Lab 6: Intro to Post-Processing & Composition

Publish **Edit**

GPR-200: Introduction to Modern Graphics Programming
Instructor: Daniel S. Buckstein
Lab 6: Intro to Post-Processing & Composition

Summary:
In lab 5, we explored the fundamentals of texture sampling and display. We explored Shadertoy's texture channels and associated uniforms, as well as a variety of sampler types and how they work. We also starter implementing some simple effects using these textures. This lab introduces the concept of multi-pass rendering, which we can implement very quickly and efficiently in Shadertoy.

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):

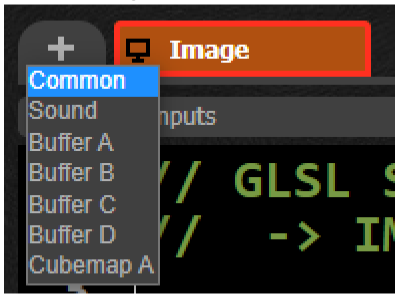
- Show the final result of the project and any features implemented, with a voice over explaining what the user is doing.
- Show and explain any relevant contributions implemented in code and explain their purpose in the context of the assignment and course.
- Show and explain any systems source code 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.**

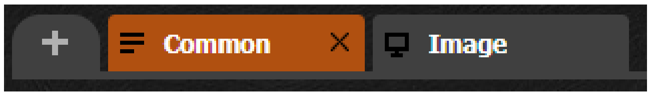
Objectives:
The outcome of this assignment is the ubiquitous and industry standard post-processing algorithm, bloom. You will learn concepts in convolution and multi-pass display.

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 Shadertoy, implement the following:

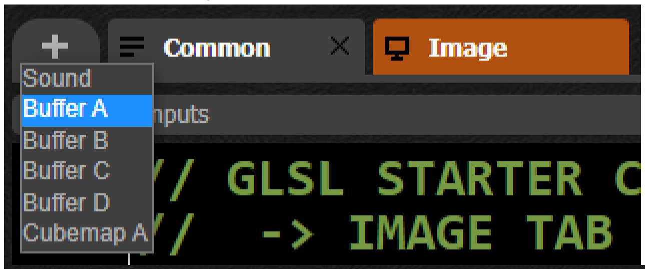
1. ***Setup***: In Shadertoy, we have exclusively been using the '*Image*' tab to do all of our drawing. For multi-pass, we will use multiple image buffers and other tabs in Shadertoy to implement a multi-pass, post-processing pipeline. Follow this process to set up your bloom pipeline:
    - Start by doing the setup we are familiar with: In a new Shadertoy, download **this file** **(https://champlain.instructure.com/courses/1693444/files/190983745/download?download_frd=1)** and open in a text editor, then copy the contents into the '*Image*' tab, replacing the default contents. The shader ***will not compile*** as-is.
        - The '*Image*' tab is called so because the target is the final display image, seen in the viewport on the left-side of the Shadertoy interface.
    - At the top-left corner of the shader text editor, next to the '*Image*' tab, there is a small tab with a plus sign (+). Click on that and select '*Common*', which will open the '*Common*' tab. Download, copy and paste the contents of **this file** **(https://champlain.instructure.com/courses/1693444/files/190983777/download?download_frd=1)** into the '*Common*' tab. The tab will appear to the left of '*Image*'.
        - The '*Common*' tab contains repeatable code that is "pasted" at the top of each of the other tabs. This is stuff that is implicitly "included" in the '*Image*' tab, and others, as you'll see, without any need for redeclaring or redefining things. In this respect, it is like a header file. See the bottom of the starter code for an example of how it is used in compiling an actual GLSL fragment shader. You will see how this can be useful as we move along.
        - After adding the new tab, your shader should compile, however, in the viewport you should not see anything. This is because we have nothing to display, even though the actual shader content ('mainImage' in the image tab) says we are

