Introduction to Computer Graphics CMPS 4213

In this assignment, you will expand your skills from building 2D scenes to working with 3D scenes. The main difference between 2D and 3D graphics is the manipulation of a third coordinate, the **z-axis**, which allows you to position shapes and faces in three-dimensional space. Instead of drawing shapes on a flat plane.

Part I Create a cube (20 points)

In a 3D scene, every object is built from a set of points (vertices), each defined by three coordinates — x, y, and z. These points are then connected to form **faces** or **triangles**, which make up the surfaces of your 3D object. For instance, a cube is made of six square faces, and each face can be divided into two triangles for rendering purposes.

When working in 3D, it is useful to separate the **geometry** (the raw data that defines where points are in space) from the **topology** (the relationships between those points that define which vertices form a face or triangle). This separation provides flexibility and control, especially when you need to modify, transform, or render your objects.

To efficiently manage 3D objects, data structures such as lists and dictionaries can be used. For example, consider two shapes—a cube and a tetrahedron. The information required to render these shapes is stored in lists. In the case of the cube, the **cubeFaces** represent triangles that define the connections between vertices, while the **cube** list itself contains the vertex coordinates. This structure maintains a clear separation between topology and geometry, ensuring that any changes made to geometry do not affect the topology. Each shape also includes a sub-dictionary containing transformation information, such as initial rotation, translation, and scaling. Additional details, such as material properties, can be included as well, although lighting calculations are not required for this assignment. Figure 2 shows a function that takes geometry and topology data and constructs an array or buffer formatted to match the expected input structure for WebGL

(you may use this function, but it is not mandatory). Keep in mind that the chosen data structure should allow easy access to all relevant information for rendering.

```
[cubeFaces,  cube,  {r: [0.0, 0.0, 0.0],
    s: [SOLIDS_SCALE, SOLIDS_SCALE, SOLIDS_SCALE],
    t: solidPositions[0]},
    materials.Brass,
    getNormals
],


[tetraFaces, tetra, {r: [0.0, 0.0, 0.0],
    s: [TETERA_SCALE, TETERA_SCALE, TETERA_SCALE],
    t: solidPositions[1]},
    materials.Ruby,
    getNormals
],
```

Figure 1 shape example1

```
function makeShape(faces, vertcies, model, getNormals) {
    let buffer = [],
        shape = {};

    faces.forEach(face => {
        face.forEach(index => {
            buffer.push(...vertcies[index]);
        });
    });
    // shape.faces = faces;
    // shape.vertcies = vertcies;
    shape.model = model;
    shape.vertexBuffer = makeBuffer(gl, buffer);
    shape.nFaces = faces.length;


    return shape;
}
```

Figure 2 Using the shape information to generate buffers for the graphics card

Part II Modeling (50 points)

You will create a set of functions that can draw each of the Platonic solids shown in Figure 3, as well as a sphere. Each function should draw the object so that its center is located at the origin.

Figure 3 Platonic Solids (image source: Wikipedia).

MathWorld provides formulas for calculating the vertices of each Platonic solid. There are many different methods for drawing a sphere, and you can find various approaches in different sources.

One approach is to start with a primitive shape—such as a tetrahedron, octahedron, or icosahedron—and then subdivide each face into four smaller triangles. This is done by adding a vertex at the midpoint of each edge and then connecting these new vertices to form three new triangles within each face.

To make the new points lie on the surface of the sphere, each newly added vertex must be "pushed out" to the sphere's surface. This is accomplished by normalizing the vertex position vector so that it lies exactly one unit from the origin (for a unit sphere).

The functions you create should be as follows:

- DrawTetrahedron
- DrawCube
- DrawOctahedron
- DrawDodecahedron
- DrawIcosahedron
- DrawSphere

Each Platonic solid function is worth **8 points**, and the sphere function is worth **10 points**.

Part III Creating a 3D scene (15 points)

Having multiple 3D shapes allows you to create more complex and visually interesting scenes. For instance, you can use cubes to construct 3D bricks for your first scene and make it 3D. By applying non-uniform scaling, you can transform these shapes to form elements such as the ground, walls, and buildings. You can also use a pyramid shape to represent a ceiling or roof structure. Extending the concept of rain from 2D to 3D adds further depth to the scene—rather than simple flat shapes, you can use 3D objects such as cubes, tetrahedrons, or other Platonic solids you developed earlier. Additionally, objects in the scene can now be rotated freely around the x, y, or z axes, allowing for more dynamic and interesting movements. The default view should show off your scene. You will now have to deal with a camera. You want the camera to show your scene off, so pick a camera location that does that, and set the location to look at (the center of interest in your scene) and then set the up vector.

Potential extensions (optional)

- Modelling, a particularly complex or creative scene.
- Interaction points for using the mouse, menus, or sliders for modifying the scene. This could be moving the camera or objects, or modifying their materials, etc. For camera movement and object orientation.
- Changing colors.
- Allow the user to rotate/scale/translate the view.

Report (15 points)

The report should be done in a word processor. Either a .doc, .docx, or .pdf file will be accepted.

1. Don't forget your name!

2. Provide an estimate of the total number of hours for the entire assignment

3. Results from parts 1 to 3.

4. Write up your experience, including any problems you ran into.

6. Mention added extensions.

5. List any sources you used.

6. Provide snapshots for parts 1 to 3. Name the images FirstLastLab2a.jpg, FirstLastLab2b.jpg, where First and Last are Your names. The snapshots should also be included in the report.

Submission All files (all source, your report, and your images) should be put into a single zip file and uploaded. I can handle, .zip, A zip file with your final source code, the report, and the images. The images need to be embedded in the report and as separate files in the zip file.