Computer Science 352 Spring 2023
Programming Assignment 9
Due 4/21/2023 by 7pm

The 9th PA for CS 352 requires you to add more features to the 3D library that you implemented for PA 8. You should create a copy of the files you used for PA 8 and place them into a pa9 directory. The files you should submit should be:

```
pa9
 ├── 3d.h and 3d.c
 ├── generator.c
 └── makefile
```

There will be a modified header file which you should download, but not modify. The **`makefile`** should produce a **`3d.o`** object file and the **`generator`** executable when **`make`** is typed. You will be required to implement several new functions described in the spec. You may add helper functions if needed too. **`make clean`** should remove any files generated by **`make`**. The generator executable should generate a 3d scene that includes at least one of each of the four shapes (cuboid, sphere, pyramid, fractal).

Binary STL File Generation

In this assignment you will be required to implement a new function:

```
void Scene3D_write_stl_binary(Scene3D* scene, char* file_name);
```

This function will be responsible for writing a Scene3D to a file using the binary STL format. A binary STL file still uses the .stl extension, but writes the data to the fine in a binary representation, which can result in more compact representations of triangles. This is especially important when we want to represent objects that have very high triangle counts. As a part of this PA, you'll implement two other functions for creating shapes that can potentially require thousands of triangles to represent them. The wikipedia page for STL has a good description of how an binary STL file should be formatted, so please read this before proceeding (https://en.wikipedia.org/wiki/STL_(file_format)#Binary_STL).

The format can be summarized as:

- The first 80 bytes are considered a header. Officially, these bytes can be anything as long as it does not begin with "solid". However, for this PA, you should set the first 80 bytes to be zeroed-out.
- The next 4 bytes should be the facet (triangle) count as an unsigned 4 byte int (**`uint32_t`**)
- After this comes the facets. Each facet should consume exactly 50 bytes.
  - The first 12 bytes should be three, 4-byte little endian floats (just **`float`** on lectura) representing the normal (just 0.0 for each)
  - The next 36 bytes should be the nine, 4-byte floats representing the coordinates of the corners of the triangle.
  - There should be a two-byte unsigned int (**`uint16_t`**) at the end, just 0

That's basically it! After you have implemented it, test it out by writing a program to generate a binary STL, and then try opening it up in either an online STL viewer such as https://www.viewstl.com/classic/ or open with blender (https://www.blender.org/).

Creating a Sphere

Next, you should implement this function:

```
    void Scene3D_add_sphere(Scene3D* scene,
        Coordinate3D origin, double radius, double increment);
```

This function should build a sphere (or really, an approximation of a sphere) in 3D space by arranging the triangles in a special way. The **`origin`** represents where the center of the sphere should be, the **`radius`** represents the radius it should have, and the **`increment`** is a value that determines how refined the sphere will look (a smaller number will make it look better, but will use more triangles).

The algorithm for creating the sphere involves converting spherical coordinates (a convenient way to specify points on the surface of a sphere) to cartesian coordinates (what STL uses, and what you are probably more familiar with). The outline of the algorithm is as follows:

- Loop through the "vertical" degrees **[increment, 180]**, with variable **`phi`,** incremented by **increment**
  - Loop through the "horizontal" degrees **[0, 360)**, with variable **`theta`,** incremented by **increment**
    - Convert the following four spherical coordinates to cartesian coordinates **(x, y, z):**
      - **(radius, theta, phi)**
      - **(radius, theta, phi - increment)**
      - **(radius, theta - increment, phi)**
      - **(radius, theta - increment, phi - increment)**
    - In order to do the conversions, you'll want to use the conversion formulas from section 4.1 of this article. If you've never worked with spherical coordinates before, read the whole page.
    - Place a quadrilateral between these four cartesian coordinates
      - Coordinates should be rounded to four decimal places using standard mathematical rounding. For example, 0.00015 rounds to 0.0002, and 0.00014 rounds to 0.0001.
      - When dealing with these coordinates, keep in mind to offset everything by the **`origin` (x, y, z)** values.

