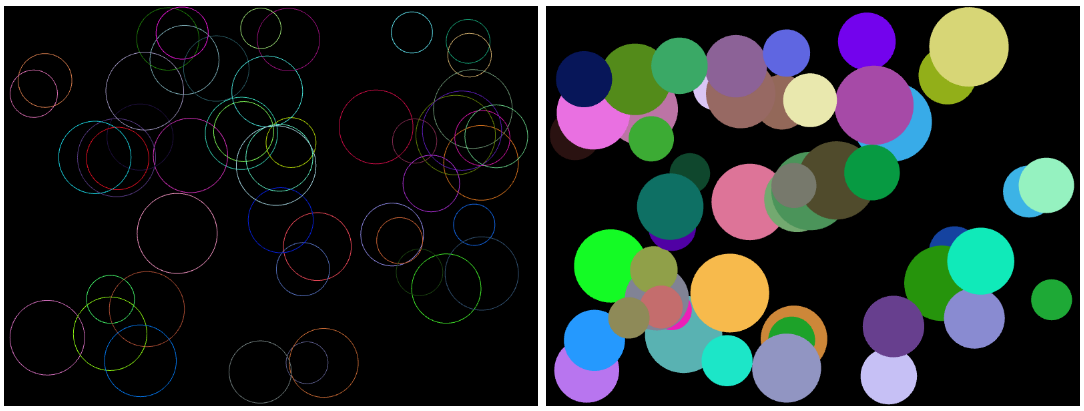
## 1 Introduction

Put each project in its own directory and zip the entire set of directories into one zip file. Upload the zip file to the Homework #3 page of the MyClasses site for this class.

This homework is to program screen saver like animations linked to a 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 50 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. The point colors and the inside color 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.

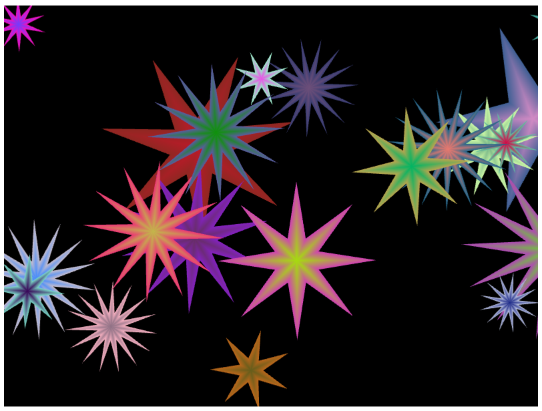
1. The data for the circle can only be loaded to the graphics card one time, at construction. The circle data is to construct a circle of random radius between 0.1 and 0.2 centered at the origin. Once the circle is constructed it is not to be reloaded to the graphics card. The color of the circle is constant but set at random when the circle is created.
2. The circle should be implemented as a class structure.
3. A common tool in animations and game construction is to have a single object for rendering and to control it with an outside class, such as a "player" class. Do the same thing here. Instead of storing the circle position and velocities in the circle class, create a separate class to do this. This controller class should store the position of the circle to be rendered as well as its velocity. It should also have an update function that will take the elapsed time and adjust the circle's position for the next frame. The graphics engine will ask this controller class for the position so it can set and load the matricies to the shaders.
4. All movement of the circle are to be done with transformations loaded to the shaders. No altering the circle data.
5. The animation is to be linked to the clock so that if the framerate changes the animation will still run at the same speed.

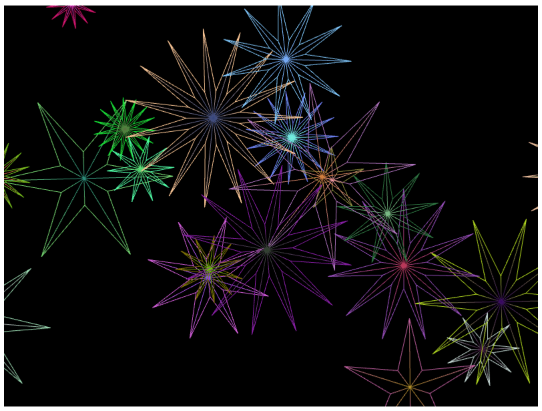
The user interface is to contain the following options.

