CSE 333/533: Computer Graphics
Lab 2: Shape Representation in OpenGL

Instructor: Ojaswa Sharma , TAs: Vishwesh Vhavle , Aadit Kant Jha

Due date: 23:59, 28 August 2023

Introduction

In this lab, we will explore the fundamental concepts of shape representation and rendering in computer graphics using OpenGL. We will delve into the face-indexed data structure that underlies most 3D models and demonstrate how to render a simple cube. Furthermore, we will explore the concept of parametric representation for complex shapes and learn how to generate a triangulated mesh from it. As a hands-on exercise, you must render a sphere using the techniques you've learned.

Objectives

1. Understand the face-indexed data structure.
2. Learn how to render basic shapes such as cubes.
3. Explore parametric representations of complex shapes.
4. Generate and render a sphere.

Face Indexed Data Structure

In computer graphics, objects are often represented using a face-indexed data structure. This structure defines a set of vertices, edges, and faces, forming the foundation for various shapes.

Mesh Class

We start by defining our mesh class that looks something like below:

```
class Mesh : Drawable
{
private:
   int NUM_V, NUM_F;
   //Vertices position of shape Vx3
   GLfloat *vertices;
   //Indices into vertices for triangles of shape Fx3
   GLuint *indices;
   glm::mat4 modelT;
   void drawEdges(unsigned int &shader_program);

public:
   //Drawable functions
   void setup();
   void draw(unsigned int &shader_program);

   Mesh(GLfloat *vertices, GLuint *indices, int nV, int nF)
   {
       this->vertices = vertices;
       this->indices = indices;
       this->NUM_V = nV;
       this->NUM_F = nF;
       this->modelT = glm::identity<glm::mat4>();
       //Call drawable's setup
       this->setup();
   }

};
```

The main components of the mesh class are:

NUM_V: Number of Vertices or V.
NUM_F: Number of Faces or F.
vertices: A **Vx3** array of GLfloat denoting the vertices of the mesh.
indices: A **Fx3** array of GLfloat denoting the indices of vertices forming F triangles.

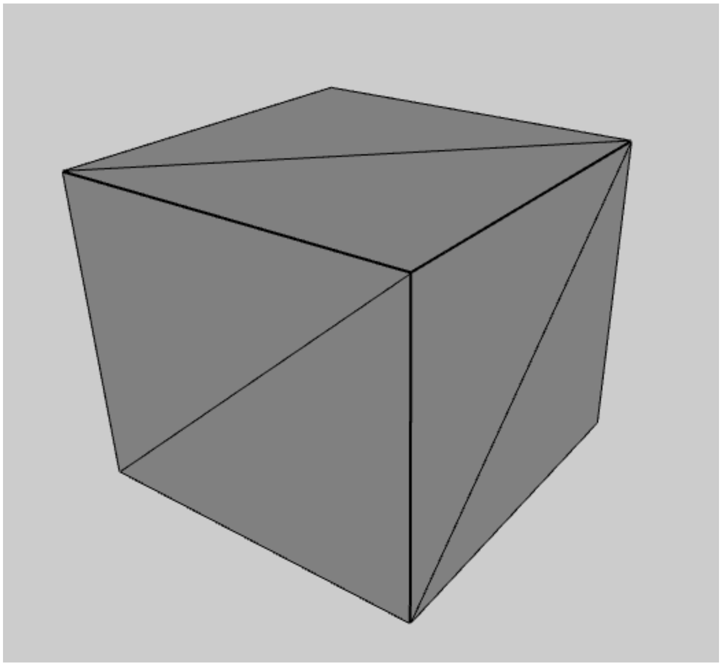
The setup() function will set the VAO, VBO, and EBO (Same as VBO but for indices) for the given meshes.

The draw() function will render the given mesh with the shader program provided to it.

Rendering a cube as a Mesh

To render a cube as a mesh, we will first define the vertices arrays as defined in the previous lab and the indices arrays. The vertices array will be of size 8x3 for eight cube vertices. The indices array will be 12x3 for 12 triangles, making up the six faces of the cube indexing into the vertices array. Finally, we get the mesh by creating an instance.

Once the cube mesh is all setup, rendering it is easy, as we call the draw() function of the mesh, passing it to the shader program.


Cube rendered using Mesh class

Parametric Representation of Surfaces

Writing out all the vertices and face indices for more complex shapes is almost impossible. There are different ways of tackling rendering more complex shapes, and we will look into one of the ways.

A parametric representation of any surface or parametric surface, in general, is any surface that can be defined by a parametric equation. For a shape residing in R^3, we require at most two parameters to define its parametric equation. We shall now examine how to compute parametric equations for some classical shapes as well as how to render these surfaces using parametric representations.

