Kungliga Tekniska Högskolan
School of Computer Science and Communication

DH2323

Citibike Bicycle Traffic Visualization

Project Documentation

Michael Hotan PN: 870522-T599 SN: 0579 5268
Alexandre St-Onge

July 5, 2014

1 Introduction

The prominence of utilizing real time sensor data to create powerful visualization has broadened uses of Computer Graphics. We were inspired by the availability of Citibike's bicycle sharing system data. bicycle transportation data provided by and. New York City, like many other populated cities, set up a bicycle sharing system. A benefit in having such a system is that it allows an intrinsic way to store traffic data. We looked to build a Unity application that depicts a bicycle scene in which traffic density reflects Citibike data.

2 Background

Citibike and New York City opened up their trip data set allowing computer scientists, data enthusiasts, and analysts to create powerful visualizations. [Ferzoco] shows how traffic trends over 48 hours using a two dimensional map. We attempted to build on concepts presented in [Dublin, 2011] except in a bicyclist context. We wanted to create a realistic bicycle traffic visualization that could represent the varying levels of traffic density found within Citibike's dataset. This included 5 essential tasks:

1. Create a background scene.
2. Simulate bike traffic.
3. Write custom shaders.
4. Create controls for augmenting displayed bike traffic.
5. Connect and augment the visualization in accordance to New York Citibike data.

I was involved with parts 3, 4, and 5. I will provide an overview for each task but only go into detail in the parts I implemented. We ended up using Blender to edit our models and Unity to create our visual scene, manage animations, and write our shaders and scripts.

3 Process

In order to facilitate our lack of experience in modelling three dimensional objects, we decided to implement a top down approach for this project. We used predefined Unity Assets for our scene and cyclist model. The cyclist model, created by [dotline11], provided a suitable base for us to customize. We augment the assets as we saw fit on order to focus more on the Graphics.

3.1 Rendering a scene in Unity

We wanted to implement a traffic simulation that was independent from where the traffic was placed. It was in our benefit that we used predefined unity assets to design our background for traffic. We imported a premade street kit [Jankowski], additional park benches [Image], and the

bicycle [dotline11]. These assets provided a base for our purposes. The work is gone over more material in my partner's, Alex St-Onge's, paper.

