# 原理

粒子系统：一系列小图片组合成的复杂模糊的对象，如火焰、烟雾、云或者烟花。组成粒子系统的粒子具有自己的属性（位置，速度，生命周期等），通过这些属性控制粒子系统的效果。

var particleSystem = viewer.scene.primitives.add(new Cesium.ParticleSystem({

// Particle appearance

image : '../../SampleData/fire.png',

width : 20,

height : 20,

startScale : 1.0,

endScale : 4.0,

// Particle behavior

life : 1.0,

speed : 5.0,

// Emitter parameters

emitter : new Cesium.CircleEmitter(0.5),

rate : 5.0,

emitterModelMatrix : computeEmitterModelMatrix(),

// Particle system parameters

modelMatrix : computeModelMatrix(),

lifetime : 16.0

}));

粒子由粒子发射器（ParticleEmitter）生成，持续一段时间，然后消亡。

Cesium粒子系统的参数由单个例子的参数，发射器的类型和摆放位置，以及粒子系统的生命周期和摆放位置决定。

有的属性是动态的，比如颜色。这里需要设置两个颜色：startColor和endColor。分别代表刚产生时和消亡时粒子的颜色。中间的颜色是根据两个颜色插值生成的。startScale和endScale的原理相同。剩下的参数都是静态的。

# 粒子发射器

粒子生成以后，它的起始位置和速度向量是由发射器决定的。发射器每秒会喷射一定数量的粒子，这个生成速率由rate参数决定。

Cesium提供几个类型的发射器。

## BoxEmitter

BoxEmitter从长方体的六个面随机发射粒子。参数为一个 Cartesian3，定义长方体的长宽高

## CircleEmitter

CircleEmitter从一个圆形面向圆的上方向发射粒子。参数为radius，圆的半径。

## ConeEmitter

ConeEmitter从一个圆锥的顶点按照一定的角度范围发射粒子。

## SphereEmitter

SphereEmitter由球心向外发射粒子。参数为radius，球的半径。

# 配置粒子系统

Cesium提供一系列选项控制粒子的行为

## 粒子发射速率

属性rate代表每一秒有多少个粒子发射，用于控制粒子在系统中的密度。

你也可以用定义burst，让粒子按照一定时间和数量发射。

bursts : [

new Cesium.ParticleBurst({time : 5.0, minimum : 300, maximum : 500}),

new Cesium.ParticleBurst({time : 10.0, minimum : 50, maximum : 100}),

new Cesium.ParticleBurst({time : 15.0, minimum : 200, maximum : 300})

],

这些burst会在给定时间发射，数量介于minmum和maximum之间。

## 生命周期

有一些属性控制粒子系统的生命周期。默认情况下，粒子系统会永久的运行下去。

如果需要粒子系统在特定的时间段内运行，就设置lifeTime为时间长度，loop设置为false

particleSystem : {

lifeTime: 5.0,

loop: false

}

也可以设置最大最小时间，让粒子随机存在于两个时间之间

particleSystem : {

minimumLife: 5.0,

maximumLife: 10.0

}

## 粒子的样式

### 颜色

除了一开始设置的基础图片属性，粒子还可以在生命周期内变化不同的颜色。通过设置startColor和endColor实现

particleSystem : {

startColor: Cesium.Color.RED.withAlpha(0.7),

endColor: Cesium.Color.YELLOW.withAlpha(0.3)

}

### 大小

粒子的大小由最大最小宽高minimumWidth, maximumWidth, minimumHeight和maximumHeight属性决定。每一个粒子的宽和高（以像素为单位），都介于最大最小宽高之间。