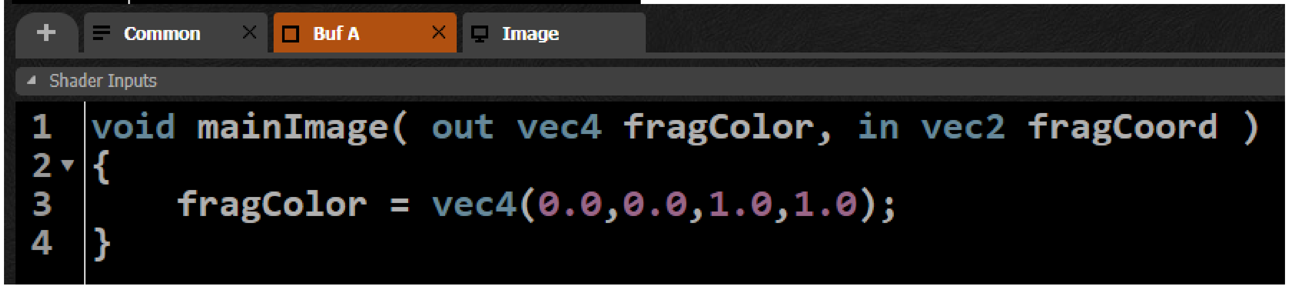
sampling from a texture. For this, we need to set up a buffer.





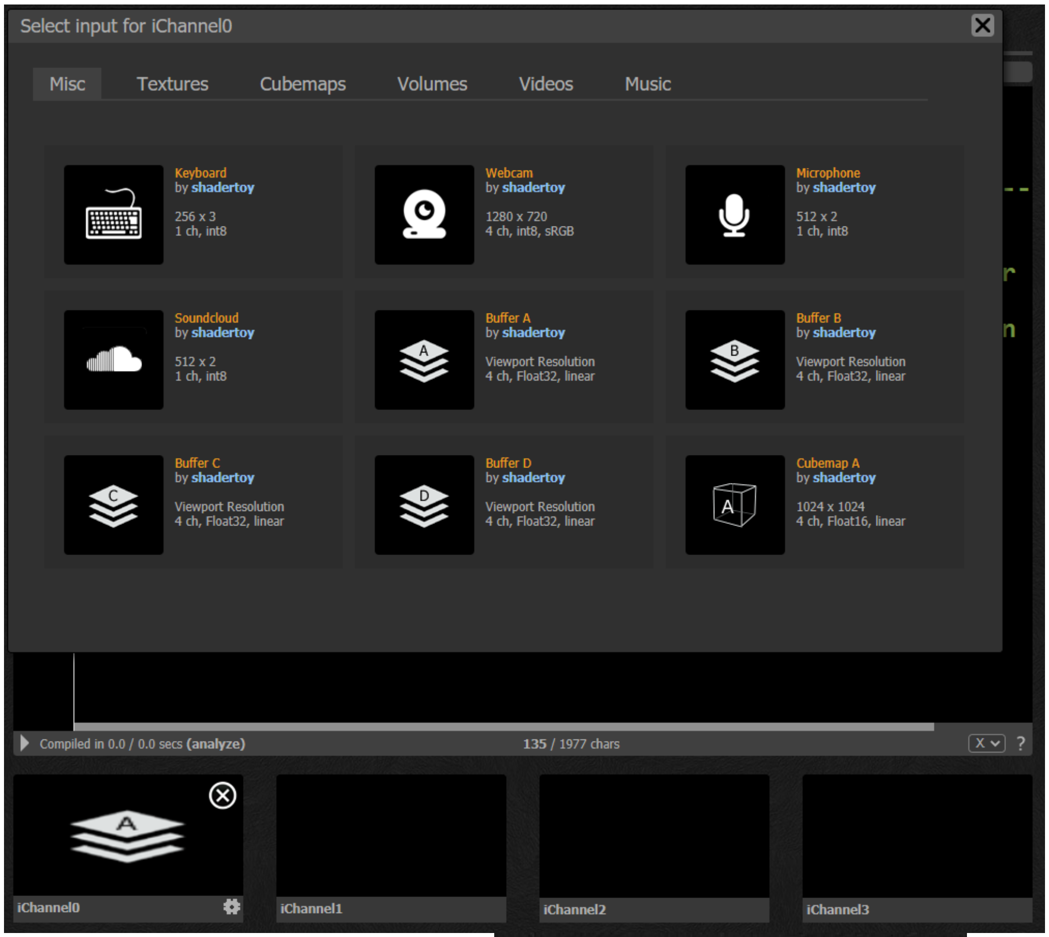
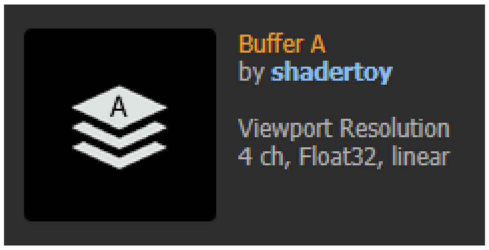
- To set up another image buffer, select the '*Buffer A*' option from the tab dropdown, in the same fashion as the previous step. This will result in a '*Buf A*' tab appearing in between '*Common*' and '*Image*'.
    - These buffer tabs are ***separate fragment shaders*** from the display image tab (default), but they all have the same format (see bottom of '*Common*'). You will note that the new shader, by default, clearly says "output solid blue", but we do not see blue yet.

