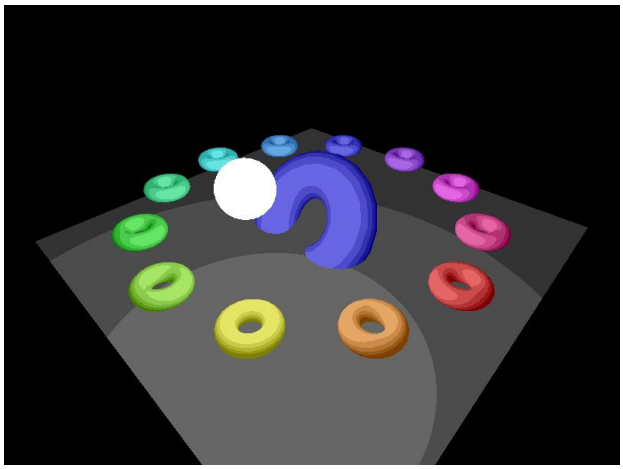
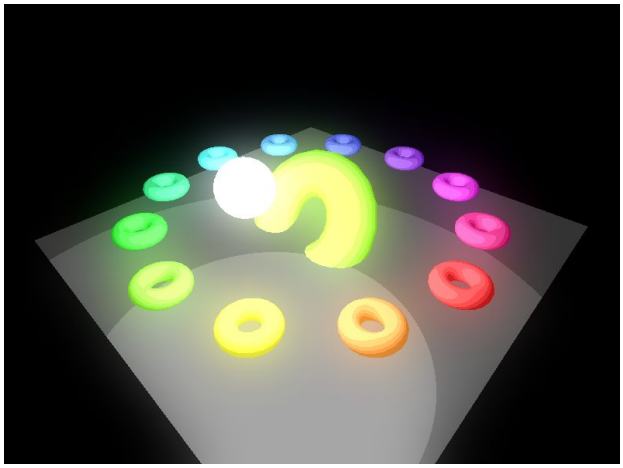
Lab 2: Bloom





Objective

Create the necessary shader programs to implement the Bloom effect. The Bloom post process effect adds a glowy aura around the bright areas in the rendered scene. The images on the right show the scene rendered without bloom (top) and with bloom (bottom). If you need a refresher on the bloom algorithm, refer to Lecture 8.

Starting Point

You may choose to use the provided framework, or your own. If you have issues running the starter code, you likely need to "retarget the solution". To do this right click on the Solution from the Solution Explorer, click "Retarget Solution" and hit okay on the box that pops up.

The C++ side of the framework is mostly setup for you. You will need to create the necessary framebuffer objects and send the appropriate uniforms to the shaders. A simple shader program using the default_v.glsl and default_f.glsl shaders is provided for you, these shaders should compile without any issues. Your task is to implement the bright_f.glsl, gaussianBlur_f.glsl and bloomComposite_f.glsl fragment shaders and use them appropriately in the DisplayCallBackFunction located in main.

Ensure that your project toggles the following modes appropriately:

- '1' = Default shading (already done!)
- '2' = Bright Pass
- '3' = Blurred Bright Pass
- '4' = Bloom

Part 0: Initialization

The starter code has no frame buffer objects created. You should read through this document and create framebuffer objects as you see fit. Frame buffer objects should be created in the initializeFrameBuffers function found in main.cpp

Part 1: The Bright Pass

The bright pass is responsible for determining which pixels are highly illuminated. This can be computed using the **pseudo code** found below:

BrightPass Filter
Extract highlights

```
float4 PixelShader(float2 texCoord : TEXCOORD0,
                   in float BloomThreshold,
                   uniform sampler2D TextureSampler) : COLOR0
{
  // Look up the original image color.
  float4 c = tex2D(TextureSampler, texCoord);

  // Adjust it to keep only values brighter than the specified threshold.
  return saturate((c - BloomThreshold) / (1 - BloomThreshold));
}
```

saturate() just clamps the interval to [0,1]

Gaming @ UOIT CHALLENGE INNOVATE CONNECT

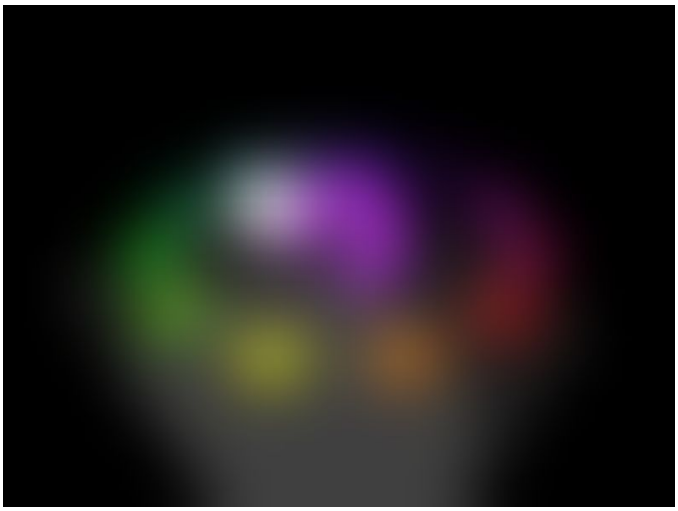
Pseudo code for extracting bright pixels, taken from the lecture slides. Note: GLSL does not have a saturate function, it's called "clamp".

The image on the right demonstrates what the bright pass image **could** look like. This image was created using a BloomThreshold of 0.2. Try playing around with different sized frame buffer objects and observe the effects on the final bloomed result.