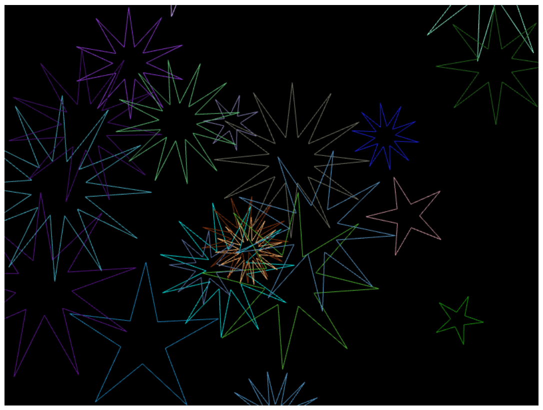
- Escape: Ends the program.
- 1: Sets the circles to render in outline mode.
- 2: Sets the circles to render in filled mode.
- F1: Polygon fill mode.
- F2: Polygon line mode.
- F3: Polygon point mode.
- F4: Toggle between 60 FPS and unlimited.
- F12: Saves a screen shot of the graphics window to a png 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 15, 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. Again the implementation fairly open but here are a few things I will be looking for.

1. Create a star class whose primary function is to render the star. The class should allow for stars of any number of points and any radius (length to the point from the center). The center of the star is always at the origin. The class should have functions to change the point color (color at all the tips of the points) and the center color. The class should load only one set of vertex data but have two EBOs, one for a filled in star and one for the outline of the star. The class should also have a boolean variable to select between the two and of course use this boolean value to determine which to use in the draw command of the star.
2. As with the bubbles program above, have a controller object for each star. This will be similar to the one for the bubbles but will also contain a rotation and rotational velocity. It should also have an update function that will take the elapsed time and adjust the star's position for the next frame. The graphics engine will ask this controller class for the position and rotation so it can set and load the matricies to the shaders.
3. Once a star is off the screen entirely the program should remove the star and remove the vertex data from the graphics card. Hence a destructor should be implemented in the star class.
4. Although the star class will allow a variable radius the graphics engine should create all the stars of radius 1 and store a scaling factor in the controller class that the graphics engine will extract and set up the appropriate scaling matrix.
5. All movement of the star are to be done with transformations loaded to the shaders. No altering the star data.
6. The initial position, velocity, and rotational velocity stored in the controller should be set at random when the new star and new controller are created.
7. The animation is to be linked to the clock so that if the framerate changes the animation will still run at the same speed.

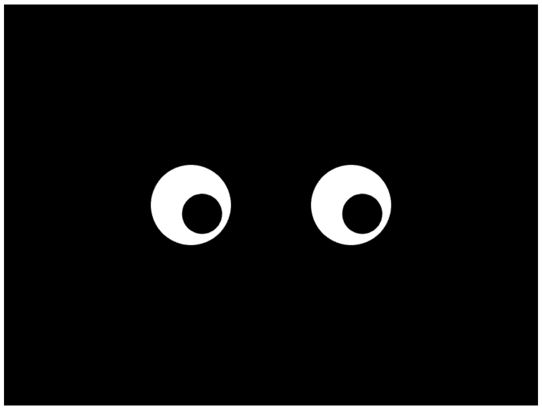
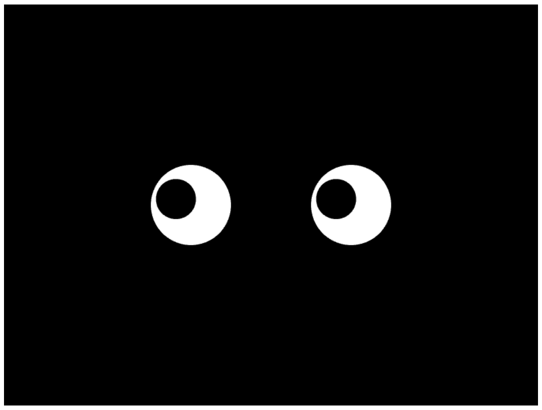
The user interface is to contain the following options.

- Escape: Ends the program.
- 1: Sets the stars to render in filled mode. So for every new star created the mode will be fill.
- 2: Sets the stars to render in outline mode. So for every new star created the mode will be outline.
- F1: Polygon fill mode.
- F2: Polygon line mode.
- F3: Polygon point mode.
- F4: Toggle between 60 FPS and unlimited.
- F12: 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.

There is not much to say about the workings of the program. As it implies, the eyes will follow the mouse pointer around the window.

The user interface is to contain the following options.

- Escape: Ends the program.
- F1: Polygon fill mode.
- F2: Polygon line mode.
- F3: Polygon point mode.
- F4: Toggle between 60 FPS and unlimited.
- F12: Saves a screen shot of the graphics window to a png file.