# 3d游戏设计作业八

# 3D游戏设计读书笔记八

## 一、 作业

本次作业基本要求是三选一

1、简单粒子制作

按参考资源要求，制作一个粒子系统，参考资源

使用 3.3 节介绍，用代码控制使之在不同场景下效果不一样

2、完善官方的“汽车尾气”模拟

使用官方资源资源 Vehicle 的 car， 使用 Smoke 粒子系统模拟启动发动、运行、故障等场景效果

3、参考 http://i-remember.fr/en 这类网站，使用粒子流编程控制制作一些效果， 如“粒子光环”

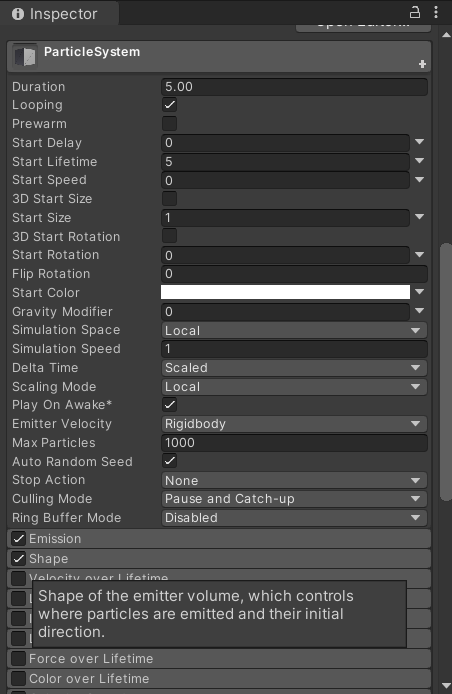
可参考以前作业

我选择3

## 二、 粒子光环

#### 1. 创建一个空对象，命名为ParticleSystem

(1)点击Add Component添加粒子系统



#### 创建CirclePosition类

定义一个辅助类CirclePosition，用来进行定位。

```javascript

public class CirclePosition

{

public float radius = 0f, angle = 0f, time = 0f;

public CirclePosition(float radius, float angle, float time)

{

this.radius = radius; // 半径

this.angle = angle; // 角度

this.time = time; // 时间

}

}

```

#### 创建NormalDistribution类

实现粒子得高斯分布

```javascript

public class NormalDistribution

{

// use Marsaglia polar method to generate normal distribution

private bool \_hasDeviate;

private double \_storedDeviate;

private readonly Random \_random;

public NormalDistribution(Random random = null)

{

\_random = random ?? new Random();

}

public double NextGaussian(double mu = 0, double sigma = 1)

{

if (sigma <= 0)

throw new ArgumentOutOfRangeException("sigma", "Must be greater than zero.");

if (\_hasDeviate)

{

\_hasDeviate = false;

return \_storedDeviate \* sigma + mu;

}

double v1, v2, rSquared;

do

{

// two random values between -1.0 and 1.0

v1 = 2 \* \_random.NextDouble() - 1;

v2 = 2 \* \_random.NextDouble() - 1;

rSquared = v1 \* v1 + v2 \* v2;

// ensure within the unit circle

} while (rSquared >= 1 || rSquared == 0);

// calculate polar tranformation for each deviate

var polar = Math.Sqrt(-2 \* Math.Log(rSquared) / rSquared);

// store first deviate

\_storedDeviate = v2 \* polar;

\_hasDeviate = true;

// return second deviate

return v1 \* polar \* sigma + mu;

}

}

```

#### 创建ParticleHalo类

1. 创建变量，定义粒子得各个属性。

```javascript

private ParticleSystem particleSys; // 粒子系统

private ParticleSystem.Particle[] particleArr; // 粒子数组

private CirclePosition[] circle; // 极坐标数组

public int count = 10000; // 粒子数量

public float size = 0.03f; // 粒子大小

public float minRadius = 5.0f; // 最小半径

public float maxRadius = 12.0f; // 最大半径

public bool clockwise = true; // 顺时针|逆时针

public float speed = 2f; // 速度

public float maxRadiusChange = 0.02f; // 游离范围

private NormalDistribution normalGenerator = new NormalDistribution(); // 高斯分布生成器

public Color startColor = Color.blue; //初始颜色

```

2. 对粒子进行初始化设置

```javascript

void Start()

{ // 初始化粒子数组

particleArr = new ParticleSystem.Particle[count];

circle = new CirclePosition[count];

// 初始化粒子系统

particleSys = this.GetComponent<ParticleSystem>();

var main = particleSys.main;

main.startSpeed = 0;

main.startSize = size; // 设置粒子大小

main.loop = false;

main.maxParticles = count; // 设置最大粒子量

particleSys.Emit(count); // 发射粒子

particleSys.GetParticles(particleArr);

RandomlySpread(); // 初始化各粒子位置

}

```

3. 将所有的粒子随机分布在圆圈轨道上.

```javascript

void RandomlySpread()

{

for (int i = 0; i < count; ++i)

{

// 使用高斯分布生成半径， 均值为midRadius，标准差为0.7

float midRadius = (maxRadius + minRadius) / 2;

float radius = (float)normalGenerator.NextGaussian(midRadius, 0.7);

float angle = Random.Range(0.0f, 360.0f);

float theta = angle / 180 \* Mathf.PI;

float time = Random.Range(0.0f, 360.0f); // 给粒子生成一个随机的初始进度

float radiusChange = Random.Range(0.0f, maxRadiusChange); // 随机生成一个轨道变化大小

circle[i] = new CirclePosition(radius, angle, time);

particleArr[i].position = new Vector3(circle[i].radius \* Mathf.Cos(theta), 0f, circle[i].radius \* Mathf.Sin(theta));

}

particleSys.SetParticles(particleArr, particleArr.Length);

}

```

4. 在Update中实现粒子得旋转。

```javascript

// Update is called once per frame

private int tier = 12; // 速度层数

void Update()

{

for (int i = 0; i < count; i++)

{

if (clockwise) // 顺时针旋转

circle[i].angle -= (i % tier + 1) \* (speed / circle[i].radius / tier);

else // 逆时针旋转

circle[i].angle += (i % tier + 1) \* (speed / circle[i].radius / tier);

// 保证angle在0~360度

circle[i].angle = (360.0f + circle[i].angle) % 360.0f;

float theta = circle[i].angle / 180 \* Mathf.PI;

//粒子在XZ平面上以半径值转圈

particleArr[i].position = new Vector3(circle[i].radius \* Mathf.Cos(theta), 0f, circle[i].radius \* Mathf.Sin(theta));

particleArr[i].startColor = startColor;

// 粒子在半径方向上游离

circle[i].time += Time.deltaTime;

circle[i].radius += Mathf.PingPong(circle[i].time / minRadius / maxRadius, maxRadiusChange) - maxRadiusChange / 2.0f;

}

particleSys.SetParticles(particleArr, particleArr.Length);

}

```

## 结果

