CSE 528 COMPUTER GRAPHICS (FALL 2024)

HOMEWORK THREE

Please note that this homework consists of a programming part only. Please refer to the TA Help Page for general requirements, and take note that all deadlines are **strictly enforced**. To prevent late submissions caused by network issues, **avoid** submitting at the last minute! Please also note that your score on Brightspace will be scaled by 50% (i.e., from $300 + 100$ to $150 + 50$) to match the overall grading schemes.

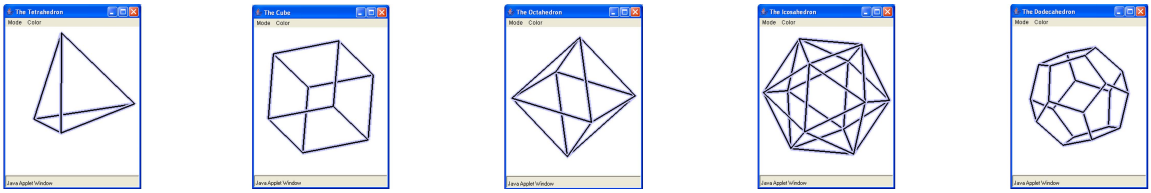
Programming Part

The primary aim of this programming part is to focus on applying 3D graphics techniques to edit, manipulate, control, and render geometric shapes in common use such as cube, tetrahedron, octahedron, sphere, cylinder, cone, ellipsoid, torus, etc. Write a program that implements the drawing of polyhedral objects and quadrics primitives. Your program should be able to: **(1) Render wireframe objects**; **(2) Render flat-shaded solid objects**; **(3) Render smoothly-shaded solid objects**; and **(4) Afford users to interactively translate, rotate w.r.t. any points/axes, scale/zoom independently along three axes, shear arbitrarily, and reflect w.r.t. any user-specified plane of the displayed objects**.

Please note that you are still required to use OpenGL in the **core-profile** mode (modern OpenGL). Deprecated functions in immediate mode (legacy OpenGL) are **not** permitted. Before you continue, please review the course materials and the handout documents carefully. If your tessellation shader is not working, you could try to disable the 3D Acceleration feature of your VMWare host.

In particular, the entire program must consist of **the following components as far as the system contents and functionalities are concerned**:

1. (**24 points**) First carefully review Figure 1. Second, write your program to display the first three shapes (Figure 1 (a, b, c)).



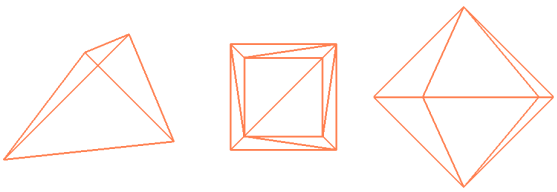
(a) Tetrahedron (b) Cube (c) Octahedron (d) Icosahedron (e) Dodecahedron

Figure 1: Five commonly-used polyhedral objects (also called *platonic solids*).

User Interaction. The user will press key **1** to enter this part. You should display the three shapes in the scene. You are free to determine their locations in the scene. However, they should **not** obstacle other shapes under the default camera setting. You may want to reset the camera when the user switches to a new mode.

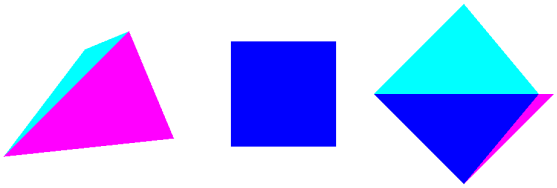
Display Modes. You should also implement local illumination for all objects in the scene with the Phong illumination model. Please note that these *display modes* should be available in all subproblems in this programming part. The user should be able to press the following keys to

switch to (1) `WIREFRAME` mode (key **F1**), which displays wireframe objects; (2) `FLAT` mode (key **F2**), which displays flat-shaded solid objects; and `SMOOTH` mode (key **F4**), which displays smoothly-shaded solid objects. Please note that, key **F3** (`NORMAL` mode) is reserved for the BONUS part. Please refer to the lecture slides and the attached materials for the implementation details of each mode as well as the Phong illumination model. Please also refer to Figure 2 for the expected visual effect of each mode.

