Naimur Rahman

Tunnel Run

Unity Infinite Racing game

Introduction

I've always been fascinated by procedural games. The possibilities are endless since we can create and update any shape we want during the game. Until this project I never really took the time to look into the subject and actually create a procedural game. Then after watching some tutorials I gave it a shot myself and came up with the idea of driving through an endless tunnel.

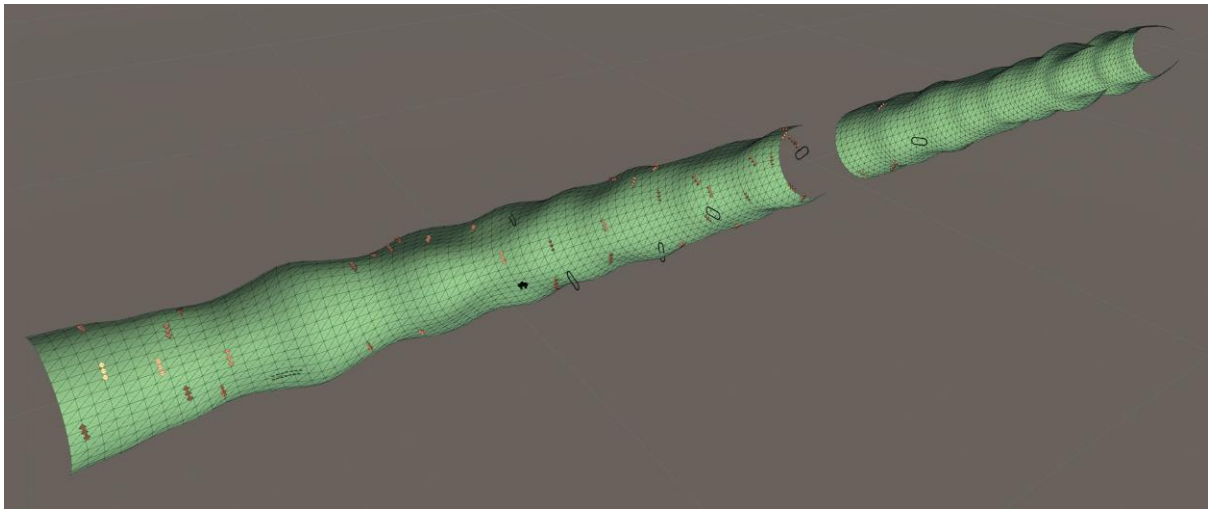
General project

The goal of this project was to create an endless racer through a cylinder with grass hills and obstacles. Initially, the idea was to create a cylinder shape and then simply update the vertices using perlin noise to move them towards the player and have some kind of grass wave effect. In theory this would work very well, but in practice it was a bit more difficult; Since the perlin noise would be updated continuously, instead of having grass hills moving towards the player, I got some weird wave like motion which didn't look like it was moving towards the player car at all.

Instead, I figured if I just created the grass tunnel once without updating the vertices afterwards, it would always be static and therefore it fixed the wave-like motion. Now the only issue was that a game with just one fixed tunnel is not endless and it still didn't look like the player car was actually moving. Therefore, I simply moved the new cylinder objects towards the player car and whenever the second cylinder (out of the two initial cylinders that make up the tunnel) wasn't visible anymore, I generated a new one and removed the invisible cylinder.

Now, to actually create some variation in the environment, I decided to add some offset to the perlin noise value and update the offset whenever a new cylinder is generated. This way each generated cylinder looks different from the last one. Also we have full control over the generated terrain using the height, noise scale, dimensions and randomness values.

The terrain is built up out of individual cylinders:



You can see through because of back face culling (we don't need outer faces since those won't be visible to the player).

Project structure

The project structure is relatively simple. There's two scenes; the main menu and the actual game scene. The game over screen is included in the game scene so we can easily reload the game scene after crashing the car to restart the game.

Please open scenes -> menu to view the menu. In the menu you can mute the music and press enter to start the game. Also, this scene contains the music object that will loop your background music throughout all scenes in the game. If you start the game directly from the game scene, everything will work fine, but there won't be any background music since the music object is located in the menu scene. One more thing to keep in mind for the menu scene; the smaller title says 'enter to play' but you'll probably want to use the title that says 'tap to play' for mobile games. So please enable the other title if you're targeting mobile platforms. Please do the same thing for the small title at the bottom of the game over screen inside the game scene.