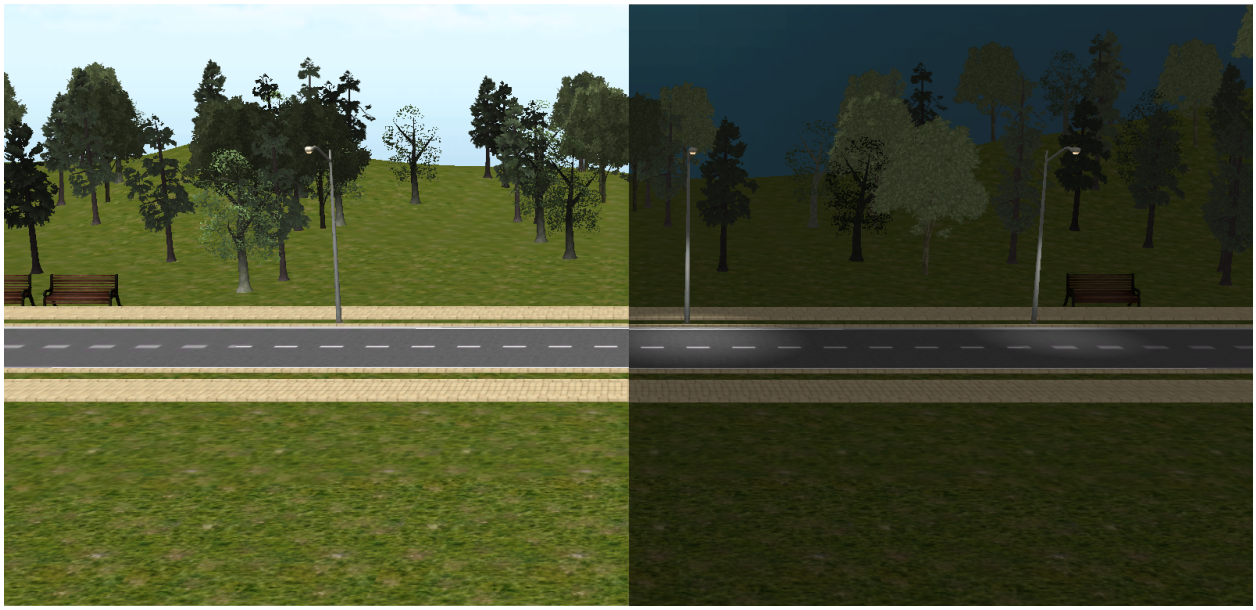

Figure 1: Park scene in Unity developed by Alex St-Onge

3.2 Rendering the cyclist

The base model for the cyclist was imported directly from the asset store. We wanted a base model that was flexible enough for us to change and adjust to display our purposes. My partner focused on changing the animation and handling collision avoidance using Unity. My part involved adding and managing the UV Mapping and texturization of the cyclist to enhance realism. This involved the combination of smart and manual Blender mapping as described in Wikibooks.

Generally UV mapping requires the 2D image to be customized for the 3D models UV mapping or visa versa. Our bike and cyclist model is relatively large compared to the texture sample and the patterns used for the bicycle and clothing components tend to use repeating patterns. Therefore we took advantage of this applied tileable textures to properly UV mapped components. We obtained the tileable textures from a free online web resource and followed the similar application process of UV mapping shown by blendtuts [blendtuts]. Figure 2 shows how tileable textures were used on multiple bicyclist components.

One of the key issues we came across was due to using someone else?s bicycle model. Some objects were not separated appropriately within the model. For example, the wheel and tire was one object. Ideally We restrained from altering the blender model too much because imperfect adjustments caused the UV mapping and texture application problems. Instead we used custom, manual UV mappings when necessary as seen in Figure 3.

The bicyclist model constructed of objects the had inconsistent extrusions and cuts leading to complicated UV maps. The models produced very complex UV maps. As a result we had to deal

Figure 2: Bike and Clothing tileable texture

with texture segregation as we applied our tilable texture. Attempting to follow some guidelines presented in [William, 1996], we adjust the size and contrast of the texture of each of our models components so that incongruent visual effects are less noticeable in the background scene. Figure 4 shows that keeping our texture details small relative to the scene?s camera distance to the cyclist allowed us to be able to utilize the same UV mapping and be able to substitute out textures.

3.3 Creating the Custom Shaders

We decided to write our own shaders in order to deal with different types of light sources for particular objects on our cyclist model. In Unity shaders are written in CG. In particular, blocks of code (passes) need to be written for every light source that affects the material. We found with the way our scene was structured there was at most two lights on a single object. Therefore, we wrote two passes for every shader that we created. This reduced the number of processing resources as more cyclist were introduced into the scene. Figure 5 shows the near optimal result of limiting our shader to just two passes.

It was essential that we wrote custom shaders for the major types of materials used by our cyclist model. In CG we had to define input structs that were used to store data for per vertex and fragment operations. Structs allowed us to hold temporary data values of specific types.

```
struct vertexInput{
        float4 vertex : POSITION;
        float3 normal : NORMAL;
        float4 texcoord: TEXCOORD0;
};
struct vertexOutput {
        float4 pos : SV_POSITION;
```

