Infinite Neon Grid Road

William Hem
William.Hem001@umb.edu
University of Massachusetts Boston

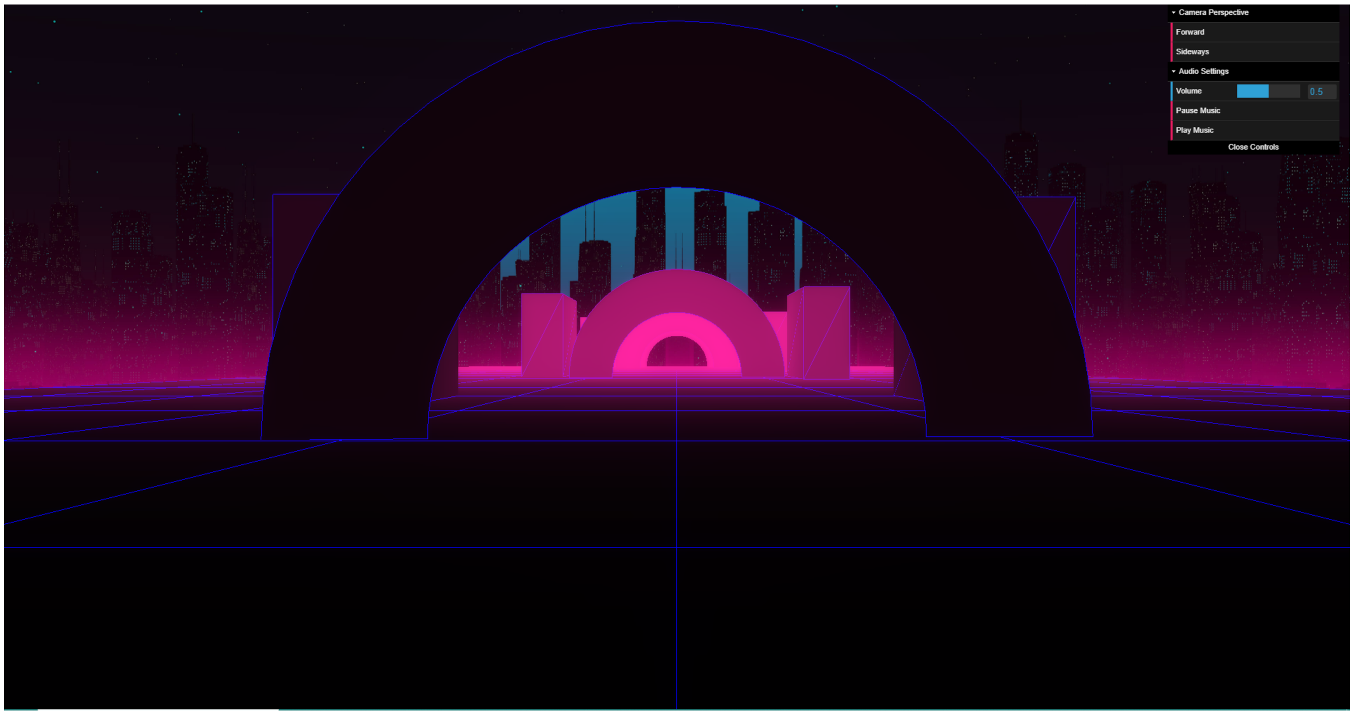

Figure 1: Neon Grid Road Scene run on local server

ABSTRACT

Neon colors have been employed in a variety of media over the years to create a futuristic-like aesthetic. The purpose of this project is to create a futuristic neon scene using the WebGL framework Three.js as well as HTML. More specifically, this project aims to create A Neon Infinite Grid road where from the camera perspective, moving forward across a grid-road is seemingly infinite. This paper will go over the implementation of the neon infinite grid road scene. More specifically, this paper will discuss the illusionary techniques used to achieve an effect of infinite and elements of the scene that help bring the scene to life.

KEYWORDS

Three.js, Wallpaper Engine, 3D Visualization, Neon, Futuristic, 3D Wallpaper, Camera Illusions

ACM Reference Format:
William Hem. 2020. Infinite Neon Grid Road. In *CS460: Computer Graphics at UMass Boston, Fall 2020.* Boston, MA, USA, 3 pages. https://CS460.org

1 INTRODUCTION

This project is a demonstration of being able to use Three.js for artistic purposes. The tron-like objects, neon colors, and effect of a never-ending scene is an aesthetic most shared by cyberpunk enthusiasts. Thus an infinite grid road can make for a great desktop wallpaper. Since HTML files can be used for a wallpaper app like Wallpaper Engine, this neon scene could actually be used as a 3D wallpaper. At the end of this paper will be a link to this neon scene that you can subscribe to if you have the app Wallpaper Engine.

