Final Project

# Development of a game
# “NEON RUSH”

國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Prepared by Group 14:
110550124 林家甫
111550205 葉莉莎

# 1 INTRODUCTION

In today's world, more and more people are becoming interested in action games that revolve around driving and stealing. Racing games, in particular, stand out as one of the most popular genres.

Recognizing this trend, our project focuses on developing a game that captures the excitement of real-life racing games.

**The work aims** to create a project that includes at least one object, a geometry shader to create a new point, line, or polygon, and implement some animation.

**Tasks of the work:**

1. Rendering a car model.
2. Setting up an environment.
3. Developing smoke from exhaust pipe.
4. Implementing movement for both the car and the camera.
5. Adding a game features

## 1.1 Game Story

**It's Summer 1985, you're driving in Miami🌴**

You are Jeff, a young bank robber. You just robbed a jewelry store and made off with jewelry worth more than 1 million dollars. However, things went wrong, and the police are in pursuit. You're driving as fast as possible, dodging obstacles in an effort to lose the police.

## 1.2 Game Introduction

The game challenges players to dodge the obstacles and score 1000 points. If the player finishes one lap, they will receive 100 points; however, when they hit obstacles, they will deduct 200 points. The player's goal is to reach 1000 points lower than 100 seconds.

# 2 IMPLEMENTATION DETAILS

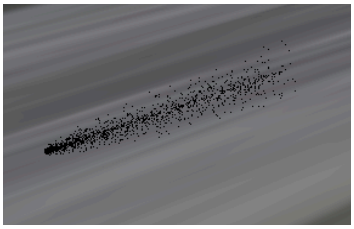
Our project consists of two main components: Particle Effects and Car Rendering.

## 2.1 Particle Effects

One of the most common ways of using geometry shaders is about rendering with billboards, namely **billboard particles**.

In the project, we wanted to have some smoke from exhaust pipes to provide interactivity and resemble a racing game.

And for that smoke, we used a particle system. We generated a bunch of particles; look at the picture.



Picture 1. Initial Particle System

Then, we decided to make our particle system more realistic. The particle system looked pretty good, but the pixels were always the same size; if we moved the camera back, they looked kind of the same size, not super realistic. We figured out that we needed to create a billboard, but it is a full-on class with its update method and everything. And, we have quite a few particles here, so one approach that works pretty well is to make a geometry shader, which takes as input a single point from the vertex shader and then transforms that and outputs a whole quad in view space, and that's how we could get billboards on the screen quickly.

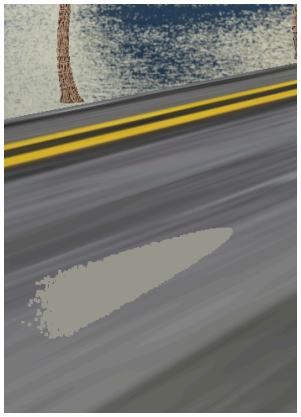
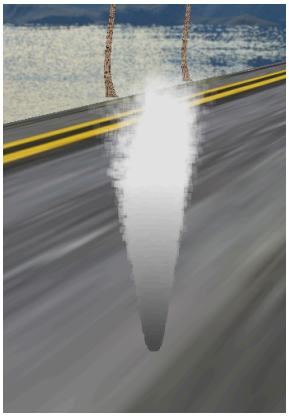
We have a **particle object** that models a single point; it has position, velocity, up_vector, and down_vector. It has a **modelTransform** that holds the position. It also has some other parameters, like the color and life of the object and its maximum lifetime.

In the **update** function, we increment our velocity according to our **up_vector** and **side_vector** values; then, we increment our **position** according to our **velocity**. **Tint** is the final color for the particle; the alpha reduces over the life of the particle, and it makes and makes the particles fade away.

**makeParticles()** basically sets up slightly randomized directions, lifetimes, and velocities for each particle and creates them.

Here are the steps we took in implementing the smoke:

1. Just one color of smoke (Picture 2.a).
2. The color transitions from dark to light gray and increases quads as their lifetime increases (Picture 2.b).
3. Blending is added (Picture 2.c).
4. Up_vector and side_vector are used to add a smoke volume (Picture 2.d).
5. Tuning some other stuff, we got this final result (Picture 2.e).

a)  b)  c) 

d)  e) 

Picture 2. Progression of Smoke Implementation

## 2.2 Car Rendering

**Reading an Object File and Mapping It to Multiple Texture Images → Failure**

We store a color for each vertex to mimic the texture images.

a)

```
219043
219044
219045 usemtl trueno_trunkb__spec_
219046
219047 f 1/1/1 2/2/2 3/3/3
219048 f 1/1/1 3/3/3 4/4/4
219049 f 5/5/5 6/6/6 7/7/7
219050 f 5/5/5 8/8/8 9/9/9
```

b)

```
80
81 newmtl trueno_trunkb__spec_
82 Ka 0.20 0.20 0.20
83 Kd 1.00 1.00 1.00
84 Ks 0.29 0.29 0.29
85 illum 4
86 map_Kd trueno_trunkb.png
87
```

Picture 3. An example using a Race Car .obj file and .mtl file

Initially, the object file will state a string after **usemtl** to show which texture image the following triangles (vertices) are going to apply. In the picture above, the following triangles are going to apply the **trueno_trunkb.png** texture image. Then we look at the color of that image, and assign the average color of that image to all corresponding vertices. In the picture the trunk triangles are assigned the color black.