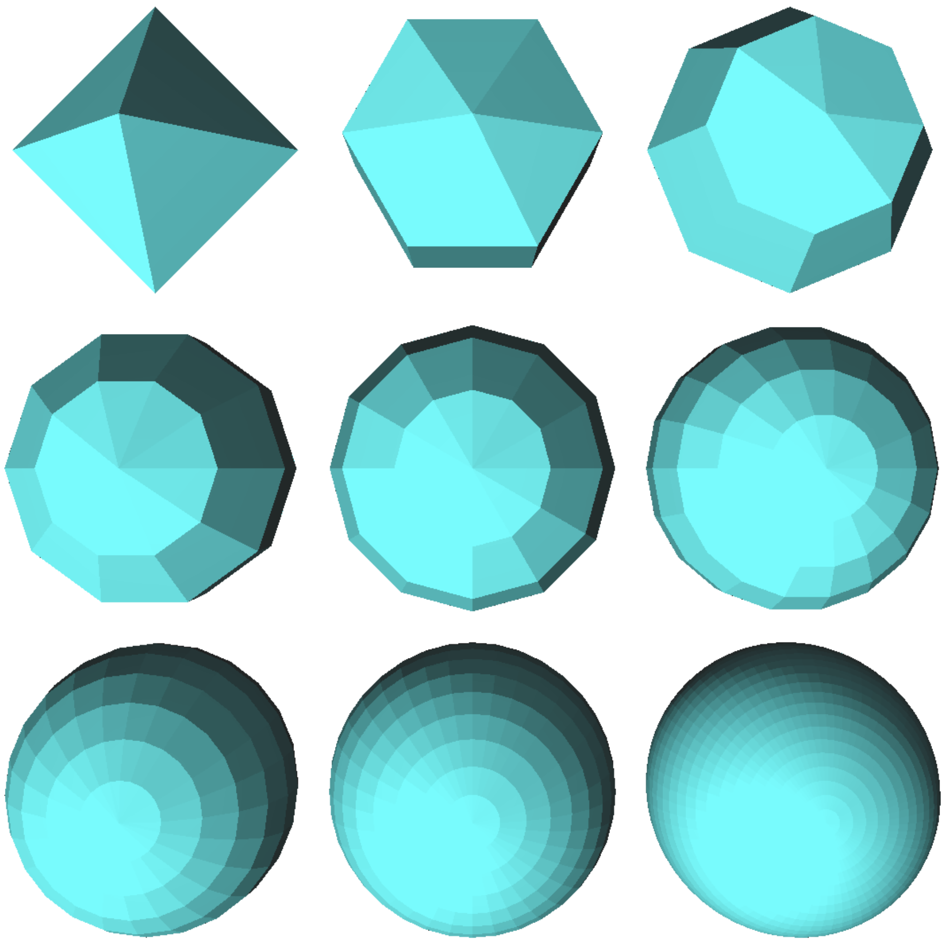
It might take a few iterations to get this algorithm totally correct. There will be a few public sphere test cases on gradescope, so you can use those to validate if you are placing the triangles correctly.

As mentioned earlier, the **`increment`** value controls how smooth and refined the sphere will look. A larger value for this will result in a jagged-looking sphere. A smaller value for **`increment`** will result in a much smoother appearing sphere, but will require more triangles to represent it. On the next page, I show an example that creates 9 spheres with different increments, and shows how the lower the increment, the smoother the shape (but also, the more triangles required)

This code:

```
    Scene3D* s = Scene3D_create();
      double increments[] = {15, 10, 5, 36, 30, 20, 90, 60, 45};
      for (int i = 0; i < 3; i += 1) {
        for (int j = 0; j < 3; j += 1) {
          Coordinate3D origin = (Coordinate3D){j*100, i*100, 0};
          Scene3D_add_sphere(s, origin, 45, increments[(i*3) + j]);
          printf("count: %ld\n", s->count);
        }
      }
      Scene3D_write_stl_binary(s, "sphere.stl");
      Scene3D_destroy(s);
```

Should produce:



(increment, triangle-count) from top-left to bottom-right:

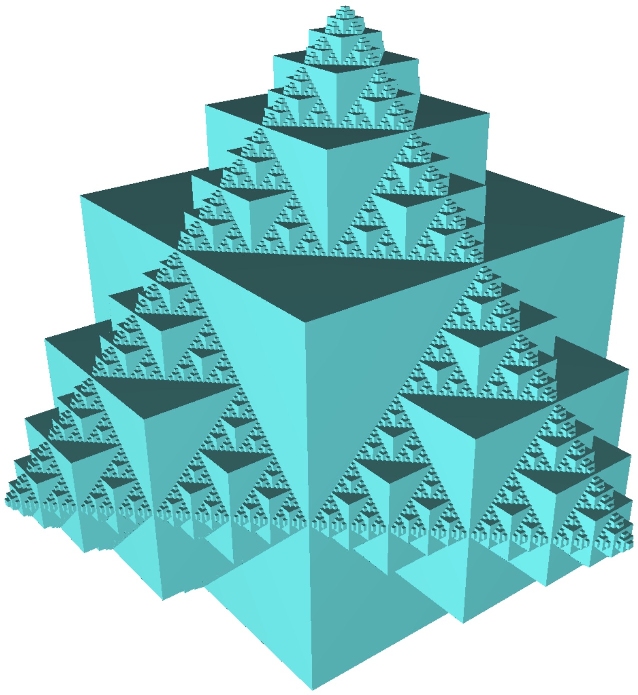
(90, 32), (60, 72), (45, 128)
(36, 200), (30, 288), (20, 648)
(15, 1152), (10, 2592), (5, 10368)

Creating a Fractal

The next function to implement is:

```
    void Scene3D_add_fractal(Scene3D* scene,
        Coordinate3D origin, double size, int levels);
```

This function will be responsible for creating a fractal in 3D space, using cubes. One kind of cube-based fractal is one called the Menger sponge, but we will be implementing a slightly different (and “easier”) one that will be named the “cubestep” for the purpose of this assignment. An example of a cubestep:



This shape can be defined (and created) recursively. Begin by creating one cube with its center at the **`origin`** and with a width / height / depth of **`size`**. For each **`level`**, the code should create 6 new cubes, each ¼ the volume of the current level (½ of the width / length / depth). The centers of each of these cubes should be placed in the center of each side of the current cube. These steps should then be repeated recursively for each created cube, until the recursive depth is reached.