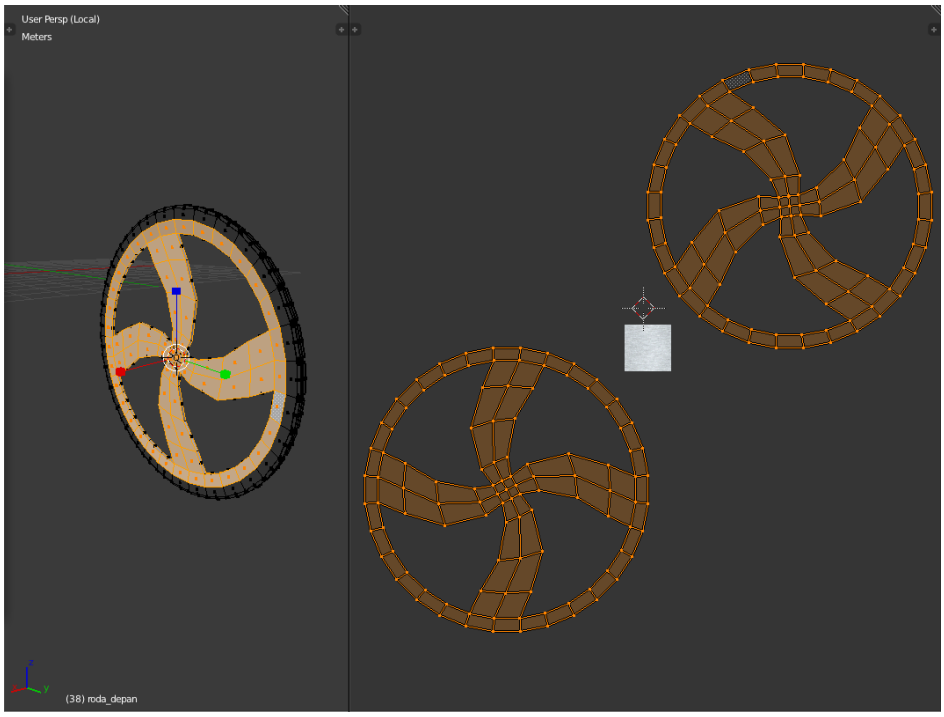

Figure 3: Manual UV mapping of the bicycle wheel

```
            float4 tex : TEXCOORD0;
            float4 posWorld : TEXCOORD1;
            float3 normalDir : TEXCOORD2;
        };
```

All of our shaders dealt with different light sources. For our metallic and clothing shaders we had to calculate for diffuse and specular reflection. We defined a unity attribute for the shininess of the material. Therefore we did not have to alter the code make metallic materials more reflective. Instead we just alter an attribute. However, we decided to include rim lighting to our metallic materials. This provides a modest glow effect around our metallic materials while non metallic material has a more modest reflection.

```
        float3 rimLighting = atten * _LightColor0.rgb *
                    _RimColor.rgb * saturate(dot(normalDirection,
        lightDirection)) * pow(rim, _RimPower);
```

All of our shaders work with textures. Textures are applied and defined as Unity attributes. This allowed us to dynamically change the texture. Textures had to be scaled and adjusted according to the appropriate shader.

```
        float4 tex = tex2D(_MainTex, i.tex.xy * _MainTex_ST.xy
            * _MainTex_ST.zw);
```

Figure 4: Overview of the full default Bicyclist. Level of texture detail is adequate for the limited user attenuation

3.4 Creating Controls

One of my primary objectives was to provide a varying levels of detail. While [Dublin, 2011] explores how represent realistic representations of pedestrians. We looked to apply similar methodology to modeling realistic cyclist scenes. We wanted to design our tool to be adjustable so we could explore how to make more realistic scenes. I wanted to have each cyclist to be able to appear realistic and different than the previous cyclist. I added to the bike generation script allowing the cyclist components to have different textures. These textures can be defined as unity scene attributes.

I created a Unity controller that controls the variability of the cyclist appearance. My goal was to reduce the amount recognition and recall of similar cyclist in the scene. [Jeffrey, 2008] addresses how effective viewpoints can be when it comes to recognition. In our situation we wanted to reduce the amount of recognition. Our camera view is orthogonal to the direction of the cyclist. Therefore, we had to supplement the higher probability of recognition with higher texture variability and lighting augmentation.

3.5 Connecting to Citibike Data

I, along with a team of other KTH students, designed and implemented a framework to help people visualize and analyze data. One of the key aspects of this framework was the ability to make analytics request as REST request. For the purposes of this project, I implemented a portion of the REST service that provides the prevalence of trips between to bike stations. This prevalence is the ratio of the total number of trips from the start and end stations weighted the by the max number of trips from the start station to any other station.

I added the ability for Users to control if they wanted to manually control simulated bicycle traffic

Figure 5: Bike under two lights. The bike frame demonstrates how reflection is highlighted on both the front and the back.

or connect to the latest aggregated Citibike data within our Unity scene. The User Interface is still very young and it was only designed for proof of concept. In actual analytic practice, we envision the scene to be supplementary to other processes. Specifically, we assumed that the parameters for the traffic simulation would be controlled externally.

4 Challenges

We used a bicyclist model that was initially created by someone else. We found parts of the model were too monolithic. For example, the bike seat and upper bike frame were all one piece. This made the task of creating UV maps and texturing more difficult. I was able to separate out the components like the wheel and the tire but in the example of the bike seat, I was unable to separate the components. Figure 2 and 6 show the difficulty of object texture separation.

