Exercise 2-a: per-vertex colors

Duplicate the solution of the previous exercise in the with *Visual Studio Code* workspace. If you solved the exercise correctly and you execute the code, you should see something similar to *Figure 1*.

In this exercise, you have to add colors to the shapes as shown *Figure 2*. To do that, you will have to:

- Modify the shaders
    - Vertex shader:
        - Add a vertex color attribute input.
        - Forward the per-vertex color attribute to the fragment shader through a varying variable of the same type.
    - Fragment shader:
        - Get the per-fragment color forwarded from the vertex shader stage through the varying variable.
        - Assign this color to the final fragment color.
- Get the new vertex attribute added to the shader in *initShaders()*, and enable it.
- Declare as many vertex-color buffer variables as necessary and initialize them in *loadSceneOnGPU()*.
- Add the appropriate code in *drawScene()* to specify that you want to send the vertex colors (using extra calls to *bindBuffer* and *vertexAttribPointer* with the corresponding color buffers).

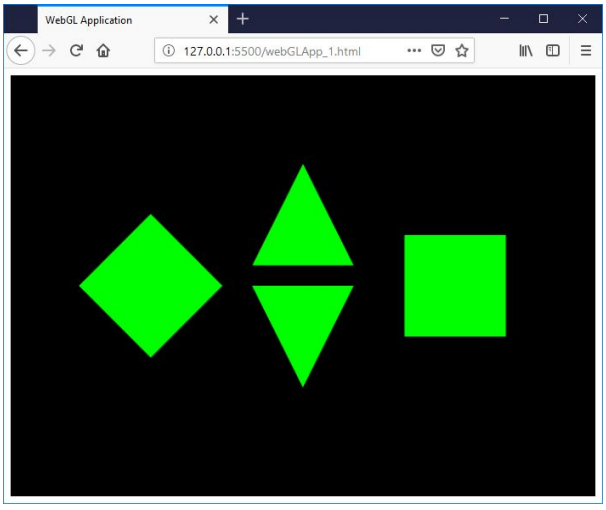

Figure 1. Exercise 1

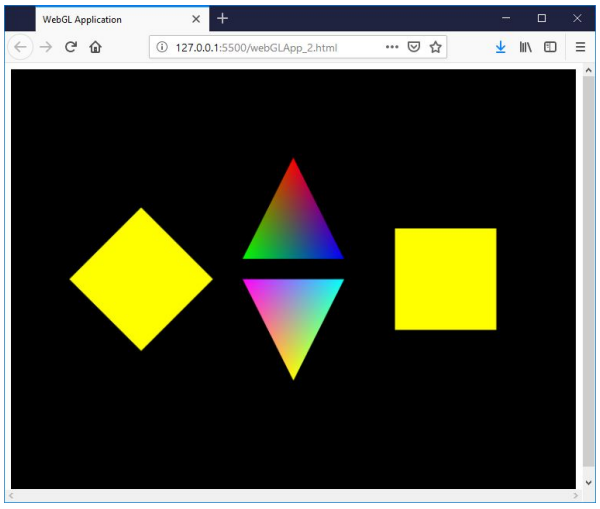

Figure 2. Exercise 2-a result

Exercise 2-b: colored screen-filling quad

Duplicate the previous exercise solution and start from that point. Draw a screen-filling quad with per-vertex colors as in *Figure 3*. To do that, you will have to:

- Remove the unnecessary code in the previous exercise (i.e. triangle primitives).
- Add per-vertex color attributes to the quad (create new buffer with colors).
- Modify drawScene():
    - Remove all unnecessary transformations to achieve a perfect, screen-filling quad. Remember that, in OpenGL, if no projection transformation is defined, the frustum is a cube with corners (-1, -1, -1) - (1, 1, 1) (also called NDC space).
    - Specify both the positions and the colors buffers using calls to *bindBuffer* and *vertexAttribPointer*.