Except for the music button and the music object, pretty much everything is located inside the 'Game' scene. This scene contains the player car, which is not a prefab since that makes it easier to reference other objects in the scene. It also contains the game manager that will generate the cylinder terrain and handle game states. Finally, it has the UI canvas for showing scores and the game over menu.

Main menu scene:

PlayerPrefs

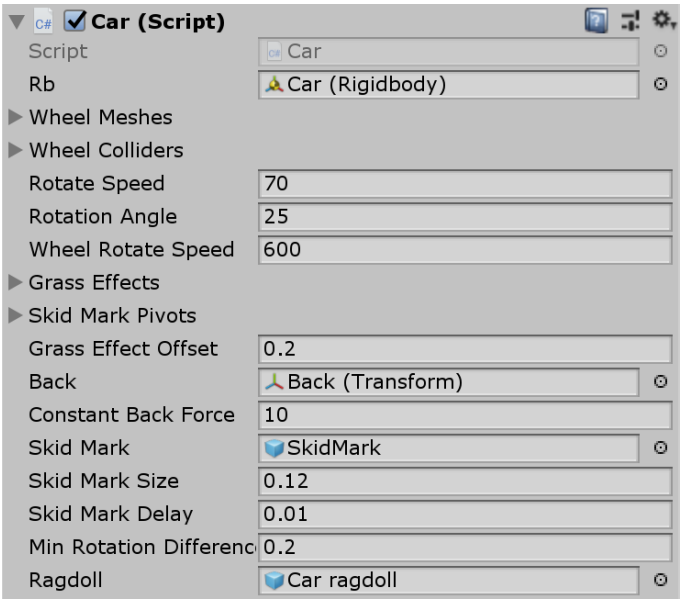
The game uses Unity's build-in playerprefs to store the best score. Also, the editor has a simple script that allows for easy playerprefs deletion. To delete the best score (and reset the music volume) please go to Window -> Delete PlayerPrefs.

General game settings

I'll describe the general game settings here. First let's clarify something that might look confusing; the general rotation speed (when the player car turns, the speed at which it moves sideways) is controlled via the directional light object. Please see the Basic Movement component attached to the light object to control the general world rotation speed. The reason for me to control this speed via the directional light is because the directional light is the only part of the world that is already there. The other parts will be procedurally generated and therefore we cannot use those to control any game settings. The directional light will also rotate to make it look like the player car is moving instead of the cylinder world and the world pieces will simply copy their rotation speed from the directional light.

Car settings

For the car settings please select the car object in the hierarchy:



Rb: reference to the rigidbody so we can add force to the back of the car.

Wheel meshes: List of the wheel meshes.

Wheel colliders: List of wheel colliders (have to be in the same order as the wheel meshes).

Rotate speed: The speed at which the player car turns.

Rotation angle: The maximum rotation angle for the player car.

Wheel rotate speed: speed at which the wheel meshes will rotate.

Grass effects: These are the grass particle effects. Please make sure to keep them in the same order as the rear wheel meshes.

Skid mark pivots: The transforms that provide the skid mark spawn positions.

Grass effect offset: The downwards offset for the grass and skid effects. Depends on wheel radii.

Back: Transform at the back of the car where force will be applied to keep the car balanced.

Constant back force: The force that will be applied to the back of the car whenever the car is not grounded.

Skid mark: skid mark prefab.

Skid mark size: The size of individual skid marks.