Writing shaders in Unity/CG was not a trivial task and required doing a lot of 3rd party research. Cg is very susceptible to syntax, type cast, and various other technical bugs. Eventually, I was able to reduce specular, diffuse, and rim lighting calculation a few lines of code once I figured out the nuances.

```
// Lighting
float3 diffuseReflection = atten * _LightColor0.rgb * saturate(
        dot(normalDirection, lightDirection));
float3 specularReflection = diffuseReflection * pow(saturate(
        dot( reflect( -lightDirection, normalDirection), viewDirection)),
         _Shininess);
// Rim lighting
float rim = 1 - saturate(dot(viewDirection, normalDirection));
```

Figure 6: Bike seat unclose that shows the combination of specular and diffuse reflection along with rim lighting.

```
float3 rimLighting = atten * _LightColor0.rgb * _RimColor.rgb *
        saturate(dot(normalDirection, lightDirection)) * pow(rim,
        _RimPower);
```

In order to dynamically change the texture of components I decided to use tileable textures instead of custom ones. [William, 1996] describes how local and global repetition effects viewer cognition. I wanted to ensure the level of detail was effective enough for the entire cyclist model to look as realistic as possible at the scene's camera position. At the current scene?s camera distance the temporal locality and the level of detail of the bicycle was deemed suitable for a natural appearance.

Currently, our server houses a database about Citibike bicycle trip data. We have a record for the starting and ending station of every bike trip within a finite amount of time. We have to calculate the relative traffic density by normalizing the numbers of trips over maximum number of trips. Ideally we would like to connect to a live traffic feed of bicycle traffic. Unfortunately, that kind of data does not currently exist.

5 Conclusion

This project allowed us to exercise implementing important graphical shading concepts within a specific context. It also allowed us to build a tool to create background bicycle scenes. We can alter the speed, traffic density, and variability of the bikes in the scene. We connect to real data via a REST API. However, in the future we would like to run scientific trials to assess what is the ideal canonical parameters for a bicyclist scene. There is also room to connect directly live traffic data. The current data available from Citibike and similar services are kept as a simple log format. There were no route data available. The project was a good exercise and challenge for my partner

and I to incorporate multiple facets of Data Analytics and Computer Graphics.

5.1 Task Matrix

Task	Team Member
Render background scene	St-Onge
Render Bicyclist	St-Onge
Animate Bicycle movement	St-Onge
Create scene controls	St-Onge and Hotan
Bind Bicycle generation to controls	St-Onge
UV - Mapping and Textures	Hotan
REST API and Server	Hotan
Unity REST Client	Hotan
Scientific Assessment of Realism	Not Complete
Connect to Real time traffic data	Not Complete

5.2 Links

- Demo Video `https://www.youtube.com/watch?v=nZhTO4bQLfs`
- Graphics Project Repository: `https://github.com/mhotan/DD2323.git`
- REST Server Repository: `https://github.com/mhotan/CityBikeDataExplorer.git`

References

blendtuts. Tileable textures in blender. URL `http://www.blendtuts.com/tileable_textures`.

Citibike. Citibike system data. URL `http://www.citibikenyc.com/system-data`.

dotline11. Bicycle animation. URL `http://www.blendswap.com/blends/view/46956`.

CATHY ENNIS, CHRISTOPHER PETERS, and CAROL O'SULLIVAN, Trinity College Dublin. Perceptual effects of scene context and viewpoint for virtual pedestrian crowds. *ACM Transactions on Applied Perception (TAP)*, 8(10):12, 2011.

Jeff Ferzoco. Citibike visualization. URL `http://linepointpath.com/111242/2771111/work/citi-bike-visualization`.

Universal Image. Parkchair. URL `https://www.assetstore.unity3d.com/#/content/850`.

Jacek Jankowski. Simple modular street kit. URL `https://www.assetstore.unity3d.com/#/content/13811`.

Gomez, Pablo ; Shutter, Jennifer ; Rouder, Jeffrey. Memory for objects in canonical and non canonical viewpoints. *Psychonomic Bulletin and Review, 2008, Vol.15(5), pp.940-944*, 15(5): 940–944, 2008.

Wikibooks. Blender 3d: Noob to pro/uv map basics. URL `http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/UV_Map_Basics`.

Lamb, Marvin R. ; Yund, E. William. Spatial frequency and attention: effects of level-, target-, and location-repetition on the processing of global and local forms. *Perception and Psychophysics*, 58(3):363–373, 1996.