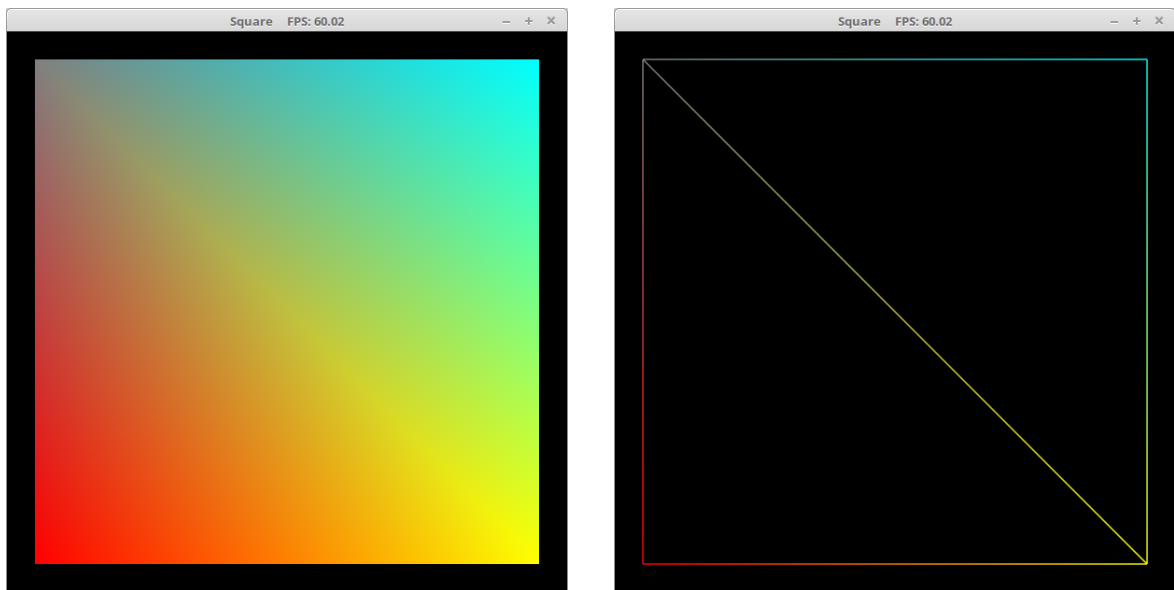


1 Introduction

The following exercises are updates of the Hello World of Graphics program. 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 #1 page of the MyClasses site for this class. Update the DoxyGen documentation to correspond with the user interface and function changes.

2 Exercises

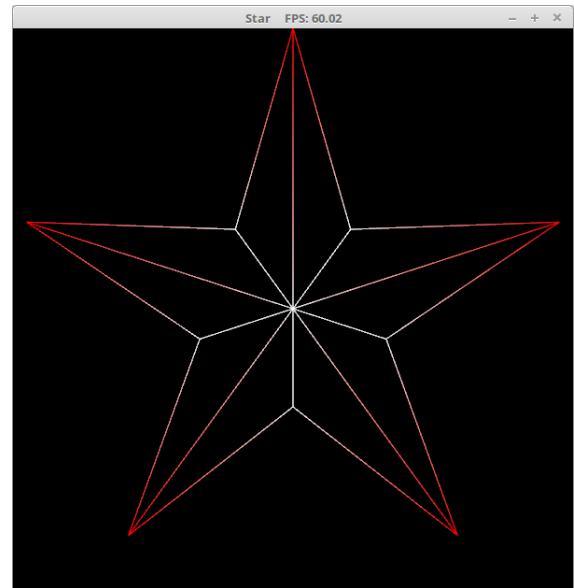
1. For the first program make the initial size of the window 600×600 and make the reset option resize to 600×600 as well. Make the title bar display “Square” and incorporate the FPS tracking in the title bar, you do not need to print the FPS to the console window. Have the image on the screen be a square like the image below. Keep the rest of the features the same as with the triangles program. Hint: A square can be produced using two triangles. Below are two screen shots, one in fill mode and one in line mode.



