# 20.3 随机性

在粒子系统中，我们希望粒子的行为相似，但又不完全相同；换句话说，我们希望为粒子系统加入一些随机性。例如，当模拟雨景时，我们不希望所有的雨点都按照完全相同的方式下落；我们希望它们从不同的位置下落，而且角度和速度都稍微有一点儿不同。为了实现粒子系统的随机性，我们在*MathHelper.h/.cpp*文件中定义了**RandF**和**RandUnitVec3**函数：

// 返回[0, 1)区间的浮点随机数.

static float RandF()

{

 return (float)(rand()) / (float)RAND\_MAX;

}

// 返回[a, b)区间的浮点随机数.

static float RandF(float a, float b)

{

 return a + RandF()\*(b-a);

}

XMVECTOR MathHelper::RandUnitVec3()

{

 XMVECTOR One = XMVectorSet(1.0f, 1.0f, 1.0f, 1.0f);

 XMVECTOR Zero = XMVectorZero();

 // 持续尝试直到获得一个半球体上或之内的点.

 while(true)

 {

 // 生成一个位于立方体[-1,1]^3之内的随机点.

 XMVECTOR v = XMVectorSet(MathHelper::RandF(-1.0f, 1.0f), MathHelper::RandF(-1.0f, 1.0f), MathHelper::RandF(-1.0f, 1.0f), 0.0f);

 // Ignore points outside the unit sphere in order to get an even distribution

 // over the unit sphere. Otherwise points will clump more on the sphere near

 // the corners of the cube.

 if( XMVector3Greater( XMVector3LengthSq(v), One) )

 continue;

 return XMVector3Normalize(v);

 }

}

上面的函数用于C++代码，但是我们还需要着色器代码中的随机数。在着色器代码中生成随机数是一件比较棘手的事情，因为HLSL没有随机数生成器。所以我们要做的是：创建一幅带有4个浮点分量的1D纹理（**DXGI\_FORMAT\_R32G32B32A32\_FLOAT**）。然后使用随机4D向量填充该纹理，这些向量的每个分量都在[1,−1]区间内。该纹理使用重复（wrap）寻址模式进行采样，这样我们便可以使用超出[0,1]区间的无限制的纹理坐标了。着色器代码只要对该纹理进行采样就可以获得随机数。对随机纹理进行采样的方式很多。如果每个粒子都有一个不同的*x*坐标，那么我们可以考虑把*x*坐标作为纹理坐标来获取随机数。不过，当很多粒子具有相同的*x*坐标时，这种方式就失效了，因为它们都对纹理中的同一个元素进行采样，随机性已不复存在。另一种方式是把当前的游戏时间作为纹理坐标。通过这一方式，在不同时间生成的粒子可以得到不同的随机数。不过，这意味着在同一时间生成的粒子会得到相同的随机数。如果粒子系统一次发射多个粒子，那么这种方式会出现问题。当同时生成多个粒子时，我们可以给游戏时间添加一个纹理坐标偏移值，使粒子对纹理贴图上的不同的点进行采样，进而获取不同的随机数。例如，我们循环20次，生成20个粒子，那么可以把循环索引（在适当的调整之后）作为纹理坐标偏移值。通过一方式，我们可以获得20个不同的随机数。

下面的代码示范了如何生成随机纹理：

ID3D11ShaderResourceView\* d3dHelper::CreateRandomTexture1DSRV(ID3D11Device\* device)

{

 //

 // 创建随机数据.

 //

 XMFLOAT4 randomValues[1024];

 for(int i = 0; i < 1024; ++i)

 {

 randomValues[i].x = MathHelper::RandF(-1.0f, 1.0f);

 randomValues[i].y = MathHelper::RandF(-1.0f, 1.0f);

 randomValues[i].z = MathHelper::RandF(-1.0f, 1.0f);

 randomValues[i].w = MathHelper::RandF(-1.0f, 1.0f);

 }

 D3D11\_SUBRESOURCE\_DATA initData;

 initData.pSysMem = randomValues;

 initData.SysMemPitch = 1024\*sizeof(XMFLOAT4);

 initData.SysMemSlicePitch = 0;

 //

 // 创建纹理.

 //

 D3D11\_TEXTURE1D\_DESC texDesc;

 texDesc.Width = 1024;

 texDesc.MipLevels = 1;

 texDesc.Format = DXGI\_FORMAT\_R32G32B32A32\_FLOAT;

 texDesc.Usage = D3D11\_USAGE\_IMMUTABLE;

 texDesc.BindFlags = D3D11\_BIND\_SHADER\_RESOURCE;

 texDesc.CPUAccessFlags = 0;

 texDesc.MiscFlags = 0;

 texDesc.ArraySize = 1;

 ID3D11Texture1D\* randomTex = 0;

 HR(device->CreateTexture1D(&texDesc, &initData, &randomTex));

 //

 // 创建资源视图.

 //

 D3D11\_SHADER\_RESOURCE\_VIEW\_DESC viewDesc;

 viewDesc.Format = texDesc.Format;

 viewDesc.ViewDimension = D3D11\_SRV\_DIMENSION\_TEXTURE1D;

 viewDesc.Texture1D.MipLevels = texDesc.MipLevels;

 viewDesc.Texture1D.MostDetailedMip = 0;

 ID3D11ShaderResourceView\* randomTexSRV = 0;

 HR(device->CreateShaderResourceView(randomTex, &viewDesc, &randomTexSRV));

 ReleaseCOM(randomTex);

 return randomTexSRV;

}

注意，随机纹理只需要一个多级渐近纹理层。我们使用**SampleLevel**内置函数对纹理仅有的一个多级渐近纹理进行采样。该函数允许我们明确地指定所要采样的多级渐近纹理层。它的第1个参数是采样器对象，第2个参数是纹理坐标（1D纹理只有一个纹理坐标），第3个参数是多级渐近纹理层（当纹理只包含一个多级渐近纹理层时，该参数应设为0）。

下面的着色器函数用于获取一个在单位球体上的随机向量：

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);

}