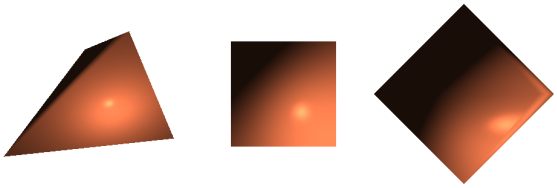


(a) `WIREFRAME` mode: wireframe only

(b) `FLAT` mode: flat-shaded objects



(c) `NORMAL` mode (see BONUS)



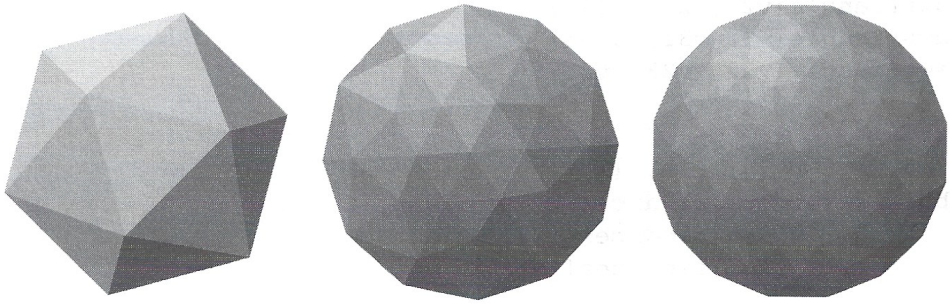
(d) `SMOOTH` mode: smoothly-shaded objects

Figure 2: Expected visual effect of different display modes.

Camera Functionalities. You should also allow the user to operate the camera for all subproblems of this programming part. The user should be able to: (1) Press key **X** to show or hide the positive x, y, z axes, with the x-axis displayed in red, the y-axis in green, and the z-axis in blue. Each axis starts from the origin and has a length of 10 (in the world coordinate system); and (2) Press keys **W**, **S**, **A**, **D**, **Up**, **Down**, or drag or scroll the mouse to adjust the camera. Hint: The camera functionalities are *already implemented* in the program template. However, you should add more code to the camera functionalities to accommodate the single-object transformation feature.

2. (**25 points**) Write a program to display an icosahedron of the unit sphere (i.e., a 20-triangle representation of the unit sphere) as shown in Figure 3 (a). Again, please refer to the program template for the details of an icosahedron. (The program template has hard-coded one unit icosahedron centered at the origin of the world coordinate system.)

   User Interaction. The user will press key **2** to enter this part. You should also implement the *subdivision* of an icosahedron. The user should be able to press key **+** to subdivide the icosahedron into an improved 80-triangle approximation of the sphere as shwon in Figure 3(b). The subdivision could be stacked, that is, the user could press key **+** again to further subdivide the 80-triangle approximation of the sphere into a 320-triangle approximation, as shown in Figure 3(c). Please note that to press key **+**, you actually press key **=** while pressing and holding key **Shift**. Hint: As shown in the pseudo-code below, the primary objective of the subdivision process is to generate a finer mesh of the approximated object:



(a) Original icosahedron (b) Subdivided once (c) Subdivided twice

Figure 3: Subdividing an icosahedron to improve the polyhedral approximation to a sphere.

```
// Subdivide a unit icosahedron
// (i.e., an icosahedron with all its vertices
// sampled uniformly from the unit sphere.)
// Takes as input a triangular facet on a unit icosahedron
// and subdivide it into four new triangular facets.
void subdivide(vec3 v1, vec3 v2, vec3 v3)
{
    // Sample new vertices so that
    // all vertices lie uniformly on the unit sphere.
    vec3 v12 = normalize((v1 + v2) / 2.0f);
    vec3 v23 = normalize((v2 + v3) / 2.0f);
    vec3 v31 = normalize((v3 + v1) / 2.0f);

    addFacet(v1, v12, v31);
    addFacet(v2, v23, v12);
    addFacet(v3, v31, v23);
    addFacet(v12, v23, v31);
    removeFacet(v1, v2, v3);
}
```

3. (**30 points**) Generalize your sphere drawing program to display an ellipsoid. In particular, your new ellipsoid drawing program should be capable of displaying a 20-triangle approximation, a 80-triangle approximation, and a 320-triangle approximation for the same ellipsoid.

