Exercise Sheet 2

Primitive Types / Shaders

Question 1.
In the lecture, you learned about the different OpenGL primitives and shaders.

(a) (1 point) Name and describe the three different primitive types for triangles (without adjacency) in OpenGL.

(b) (4 points) In Figure 1, you see three different polygons. For each of the polygons and each of the three primitive types for triangles, specify an (optimal) vertex buffer (the names of the vertices in the correct order), the number of OpenGL draw calls, and the parameters of these calls necessary to correctly draw the polygon. The front sides of the triangles should be defined counterclockwise.

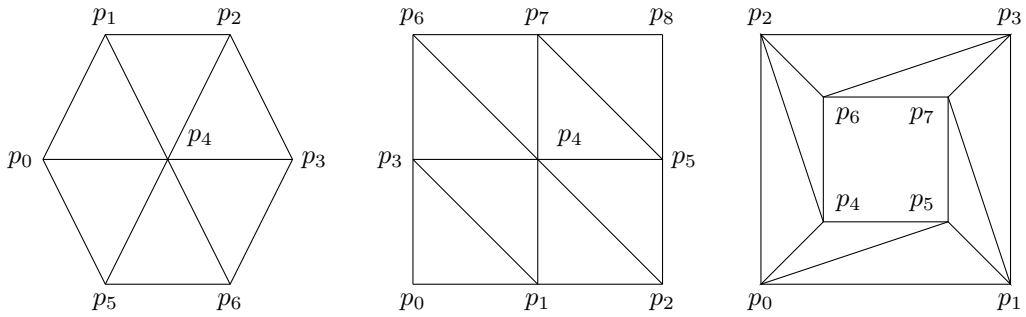

Figure 1: Polygons

Question 2.
For this exercise sheet, a framework is available for download in which the programming tasks are to be solved. In this exercise we want to visualize the Mandelbrot Set.

Details see here: https://en.wikipedia.org/wiki/Mandelbrot_set.

In the end, the image should look approximately like this:

(a) (3 points) Find the function **createJuliaVertices** and insert to **vertices** the corner points of a square (value range: $[0,1]^2$, $z = 0$, $w = 1$). The values for **complex** should lie in the value range $[-2.5, 2.5] \times [-1, 1]$.

Now, in the **main** function retrieve the vertex attributes from the shader, using the function **glGetAttribLocation**. Adjust these to the used vertex type **julia_vertex**, with the help of the function **glVertexAttribPointer**.

(b) (5 points) Complete the vertex shader (**"julia.vert"**) so that

- the square is output in the range $[-1, 1]^2$. The square should now fill the screen.
- the value range of **complex** is scaled and shifted to correspond to a magnification of the image by **zoom** with a midpoint at **zoomPosition**.

(c) (6 points) Edit the fragment shader (**"julia.frag"**) to paint a Julia set[1]. Use the iterative function

$$z_{n+1} = z_n^2 + c$$

The initial values are given in the shader (z0 = **z**, c = **c**), to multiply complex numbers the function **multiply_complex** can be used. Display the set of iteration steps until the square of the magnitude of the number z (can be implemented with the dot product) exceeds the value **bailout**. The upper limit for iterations is **maxIteration**. If this value is reached, the color black should be output. If not, the ratio of iteration steps to the upper limit for iterations should be used to linearly interpolate between the color values **color0** and **color1** in the HSV color space. Use **hsv2rgb** to convert the result to the RGB color space. The result color should be stored in **color**.

(d) (1 point) Ask for the locations of the uniform variables **time**, **zoom** and **zoomPosition** from the shader and set the corresponding variables in the **main** function. The application should now display an animated version of the Julia set that can be zoomed with the mouse wheel (and the mouse position).

All areas in which you need to change source code are marked with a comment of the form //TODO:.

Submit the written assignments, the modified framework, and the result images for sub-tasks 2 a), 2 b), and 2 c) by **Monday, June 02, 2025, 11:55 PM** via the Moodle platform in the form of a ZIP file.

[1] https://de.wikipedia.org/wiki/Julia-Menge