# 20.8 雨景

我们还实现了一个雨景粒子系统。雨景粒子系统的行为定义在rain.fx文件中。它的实现思路与fire.fx相似，只是销毁、生成和渲染规则不同。例如，雨点粒子会沿着稍微倾斜的角度加速下落，而火焰粒子会加速上升。而且，雨点粒子会被扩展为直线而不是四边形（参见图20.6）。雨点粒子在高于摄像机的随机位置上发射；“下雨”的范围总是在摄像机的视域范围之内，这样我们就不必在整个世界空间中发射雨点粒子了。也就是，只要在靠近摄像机的位置上发射雨点粒子就完全可以表现出下雨的感觉了。注意，雨景粒子系统不使用混合技术。

****

**图20.6 由粒子系统演示程序生成的雨景。**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GLOBALS \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

cbuffer cbPerFrame

{

float3 gEyePosW;

// for when the emit position/direction is varying

float3 gEmitPosW;

float3 gEmitDirW;

float gGameTime;

float gTimeStep;

float4x4 gViewProj;

};

cbuffer cbFixed

{

// Net constant acceleration used to accerlate the particles.

float3 gAccelW = {-1.0f, -9.8f, 0.0f};

};

// Array of textures for texturing the particles.

Texture2DArray gTexArray;

// Random texture used to generate random numbers in shaders.

Texture1D gRandomTex;

SamplerState samLinear

{

Filter = MIN\_MAG\_MIP\_LINEAR;

AddressU = WRAP;

AddressV = WRAP;

};

DepthStencilState DisableDepth

{

DepthEnable = FALSE;

DepthWriteMask = ZERO;

};

DepthStencilState NoDepthWrites

{

DepthEnable = TRUE;

DepthWriteMask = ZERO;

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// HELPER FUNCTIONS \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

float3 RandUnitVec3(float offset)

{

// 使用游戏时间在采样的随机纹理之上添加一个偏移.

float u = (gGameTime + offset);

// coordinates in [-1,1]

float3 v = gRandomTex.SampleLevel(samLinear, u, 0).xyz;

// 规范化到一个单位球上

return normalize(v);

}

float3 RandVec3(float offset)

{

// Use game time plus offset to sample random texture.

float u = (gGameTime + offset);

// coordinates in [-1,1]

float3 v = gRandomTex.SampleLevel(samLinear, u, 0).xyz;

return v;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// STREAM-OUT TECH \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define PT\_EMITTER 0

#define PT\_FLARE 1

struct Particle

{

float3 InitialPosW : POSITION;

float3 InitialVelW : VELOCITY;

float2 SizeW : SIZE;

float Age : AGE;

uint Type : TYPE;

};

Particle StreamOutVS(Particle vin)

{

return vin;

}

// The stream-out GS is just responsible for emitting

// new particles and destroying old particles. The logic

// programed here will generally vary from particle system

// to particle system, as the destroy/spawn rules will be

// different.

[maxvertexcount(6)]

void StreamOutGS(point Particle gin[1],

inout PointStream<Particle> ptStream)

{

gin[0].Age += gTimeStep;

if( gin[0].Type == PT\_EMITTER )

{

// time to emit a new particle?

if( gin[0].Age > 0.002f )

{

for(int i = 0; i < 5; ++i)

{

// Spread rain drops out above the camera.

float3 vRandom = 35.0f\*RandVec3((float)i/5.0f);

vRandom.y = 20.0f;

Particle p;

p.InitialPosW = gEmitPosW.xyz + vRandom;

p.InitialVelW = float3(0.0f, 0.0f, 0.0f);

p.SizeW = float2(1.0f, 1.0f);

p.Age = 0.0f;

p.Type = PT\_FLARE;

ptStream.Append(p);

}

// reset the time to emit

gin[0].Age = 0.0f;

}

// always keep emitters

ptStream.Append(gin[0]);

}

else

{

// Specify conditions to keep particle; this may vary from system to system.

if( gin[0].Age <= 3.0f )

ptStream.Append(gin[0]);

}

}

GeometryShader gsStreamOut = ConstructGSWithSO(

CompileShader( gs\_5\_0, StreamOutGS() ),

"POSITION.xyz; VELOCITY.xyz; SIZE.xy; AGE.x; TYPE.x" );

technique11 StreamOutTech

{

pass P0

{

SetVertexShader( CompileShader( vs\_5\_0, StreamOutVS() ) );

SetGeometryShader( gsStreamOut );

// disable pixel shader for stream-out only

SetPixelShader(NULL);

// we must also disable the depth buffer for stream-out only

SetDepthStencilState( DisableDepth, 0 );

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DRAW TECH \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct VertexOut

{

float3 PosW : POSITION;

uint Type : TYPE;

};

VertexOut DrawVS(Particle vin)

{

VertexOut vout;

float t = vin.Age;

// constant acceleration equation

vout.PosW = 0.5f\*t\*t\*gAccelW + t\*vin.InitialVelW + vin.InitialPosW;

vout.Type = vin.Type;

return vout;

}

struct GeoOut

{

float4 PosH : SV\_Position;

float2 Tex : TEXCOORD;

};

// DrawGS将点扩展成一条线.

[maxvertexcount(2)]

void DrawGS(point VertexOut gin[1],

inout LineStream<GeoOut> lineStream)

{

// 不绘制发射器粒子.

if( gin[0].Type != PT\_EMITTER )

{

// Slant line in acceleration direction.

float3 p0 = gin[0].PosW;

float3 p1 = gin[0].PosW + 0.07f\*gAccelW;

GeoOut v0;

v0.PosH = mul(float4(p0, 1.0f), gViewProj);

v0.Tex = float2(0.0f, 0.0f);

lineStream.Append(v0);

GeoOut v1;

v1.PosH = mul(float4(p1, 1.0f), gViewProj);

v1.Tex = float2(1.0f, 1.0f);

lineStream.Append(v1);

}

}

float4 DrawPS(GeoOut pin) : SV\_TARGET

{

return gTexArray.Sample(samLinear, float3(pin.Tex, 0));

}

technique11 DrawTech

{

pass P0

{

SetVertexShader( CompileShader( vs\_5\_0, DrawVS() ) );

SetGeometryShader( CompileShader( gs\_5\_0, DrawGS() ) );

SetPixelShader( CompileShader( ps\_5\_0, DrawPS() ) );

SetDepthStencilState( NoDepthWrites, 0 );

}

}