   User Interaction. The user will press key **3** to enter this part. The interactions for subdivision remain the same as the icosahedron (sphere).

4. (**30 points**) Carefully study the program template and pay special attention to the drawing instructions of parametric surfaces. Display a sphere, a cylinder, and a cone with the tessellation shader. You could put these three objects in any size and anywhere in the scene, as long as all of them are completely visible (i.e., **not** occluded by each other) upon switching to this part.

   The major topic of this part is to display quadric primitives with OpenGL's tessellation shaders. You will learn to assemble quadric surface meshes from their parametric equations with the

help of OpenGL tessellation shader. For this part, you are **not** allowed to display these shapes by manually subdividing polyhedral objects.

For quadric surface meshes which are assembled from their parametric equations, you could just derive vertex normal from parametric equations. You are **not** required to handle face normal (which is used in the `FLAT` display mode). That is, for this part, the `FLAT` mode could look the same as the `SMOOTH` mode (without sharp edges.)

Meanwhile, you will have to be more careful about how you are going to deal with these special cases, especially cylinders and cones because these shapes should be considered to be *closed* to actually represent solid objects.

User Interaction. The user will press key **4** to enter this part.

5. (**60 points, 20 points each for (a), (b) and (c)**) (a) Based on this idea (explained above) currently used to implement parts above, draw a torus. Although you are free to select the number of polygons to approximate the torus, I suggest you start to sample a continuous torus using at least $15 \times 15 = 225$ different points to get a good approximation of a torus to start your program. Note that a torus's display quality at this stage should be reasonably good! (b) Once again, based on the subdivision strategy detailed above, generalize your program to handle the refinement of (a) for a torus, for example, 15×15 now becomes 30×30, at this stage, you could either directly work on the quadrilaterals or split one quadrilateral along one main diagonal into two triangles. Either strategy suffices here! (c) Perform one more level of refinement for a torus so that now the resolution becomes 60×60.

   User Interaction. The user will press key **5** to enter this part. The subdivision procedure should be consistent with previous parts. Hint: For your convenience, you adjust the tessellation granularity of a tessellation shader to achieve visual effects similar to those of a manual subdivision. You are also welcome to perform subdivision manually.

6. (**30 points, 15 points each for (a) and (b)**) (a) Based on the ideas explained above and in the class, please extend your program to the drawing of any super-quadrics. The implicit and parametric representations of super-quadrics are available in the textbook or via search over the Internet. Meanwhile, you will have to be more careful about how you are going to deal with special cases because some shapes should be considered to be close to actually representing solid objects. (b) Based on this idea (explained above) currently used to implement functionalities above, draw a dodecahedron (refer to Figure 1 (e)) and your program should support at least two levels of refinement based on the similar ideas presented above and also in the handouts. (Hint, search the Internet or mathematics/geometry textbooks for a reasonable approximation of dodecahedron geometry). If you have any questions, you are strongly encouraged to discuss possible challenging issues with Hong, Xi, and/or Chahat!!!

   User Interaction. The user will press key **6** to enter this part. You could arrange the two shapes in one scene, as long as they don't obstacle each other under the default camera setting. Moreover, the parameters of the super-quadrics should be read *dynamically* from the configuration file `subid_hw3/etc/config.txt`. You could determine the format of this line by yourself, but please detail it in your `README` file.

