

1 Introduction

Put each project in a separate folder, zip all the projects into one zip file and upload to the assignment 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. Of course, include your name as an author on any files that you edited.

This homework is to program screen saver like animations linked to the SFML clock so that the animation looks the same on any framerate. In addition, each program should be using the aspect ratio scaling matrix so that the circles look like circles and the stars look like stars.

2 Exercise #1: Bubbles

This program is to produce between 10 and 60 random circles (bubbles) that bounce around the screen. When a bubble hits the edge of the screen it should bounce off in a natural manner and change its color. The initial color and new colors on the bounce are to be chosen at random. The initial starting point of each bubble and its direction and speed are also to be chosen at random.



I am going to leave the specifics on the implementation fairly open but here are a few things I will be looking for.

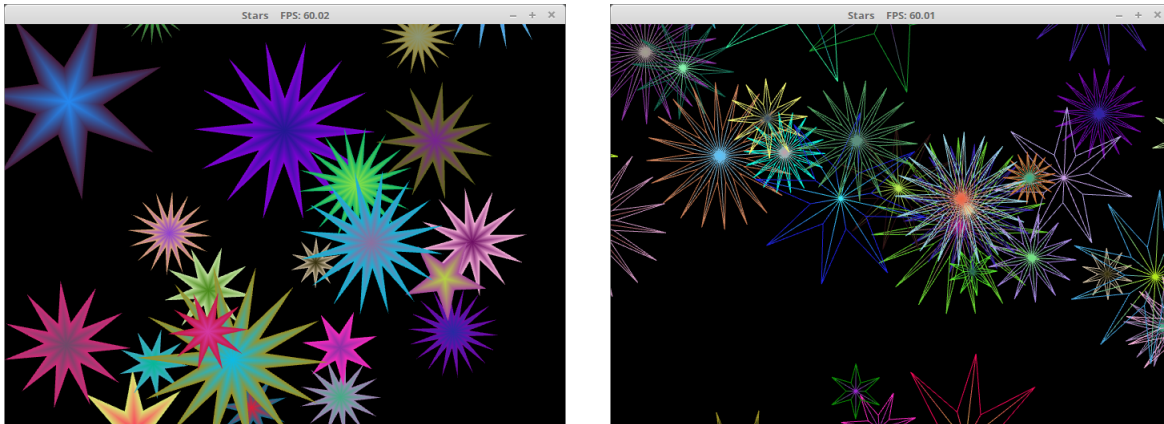
1. The data for the circle can only be loaded to the graphics card one time, at construction. The circle data is also to construct a circle of radius 1 centered at the origin.
2. All sizing and movement of the circle are to be done with transformations loaded to the shaders.
3. The color of the circle is constant and can be taken care of in the fragment shader.
4. The animation is to be linked to the SFML clock so that if the framerate changes the animation will still run at the same speed.

The user interface is to contain the following options.

1. Escape: Ends the program.
2. S: Toggles the vertical synchronization, and framerate, on and off so that you can test the program at 60 frames per second or the maximum speed of the card.
3. F10: Saves a screen shot of the graphics window to a png file.

3 Exercise #2: Stars

The Stars screen saver program that places a random star on the screen every tenth of a second. Each star has a random starting position on the screen, random number of points from 5 to 20, random center color, and random point color. The stars are also given a random velocity and rotational velocity. When the star is created it will animate across the screen and eventually off the screen.



For this program we are again going to utilize the vertex shader transformations to do our animation. Unlike the bubbles program we are going to be more restrictive. First create a Star class that has the following declaration. These are the only functions and data to be included in this class. If one were making a general star class then they would add in much more functionality but that is not the point of this exercise.

```

1 class Star
2 {
3 private:
4     int points;           ///< The number of points on the star.
5     GLuint VAO;           ///< Vertex Array Object ID.
6     GLuint ArrayBuffer;  ///< Vertex and Color Information Array Buffer ID.
7     GLint vPosition;     ///< Array Buffer position index.
8     GLint vColor;        ///< Array Buffer color index.
9
10 public:
11     Star(int numpoints = 5, GLfloat outerRadius = 1, GLfloat innerRadius = 0.35,
12          GLfloat ptred = 1, GLfloat ptgreen = 0, GLfloat ptblue = 0,
13          GLfloat ctred = 1, GLfloat ctgreen = 1, GLfloat ctblue = 1);
14     ~Star();
15
16     void draw();
17 };

```

The implementation of the class should create a star with the desired number of points, inner and outer radius, point and center colors. One of the points of the star is to be pointing straight up. As with the bubbles program, each star will be loaded into graphics memory only one time, transformations will take care of the rest.

