CS 300 | Advanced Computer Graphics I
Assignment 0 | Shape Library

Description

In this assignment, the main task to generate some 3D shapes procedurally and render them using OpenGL. Input file contains the scene information that should be represented on the application. Every item will be followed by its value separated by a space. A parser is provided that loads the required information. You will find the following information in the file:

- The camera vertical field of view, projection plane width and height, near plane, far plane, position, target and up values
- Objects in scene:
  - Every object will start with the keyword `object` followed by the name.
  - All properties read after an object will be applied to the last object added:
    - `translation`: Translation of the object
    - `rotation`: Rotation of the object in Euler angles. Order of rotations X first, Y after and Z last.
    - `scale`: Scale of the object
    - `mesh`: shape of the object
      - `PLANE`: plane aligned with the XY-plane with four vertices
      - `CUBE`: 6 faced cube where each side has four vertices
      - `CONE`: adjustable sliced cone divided radially in slices. The number of slices needs to be passed as a parameter upon initialization of the shape. Default slice number 4.
      - `CYLINDER`: adjustable sliced cylinder divided radially in slices. The number of slices needs to be passed as a parameter upon initialization of the shape. Default slice number 4.
      - `SPHERE`: adjustable sliced sphere divided into slices and rings (stacks). These numbers must be passed as parameters upon initialization and the number of slices will be double the rings value. Default slice number is 4 and 2 rings.
      - Other: Any other name will be interpreted as a mesh name that should be loaded. Not updated with slice changes and normals/texture coordinates are loaded instead of computed.

The application should:

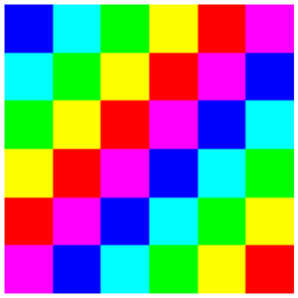
1. Read the input file
   a Load/Generate mesh data
   b Generate Texture
2. Upload data to OpenGL
3. Accept from the keyboard the user input in order to move the camera
4. Build the camera and perspective transformation according to the user input
5. Set shader program and uniforms
6. Draw each mesh
   a Vertex Shader: Transform vertex data, set gl_Position and pass texture coordinate to the fragment shader
   b Fragment Shader: Sample texture and output color
7. Finish frame

Shapes

- 5 shapes will be generated: plane (aligned with the XY-plane), cube, cone, cylinder and sphere.
- Each shape will be centered at the origin and will have one-unit size, i.e. they all fit exactly in a 1x1x1 axis aligned bounding box centered at the origin. The lower left back corner of the AABB is at (-0.5, -0.5, -0.5) and the upper right front corner of the cube is at (0.5, 0.5, 0.5).
- For each vertex the following information must be calculated: position, normal and texture coordinate. Two types of normal should be visible:
  - Face normal: compute the normal for the triangle and render it on the three vertices. That will produce more than one normal for each position and we will consider those as two different vertices.
  - Averaged normal: the result of averaging every normal for a single vertex position. In the cases where a vertex has the same normal for two faces, it will be included only once in the average.
- Each shape will have the following mapping for the texture coordinates:
  - Plane: Planar mapping
  - Cube: Cube mapping
  - Cone: Cylindrical mapping
  - Cylinder: Cylindrical mapping
  - Sphere: Spherical mapping

Rendering

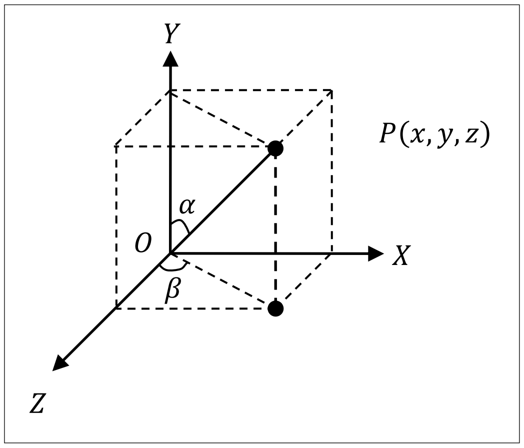
- The rendering function must be done from scratch, i.e. you must not use a third-party library to draw your shape.
- In addition to the regular shape rendering, must be able to render the shape's normal using lines.
- **Rendering must be done using OpenGL vertex array, either with glDrawArrays or glDrawElements function** (please read the documentation for both and make an educated decision)**.**
- Loaded mesh will always be Wavefront OBJ format and they will always have positions, texture coordinates and normals. The tinyobj library can be used in order to load the mesh data or you can write your own parser.
- Generate a texture procedurally that looks look like the following:



Blue: (0,0,1) ●
Cyan: (0,1,1) ●
Green: (0,1,0) ●
Yellow: (1,1,0) ●
Red: (1,0,0) ●
Purple: (1,0,1) ●

Input

- Camera will be controlled using spherical coordinates defined in the diagram. The target will be the origin of that coordinate system. The input should alter r, α and β as follows:
  - W: Make α angle smaller. It should move the camera towards the top of the target.
  - S: Make α angle greater. It should move the camera towards the bottom of the target.
  - A: Make β angle smaller. It should move the camera towards the left rotating around the target.
  - D: Make β angle greater. It should move the camera towards the left rotating around the target.
  - E: Make r greater. It should move the camera away from the target.
  - Q: Make r smaller. It should move the camera closer to the target.



- N: Toggle normal rendering
- T: Toggle texture-mapping on/off
- F: Toggle face/averaged normal
- M: Toggle wireframe on/off
- +/- or Z/X: Increase/reduce number of slices (4 is the minimum number of slices)

Assignment Submission

Please refer to the syllabus for assignment submission guideline. Failure to the submission guidelines correctly might cause you to lose point.

Grading Rubrics

The following is a rough guideline on how your assignment will be graded and the weight of each part.

Feature	Grade %
Plane generation	5%
Cube generation	5%
Cone generation	10%
Cylinder generation	10%
Sphere generation	20%
Mesh loading	10%
Texture generation and mapping	10%
Normal rendering	10%
Shaders	10%
Code quality	10%