

## 1 Introduction

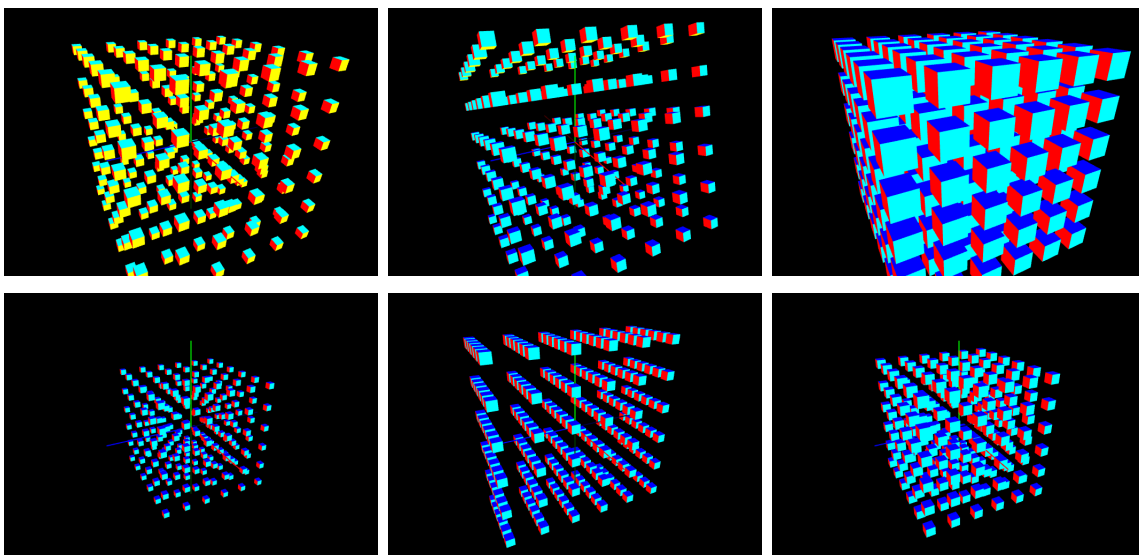
Write each program using either Windows or Linux using Code::Blocks or using the Mac and XCode, zip up all your work and upload the projects to the Assignment #6 page of the MyClasses site for this class.

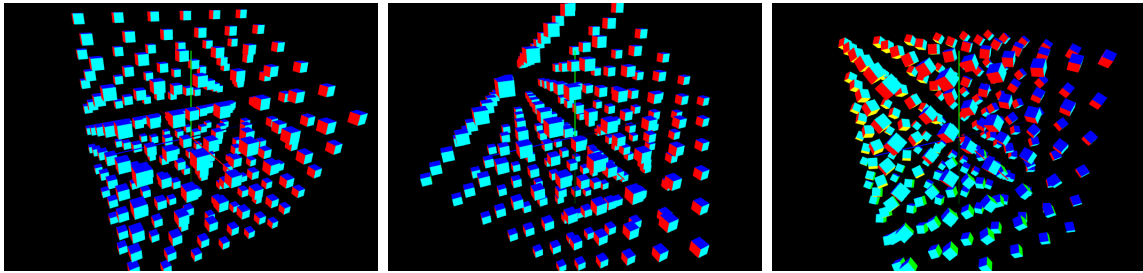
In these exercises you need to update the DoxyGen documentation. Fully document all new data members and methods to any of the classes, and update the documentation for anything that has changed, including the documentation in the main that gives an overview of the program.

## 2 Exercise #1

The exercise is an update of the Cameras program example. You are to add 9 animations to the grid of boxes. The animation number is to be chosen by the user pressing the numeric keys 1–9. The animation is to be toggled on and off by pressing the S key. The animations are scales, rotations, and translations of the box grid. Run each of the animations in the example program, watch the movement and sizes of the boxes closely. From the movement, determine the transformations, and their order. Also add in the user interface to turn the axes on and off by using F5 and F6. You may lock the animations to the framerate although that is not the best design and set the vertical synchronization on at 60 FPS.

The only code in the project that will need to be updated are to the UI.cpp, GraphicsEngine.cpp and GraphicsEngine.h files. The cube data should not be changed nor reuploaded to the graphics card. In addition, there should be only one cube uploaded to the graphics card. All animations can be done just with updating the transformations that are applied to the cubes. Keep in mind that the red axis is the  $x$  axis, the green axis is the  $y$  axis, and the blue axis is the  $z$  axis. The bright end is the positive end and the dark end is the negative end.





### 3 Exercise #2

This exercise was inspired by the film *Polar Express*. One of the scenes is where the main characters are on the front of the train and they are heading toward a very steep down grade. The scene is animated with the camera at the front of the train close to the tracks but slightly above them, so the viewer sees a roller coaster like animation of tracks moving underneath them.

The exercise is an update of the Cameras program example.

1. Remove the current objects that were being rendered. Keep in the axes for a frame of reference.
2. Keep the scale of the objects at  $[10, 10] \times [10, 10] \times [10, 10]$ , as the original was.
3. Create a track class that will create the track. The track is constructed with the curve

$$\begin{aligned} x &= r \cos(\theta) \\ y &= \sin(3\theta) - 2 \cos(2(\theta + 0.2)) + 2 \sin(7\theta) \\ z &= r \sin(\theta) \end{aligned}$$

Where  $r$  is the radius of the circle it is on and  $0 \leq \theta \leq 2\pi$ . The rails of the track are really two of these curves, one with a radius of 10 and the other with a radius of 9.6. The railroad ties connect the two rails perpendicularly. I personally used 500  $(x, y, z)$  data points for each curve, that is, 500 values of  $\theta$  between 0 and  $2\pi$ . Also, each railroad tie is between the data points on the two curves, so there are 500 ties.

4. Change the  $y$  field of view of the perspective function to  $75^\circ$ .
5. Add in the user interface to turn the axes on and off by using F5 and F6.
6. When the user hits F11 the camera will be the spherical camera and when the user hits F12 the camera will be the “roller coaster camera” that you will create.
7. Create the “roller coaster camera” by using the yaw-pitch-roll camera. You will not need to alter the yaw-pitch-roll camera in any way, you will simply use the accessor functions of the one we created. The camera is to follow the track directly between the rails (so at a radius of 9.8). It should be placed a little above the track or the

rails will come at you instead of under you. I used a height of 0.1 above the track. Lock the animations to the framerate although that is not the best design and set the framerate to 60 frames per second and the position of the camera should increase by 0.003 radians per frame. This will make the camera do one cycle of the track every 34.9 seconds. The viewing vector should be along the tangent vector to the camera motion. This is derived by taking the derivative of the position function, in  $(x, y, z)$ , with respect to  $\theta$ . This is,

$$\begin{aligned}x &= -r \sin(\theta) \\y &= 3 \cos(3\theta) + 4 \sin(2(\theta + 0.2)) + 14 \cos(7\theta) \\z &= r \cos(\theta)\end{aligned}$$

