Università della Svizzera italiana

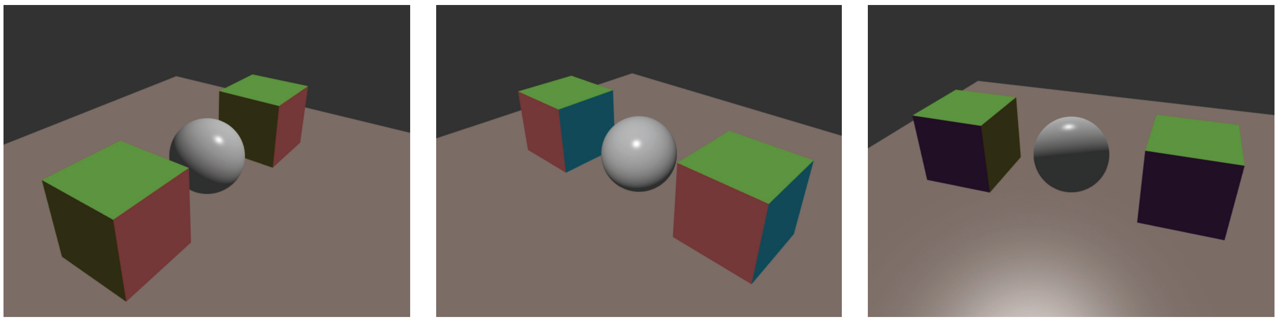
Faculty of Informatics

Computer Graphics (Fall 2022)

Assignment 9: Transformations and Lighting

November 17, 2022

In this assignment, you will first implement a basic transformation pipeline, which includes defining arbitrary camera position, a correct perspective projection, as well as setting up the position and orientation of objects in the scene. Next, you will add more objects to the scene (cube, sphere, and a plane), and position them using the transformation pipeline to form a simple scene. In the next part of the assignment, you will add a directional light to your scene. If you complete all the tasks, your scene should look similar to the one below:



Template Update

The updated template includes now the solution to the previous WebGL assignment. You can either start from the template or use it as a guide for continuing from your solution. We strongly encourage the latter. While designing the new assignment, we take care that it is easy for you to copy new snippets of code we provide. Besides the usual comments and hints on how to complete the assignment, we also added to the template:

- New sliders for controlling the positions of camera and light
- New geometry definitions:
  - plane (vertices, colors, and normals)
  - sphere (vertices, and color)
  - empty definition of a function which computes normals (see exercise 3)

Again, feel free to modify the code as well as the geometry to make the scenes more interesting.

Important: Note that in the previous assignment, we defined the cube directly in the normal device coordinates, which is a left-handed coordinate system. With the introduction of the transformation pipeline, the word and object spaces are right-handed coordinate systems. To fix our cube for this and next assignments, you will have to change the sign of z component of vertices. The rest of the geometry data provided in **geometry.js** is already correctly defined and does not need any modifications.

Exercise 1 [5 points]

In this exercise, 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

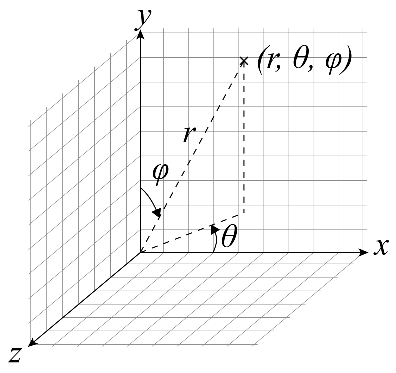


Figure 1: The definition of azimuthal and polar angles used in the template.

use spherical coordinates[1], where θ, ϕ, and r are the azimuthal angle, polar angle, and distance to the center of the coordinate system, respectively.
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*[2] library which is already included in the template. Please familiarize yourself with it since it will be useful also for other assignments. 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. After solving this exercise, you should be able to look around the cube from all possible directions.

Exercise 2 [4 points]

In this exercise, you have to add more objects to the scene. You can use the definitions provided in the **geometry.js** file. You should add at least one more cube, one sphere, and a plane. Define model matrices for each of the objects in such a way that both cubes and the sphere are located on the plane. The shapes should also not intersect each other. We suggest, you create separate VAOs for each shape.

Exercise 3 [4 points]

In this exercise, you finally add lighting. 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 Phong reflection model. For the light computation you will 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. In the **geometry.js** file, we already provide you with definitions of normals for sphere and plane. The normals for cube are missing. You should implement a function which computes the normal vectors directly from the definition of the triangle vertices. The function should be general in a sense that it works for an arbitrary array of vertices defining triangulated shape, not only cube. In the template, there is already an empty function for this task. Please look for more instructions on this exercise into the template.

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

Exercise 4 (2 points)

A triangle formed by points $A = (0, 0, 0, 1)$, $B = (0.5, 0, 0, 1)$, $C = (0, \frac{1}{\sqrt{2}}, 0, 1)$ is rendered with the following model-view-projection matrix (i.e., the product of projection, view, and model matrix):

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & -0.25 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Determine the percentage of the rendered image which is covered by the triangle.

Bonus [3 points]

As a bonus, you can add to your scene a point light source. To this end, add additional controls for setting the position of the light in the 3D space. We suggest using sliders for controlling (x, y, z) position, such that the light can be placed anywhere in the scene. Remember that in contrast to the directional light, now the direction of the light is different for each vertex. Also, you should implement the intensity fall-off related to the distance between the surface and the light source and handle the case when it approaches zero. To make the rendering intuitive for debugging, you can draw in the scene a little sphere at the location of the light source.

Submitting the assignment

Submit an archive file (ZIP) with all the files required for running your application. Your solution should contain one *.html* file which includes the solution to one or more parts of this assignment. Please note, that the assignment is formulated in such a way that the Exercise 1 is a prerequisite for the Exercises 2 and 3. We suggest doing the exercises in the order specified in this document.
For Exercise 4, please include into the ZIP file a PDF or image with the solution.
Please specify in the comment in iCorsi which exercises you solved.

Solutions must be returned on November 24, 2022 via iCorsi3