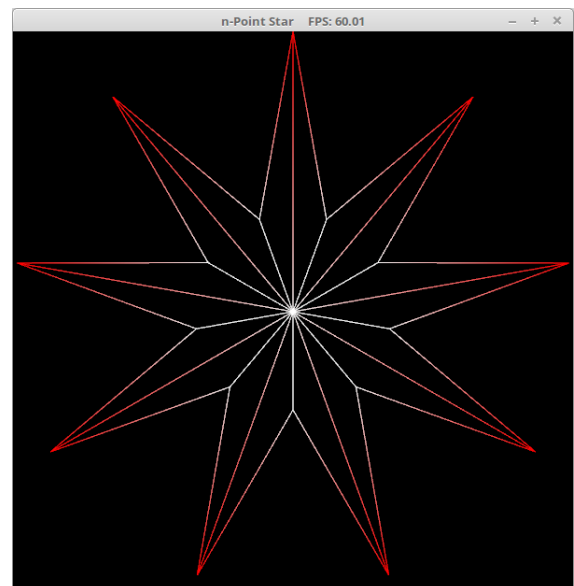
2. Create a program to produce a 5-point star with the coloring of the star being like the image below, that is, red at the points and white in the center. The title bar should also display the title “Star” with FPS in the title bar. As with the previous exercise, make the initial size of the window 600×600 and make the reset option resize to 600×600 as well. Keep the rest of the features the same as with the triangles program. Below are two screen shots, one in fill mode and one in line mode. Note that one point of the star is always pointed straight up.

Hints: Each point of the star can be done with two triangles. Also recall from trigonometry that a point on the unit circle can be represented by $(\cos(\theta), \sin(\theta))$.

So if we wanted to space n points around the unit circle evenly we could use the points $(\cos(i \cdot 2\pi/n), \sin(i \cdot 2\pi/n))$ where $0 \leq i < n$. We can also scale the circle by multiplying by a constant. So n evenly spaced points around a circle of radius r would be $(r \cdot \cos(i \cdot 2\pi/n), r \cdot \sin(i \cdot 2\pi/n))$ where $0 \leq i < n$.



- Update the above star program to produce an n -point star with the coloring of the star being like the image below where n is chosen randomly between 5 and 15. So on each run the number of points to the star could be different. Hint: Use the `rand` and `srand` functions from the `ctime` library and set the seed of the random number generator to the value of the system clock. Then generalize what you did for the 5-point star to the n -point star. Note that one point of the star is always pointed straight up.

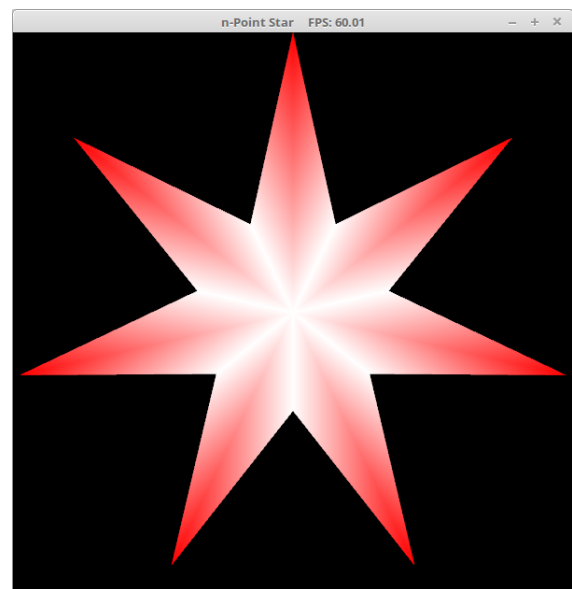
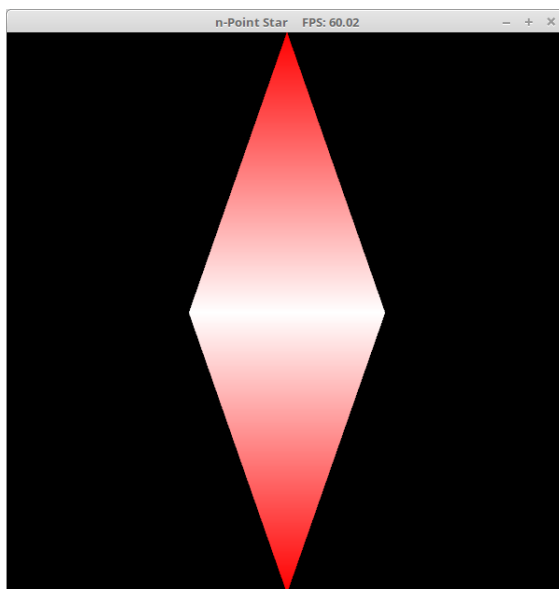


