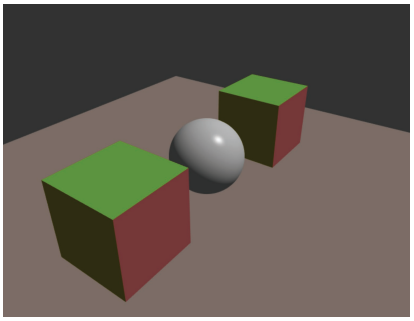
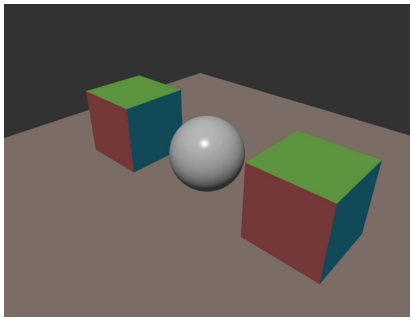
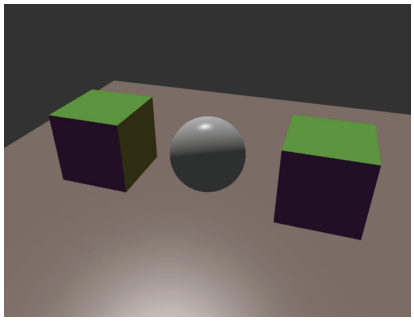
Università
della
Svizzera
italiana

Faculty
of Informatics

Computer Graphics

Assignment 5: Rendering Graphics Pipeline with WebGL

In this assignment, you will implement a simple rasterization pipeline to render a scene with basic objects. You will start with a template that already includes the rendering of a simple triangle. As you progress, you will add new geometries, implement a transformation pipeline, and incorporate illumination handling. Your final results should resemble the figure above. The specific arrangement of objects in your scene is not critical; however, your scene must include a ground plane with several objects, such as at least one sphere and two cubes, placed directly on the plane. Make sure that everything is illuminated by a light source. Additionally, you will implement camera and light controls, allowing you to interactively change their positions. You can see the video included in the assignment that demonstrates interaction with correctly implement renderer.

Template

The provided template consists of several files:

- **template.html** - It contains the code for rendering the triangle presented during the lecture with several comments providing tips for completing the assignment.
- **gl-matrix-min.js** - *glMatrix*[1] library containing a set of functions for vector and matrix operations.
- **gemetry.js** - Javascript file for defining geometry. Currently, the triangle definition is included in the main file (template.html), but it is advised to keep the rest of the geometry required for this assignment in a separate file. Currently, the file contains the definitions of empty arrays for the plane and the cube, as well as a function that generates the vertices of the sphere. Note that since the code generates a unit sphere, the vertices can be reused as normal vectors.

The template also contains the definition of sliders for controlling the positions of the camera and light. The main code defines already variables that read the values from the sliders.

Exercise 1 [10 points]

In this exercise, you need to add definitions of the objects' geometry, implement the transformation pipeline, i.e., define all the matrices based on the slider values, and render the scene accordingly.

[1] `http://glmatrix.net/`

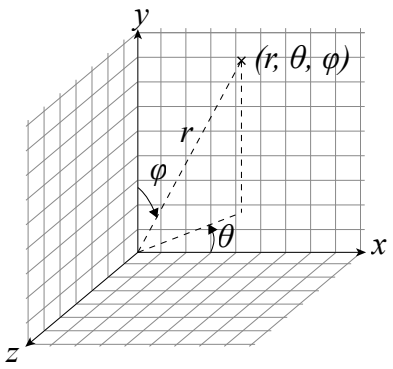


Figure 1: The definition of azimuthal and polar angles used in the template.

Object definitions

We suggest to define the objects as follows:

- **Ground plane** - two triangles defining a square of size 1×1, centered around the center of the coordinate system on the XZ-plane.
- **Cube** - 12 triangles defining a unit, axis-aligned cube around the center of the coordinate system.
- **Sphere** - the code included in the template generates the vertices of the sphere, which can be reused as normal vectors. There is also an array containing colors which is set to white.

Note that besides vertices of the triangles, you also need to define per-vertex normal vectors and colors.

Transformation pipeline

You should add to your renderer the transformation pipeline, as discussed during the lecture. This includes adding projection, view, and model matrices. For defining the position of the camera, we will use spherical coordinates[2], where θ, ϕ, and r are the azimuthal angle, polar angle, and distance to the center of the coordinate system, respectively. See Figure 1 for visualization.
You can assume that the camera should look directly towards the center of the word coordinate system. For the definition of the matrices you can use *glMatrix*[3] library which is already included in the template. The functions that you may find in particularly useful now are: **vec3.fromValues**, **mat4.create**, **mat4.lookAt**, **mat4.perspective**, **mat4.fromTranslation**, **mat4.scale**.
Your solution should use input from the provided sliders or should implement similar functionality, for example, based on the mouse input. Upone correct implementation of the transformation pipeline, you should be able to look around the cube from all possible directions.

Completing the exercise

To render the scene, you will need to expand the rendering pipeline by adding new VBOs and VAOs and all the code required for rendering the geometries. You will also need to modify the shaders to handle the matrix transformations. We advise you to complete the exercise in the following steps:

- Add definition of the plane, only vertices.
- Add definitions of the projection and the view matrices and render the plane.
- Add the model matrix for the plane and make sure you can rotate, scale, and translate the ground plane.
- Start adding new objects.

[2] `https://en.wikipedia.org/wiki/Spherical_coordinate_system`
[3] http://glmatrix.net/

- Define per-vertex color information, and incorporate it into your rendering to change colors of your objects.

Note that if you want to render two instances of the same object, e.g., two cubes, you do not need to create two different sets of buffers. You can simply draw the triangles twice, but in the meantime change the model matrix that is suplied to the vertex shader.

Exercise 2 [5 points]

In this exercise, you need to add lighting computation. You should add one directional light source. The template already includes sliders for controlling its direction using azimuthal and polar angles. The lighting should be modeled using the Phong reflection model. For the light computation, you will first need to specify the normal vectors in the geometry file (for the cube and the plane). You can do it manually or write a little function that computes the normals automatically from triangles. You will then need one additional attribute associated with each vertex, i.e., normal vector. For that, you will need to create additional buffers where the normal vectors will be stored, as well as set up the additional attribute similarly as it is now done for position and color attributes. Finally, you can use the normal vectors and, optionally, material parameters provided to the shaders as uniforms to shade the geometry. We ask you to include one specular sphere in the scene, as it will let you verify that the lighting computation is correct.

Submission

Your submission **must** contain one ZIP-file with:

- a readme file or a PDF document file with information about which exercises you solved, the authors of the solutions, and the explanation of encountered problems, if any,
- a directory named *code* containing all the source code required for running your web-based renderer.

The ZIP file name must be of a form *surname1_surname2.zip* for a team of two, and *surname.zip* for a single submission. Only one person from the team should submit the solution.

Solutions must be submitted via iCorsi by the indicated there deadline.