Final Project Final Draft

CS 250, Spring 2018

Due Thursday, April 19

In the first draft of the final project, you demonstrated your knowledge of modeling transformations and camera transformations, but with the final rendering performed using OpenGL. With this assignment, you will demonstrate your mastery of 3D rendering techniques, which will also include clipping, but with the rendering done entirely in software. You are welcome to modify your first draft of the project to suit the new requirements, or you may wish to start anew. The requirements for the project are the same as the first draft, with some minor modifications, plus some additional ones.

I will provide you with the driver file `cs250final_proj.cpp`, which is responsible for copying the frame buffer to a window. You are responsible for setting the pixels in the frame buffer. This is done by implementing the class `DrawFrame` which has public interface

```
class DrawFrame {
  public:
    DrawFrame(Raster &r);
    ~DrawFrame(void);
    void operator()(double dt, int cam_num);
};
```

(you are free to declare the private section as you see fit). The driver creates a `DrawFrame` object prior to the first frame of animation, and calls the ()–operator of the object with each subsequent animation frame. When the driver is run, the frame rate will display in the window title bar, and the space bar of the keyboard is used to cycle through the three cameras.

`DrawFrame(r)` — (constructor) creates an object for drawing to the frame buffer associated to the passed–in raster object. The width and height of the frame buffer correspond to the width and height of the window that the driver program `cs250final_proj` writes to. You are to assume that these values do not change over the course of the animation; i.e., that the window will not be resized.

~DrawFrame() — (destructor) destroys the frame buffer drawing object. Depending on your implementation, this may be trivial.

`operator()(dt,cam_num)` — draws to the frame buffer. This function is called with each new animation frame. This is where you will set the pixels in the frame buffer associated to the raster object r specified in the constructor. The value of dt gives the time (in seconds) that has elapsed since the last animation frame, and `cam_num` gives the number of the camera to use: either 0,1, or 2 (see below).

Here are the requirements for this assignment.

- There must be a total of at least 50 objects in the scene. All objects must be represented by 3D triangular meshes. If you wish, all objects may be cubes, or composed of cubes.
- The scene must have a well–defined notion of up. For instance, this may be achieved by having several objects aligned on a common plane. All cameras will use this direction as the *relative* up direction.
- At least two of the objects must form an earth/moon pair. That is, one of the objects must be rotating about an axis passing through its center, and the second object must be orbiting the first object.
- A third object must be moving through the scene. Two of the cameras will view the scene from the point of view of this object, the *moving camera object.* This object must be rendered in red, or some other obvious color, and must be aligned along its direction of motion. You may *not* use the moon object as the moving camera object.
- There must be three (perspective) cameras: numbers 0, 1, and 2. The value of `cam_num` specifies which camera to render the scene with.
- Camera #0 should be a stationary camera. Both the earth/moon pair and the moving camera object should be visible using this camera.
- Camera #1 should move along with the moving camera object, with the camera look–at vector parallel to the direction of the moving camera object's *forward motion.*
- Camera #2 should also move along with the moving camera object. However, the camera should always look directly at the earth/moon pair.
- The scene should be rendered with the correct aspect ratio, regardless of the window geometry.
- Animation must be time–based (as opposed to frame–based).
- To improve the frame rate, clipping and culling must be used.

As with the first draft, the project will not be graded on artistic merit, although I will give a bonus point or two if it looks nice. You will be graded primarily on the correctness and efficiency of your code.

Your submission for this project should consist of two files named `DrawFrame.h` and `DrawFrame.cpp`. If you make your own triangular meshes, they should be declared and implemented in thes files. You may include the header files

```
Affine.h  Mesh.h             CubeMesh.h   Camera.h  Projection.h
Raster.h  RasterUtilities.h  HalfSpace.h  Clip.h    SnubDodecMesh.h
```

as well as any *standard* C++ header file. You may **not** use OpenGL, GLEW, or SLD2 in this assignment.

I will compile and link two versions of your code using my implementations of the above packages. One version will be compiled with the `CLIP_PIXELS` flag (which forces per pixel clipping to the frame buffer), and one without. Ideally, both versions should run without crashing.