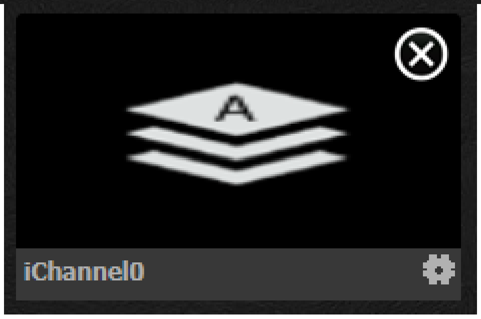




```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    fragColor = vec4(0.0,0.0,1.0,1.0);
}
```

- Finally, to visualize the contents of a buffer, we sample it as a texture in the 'Image' tab. The code to do this has already been provided, but nothing is set in the channel box.
    - Click on the preview box for *iChannel0* and select '***Misc/Buffer A***'. This means that the ***output of the 'Buf A' tab's shader is an input for the 'Input' tab's shader***. Upon successfully linking and compiling, you will see the contents of '*Buf A*' displayed in the viewport, because the display shader is sampling from that buffer as a regular 2D texture!


Select input for iChannel0
Misc
Textures
Cubemaps
Volumes
Videos
Music
Keyboard by shadertoy 256 x 3 1 ch, int8
Webcam by shadertoy 1280 x 720 4 ch, int8, sRGB
Microphone by shadertoy 512 x 2 1 ch, int8
Soundcloud by shadertoy 512 x 2 1 ch, int8
Buffer A by shadertoy Viewport Resolution 4 ch, Float32, linear
Buffer B by shadertoy Viewport Resolution 4 ch, Float32, linear
Buffer C by shadertoy Viewport Resolution 4 ch, Float32, linear
Buffer D by shadertoy Viewport Resolution 4 ch, Float32, linear
Cubemap A by shadertoy 1024 x 1024 4 ch, Float16, linear
Compiled in 0.0 / 0.0 secs (analyze)
135 / 1977 chars
iChannel0
iChannel1
iChannel2
iChannel3




- A good way to test that you are in fact sampling from another buffer is to change the result of the shader in the '*Buffer A*' tab, without changing the contents of the '*Image*' tab. You will see whatever you are doing in '*Buffer A*' the viewport because the '*Image*' tab is sampling from it.
- **The tabs are henceforth simply referred to by name: *Common*, *Buffer A*, *Buffer B*, *Buffer C*, *Buffer D* and *Image*.**

2. **Buffer A**: In the first pass, render your ***scene***.
   - Download and copy the contents of **this file (https://champlain.instructure.com/courses/1693444/files/190983747/download?download_frd=1)** into *Buffer A*. You will notice that this is a simplified version of some of the other stuff we've seen; recall that most of the reusable code has been moved to *Common* and is still accessible in all buffer tabs, including *Image*. The output you will see at first is just a testing blue-cyan-magenta-white gradient (complementary to our familiar black-red-green-yellow).