4. Update the above n -star program to add in some user interface from the keyboard. Have the program start by displaying a 5-point star, when the user hits 2 replace the star by a 2-point star, when the user hits 3 replace the star by a 3-point star, when the user hits 4 replace the star by a 4-point star, when the user hits 5 replace the star by a 5-point star, and so on up to 9 creates a 9-point star and 0 produces a 10 point star. I would suggest creating a new function for the graphics engine called `updatePoints(GLint numpts)` which takes in the number of points to be used, creates the star, and loads it to the graphics card.

A note on memory management on the graphics card. If you keep generating new vertex array objects and buffers inside them, it is like doing the `new` (or `malloc`) command over and over again. That is, you will have a memory leak on the graphics card. It is unlikely that you will run out of memory with this program but we would like to be as efficient as possible. The first time you call `updatePoints` you will need to generate your arrays and buffers. Then for every subsequent call to `updatePoints` you should delete your VAO and Buffer before you generate another one. This is like doing the `delete` (or `free`) command to release the memory locations for use. To do that in OpenGL use the commands,

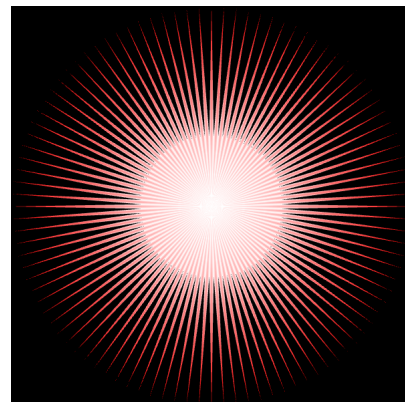
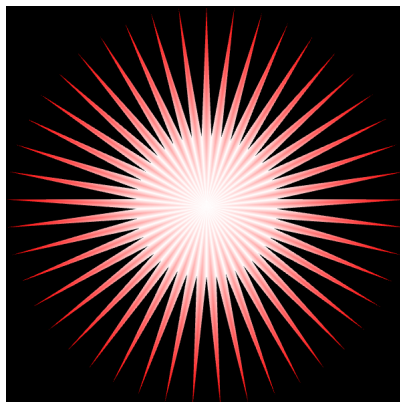
```
glDeleteBuffers(1, &Buffer);  
glDeleteVertexArrays(1, &VAO);
```

where VAO and Buffer are holding the handle to the object and buffer. Then when you generate a new VAO and Buffer they will use the same handle. Note, just like the other programs, one point of the star is pointed straight up.