**How do we assign those vertices into black color?**

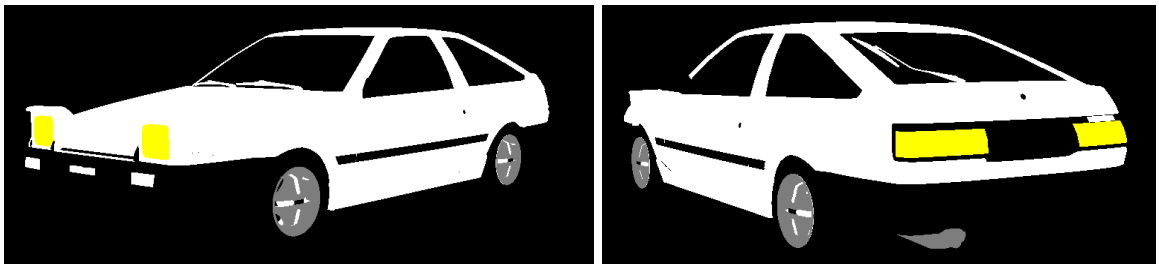
We add a vertex attribute to the VAO and assign it a value. This value determines the color that will be applied to the vertices.

```
   // each vertex has its own color
   glBindBuffer(GL_ARRAY_BUFFER, VBO[3]);
   glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (colors.size()), &(colors[0]),
GL_STATIC_DRAW);
   glVertexAttribPointer(3, 1, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 1, 0);
   glEnableVertexAttribArray(3);
   glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Snippet 1. object.cpp

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoord;
layout (location = 3) in float aVertexColor; // each vertex has its color
```

Snippet 2. Vertex shader



Picture 4. Result model

**Modification**

Since we read the line **usemtl** in the object file to assign vertex colors, for the objects binding texture images, we delete the line **usemtl** in those objects like the palm tree and road. Therefore, TAs should use the object files we provide (in E3 or from the GitHub link: https://github.com/chiafu2018/ICG-Project).

# 3 DISCUSSION

## 3.1 Role Play

**Proposer:** Liza

It would be cool to make a car, take a car model, and use a geometry shader to make particles effect to produce smoke from pipes or from wheels when turning. It's gonna be interacting. Additionally, it would be great to implement this as a game. Like the race car we all have played online before, we can make it run for round laps where players must dodge obstacles. Moreover, we can add a neon effect and set the environment (cube map) in a purple theme to add the retro wave elements inside.

**Critic:** Jeff

Everything is perfect, but I think the track is too complicated to implement. As a result, I think we can set the race track straight. When the car reaches the finish line, reset the car to the origin (starting line) and call it a lap. (score +100) Last, we can play retro wave music in the background when we record the video.

**Negotiator:** Both

It would look good to allow the camera to move with the car and make it in three dimensions: front, back, and top. By doing so, we can see the smoke effect and the front of the race car.

# 4 WORK ASSIGNMENT

**Liza's assignments :**

- Develop and implement the **smoke effects** using particle systems.
- Handle **camera movement**.

**Jeff's assignments:**

- Manage the **car rendering**, including reading object files, assigning vertex colors, and refining the car's visual representation.
- Implement **car movement and game logic**

# 5 REFERENCES

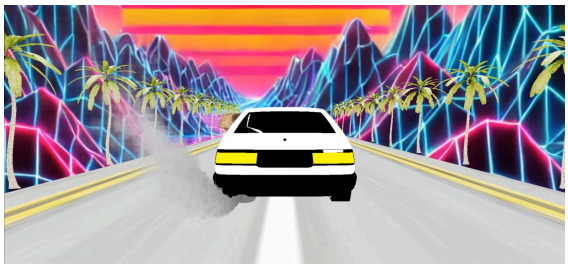
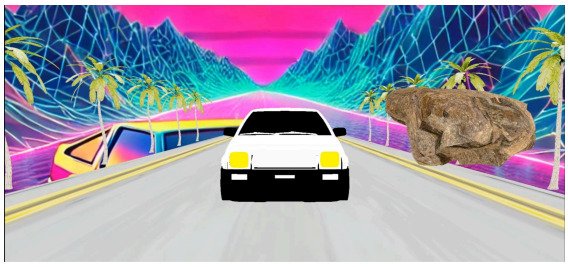
All the objects in this project were found in sketchfab, an online free library.
Online Free 3D Model Library Link: https://sketchfab.com/

# 6 RESULTS

**The purpose** of this project, namely creating animation, creating at least one object, and implementing geometry shade, was achieved.

The video link, and pictures of the result are shown below.

**Video Link:** https://youtu.be/d8vG-NwXv3o

a)  b) 

Picture 5. In-Game Screenshots

```
Game Start!
Lap:1 Score:100
Lap:2 Score:200
Lap:3 Score:300
Lap:4 Score:400
Lap:5 Score:500
Lap:6 Score:600
Lap:7 Score:700
Lap:8 Score:800
Lap:9 Score:900
Lap:10 Score:1000
You Win! Jeff escaped successfully
```

Picture 6. Game Results Terminal