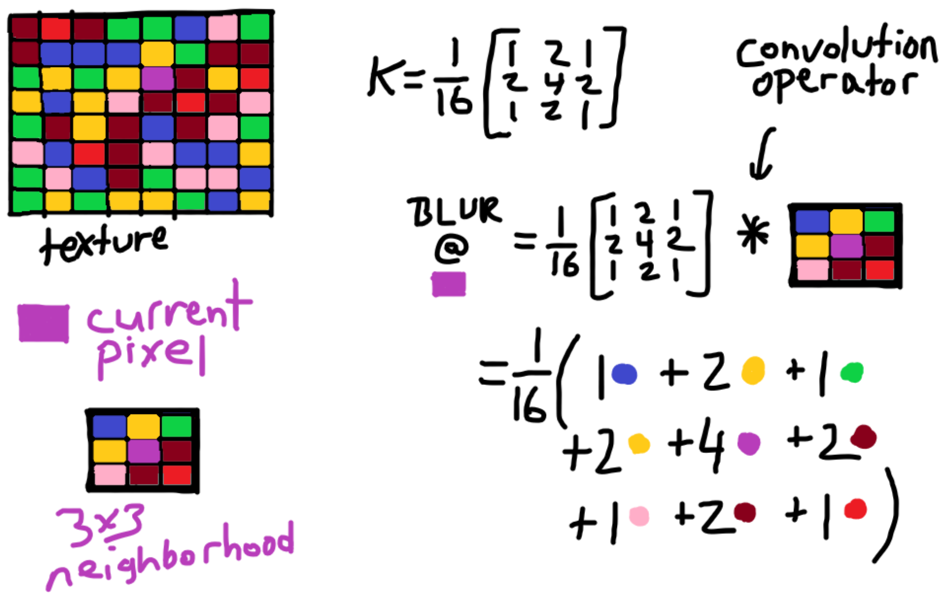
- ***Scene***: Draw a scene in *Buffer A* that includes a ***rotating cube map*** as the background. With our current setup, the viewport should automatically update upon compiling to show the contents of *Buffer A* because we are sampling from it in *Image*.
  - You have officially set up a multi-pass pipeline! *Buffer A* draws the scene and stores the result, and *Image* samples from the output of *Buffer A* and displays it.
    - *Buffer A* -> *Image* -> **Display**
  - The result of this pass is a scenic view using whatever cube map you selected:

3. **Buffer B**: In the second pass, sample the scene buffer and perform ***tone mapping*** or a ***bright pass***.
   - Add a new tab, *Buffer B*. Start with the same code as you did for the *Image* tab. This will not change the output. Set *iChannel0* to '*Misc/Buffer A*' in the same fashion, while the channel reference in *Image* should change to *Buffer B*.
     - You now have three passes in your pipeline: *Buffer A* renders the scene to a render target, *Buffer B* samples from *Buffer A* and outputs it to its render target, finally *Image* samples from *Buffer B* and displays it!
       - *Buffer A* -> *Buffer B* -> *Image* -> **Display**
   - ***Bright pass***: The next stage in the bloom effect after drawing the scene is called a *bright pass*, a form of *tone mapping* which extracts the brightest portions of the image while filtering out the dark. Implement a bright pass effect.
     - There are many ways to do this effect, but some may use the concept of *luminance* to determine how bright a pixel is. Remember to use external tools like Desmos graphing calculate to help visualize falloff functions.
     - Here is my result, with the brightest areas of the image still bright, while the rest is dimmed:

4. **Buffer C**: In the third pass, sample the tone mapped/bright passed buffer and perform a ***Gaussian blur***.
   - Set up this pass the same way as the previous. You now have a four-pass pipeline.
   - ***Gaussian blur***: Blurring is an example of ***convolution*** and operates on a group of pixels, not just one. A ***convolution kernel*** is a grid/matrix of weights, which are multiplied by the respective pixel in a neighborhood of pixels in the image (e.g. in a shader, some area of pixels surrounding the one you are processing).
     - Here is an example of a 3x3 Gaussian convolution kernel:

       $$K_{3\times 3} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

       The weight of 4 in the middle would be the multiplier for the pixel at the center of the neighborhood (the one your are processing), and the other weights are multipliers for pixels at the respective positions relative to the central pixel. The fraction (1/16) is to normalize the final result; it is the reciprocal of the sum of weights (in this case, 16). It is called a Gaussian kernel because the distribution of weights is a Gaussian distribution; if you were to plot out the weights as heights on a 2D grid, the resulting curved surface would be a bell shape. In so many words, it's a fancy way of saying "weighted average" for textures, over some small area in the texture.
     - For this assignment, test convolution using the above 3x3 kernel and implement at minimum a ***5x5 kernel***. *Hint: look at Pascal's triangle to help you figure out the weights*.
     - Here's a graphical example of convolution using the above kernel and a 3-pixel by 3-pixel region of a texture:


texture
current pixel
3x3 neighborhood
$K = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$
convolution operator
BLUR @ $= \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} *$
$= \frac{1}{16}(1 + 2 + 1 + 2 + 4 + 2 + 1 + 2 + 1)$

- Here is my blur pass (again, different methods):

- ***Note: DO NOT use the blurred cube maps in Shadertoy to fake this effect; implement proper Gaussian blurring!***

5. **Image**: In the final pass, composite the final image by ***compositing*** the results of the ***scene*** and the ***blur*** buffers.
   - The current setup in *Image* has *iChannel0* set to the output of *Buffer C*. Now, reference the result of the original scene, *Buffer A*, in another texture channel. Sample both textures and store the colors.
   - ***Composite & final display***: Finally, create the final image by ***compositing*** the two color samples; the final image looks like the original scene but has light flooding over edges of bright areas.
     - Try a few different composition methods, including ***add*** (it's just addition), ***mix*** (GLSL function) and ***screen*** (look up the formula online).
     - Demonstrate ***at least these three methods*** and ***one other*** compositing mode in your video (Photoshop has many great examples for blending layers, which is

basically what you are doing, and rest assured that they are all very simple formulas).

- Here is my final image (again, many ways to get yours):

6. ***Optimize***: Per our usual practice, once you make it through the above effects, review and optimize your code. Note: this includes avoiding redundant blocks of code wherever possible; any repeatable code should be encapsulated in data structures and functions, and any code that is reusable in multiple passes should be moved to the *Common* tab.
7. ***Organization***: Remember to commit frequently. Furthermore, your repository should now have ***one text file for each shader tab (i.e. "common.txt", "bufA.txt", etc.)***.

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

- ***Multiple blur passes (+1)***: Improve the efficiency of your bloom pipeline by splitting your *blur* pass, which uses a 2D convolution kernel (mathematically a large square matrix), into *two blur passes*, each using a 1D kernel (a row/column matrix) sampling in perpendicular directions. Optimize this process as much as possible and be sure to explain how it is more efficient; it will produce identical results as the non-bonus requirements, however it is far more optimized and the code is cleaner. You will need to add the '*Buffer D*' tab into your pipeline, sampling from *Buffer C* and referenced in final display pass in *Image*.
    - Demonstrate this effect using a 3x3 kernel, and at minimum a 5x5 kernel.
    - Here is the math showing how the final 3x3 blur output is identical, but broken into a horizontal and vertical blur pass:

$$K_{3\times3} = K_{3\times1}K_{1\times3} = \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix}\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$$

- ***Additional convolution algorithm (+1)***: Implement an additional convolution algorithm as part of the final processing/compositing stage (e.g. edge detection, see "Sobel" operator/filter as an example).

- ***Custom cube map (+1)***: Add the '*Cubemap A*' buffer tab into your pipeline and use it to generate a custom cube map to be referenced in '*Buffer A*'. There are some tutorials on this, but you must make your own background effect. Do something creative and trippy to envelope the viewer in a strange land.
- ***Sound (+1)***: You will also notice there is a '*Sound*' buffer which you can use to process audio in Shadertoy, and play it back through another shader. Yes, sound is ***just data*** so you can process it using a shader. Do something interesting with the sound tab (e.g. distort the sound much like you might distort an image coordinate, then play it back).
- ***Interactive cube map part 2 (+1)***: Use one of the buffers (e.g. *Buffer D*) as an ***accumulation buffer*** for ***mouse input***. An accumulation buffer is one that references or samples from itself. In the same fashion as the previous lab, the cube map view direction should be controlled using the mouse, however you should be able to release the mouse and click again at any point to resume looking from the same viewpoint. This is against the tradition in that a new mouse click would typically reset the view.

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'.
- ***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