2 RELATED WORK

Framework used: Three.js [1]
Wallpaper Software: Wallpaper Engine [2]

Disclaimer: This project was created with fun intentions not for monetary gain.

3 METHOD

When one runs this Three.js scene, the scene is already moving forward. They could be drawn to the geometries in the scene with the black surfaces and blue outlines which emulates neon in a dark room. This of course is a commonly used aesthetic associated with the futuristic movie Tron. There's also background music which employs synths and electronic riffs. This helps to add that sense of futurism in the scene. Finally, one might also be drawn to the menu at the top right-hand corner which allows you to change the camera perspective from forward to sideways and manipulate the audio. This menu was created using Dat.GUI with the purpose of making this scene more viable for a wallpaper use.

The most important aspect of this scene is the fact that the grid road is seemingly infinite. This employs two illusionary techniques. The first technique is how the scene moves forward. This is done by doing translation in the Z axis. After the grid or structures reach a certain distance, they are moved to the front of the camera. The camera in the scene always stays stationary throughout unless one changes the perspective from the menu. The grid's were created with GridHelper() in three.js with two grid structures used to help maintain the illusion of infinite. There's a black plane beneath these grids which is separated by 1 in the y direction. This plane helps to provide a "floor" for these grids which by default have empty holes. The grids are then translated towards the camera (or sideways depending on perspective) which helps achieve the effect of moving forward.

The second illusionary technique is making the grids and structures appear in front of the road more naturally. This is done in two ways. The first is with Fogexp2() in Three.js which blends with the fog color in the image background. The second is by having 1 of each structures (left buildings, arches and right buildings; not grids) be at the position where we translate the other buildings in front of the road. This helps makes it so that the structures don't "pop of-nowhere". Combing these techniques helps to make the buildings appear more gradually rather than instantaneously.

The result of combing all this is a basic neon infinite grid road.

3.1 Implementation

The first implementation to go over is the camera perspective implementation. This is done by changing the camera position and field of view (fov). The field of view is changed to avoid the structures and grids in the scene that would seemingly vanish (translating the objects to the front). The z position where the grids move to the front is also changed as it was noticed that different camera perspectives works for different z distances since one could notice instantaneous vanishing if this was not changed. The camera projection matrix is then updated to reflect the change in fov. Below is a snippet of what this looks like for the sideways camera (this is all put into a switch statement).

```
 case('Sideways'):
           camera.position.set(1344, 0, 2);
           camera.fov = 45;
           grid_scale = 2400;
             break;
         }
         camera.updateProjectionMatrix();
```

Another implementation to consider is how the meshes were created. Three.js doesn't offer an immediate way to create neon structures. Instead to create a neon effect, each structure has two meshes that are merged with one another. The first mesh is the black geometry and the second mesh is either a wireframe geometry or an edge geometry of the first mesh. The second mesh emulates a neon color (in this project blue). Three.js offers both types of "skeletal" geometries. Once this is created, the second mesh becomes a child of the first mesh by using the add method in a mesh. The result of this is a geometry that looks retro. Shown below is a sample code for the arches in the scene.

```
// Create Geometry
      var arch_geometry = new THREE.RingBufferGeometry(
                            500, 300, 30, 30, 0, 3.15 );
       var arch_material = new THREE.MeshBasicMaterial(
          { color: 0x000000, side: THREE.DoubleSide } )

        var arch_mesh = new THREE.Mesh(
            arch_geometry, arch_material );

        // Create Frame
     var edge_geometry = new THREE.EdgesGeometry( arch_geometry );
     var edge_mesh = new THREE.LineSegments( edge_geometry,
            new THREE.LineBasicMaterial(
            { color: 0x0000ff, linewidth: 2} ) );

        // Combine Meshes of Edges and Geometry
         arch_mesh.add(edge_mesh);
```

The way the translations are done and checked for each structure is implemented in the animate() function. The final z position is chosen based on the initial position of the last structure of that type plus the displacement between the same structure. Below is a sample for the translation of arches where the z position to check is 4500 [2400 (last arch z position) + 2100 (displacement)]:

```
    center_arches.forEach(element => {
          element.translateZ(15);
          if (element.position.z > 4500) {
            element.position.z = -6000;
          }
        });
```

By repeating this for each structure type, one gets the general format of how translation is done. Three.js Audio was used to insert the track while Dat.GUI displayed features such as audio control and camera control.

