Programming Assignment 1: Triangle Meshes and Transformations
CMSC427: Computer Graphics, Fall 2020

Submission deadline: Tuesday, September 29th, 11:59am

In this assignment you will first write code to construct your own 3D objects consisting of triangle meshes. You should build on the Java code that you got to know in the initial assignment. In the second part of the assignment, you practice to manipulate 3D transformations and to set up an animated scene.

The deadline to submit your solution via ELMS/Canvas is Tuesday, September 29th, 11:59am. Please submit only the *Java files that you modified* plus a short report of your technical challenges, your solutions, your design choices and screenshots of your results for each part. Grading will take place in a meeting with the teaching assistant. Please sign up for a time slot via the list linked on ELMS/Canvas. Signing up and attending this meeting is considered part of the assignment, and we will impose the late penalty for students who fail to do so.

1 Based on Project 0 code

We assume you have completed Project 0 and have the base code running. If you have problems with the base code, please work with the TA immediately. This project can be prototyped in Processing if that helps, but you are required to deliver the implementation on the base code in the end.

2 Cylinder (30% Points)

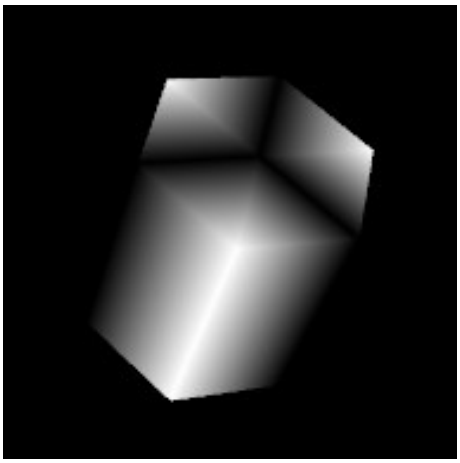
Write a function that generates a triangle mesh for a cylinder. The function should return arrays of vertices, colors and indices, which then can be used similarly to the cube mesh in the existing Java code in `simple.java`. Note that the mesh data structure used in the Java code is called (again, see the cube mesh as an example) a *face-vertex mesh*. See also the article on polygon meshes on Wikipedia for additional explanations about it.

Your function should accept a parameter for the resolution of the triangles mesh. This parameter indicates how many segments are used to construct the cylinder. The top and

the bottom of the cylinder should be closed using a disk. Use a color scheme which assigns different colors to the triangles (similar as in the figures below).

You do not need to compute vertex normals or texture coordinates (this is optional, if you want to try; see the results by pressing the 'd' or 'm' key when the application is running).

Caution: Take care of the order of the vertices for the triangles. The vertices must be ordered counterclockwise, if the surface is observed from the outside. This orientation is defined as "front facing". If the vertices are ordered clockwise, the triangle is not drawn when being observed from outside. This orientation is called "back facing".



Cylinder with 6 Segments

Cylinder with 50 Segments

3 Torus (30% Points)

Write a function that generates a torus, similar to the cylinder above. This Wikipedia page contains useful information about the mathematical definition of a torus.

4 Displaying the Meshes

Add a key listener in `simple.java`, similar to the existing key listeners, to allow the user to select which mesh should be rendered. For example, key 'c' displays the cylinder, and 't' the torus. Note that you can add and remove shapes (meshes) from the scene using `SceneManager.addShape` and `SceneManager.removeShape`.

5 Animation (40% Points)

Construct an animated scene consisting of a ground plane, and at least one moving object. The moving object needs to consist of at least three parts, and at least one of the parts needs to also move relative to the object. Use cylinders, tori and cubes to compose your scene. Model the scene by transforming (scaling, rotating, translating) the base objects (cylinders,

tori, cubes). Use the timer event to update the transformations and animate the object in the scene. Possible examples for scenes that meet these requirements are:

- A vehicle with spinning wheels, moving in a circle on a ground plane. The wheels rotate relatively to the vehicle, which in turn moves relatively to the ground plane.
- An airplane with a spinning propeller, circling above a ground plane.
- A helicopter with spinning rotors, circling above a ground plane.

Hint: Store all steps of the transformation of a scene part in separate matrices. Some matrices will be used by several scene parts. In each animation step, update selected transformation matrices that implement the animation. Then, multiply the matrices together (in the right order), and assign the final result to the scene part.

For example, a wheel of a car is rotated around its center (to turn around in the animation), and positioned relative to the car. The car in turn is translated away from the scene origin, and rotated around the scene origin (to move in a circle). So the overall transformation of the wheel could be written mathematically as

$$M_{\text{rotate car}} M_{\text{translate car}} M_{\text{translate wheel relative to car}} M_{\text{rotate wheel}}(\text{wheel}),$$

where each M is a separate matrix.