Assignment 3: Geometric Modeling of a Scene

Loyola Marymount University, CSMI-371: Computer Graphics *Professor Alex Wong*

Given a real world scene, your goal is to replicate it using hierarchical modeling of the objects. You will begin by building prisms from planes using 3D translations and rotations. These prisms can then be used to form parts of objects.

The skeleton code is provided in a separate `assignment3.cpp` file

I have provided you with a function `init_plane` which initializes a square plane of unit lengths centered at the origin (0, 0, 0). You will utilize this initial plane to form cube, which can then be transformed and used to form objects.

I have also provided you with the `deg2rad`, `vector2array`, `to_homogeneous_coord`, `to_cartesian_coord`, and `init_color` functions.

- `deg2rad` -- converts degree to radians
- `vector2array` -- converts a vector into an array
- `to_homogeneous_coord` -- converts Cartesian coordinates to homogeneous coordinates
- `to_cartesian_coord` -- converts homogeneous coordinates to Cartesian coordinates
- `init_color` -- creates a color map for the scene

As `vector2array` dynamically allocates space and copies the elements of the vector into the array. You need to remember to deallocate the arrays created via `vector2array` after you have rendered your scene to prevent memory leak.

Since arrays are static, it is easier to work with the `vector` class before producing the final array of vertices that will be used for rendering.

You will need to first implement the `build_cube` function, which creates an unit cube. Then apply transformations to the cube to create objects in the scene (`init_scene`). Once the scene is built you will apply rotation to the scene such that the entire scene spins while the camera stays still. You may set the parameters of your camera in `init_camera`.

For the `mat_mult` function, please implement it such that it multiplies a transformation matrix A by the entire points matrix B rather than transformation A and point by point in B.

The `math` header file that I have included should be sufficient in doing the operations needed for this project.

You will complete the following functions for the assignment:

1) `translation_matrix (float dx, float dy, float dz)`
2) `scaling_matrix (float sx, float sy, float sz)`
3) `rotation_matrix_x (float theta)`
4) `rotation_matrix_y (float theta)`
5) `rotation_matrix_z (float theta)`
6) `mat_mult(vector<GLfloat> A, vector<GLfloat> B)`
7) `build_cube()`
8) `init_camera()`
9) `init_scene()`
10) `display_func()`

Note: functions 1-7 serve as helper functions to generate the objects in the scene (i.e. `build_cube` will create a cube in which you can then modify to form parts of an object). I would suggest using `vector2array` as the final step since going from an array to vector requires you to keep track of the number of elements in the array (which could be hard after generating thousands of points).

Submission:
You will submit the following to Bright Space:

1) `assignment2.cpp`
2) Either 6 different viewpoints taken from different angles of you scene submitted as JPEG, JPG or PNG with the names: view{1,2,3,4,5,6}.{jpeg, jpg, png}
   or a short video panning over regions of your scene
3) A photo in the form of JPEG, JPG or PNG (scene.{jpeg, jpg, png}) taken of the scene with which you are basing your model on

Grading:
I will be compiling the assignment using the following command:

```
g++ -o assignment3 assignment3.cpp -lglut -lGLU -lGL
```

Your code must compile for me to assign points!

Your assignment will be graded on:

1) the correctness of your implementation of the hierarchical modeling procedure
2) the correctness of your implementation of your camera and scene modeling
3) effort placed recreating the real world scene

Late Policy:
For each day the assignment is late, 50% of its worth will be deducted, e.g. 100% on time, 50% 1 day late, 25% 2 days late, etc.