S.C.O.P.E: A Small City of Procedural Elements

Ayushi Tandel (atandel@stanford.edu) & Samaksh Goyal (sagoyal@stanford.edu)

Overview

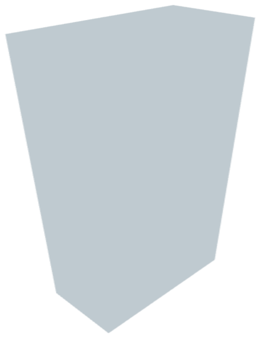
We present SCOPE: a small city of procedural elements. We procedurally generated a city using WebGL2 and constructed a predefined path for users to traverse it. (If a user takes a peek under the hood they can also edit the code so the camera follows the path of their choosing). Video Link - please download for best results. Link to GitHub Repo.

Tasks

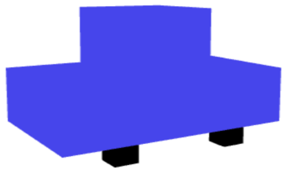
Procedurally Generate City

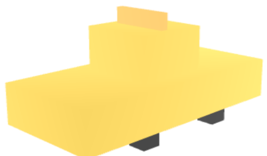
1. Create assets for the ground plane, buildings, lampposts, cars, taxis, and a camera. We created each of these assets out of triangles. We manually constructed a set of predefined base objects by enumerating positions, normals, and colors for each triangle vertex. All assets were centered around (0,0,0) to make further transforms (like scaling and translating) easier. These are the building blocks of our city.

(a) Ground


(b) Building

(c) Lamppost


(d) Car


(e) Taxi

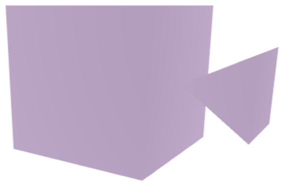

(f) Camera

Figure 1: Assets Created for S.C.O.P.E

2. We used transforms on the assets to create ”novel” objects and place them throughout the city. We used a floor plan, shown in Figure 2, to map out the placements of assets throughout the city. Figure 2 also shows an intermediary version of the city with the ground plane, buildings, and lampposts added.

3. Color all objects on the screen. Our assets were initialized with colors, as R,G,B values, for each vertex. We have one shader that colors the ground plane as a constant, fully opaque surface. The second shader, which is used on the other assets in the image, weights the intensity of the color based on the depth of the object, which we calculated as the distance between the object and the camera.

