Project 8: Simple Demo Scene

Kyler Harden, Angel Velzaquez

CST-310 Computer Graphics

November 12, 2024

Professor Citro

Background:

Project 8 is an open-ended project so we thought it would be cool to implement a light trail effect inside of open gl. A light trail effect is where you have a light that is moving around on screen with a trail following its path that slowly fades. We would like there to be multiple light trails moving around the screen circling around all three axes to give a 3D look. To achieve the circler movements trigonometry is used to update the coordinate positions as the lights move away and back towards the origin.

Mathematical Concepts:

We utilize trigonometry to create nonlinear movement for each light trail. This is achieved using sin and cos along with radius, trails speed, time, and trail phase. For each trail we start by calculating t, this gives us the value that is used in the sin and cos statements. Time is added in here to allow the entire image too to look like it is rotating as the light trails dance:

float t = time \* trail.speed + trail.phase;

Next we calculate the x, y, and z of the new point using the radius times the gemometric translations.

newPoint.x = trail.radius \* sin(t \* 1.1f) \* cos(t\*0.7f) ;

newPoint.y = trail.radius \* sin(t\*0.8f) ;

newPoint.z = trail.radius \* cos(t\*1.3f) \* sin(t\*0.9f) ;

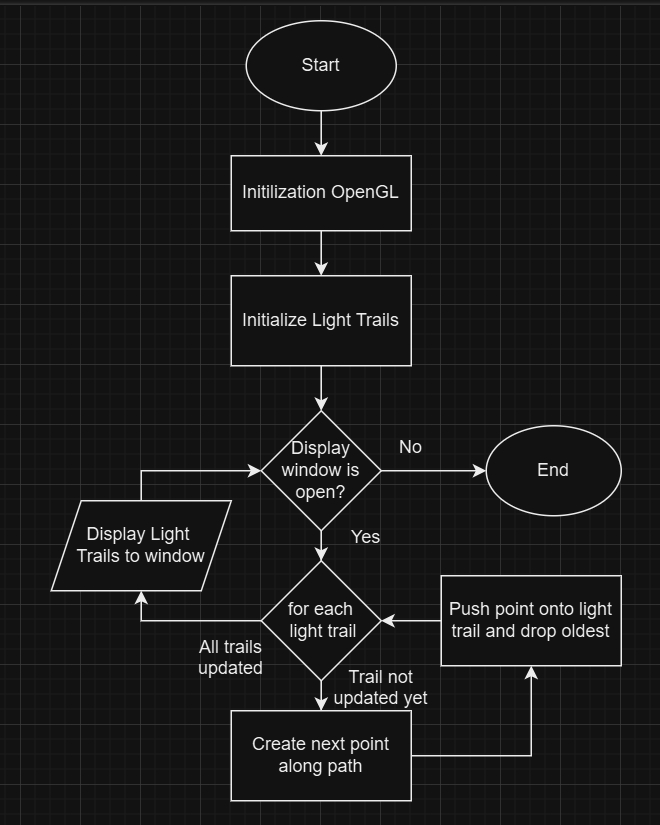
This new point is added to the front of the trail as it is the latest point, and all other points shift one to the back with the last one being dropped off.

Implementation:

To create light trails we need to start with creating points moving around the screen utilizing the geometric concepts laid out in the Mathematical Concepts section. Once we had the points moving in a non-linear looping fashion then we needed to get the trails added to them. To achieve this, we created a vector of points so that way we can keep the last 50 points of each trail with the goal of creating a smooth looking line. At first implementation of the trails it didn’t go well as we had large gaps between each point making it look more like a row of dots instead of one fluid line. To rectify this, we spent time adjusting the speed, radius, and geometric updates of the trail, and the rate the update function is called to close the gaps to make it finally look like one fluid line. Once we had one line working, we worked on adding more to make the graphic more interesting to watch. We wanted to be able to dynamically specify the number of lines, so we introduced random number generation to the initialization of the trails for their speed, phase, radius, and color.

Flowchart:

The following is a flowchart that shows the process of the application. The application starts by initializing OpenGL and the light trails before it enters a loop that runs while the display window is open. Inside this loop the application loops through each light trail updating the position of the head and then it will display the updates before checking it the window is still open and repeat the process.



Screenshots:

The following screenshot is an example of running the light Trail program utilizing 15 light trails.

A screenshot of a computer

Description automatically generated

The following is a screenshot of the light trail program with just 2 trails highlighting the ability for the program to handle fewer light trails being displayed.

A screenshot of a computer

Description automatically generated

The following screenshot is an example of running the light trail program with 200 light trails highlighting it’s ability to scale upwards.

A screenshot of a computer

Description automatically generated

Youtube Link: <https://www.youtube.com/watch?v=k5ylLeNjdGk>