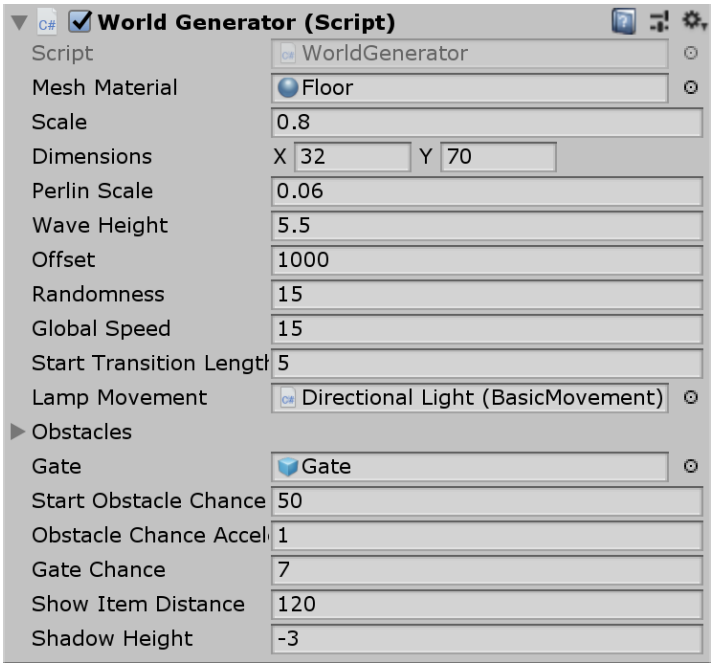
Skid mark delay: The delay in between individual skid marks.

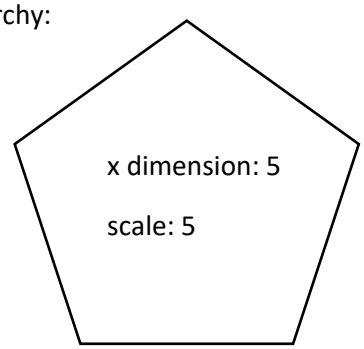
Min rotation difference: The minimum rotation change compared to last frame for skidmarks to show up.

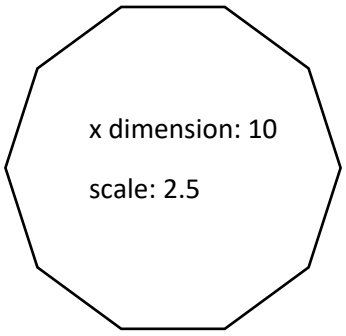
Ragdoll: Prefab version of the car that only consists of meshes with rigidbodies so it will fall apart immediately. This will be instantiated when the car crashes so it looks like the car breaks.

World generator settings

For the world settings, please select the game manager in the hierarchy:







I'll first explain the most important values. The **scale** and **dimensions** together determine the size and detail of your terrain. The scale means the distance between individual vertices and the dimensions stands for the number of triangles that will make up the mesh. For example, if you change the default scale of 0.8 to 1.6 and set the x dimension to 18 instead of 32, the mesh will mostly look the same, just way less detailed. The reason behind this is because you're basically adding half the points in 3d space, but spread them twice as far apart which results in the same size. I would recommend (especially for the default world size) to keep about the same level of detail since it looks good and drives smooth, but it won't completely slow down the game. The y dimension matters a lot less than the x dimension. Basically, the y dimension is the depth of the generated cylinders, so a bigger value means you can look further into the tunnel without noticing new tunnels being generated.

Another important value is the **perlin scale**. You can adjust this value to adjust the roughness of the terrain. For example, decreasing the value will smooth out the terrain so there's less hills and therefore smoother terrain, while increasing this value will make the terrain more rough in that there will be more small hills which makes controlling the car harder. After perlin scale, we can also set the **wave height**. This will control the height of the grass hills inside the tunnel. A smaller value will of course make for a smoother drive, while higher values create bigger hills and more difficult terrain.

Then there's the **offset** value. This value controls the total shape of the terrain. Leaving all values the same including the offset should result in the same terrain each time. The start offset doesn't necessarily matter that much, but we do need to make sure to change the offset each time to keep generating new terrain during the game. To do so, please use the **randomness** value. This value will be added to the offset each time after generating new terrain. A bigger randomness value makes for a bigger difference between the current terrain and the previous terrain, where a small randomness value would barely change the terrain compared to the previous terrain.

