

KUNGLIGA TEKNISKA HÖGSKOLAN
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION



DH2323

A Nice Title

Project Documentation

Michael HOTAN
Alexandre ST-ONGE

May 6, 2014

1 Introduction

For this project, we designed and implemented a Unity application to simulate bike traffic in a park scene. In phase 1 of the project we first created a park scene containing a road, park bench, trees and light sources. In phase 2 we started working on the bike traffic. Finally, in phase 3 we worked on shader to apply to our cyclist model.

2 Documentation

2.1 Rendering a scene in Unity

The first step in this project was to create a realistic park scene to use as a background for our application. The simplest way to do this was to use the built-in terrain feature of Unity. With the terrain object we were able to easily obtain a grassy plain with trees and hill. Next we used Jacek Jankowski's street kit[4] to setup a road for our future bike traffic. From there, since we knew we wanted to work with some shader for our bike model, we added some street lights that we got from the asset store[1] and some park bench[3]. Appropriate material for the sky were used with Unity's skybox to obtain a nice sky for the scene. For the daylight scenario, a single directionnal light is used for the sun, while the nighttime scenario uses one spotlight attached to each of the four street lights.



Figure 1: Park scene in Unity

2.2 Rendering the cyclist

Next step in our project was to find a 3d model for the bike and the cyclist. We used a 3d model from www.blendswap.com [2], as seen in figure 2. The model already had a basic animation so it was perfect for our need. Since it was a blender file, using it in Unity was not very simple. First, the animation needed to be adapted for our need. I had to change the animation slightly to make it more "loopable" in Unity. Also, the initial animation had the model moving forward, causing it to "teleport" back to its initial position at every loop, I then changed it so the animation now plays in-place. Having in-place animation is needed since we wanted to use Unity to move the bike ourselves. Once everything looked great in blender, the model was loaded into our Unity project. The bike was then scaled and rotated accordingly and was saved into what Unity call a "prefab". Having the model as a prefab allow us to easily instantiate the model during runtime.

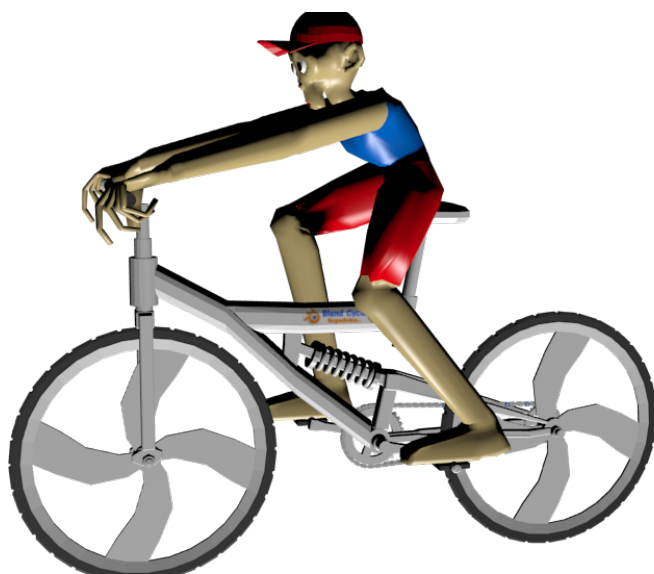


Figure 2: Bike 3d model found on blendswap.com

2.3 Generating Traffic

The final step was to generate traffic. There is multiple way to do this. The first method we try was simply to add a rigidbody onto the bike and add a constant force in the direction we want with a script. Since the animation is automatically playing and looping, this is a simple and easy way to have a bike rolling from one point to the other in a straight line. As the density of bike goes up, the chance of collision is much higher too and since we have not implemented any path-finding the simulation quickly becomes a mess. Luckily, Unity provides us with a built-in pathfinding solution with NavMeshAgent. First I had to define the navigable area in the scene with a NavMesh. We can do this in the navigation window in Unity by disabling navigation for the terrain and enabling it for our set of road. Next, we need a target and a spawn point for our bike. We can create these with simple empty GameObject placed at the appropriate position. We can then add a NavMeshAgent

on our bike prefab. We now have to setup the agent with a script. This is done in NavBike.cs in the Update() method with a simple line of code:

```
agent.SetDestination(target.position);
```

Once the destination is set, the agent will move the object toward the target while avoiding any obstacles (i.e.: other bikes) in the way. Now that we have a simple way of moving bike from one side to the other we can work on making the traffic nicer. This is done in the GenerateBike.cs script. I modeled the traffic according to a simple equation where the flow is equal $\mathbf{u} \cdot \mathbf{k}$ where \mathbf{k} is the number of vehicles per unit of distance and \mathbf{u} is the average speed of the vehicles. To be able to observe how these two variables affect the flow, I use two sliders: one for the density and one for the speed. If we want more bikes per distance we need to spawn more bikes, this is done by sliding the density slider to the right. The slider affect the intervals between each spawn. The Random.Range function is used to make the spawn intervals look more natural. The next slider affect the bike's speed. Before being instantiated, each bike is assign a speed randomly chosen in an interval around the slider's value. This allow us to choose the average speed with the slider, while still having faster and slower bike. The ComputeFlow() method is there to quantify the traffic flow so we can show it on screen. The flow is calculated using the equation presented earlier.



Figure 3: Screenshot of the application running with the GUI in the top left corner

2.4 Conclusion

For this project I had the responsibility of generating bike traffic at runtime in a realistic manner. While my partner is working on finding realistic way of rendering the scene with shader, I was in charge of finding a realistic traffic flow. While there is a lot of research done on the subject of traffic flow for computer generated scene, a lot of them were too much complex for my level of understanding and I found that they didn't really apply to our scenario. Indeed, bike traffic is

very different from car traffic. Car traffic is organized, dense and follow a numbers of well defined rules. On the other hand, bike traffic is a lot more sparse and organic. I actually spend some times in parks observing bike traffic for this project and observe a low traffic flow at around 8 or 10 bikes per minutes. The speed variation between bikes is also considerable, which is why I used a random number generator to assign speed over an interval instead of using the same speed for everyone. Working on this project allowed me to learn about how to create scene with a framework like Unity. It can be difficult to put all these 3d models together, you have to think about scale, positions and rotation, and if you want to generate object by script you need to understand these concept even more. By choosing to use the navigation framework provided by Unity, my goal was to make sometime reusable and I'm confident that my work could be easily in any scene where you would want to add bike traffic between two points.

3 Perception Studies

References

- [1] Adam Badke. *Urban Props*. URL: <https://www.assetstore.unity3d.com/#/content/708>.
- [2] dotline11. *Bicycle Animation*. URL: <http://www.blendswap.com/blends/view/46956>.
- [3] Universal Image. *ParkChair*. URL: <https://www.assetstore.unity3d.com/#/content/850>.
- [4] Jacek Jankowski. *Simple Modular Street Kit*. URL: <https://www.assetstore.unity3d.com/#/content/13811>.