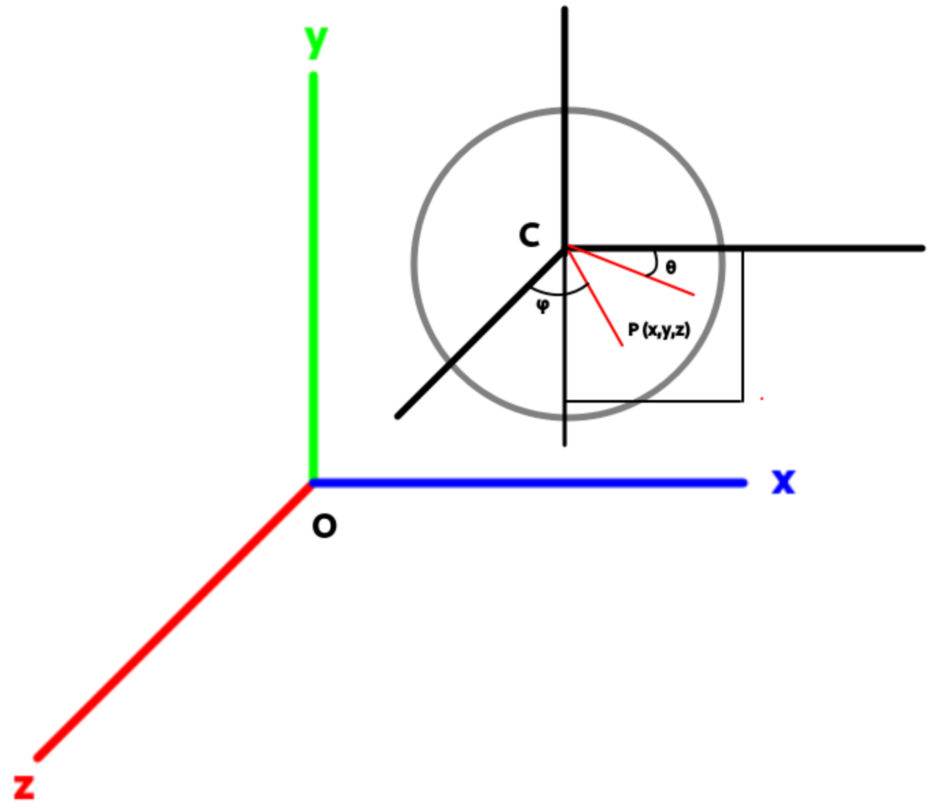
Parametric Equation of Sphere

Let us try to formulate the parametric equation for a sphere in R^3. By definition, any point on the surface of a sphere of radius **r** and center **C(h,k,l)** would follow the equation:

$$\mathbf{P(x,y,z): (x-h)^2 + (y-k)^2 + (z-l)^2 = r^2}$$

Now, this is what we call an implicit representation. To get the parametric equation, we need to consider two parameters θ and φ where:

φ: the angle vector CP makes on the z-axis.
θ: the angle the projection of vector CP on the x-y plane makes with the x-axis.



Now, if we write out the x,y, and z coordinates of the point P, we will get:

$x(\varphi,\theta) = h + r \sin(\varphi) \cos(\theta)$
$y(\varphi,\theta) = k + r \sin(\varphi) \sin(\theta)$
$z(\varphi,\theta) = l + r \cos(\varphi)$

where h,k,l, and r are constant.

This forms the parametric equation of the sphere. Now, one thing to note is that we need to specify the domain of **(φ,θ).** Simply looking at our sphere, we can see that:

$0 \leq \varphi \leq \pi$ and $0 \leq \theta < 2\pi$

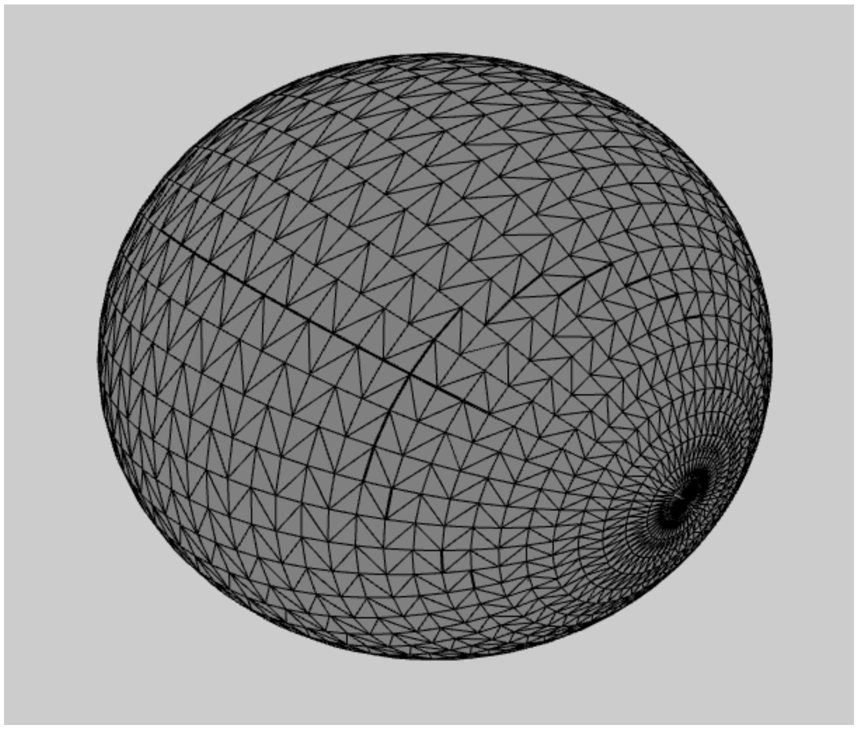
Rendering a Parametric Surface

Now that we have an idea of what parametric equations look like. The next step is to utilize them to render a surface in OpenGL.

To render any parametric surface in OpenGL:

1. Generate a plane of u-v values ranging from the specified surface's domain.
2. For each u,v value, compute x(u,v), y(u,v), and z(u,v). This will give you the vertices array.
3. The order in which you generate the u-v grid will define the indices array.
4. Instantiate a mesh with the generated vertices and indices array.

By doing this, you can render any parametric surface stored with the help of the face-indexed data structure.



Parametric Representation of Sphere

Exercise

Your task is to implement what you have learned and render a sphere, as shown in the example above. Note that the orientation of your triangulation might differ, and that's okay.

Deliverables

Upload the code zip file and image of the output, a sphere.

Name the zip file as **lab02_<name>_<roll number>.zip**
Example: lab02_vishwesh_2020156.zip

References

- Parametric Surfaces
- Face-Indexed Data Structure
- More Parametric Surface Examples
- Parametric Surfaces: Handout