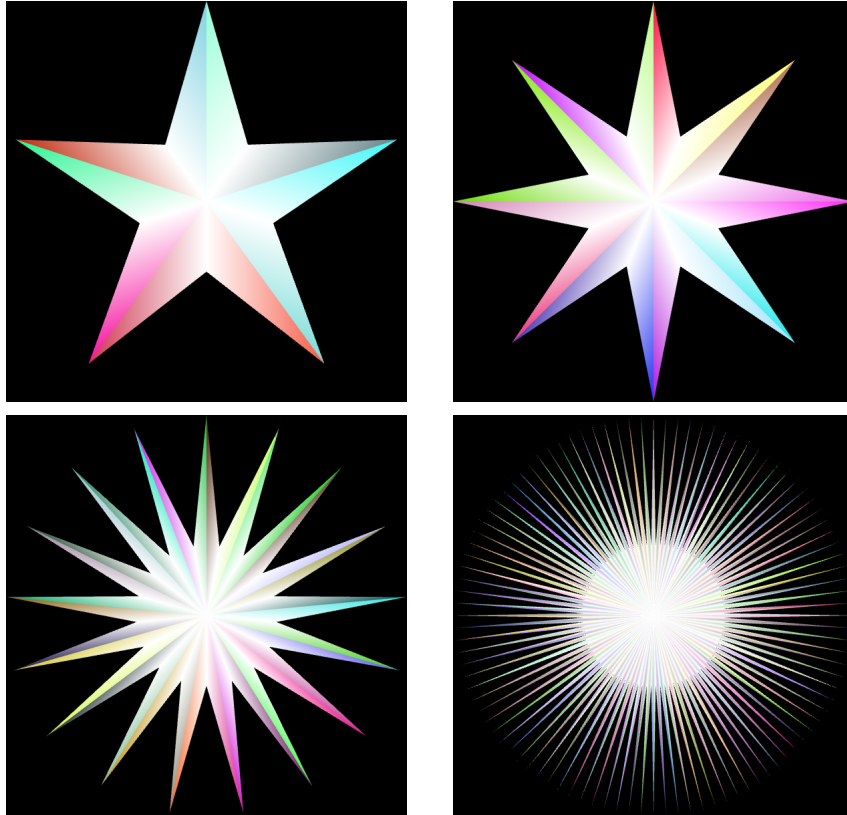




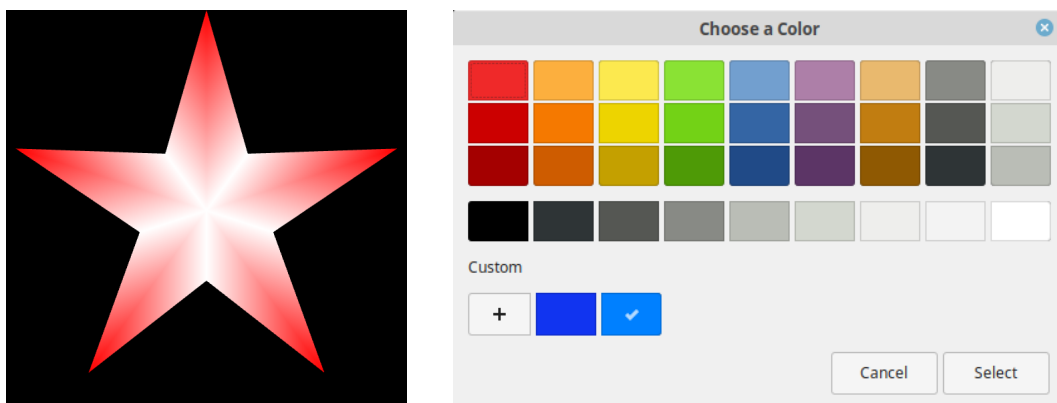
5. Update the above n -star program to add in one more user interface option. If the user presses the up arrow key the number of points will be incremented by 1 and if they press the down arrow key the number of points will be decremented by 1. The minimum number of points should be 2 and the maximum number of points should be 100.

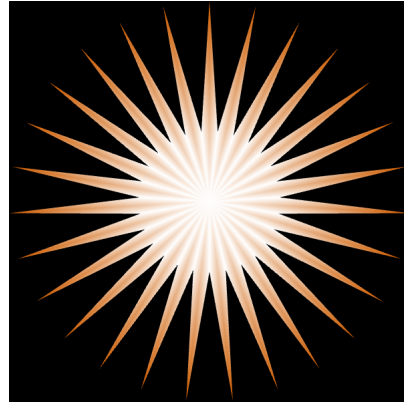


6. Update the above n -star program to randomize the colors of the points. The center should still be white. Specifically, any time that `updatePoints` is called load in random colors for the points.



7. (Optional Extra Credit) Update the n -star program from #5 to let the user select the color of the points. In the `Advanced_Topics` folder there is a folder `TinyFileDialogs_Code_and_Example` that contains the source code for cross platform dialog boxes, one of which is a color selector. Read through and run the example and then incorporate a color selection dialog into your program if the user selects C from the keyboard. The user selected color should then be used until the user changes the color again. If the user selects Cancel the old color should be unchanged. The program should start out with red as the point color.





8. (Optional Extra Credit) Update the n -star program from above to alter the size of the star points. Have the right arrow make the points longer and the left arrow key make them shorter. The maximum length should be 1 so that the star stays inside the window. The minimum value should be the value you use for the inner radius of vertices. The user color selection and the up and down arrows for increasing and decreasing the number of points should also be implemented.

