Università
della
Svizzera
italiana

Faculty
of Informatics

Computer Graphics (Fall 2022)

Assignment 8: Hello WebGL!

November 10, 2022

Welcome to the first WebGL assignment

Your first task is to build upon the triangle rendering presented in the lecture. First, you should add a color attribute to give each corner of the triangle a different color. Second, you should further extend the code to render a cube with each face rendered in a different color. You can download a template from iCorsi. It includes the code presented during the lecture, as well as comments/hints on how to complete this assignment. For development, we suggest using the latest version of *Firefox* or *Chrome* web browser and *Atom* editor. Before starting the assignment, we encourage you to familiarise yourself with the template and analyze how it works. Future assignments are designed to add functionality to the project; therefore:

- keep your code nice and clean,
- feel free to customize and refactor it to adapt it to your needs,
- e.g., feel free to split the code into different files and define your functions to avoid code repetition.

If you follow our guidelines, at the end of the course, you will be rewarded with a small rendering engine capable of adding new objects and simulating effects such as different kinds of illumination, advanced texture mapping, shadows, animation, etc.

Template Description

The provided template consists of several files:

- **template.html** - It contains the code for rendering the triangle presented during the lecture with hints on how to complete the assignment.
- **gl-matrix-min.js** - It contains a set of functions for vector and matrix operations. We will be using these functions in following assignments. For now, it is used for rotating the object with a slider that you can find next to the rendered image.
- **gemetry.js** - This is a Javascript file for defining geometry. Currently, the triangle definition is included in the main file (template.html). Since already the definition of the cube is more complex, we suggest keeping all the geometry definitions in a separate file. In the future, you may also want to keep there functions, which procedurally generate the geometry.

Exercise 1 [7 points]

Your first task is to add color for each vertex of the triangle. To this end, you should :

1. Add three RGB color definitions to the array defining the per-vertex color in Javascript code. Specify each component (R,G,B) with values in range $[0.0 \ldots 1.0]$.
2. Create a buffer for storing the color values on GPU. This should be done the same way as for vertices. After this step, your application should still run without any errors and show the same triangle as in the beginning.

3. Modify the shaders to accept a new attribute color. Remember how the data flows in the graphics pipeline. The attributes are defined for each vertex. Therefore, your color information goes into the vertex shader first. The vertex shader should then pass the color information further to the fragment shader. Here, each fragment receives its interpolated color value. To realize this, you need:

   (a) define an input variable (attribute) for the vertex shader, similarly as it is already done for vertex position, but now for input color;

   (b) define the output variable for the color in the vertex shader, which should be the variable passed to the fragment shader;

   (c) in the vertex shader code, copy the value from the input color variable to the output variable to define the flow of the data to the fragment shader, at this point you do not need to modify it;

   (d) add the input color variable to the fragment shader; note that in order to make it work the name of the color output variable in the vertex shader has to be the same as the input variable in the fragment shader, only then the graphics pipeline will know how the data should flow between vertex and fragment shaders;

   (e) use the input color variable in the fragment shader to color the fragment, note that the final color of the fragment has type *vec4* while your color variable *vec3*, you can create the *vec4* out of *vec3* in a very easy way by concatenating it with value 1.0. In GLSL, you can do it easily by $b = vec4(a, 1.0)$. Here b is *vec4* with three first components being a and last being 1.0.

4. in the last step, you should update the information about the attributes and buffers in the vertex array object (VAO). To this end, similarly as it is done for vertex position attribute, after creating and binding VAO, bind the new buffer, enable attributes, and bind the buffer to the attribute.

After completing all the above steps, you should see a triangle with a color smoothly changing between the corners.

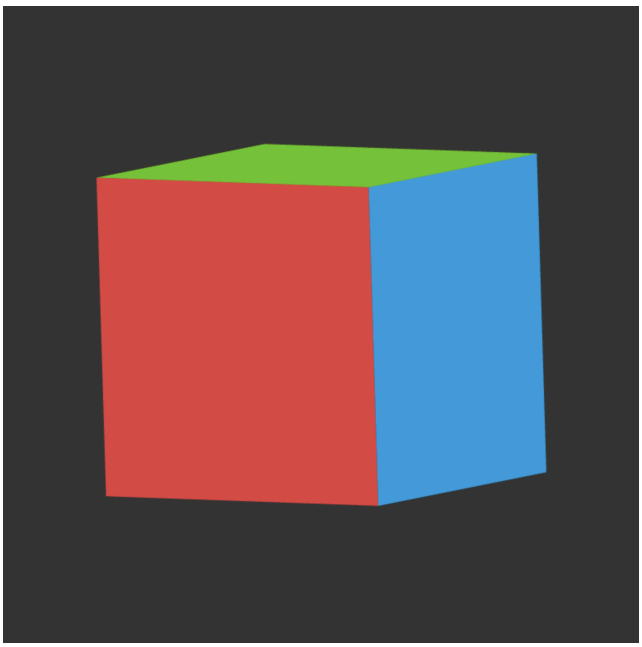
Exercise 2 [8 points]

Your next task is to extend the application to render a cube. To this end, you need to modify the *geometry.js* file, where you have to add vertices of all triangles which form the cube as well as define corresponding, per-vertex colors. While completing this task, you have to remember that:

- The arrays defining the vertices and colors are one-dimensional arrays. WebGL pipeline interprets each consecutive triplet as a data for one vertex.

- Our simple rendering application displays only triangles defined within $[-1 \dots 1]^3$ cube. Everything else won't be displayed. You should make sure that the whole cube is visible as it is rotated with the slider.
- The coordinate system is defined as follows: the x-axis is pointing right, the y-axis is pointing up, and the z-axis is pointing towards the screen.
- For the proper rendering of the cube, you should enable depth test and face culling. Please look for the commands in the code and uncomment them.
- Remember to define the triangles such that the order of the vertices for each triangle is counter-clockwise. WebGL uses the order of the vertices to determine the orientation of the triangle with respect to the camera (back vs. front). If the order is incorrect, you won't see triangles.
- Remember that function $gl.drawArrays$ takes as an input the number of vertices. You have to increase it to match the number of vertices to be drawn. Also, make sure that when creating the buffers the data for cube is now transferred to the GPU.
- Now, the rotation sliders becomes handy. You can use it to rotate your object and see how it looks from different directions.
- Remember to color different sides of the cube in different colors. Otherwise, it will be very hard to see a cube there. In the next lecture, you will learn how to implement shading. Having a single color cube won't be a problem any more ;)

After completing the task, you will be rewarded with a view of a cube! If you got it right, it means that you got a basic understanding of how the general graphics pipeline is realized using WebGL. You are now ready to shade the cube... (to be continued).



Bonus [2 points]

You can further extend your application and combine the two exercises in one. For this, you should define two separate VAOs, one for the triangle and one for the cube. The application should have an additional input method, e.g., button, to switch between rendering the triangle and the cube.
Hint: If you define two VAOs correctly, the only thing you need to do at the drawing stage is to bind the right one.

Submitting the assignment

Submit an archive file (ZIP) with all the files required for running your application. Unless you complete the bonus exercise, your submission should include a separate HTML file for each exercise. One for rendering the triangle and one for the cube.

Solutions must be returned on November 17, 2022 via iCorsi3