Complete the remaining activations.
   - Vertex shader: This program uses the pass texture coordinate vertex shader (implemented) because it is a screen-space effect.
   - Fragment shader: Open "*fs/02.../postDeferredLightingComposite_fs4x.glsl*" and follow the to-do prompts.

3. **Testing & demonstration**: You must thoroughly test your render pipeline and shaders. Your demonstration must show evidence of the following:
   A. **Walkthrough and justification of architecture**: Demonstrate that the above requirements have been met in code.
   B. **Framework**: Demonstrate completion of the pipeline setup using animal3D data structures and functions.
   C. **Shaders**: Demonstrate the functional rendering pipeline and completion of the shaders implemented throughout the project. Furthermore, you must demonstrate that the shaders are your own and not the encoded files.
   D. **Takeaways**: Discuss personal and professional takeaways from this project.

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

- ***Additional screen-space effects (+2)***: Research and implement an additional screen-space effect, such as edge detection, screen-space ambient occlusion, depth-of-field or motion blur.
- ***Parallax occlusion mapping (+2)***: As part of the forward shading pipeline and g-buffer pass in the deferred pipelines, implement parallax occlusion mapping (POM) to offset texture coordinates before they are used in the final effect.
- ***Dancing lights (+1 CPU only, +2 CPU+GPU)***: Make the lights move around the scene by either seemingly randomizing their movement or having them follow an interesting curved path over time. Play around with other properties such as color and size for the most interesting effect. For debugging purposes, the lights can be rendered as physical objects by revealing them with the 'H' key (they are all drawn the same size here, and there is a quick way to fix that as well).

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

- ***Reminder: You may be referencing others' ideas and borrowing their code. Credit sources and provide a links wherever code 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. ***This principle applies to all evaluations***.
- ***Reminder: You must use version control consistently (zero on organization)***. Commit after a small change set (e.g. completing a section in the book) and push to your repository once in a while. Use branches to separate features (e.g. a chapter in the book), merging back to the parent branch (dev) when you stabilize something.
- ***Visual programming interfaces (e.g. Blueprint) are forbidden (zero on assignment)***. The programming languages allowed are: C/C++, C# (Unity) and/or Python (Maya).
  - If you are using Unity, all front-end code must be implemented in C# (i.e. without the use of additional editors). You may implement and use your own C/C++ back-end plugin. The editor may be used strictly for UI (not the required algorithms).
  - If you are using Unreal, all code must be implemented directly in C/C++ (i.e. without Blueprint). Blueprints may be used strictly for UI (not the required algorithms).

    - If you are using Maya, all code must be implemented in Python. You may implement and use your own C/C++ back-end plugin. Editor tools may be used for UI.
    - You have been provided with a C-based framework called *animal3D* from your instructor.
    - You may find another C/C++ based framework to use. Ask before using.
- ***The 'auto' keyword and other language equivalents are forbidden (-1 per instance)***. Determine and use the proper variable type of all objects. Be explicit and understand what your data represents. Example:
    - auto someNumber = 1.0f;
        - This is a float, so the correct line should be: float someFloat = 1.0f;
    - auto someListThing = vector<int>();
        - You already know from the constructor that it is a vector of integers; name it as such: vector<int> someVecOfInts = vector<int>();
    - Pro tip: If you don't like the vector syntax, use your own typedefs. Here's one to make the previous example more convenient:
        - typedef vector<int> vecInt;
        - vecInt someVecOfInts = vecInt();
- ***The 'for-each' loop syntax is forbidden (-1 per instance)***. Replace 'for each' loops with traditional 'for' loops: the loops provided have this syntax:
    - In C++: for (<object> : <set>)...
    - In C#: foreach (<object> in <set>)...
- ***Compiler warnings are forbidden (-1 per instance)***. Your starter project has warnings treated as errors so you must fix them in order to complete a build. ***Do not disable this***. Fix any silly errors or warnings for a nice, clean build. They are generally pretty clear but if you are confused please ask for help. Also be sure to test your work and product before submitting to ensure no warnings/errors made it through. This also applies to C# projects.
- ***Every section/block of code must be commented (-1 per ambiguous section/block)***. Clearly state the intent, the 'why' behind each section/block. 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.
- ***Immediate mode is forbidden (zero on assignment)***: In this course we are studying modern graphics engineering principles; immediate mode refers to an ancient and deprecated set of OpenGL functions. The tutorials followed use the correct techniques; do not use tutorials on the internet that will lead you astray.

Points 10

Submitting a text entry box

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