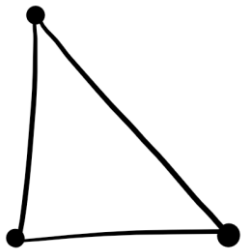
Other settings:

- **Mesh material**: The material for the generated mesh. You can use tiling and offset on your material to fit it with the terrain to your liking.
- **Global speed**: The global moving speed (car speed).
- **Start transition length**: The number of rows of vertices involved in transitioning from the end of the last cylinder to the hills in the new cylinder. This makes sure to correctly connect terrain cylinders without a gap or a sudden hill. A smaller value means a less subtle transition. For example, 1 means a sudden hill in between the cylinders, where for example 7 is much smoother so the player barely notices the transition.
- **Lamp movement**: Reference to the script that will rotate the directional light.
- **Obstacles**: Array of obstacles, you can add your own obstacle here as well.
- **Gate**: The gate object, you can replace it by your own gate but it's easier to just modify the gate prefab since that will be updated automatically.
- **Start obstacle chance**: The chance of obstacles being spawned. Actually it's more like the opposite of chance, since the chance of obstacles being spawned is 1/start obstacle chance. A bigger value means less obstacles, a smaller value means more obstacles.
- **Obstacle chance acceleration**: How much bigger does the chance of obstacles being spawned get after each newly generated cylinder. So for example, with a value of 1 the number of obstacles increases very gradually, while 5 would mean the player sees a lot of obstacles very early in the game.
- **Gate chance**: For each spawned item, the chance of that item being a gate is 1/gate chance. So a bigger gate chance means less gates, where a smaller gate chance means more gates which makes the game easier.
- **Show item distance**: The distance at which gates and obstacles first appear.
- **Shadow height**: The height (from the world centre, that's why it's negative; because the y position of the car is below 0) at which shadows appear. Any object below that height will have shadows and objects above that height won't have shadows. This it to prevent weird shadows from items that are not visible to the player.

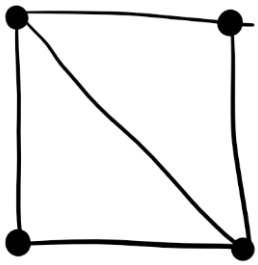
Procedural terrain generation

The main target of this project was to procedurally generate an endless terrain to race through. I will try to explain the basic steps to generating tunnels like the terrain you see in the game.

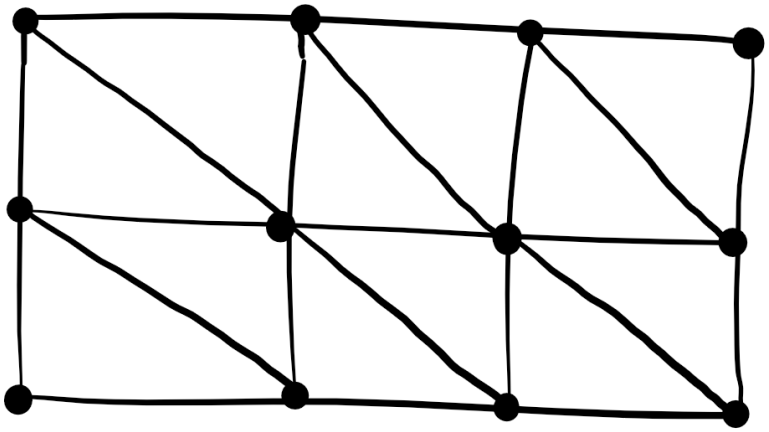
1) I started by generating just one triangle. Basically all that takes is to create a new mesh and add three vertices (one for each corner) and three indexes in the triangles array to reference our vertices. Then you get something that looks like this:



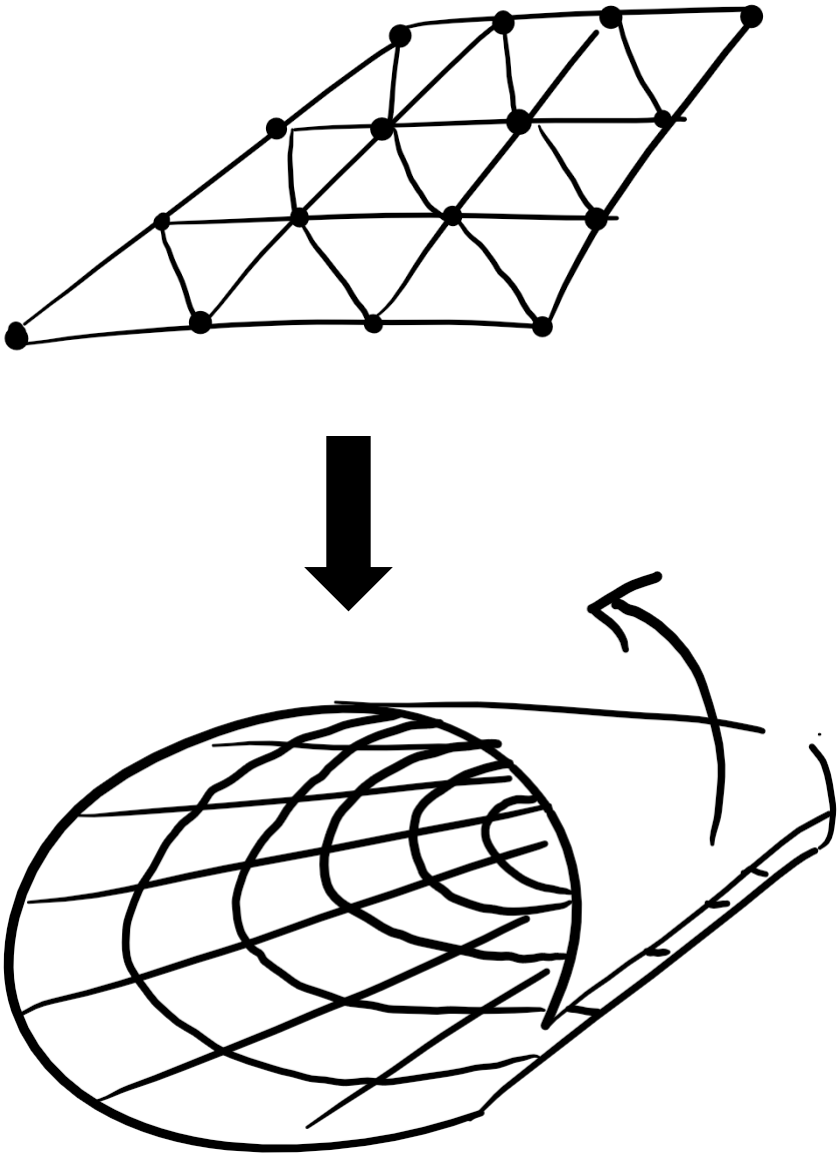
2) Then by adding only one vertice (one point in 3d space) and three new values in the triangle array (three new corners) we can add another triangle and create a square:



3) Next, using two for-loops, I generated an entire plane out of squares, which looks something like this:

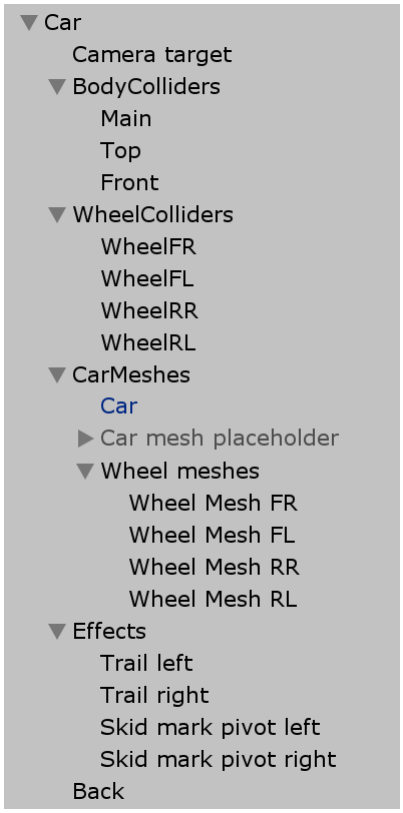


4) And last but not least, for the cylinder, the plane needs to be wrapped into a new shape using cosines and sinus on the vertices:



Car setup

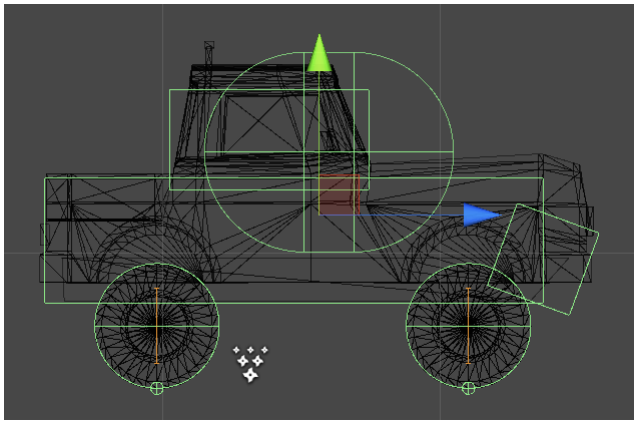
The car is using Unity's build-in wheel colliders for suspension physics, and the meshes are separate objects. Therefore the car is easily replaceable. If you want to change the look of the car, please take a look at the hierarchy:



As you can see, there's the body colliders, wheel colliders and car meshes all under separate parent objects. To change the car, it's best to simply replace the wheel meshes and body mesh. For the wheel meshes, make sure when assigning them to the car script array to keep the same order so they correctly correspond to the wheel colliders.

For the body mesh, just make sure to correctly position the mesh above the wheels and scale the mesh if neccasery.

Then after replacing the car meshes, please take a look at the body colliders:



These colliders are resized based on the actual mesh, so they correspond to the car body. Whenever you change the tire size or the car body size, please make sure to either add new colliders or resize the existing colliders to match your car. Also please notice the small box in the front. This box should keep the car stable in case it lands on the nose of the car.

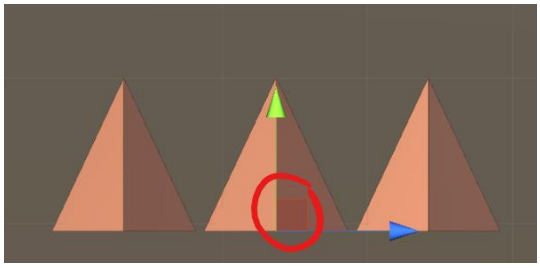
Furthermore, the big round collider on top is a trigger that will be triggered when the car flips over. This way we can stop the game whenever the car loses balance and falls over.

Game items

The game currently consists of two items, the spike obstacles and the gates that the player should drive through to score points. The main components needed to set up any obstacle are the following:

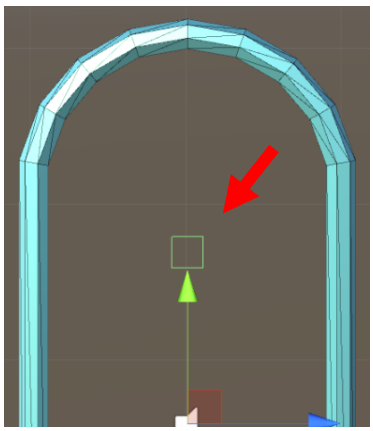
- The obstacle should have a regular collider
- The obstacle should have the Obstacle script
- And finally, please tag your obstacle ‘Item’ since it’s a game item

Also please make sure to have the origin of the obstacle at the bottom centre so it will be positioned correctly onto the terrain:



After creating your obstacle, please add it to the prefabs folder (which automatically creates a prefab) and add the new prefab to the obstacles array in the world generator.

Except for obstacles, we also have the gates to score points. These look largely similar to the obstacles, they just have one extra feature which allows us to score points. For scoring points, the gates have an extra collider, which is a really small box collider set to ‘is trigger’. Now all the gate script will do is check for the trigger and add some points to the game manager (it will also play some audio). The most important thing about setting up gates is please make sure to correctly centre the trigger and make it as small as possible so it won’t be triggered when we hit the gate itself (because hitting the gate means the game is over):



Also just like the obstacles, please make sure to tag the gates ‘item’.

Conclusion

I hope this document gives a clear overview of the procedural racing asset. I have tried to explain everything as clear as possible, but please don't hesitate to contact me for any questions or suggestions via:

aminrahman24@gmail.com

Additionally, since this is the very first release I would really appreciate any review or feedback on github.
https://github.com/adamtrinity