Every tenth of a second the program is to generate another star on the screen. The star is to be given a random number of points between 5 and 20, a random scale factor between 0.1 and 0.5, random center, velocity, and rotational velocity that will produce a similar effect to the demo program.

You may limit the number of stars on the screen if you would like or use a storage structure that does not limit the number. If you choose to limit the number of stars the program can handle make sure the limit is fairly large, for example 200 to 250 or so. In either case you will want to remove stars that are off the screen.

When the graphics engine creates a new star, the outer and inner radius must be 1 and 0.35 respectively. The star will have a scaling factor to change its size. This way all of the stars will have the same inner to outer ratio and hence look like each other.

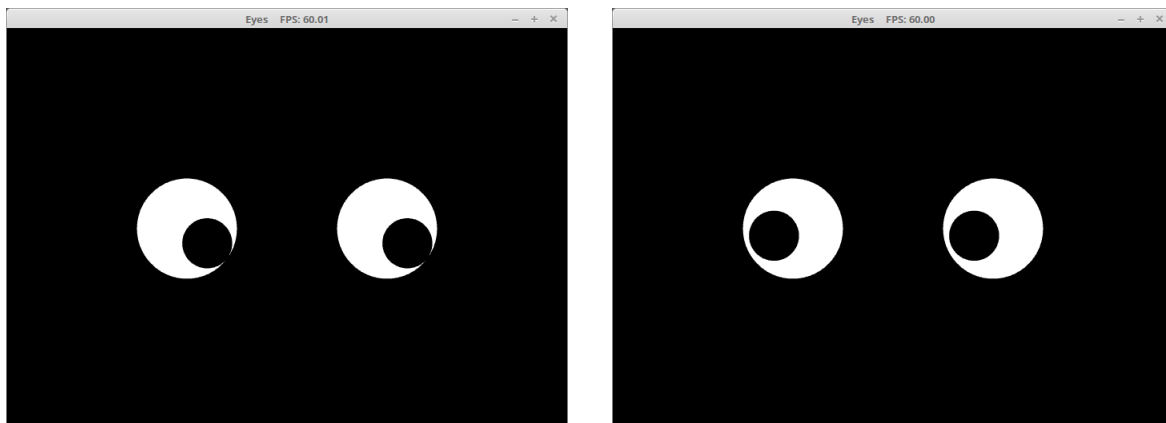
The user interface is to contain the following options.

1. Escape: Ends the program.
2. M: Toggles between filled and wire-frame modes.
3. S: Toggles the vertical synchronization on and off so that you can test the program at 60 frames per second or the maximum speed of the card.
4. F10: Saves a screen shot of the graphics window to a png file.

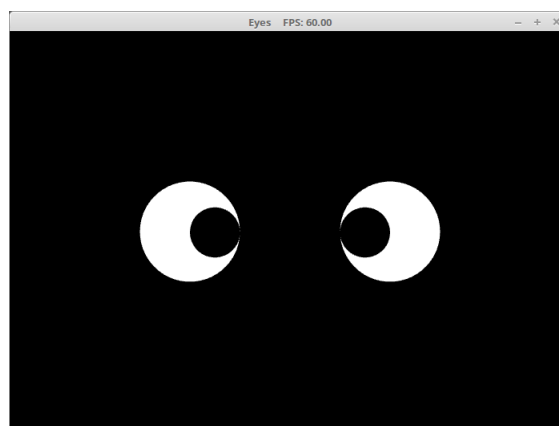
4 Exercise #3: Eyes

There used to be a Windows program that was called eyes. It was simply an icon that was on your desktop that had two eyes that followed your mouse pointer as you moved it around the screen. I never installed it because I thought it looked a bit creepy. This program is an emulation of this old application.

In together mode,



In independent mode,



The user interface is to contain the following options.

1. Escape: Ends the program.
2. M: Toggles between filled and wire-frame modes.
3. S: Toggles the vertical synchronization on and off so that you can test the program at 60 frames per second or the maximum speed of the card.
4. I: Toggles the eye mode from together to independent. When the eyes are together they move like real eyes would to follow the cursor. When the eyes are independent they will each look at the cursor from their own position.
5. F10: Saves a screen shot of the graphics window to a png file.