Part 2: Blurred Bright Pass

We have now figured out which pixels should have a glow around them. The next task is to blur the image created in the bright pass. This effectively adds the auras around the bright pixels. To do this, you need to implement a **2D gaussian blur**. The process of performing a gaussian blur is the exact same as performing a box blur with the only difference being each sample is multiplied by a weight. A useful tool to compute the weights can be found here: http://dev.theomader.com/gaussian-kernel-calculator/

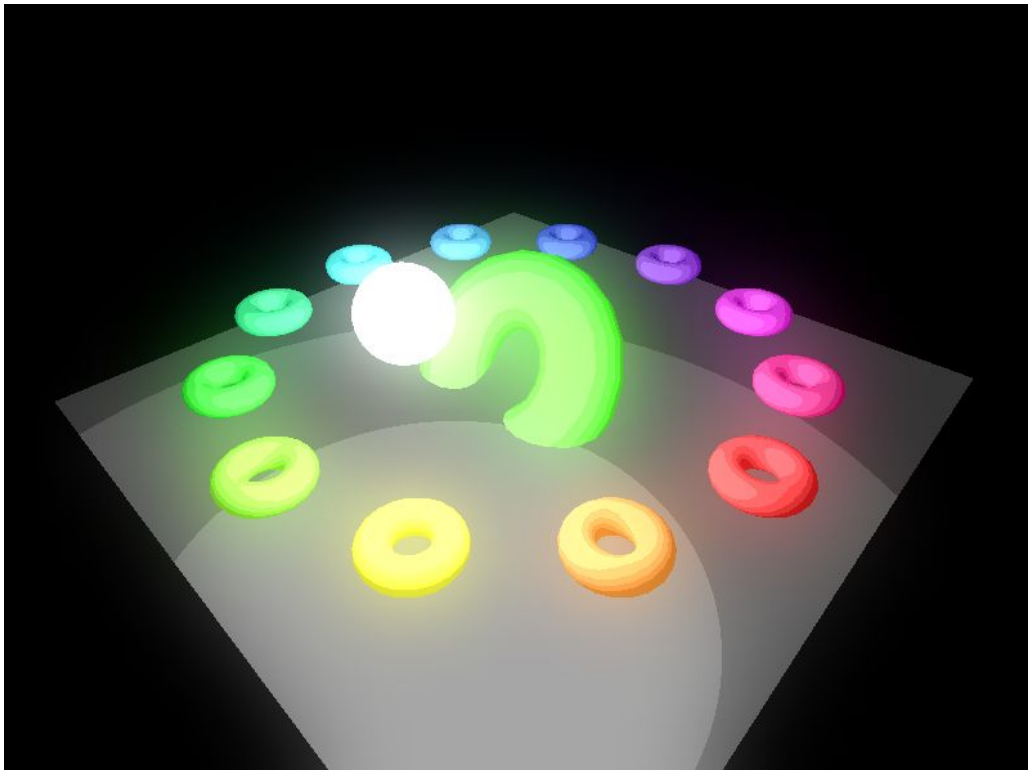
Alternatively you can use Excel to compute the weights.



The image on the right shows what the blurred bright pass **could** look like. If you just use a standard 3x3 kernel on the full resolution image, the blur effect will be much less noticeable. Think about how you can make your bright pass this blurry. What size kernel should you use? How many blur passes should you do? How large should your FBO be? The image on the right uses a 3x3 kernel with a very small FBO and several blur passes.

Part 3: Compositing

This is the last step in the bloom algorithm, to create the final bloomed image you need to bind the original scene texture and the blurred bright pass texture. You then sample from both textures in the bloomComposite_f shader and simply add them together. The final result should look something like:



If your bloom does not look this glowy, think about how you can modify the parameters you are using throughout the algorithm to exaggerate the effect.

Grading

You are being graded on functionality and correctness.

IF YOUR CODE DOES NOT COMPILE YOU GET A ZERO FOR THE CODING PORTIONS OF THE LAB!!!

Initialization (/2)
- Did you create your frame buffer objects correctly?

Quality of bright pass (/4)
- Does your bright pass image look correct?

Quality of blurred bright pass (/4)
- Does your blurred bright pass image look correct?

Quality of bloom effect (/4)
- Does your final composited bloom effect look correct?
- The final result should have a **noticeable** glow around the bright pixels

Lab report with all explanations and requirements (just answer the following) (/4)
- What is a fullscreen quad? Why is it useful for post processing? (/1)

- This bloom implementation adds a glow to all of the bright pixels on the screen, how would you modify this implementation to only bloom certain objects? For example, what if you only wanted to bloom particles and nothing else. (/3)

Project directory submitted and is clean (/2)

- Do not submit the intermediate directory and the .db file. If your submission is extraordinarily large it may not be marked and you will receive a grade of 0.

****Project compiles and runs without error****

- IF YOUR CODE DOES NOT COMPILE (SHADERS TOO) YOU GET A ZERO FOR THE RELEVANT SECTIONS

Lab Report Submission Guidelines:

- **Report is due when you submit the code. Put the answers in a PDF document in the root directory of the code zip file.**

Video Submission Guidelines:

- **You must submit a video, failure to do so will result in a grade of 0**
- Video should be concise, run the program, demonstrate the different toggle modes and describe what is happening.
- You should briefly explain how each mode works technically (i.e. Outlines are achieved by...)
- A good approach to making the video is to imagine you are demonstrating your work to someone who does not know what “bloom” is and describe to them how each mode contributes to the final effect.