Assignment 1: A Story-Telling Canvas

Welcome to the journey of Computer Graphics! Your first assignment serves as both an introduction to GLSL programming and an opportunity to build connections among us. The goal is to create your first GLSL program on a virtual canvas and tell a short story about yourself. Let's get started!

1 Reading

- Course slides for GPU rendering pipeline and GLSL programming
- Starter Code Tutorial: Hello GLSL
- Students' work from previous years for inspiration (check the pdf file on Canvas)

2 Starter Code

To access the latest starter code for our computer graphics course, please visit the following GitLab repository: https://github.com/cg-gatech/cs3451. I recommend using the 'git clone' command to download the complete codebase directly from the project webpage. Make sure to follow the tutorial available on the project page, as it provides detailed instructions on using CMake to compile the source files. These instructions encompass the installation of CMake, generating project files, compiling the code through the command line, and executing the resultant program. The source files for this assignment is located in `assignments/a1`. You task is to work on the fragment shader `a1_frag.frag`. Upon successful setup and execution, you should see a window appear and a triangle rotating on the screen.

Figure 1: Example of window content. A triangle is spinning around.

3 Implementation Tasks

Overview You are asked to generate an image by specifying the color for each pixel in the provided `mainImage` function. Please carefully read the starter code and the comments before starting your implementation. The mechanism underlying the painting process is very simple: **for a given position** (x, y) **on the screen, you return a `vec4` color** (**red**, **green**, **blue**, **alpha**). Note that the current canvas does not support alpha blending (for transparency), so the alpha value (the last component of the vec4) should always be set to one.

The input coordinates (x, y) for the drawing operation are constrained within the canvas' domain size, which defaults to $(1280, 960)$. In this context, the first element 1280 specifies the width, and the second element 960 indicates the height. Therefore, in your drawing implementation, you should only consider coordinates ranging from $(0, 0)$ to $(1280, 960)$. Operations conducted outside this defined range will be automatically discarded. Note that the coordinates $(0, 0)$ **represent the bottom-left corner** of the canvas, while $(1280, 960)$ corresponds to the top-right corner.

Additionally, in the provided fragment shader `a1_frag.frag`, there are several useful variables that you might find useful for enhancing your drawing operations:

- **iTime:** This variable measures the elapsed seconds since the start of the program. It can be used to introduce elements of animation into your drawing.
- **iResolution:** This indicates the size of the window, which defaults to 1280*960. It can be utilized to calculate a fixed point on the window, such as the center. For example, the code `vec2 center = vec2(iResolution / 2.f)` will return the window's center position.
- **fragCoord:** This variable represents the input position on the canvas. The origin is located at the bottom-left corner, with positive extending from left to right and from bottom to top.
- **fragColor:** This variable is the output rgba color of the given position.

Step 1: Draw a Circle on Screen You are asked to implement a function `inCircle(vec2pos, vec2center,floatradius)` which takes the query position, circle's center, and circle's radius as input and returns a boolean variable to indicate if the query position is inside the circle or not. This check function will be called in `drawCircle(vec2pos,vec2center,floatradius,vec3color)` to calculate a color for each given position. After your implementation, uncomment the function call of `drawCircle` in `mainImage()`. If everything is implemented correctly, you should be able to see a white circle in the center of the screen, as shown in Figure 2.

Step 2: Draw a Rectangle on Screen Let's reinforce the idea of "calculating a color for a position" in a fragment shader by practicing another function implementation for a rectangle. In `inRectangle(vec2pos,vec2leftBottom,vec2rightTop)`, you are asked to implement an indicator function to check if a given position is inside the rectangle. The rectangle is defined by two input vectors `leftBottom` and `rightTop`. After implementing this function, don't forget to uncomment the function call of `drawRectangle` in `mainImage()` to test it. If everything is implemented correctly, you should see a white rectangle in the center of the screen, as shown in Figure 3.

Figure 2: Step 1: Draw a circle on screen.

Figure 3: Step 2: Draw a rectangle on screen.

Step 3: Draw an animated circle Here, we present an example of drawing an animated circle by passing a temporally varying radius to `drawCircle()`. There is no specific programming requirement for this task. To observe how it works, uncomment the line after Step 3 in `mainImage()` (refer to Figure 4) and attempt to replicate this animation in your customized scene.

Figure 4: Step 3: Animate the circle.

Step 4: Draw the union of two shapes Finally, we demonstrate how to draw two shapes on the screen simultaneously. The approach involves calling both `drawCircle()` and `drawRectangle()` and combining their colors in the same fragment (consider why this works). This method can easily be extended to draw multiple overlapping objects. To see the union of the two objects on the

screen, uncomment the line after Step 4 in `mainImage()` (refer to Figure 5). There is no specific programming requirement for this task. Try to apply this concept in your customized scene.

Figure 5: Step 4: Draw the union of two shapes.

3.1 Creative Expression: Create Your Customized Scene

So far, we have demonstrated how to draw a triangle, a circle, and a rectangle, animate these shapes, and combine them. Now, it is your turn to create your own scene by drawing and animating shapes on the screen. Feel free to modify any part of the starter code to customize your image and aim to tell a story with your creation. Seek inspiration from the examples provided. The theme for this Creative Expression is **"Telling a story about yourself."**

4 Submission

To complete your assignment submission, please include the following elements:

- Your source code: Submit all source files you have modified.
- Screenshots demonstrating your implementation's correctness for Steps 1 and 2.
- An image or a video demonstrating your customized scene.
- A concise paragraph describing your technical implementation.

5 Grading

This assignment carries a total of 8 points. The grading will be based on the following criteria:

1. Technical contributions (5 pts):
   - **Step 1 (draw circle):** 2 points
   - **Step 2 (draw rectangle):** 2 points

- **Step 3 and 4:** 1 point

2. Creative expression (3 points): This aspect focuses on your ability to create a new image by composing different shapes and colors on a canvas screen. Use different **shapes**, **colors**, and **animations** to express your theme.

6 Sharing your Work

You are encouraged to share your graphical work with the class. If you want to do so, please upload your image to Ed Discussion under the post **A1 Gallery: A Story-telling Canvas**. This is an excellent opportunity to engage with your peers and gain recognition for your work. We look forward to seeing you shine with your excellent renderings!