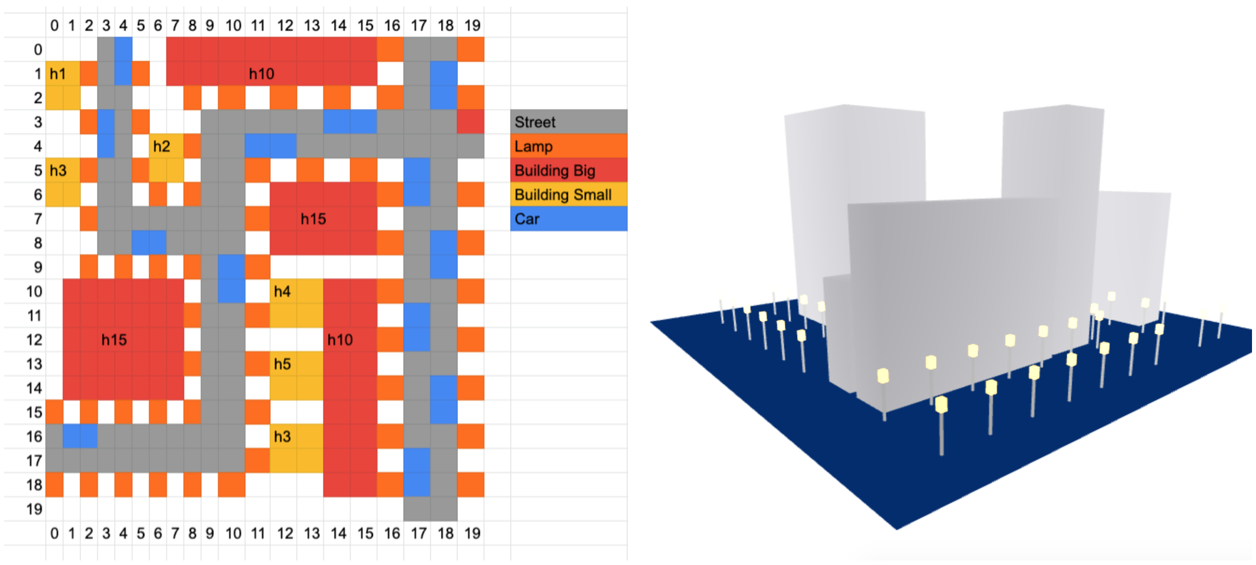


Figure 2: City Plan and Intermediate City

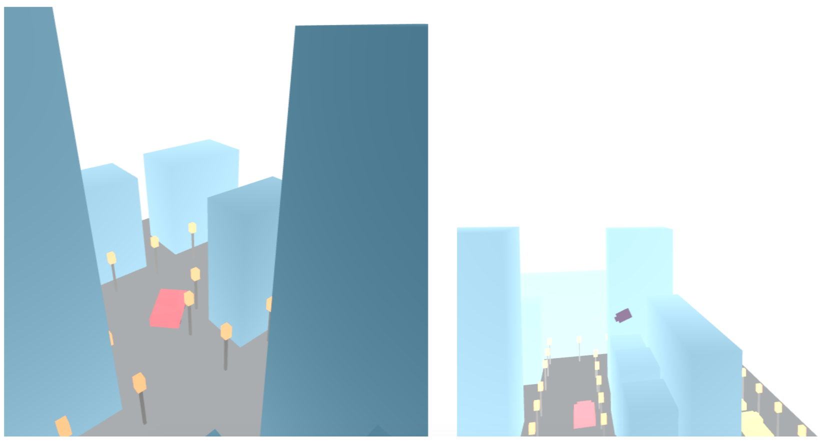


Figure 3: Split View: First Person View(left) & Camera View(right)

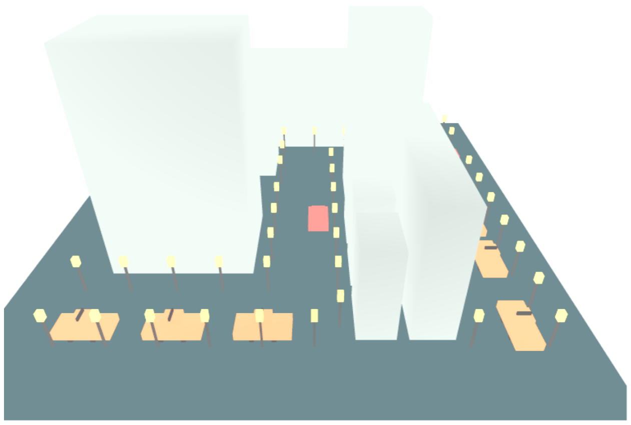
Thus, objects closer to the camera appear brighter and more intense colored than objects that are farther away.

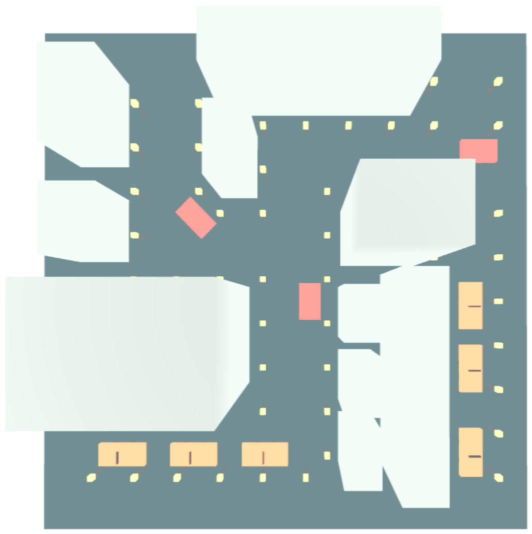
Camera Dolly

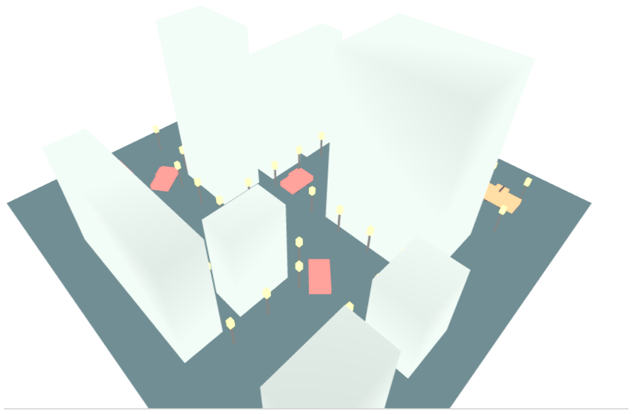
1. Allow the user to view different directions of the city using mouse controls. We implemented a split screen view of the city, as shown in Figure 3. On the left side, we see the first-person view and on the right side, we see the global view and how the camera is moving throughout the city. Every time the mouse is moved, the camera is shifted by a distance proportional to how much and what direction the mouse was moved. The split-screen is re-drawn whenever the mouse is moved as a result, giving the user an interactive experience of travelling through the city.

2. We are able to figure out which direction the user wants to turn by adding an event listener to the mouse. When a user move their mouse rightward we move the camera's target viewing direction rightwards. More concretely this direction computed in the camera's look at matrix. This works in all directions.

Final City


(a) Front View


(b) Top View


(c) Back Left View

(d) Front Right View

Acknowledgements

- We would like to thank Kayvon and Haotian for thier guidance. Also we would like to thank Gregg Tavares for his incrediable WebGL Tutorials and Examples can be found at `https://webgl2fundamentals.org/`.