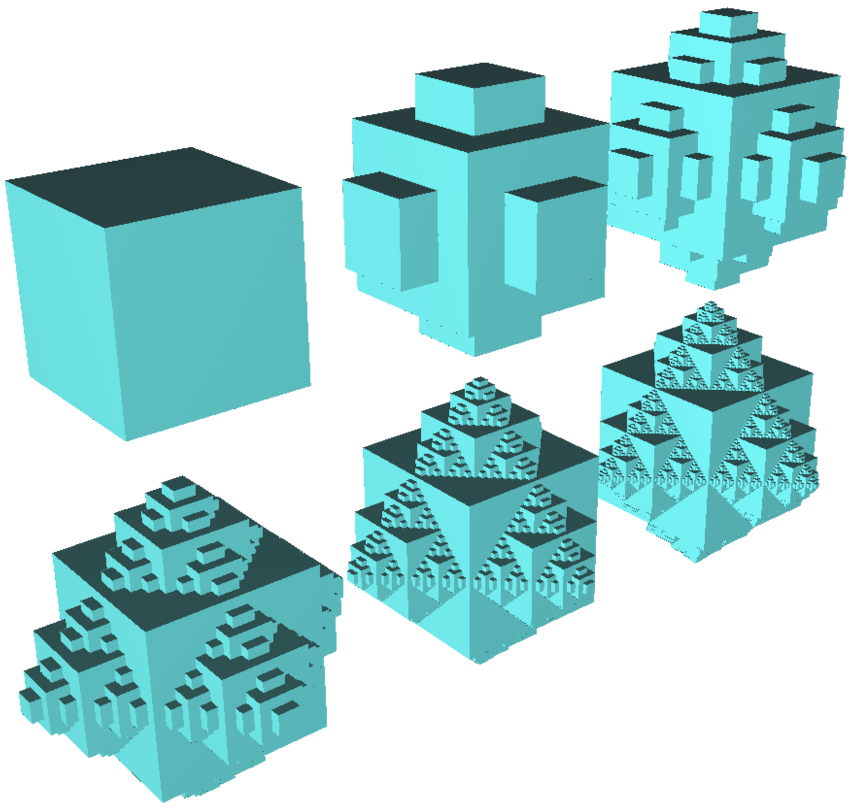
The example on the right shows this operation happening to 7-levels. You can and should test your fractal generator with various sizes. Though as a warning, the cost of this algorithm grows significantly as the number of levels increases, so beware! For example, when run with a depth of 8 on my laptop, it creates an object with 8,062,152 triangles, and a file size of 400 megabytes (and that is with BINARY format!).

The next page shows an example of creating several fractals.

This code:

```
        Scene3D* s = Scene3D_create();
         int levels = 1;
         for (int i = 1; i >= 0; i--) {
           for (int j = 0; j < 3; j++) {
             Coordinate3D origin = (Coordinate3D){j*100, i*100, 0};
             Scene3D_add_fractal(s, origin, 50, levels);
             printf("count for level %d: %ld\n", levels, s->count);
             levels += 1;
           }
         }
         Scene3D_write_stl_binary(s, "fractal.stl");
         Scene3D_destroy(s);
```

Should produce:



(level, triangle-count) from top-left to bottom-right:

(1, 24), (2, 168), (3, 1032)
(4, 6216), (5, 37320), (6, 223944)

Consistent Quadrilaterals

For this assignment, you should use the same `Scene3D_add_quadrilateral` function to draw any quadrilaterals. This includes any quadrilateral needed for a sphere or fractal.

PA 8 Functions

The functions from PA 8 should be included in your code and function the same way.

Submitting Your PA

Before you submit, you should ensure that:

- The file structure and file / directory names match what is required.
- The code compiles, runs, and functions correctly on lectura.
- The code is free form memory errors and leaks.
- Remove all extra and non-required files. If you are on a mac, you should avoid zipping the file on the mac, because it might include one or more hidden / extra files. Instead, zip on lectura.
- Follow the rules from the style guide.

Once you are ready to submit, zip up the **pa9** directory by running:

```
$ zip -r pa9.zip ./pa9
```

Then, turn this file to the PA 9 dropbox on gradescope.