7. (**101 points, building a small-scale city that has urban structures utilizing all the shapes you have created and then performing your flight simulation**) Add a small-scale city that comprises all the urban structures supported by your shape repository from (1) to (6) above and

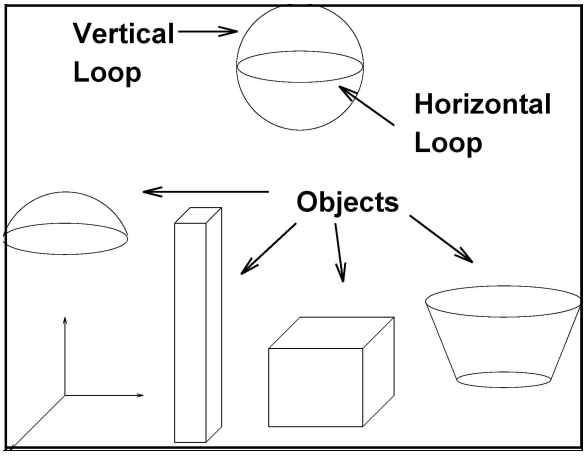


Figure 4: Flight simulation over your universe that comprises a small-scale city with 8-12 urban structures.

then fly over your universe (virtual world) that you have just created (including a few urban structures, e.g., skyscrapers, dome, stadium, and modern architectures). You are welcome to use NYC as a possible template to arrange their spatial distributions in order to imagine a few interesting landmarks, but you should utilize all the shapes from (1) to (6) above. In the interest of time, 8-12 urban structures with detailed geometry shape suffice for this purpose. As far as the flight simulation is concerned, there is a synthetic camera installed in the cockpit of the plane. The plane is allow to do either a **vertical** loop or **horizontal** loop above your universe. You should make use of an object hierarchy for rendering your universe (i.e., make use of the matrix stack). Note that, you should also switch between different display modes (in order to speed up your flight simulation). There are many methods in modeling the above-suggested objects. For example, one suggestion would be: modeling the stadium using truncated cone, modeling the dome using truncated sphere, and modeling the tower and high-rise buildings using a set of blocks (refer to Figure 4). Please note that, this is ONLY an illustrative figure, you are welcome to arrange the spatial distribution of your universe using your own imaginations. Your flight simulation should also try to cover your universe.

User Interaction. The user will press key **7** to enter this part. The user should be able to press key **H** for a horizontal loop, or press key **V** for a vertical loop. When the loop is completed, the camera should go back to its original position (its position, look-at direction, etc. before the loop.)

8. **(BONUS PART, up to 100 points for this project)**. You are welcome to explore different options. At the same time, I could suggest many possible extensions and generalizations. As a concrete example (as one of my suggestions here), you could try to render flat-shaded solid objects by mapping the normal (n_x, n_y, n_z) of each face to (R, G, B) components respectively (i.e., n_x to R, n_y to G, and n_z to B), as shown in Figure 2 (c). An another constructive suggestion, if you wish, you could explore the Internet to understand the current state-of-the-art methods for different modeling and design methods, and you are welcome to follow either existing ideas or explore your own ideas. As far as the shape modeling is concerned, you might think about how you could provide global and/or local editing tools to allow users to better control the modeled urban structures. For example, you could deform them either globally or locally. Alternatively, you could add a terrain (e.g., mountains, lakes, rivers, etc.), a theme park,

other urban structures, and/or a road system, in order to enhance the overall cityscape. As far as the urban structures are concerned, you are also welcome to explore different methods from what I had already suggested above. For example, you could (a) add more buildings into your universe, (b) make your objects more photo-realistic, etc. As far as your enhanced flight experiences, you might consider to allow your self-defined camera to follow an arbitrary spline specified by users, etc. It may be noted that, since these are the bonus parts, you are strongly encouraged to explore your own ideas. However, you are welcome to discuss with Hong, Xi, and/or Chahat about your ideas before you go ahead with your implementation for **the bonus part**. Please note that we are rewarding up to **100 points** for this part, so a reasonable amount of time investment is necessary. **PLEASE MANAGE YOUR TIME WISELY, HAVE FUN, AND ENJOY WORKING ON THE PROJECT!**