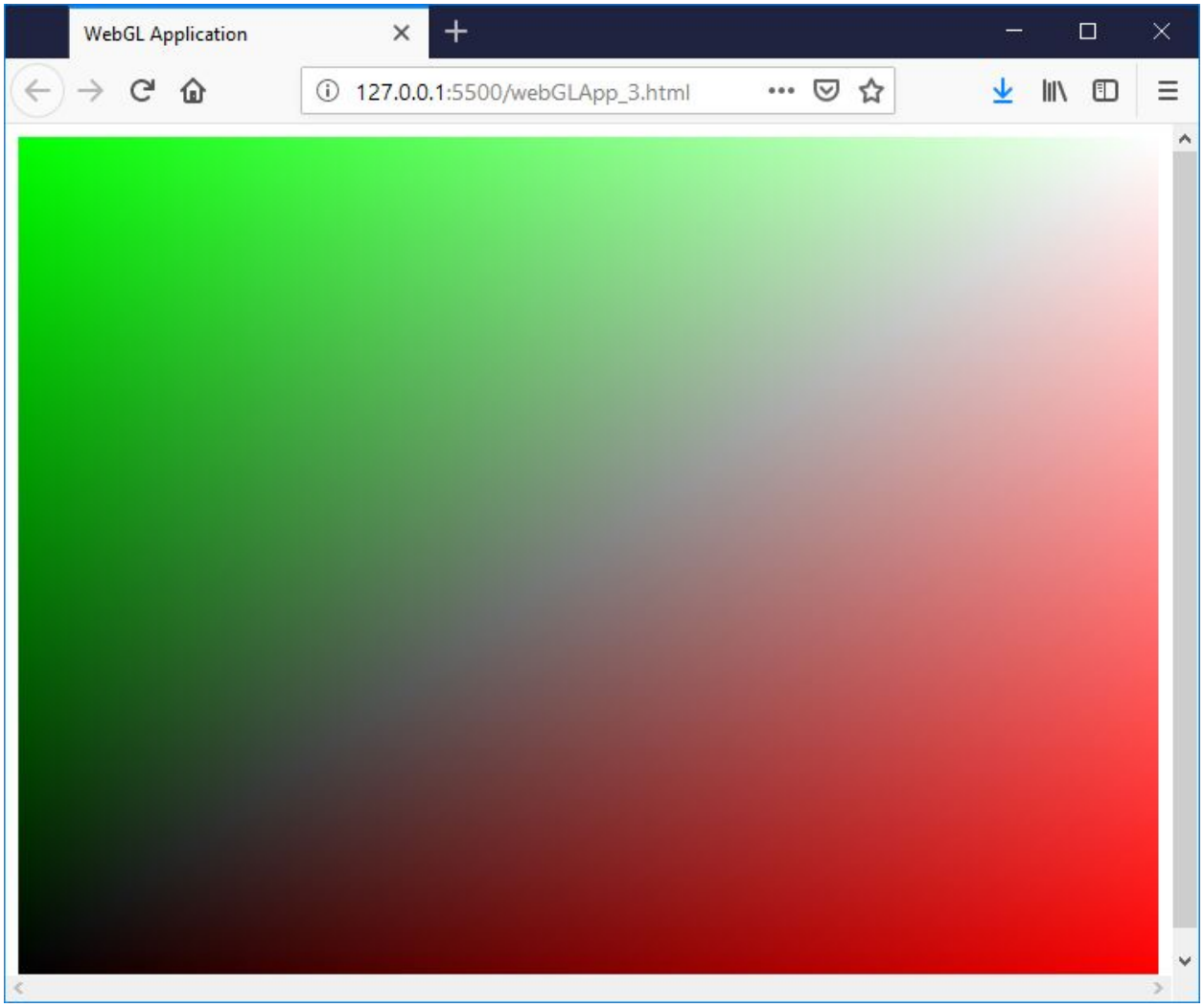

Figure 3. Exercise 2-b result