3.2 Milestones

3.2.1 Milestone 1. Piecing together everything and how it would be implemented. In particular discovering the existence of gridHelper and EdgeGeometry/WireframeHelper to create neon meshes. Also looking online to find the right music and background for the scene.

3.2.2 Milestone 2. Implementing a sufficient way to create the geometries, figuring out how much to space each of them apart, and calculating when to move the geometries to the front.

3.2.3 Milestone 3. Implementing the Z translation for all geometries besides the black plane since this is the foundation of how the scene looks like it's moving forward

3.2.4 Milestone 4. Implementing Dat.GUI with audio and camera control. Volume control was a little complicated because dat.GUI works with objects and not single variables, but this was remedied by using a 1 element dictionary which now allows sliders to control the volume.

.

3.3 Challenges

Describe the challenges you faced.

- Challenge 1: Implementing Bloom Effect. Overall the bloom effect slightly lights up the scene, but it's not to my expectation. In trying to reference the unreal bloom selective in three.js post-processing example page, I found out the bloom effect greatly affects the fog and changed the background to all black. This is an apparent limitation for the bloom effect so it could not be remedied so easily. In the end the bloom effect only plays a partial role in the scene.

- Challenge 2: Grid Helper limitations. The original purpose of the Dat.gui was to allow for color customization of the grid, but further research into gridHelper found that the grid color couldn't be changed if at all. This discovery made implementing Dat.GUI useless until I figured out other uses for it like camera perspective and audio control.

- Challenge 3: Skybox and camera movement. Originally I was going to use a skybox and orbital camera to give the scene more of a 3D touch, but it became apparent that the background image, though it could be cropped, came out terribly when trying to make a skybox with it along with 5 other images. In addition orbital camera controls could allow the user to see pass the illusion of the scene so there was no reason to use it. Plus it might affect how the scene is used as a wallpaper. In the end I killed two bird with one stone by implementing camera control in dat.GUI.

4 RESULTS

The end product of this project is a neon road that's seemingly infinite and has some structures along the way. Now that the scene is implemented, we can use a third party app like Wallpaper Engine to turn the scene into a 3D Wallpaper. This final product of this scene can be seen in Figure 2 being used as a wallpaper. The link to this wallpaper is below:

https://steamcommunity.com/sharedfiles/filedetails/?id=2328318679

Live render of scene hosted on website: https://glisync-00.github.io/cs460student/

It should be noted that the scene with implemented to account for window size so this scene should be able to run and look good in all devices compatible with three.js.

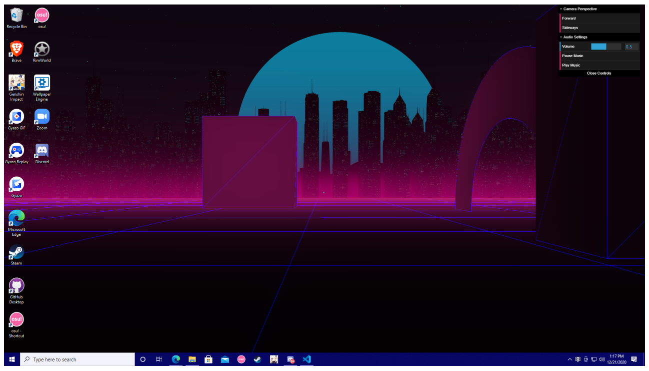

Figure 2: Neon Scene in sideways perspective used as a wallpaper

5 CONCLUSIONS

In short this project was overall fun to work on. With the use of Wallpaper Engine, this shows how much potential three.js has beyond being a library for web development. Three.js can be used for artistic purposes and to create scenes viable for wallpapers or animations. Despite the limitations of certain aspects of Three.js, the end product still turned out looking good and fitting the aesthetic of neon futurism.

REFERENCES

[1] Ricardo Cabello et al. 2010. Three.js. *URL: https://github. com/mrdoob/three.js* (2010).
[2] Tim Eulitz and Kristjan Skutta. 2016. Wallpaper Engine. *URL: https://store.steampowered.com/app/431960/Wallpaper$_E$ngine/*(2016).