particleSystem : {

minimumWidth: 30.0,

maximumWidth: 60.0,

minimumHeight: 30.0,

maximumHeight: 60.0

}

大小也可以和颜色一样随时间渐变，通过设置startScale和endScale属性来实现：

particleSystem : {

startScale: 1.0,

endScale: 4.0

}

### 速度

发射器控制粒子的起始位置和方向，粒子的速度由minimumSpeed和maximumSpeed控制。

particleSystem : {

minimumSpeed: 5.0,

maximumSpeed: 10.0

}

### 力场

重力和风力这样的力场也是可以应用在粒子系统中的。

每个粒子系统中都可以有一个力场的数组，存放多个力场回调函数，用于在运行时修改粒子的属性。力场回调函数是接收两个输入参数，一个是粒子，还有一个是模拟时间间隔。大多数基于物理的效果会修改速度向量（以修改粒子的方向和速度）。以下代码创建了一个重力场：

var gravityScratch = new Cesium.Cartesian3();

function applyGravity(p, dt) {

// Compute a local up vector for each particle in geocentric space.

var position = p.position;

Cesium.Cartesian3.normalize(position, gravityScratch);

Cesium.Cartesian3.multiplyByScalar(gravityScratch, viewModel.gravity \* dt, gravityScratch);

p.velocity = Cesium.Cartesian3.add(p.velocity, gravityScratch, p.velocity);

}

该函数计算了重力速度向量，然后用重力加速度（-9.8m/s2）修改粒子的速度。

给粒子系统添加力场数组

particleSystem: {

forces: [applyGravity]

}

### 位置

粒子系统用两个4\*4的矩阵设置位置

modelMatrix：将粒子系统从模型坐标系转换成世界坐标系。

emitterModelMatrix：在粒子系统的本地坐标系变换发射器位置的矩阵

你可以只使用两者中的一个，另一个默认为单位矩阵。为了方便起见，两个矩阵我们都提供了。为了练习矩阵操作，下面我们要做一个练习，将粒子发射器放置于另一个entity旁边。

首先要创建一个entity，打开Hello World的Sandcastle实例，添加一下代码：

var entity = viewer.entities.add({

// Load the Cesium plane model to represent the entity

model : {

uri : '../../SampleData/models/CesiumAir/Cesium\_Air.gltf',

minimumPixelSize : 64

},

position : Cesium.Cartesian3.fromDegrees(-112.110693, 36.0994841, 1000.0)

});

viewer.trackedEntity = entity;

上面的代码给场景添加了一个飞机模型，然后我们要把一个火焰效果的粒子系统放置在飞机的一个引擎上。为此，我们要创建一个模型矩阵，将飞机的位置和朝向复制出来。下面的代码可以获取飞机模型的矩阵：

function computeModelMatrix(entity, time) {

var position = Cesium.Property.getValueOrUndefined(entity.position, time, new Cesium.Cartesian3());

if (!Cesium.defined(position)) {

return undefined;

}

var orientation = Cesium.Property.getValueOrUndefined(entity.orientation, time, new Cesium.Quaternion());

if (!Cesium.defined(orientation)) {

var modelMatrix = Cesium.Transforms.eastNorthUpToFixedFrame(position, undefined, new Cesium.Matrix4());

} else {

modelMatrix = Cesium.Matrix4.fromRotationTranslation(Cesium.Matrix3.fromQuaternion(orientation, new Cesium.Matrix3()), position, new Cesium.Matrix4());

}

return modelMatrix;

}

上面函数返回的矩阵可以将粒子系统放置在飞机模型的中心点。如果我们想要将粒子系统放置在飞机的一个引擎上，那么就需要使用EmitterModelMatrix。在模型的坐标系统中，创建一个相对于中心点的模型矩阵，配置旋转和平移参数。代码如下：

function computeEmitterModelMatrix() {

hpr = Cesium.HeadingPitchRoll.fromDegrees(0.0, 0.0, 0.0, new Cesium.HeadingPitchRoll());

var trs = new Cesium.TranslationRotationScale();

trs.translation = Cesium.Cartesian3.fromElements(2.5, 4.0, 1.0, new Cesium.Cartesian3());

trs.rotation = Cesium.Quaternion.fromHeadingPitchRoll(hpr, new Cesium.Quaternion());

return Cesium.Matrix4.fromTranslationRotationScale(trs, new Cesium.Matrix4());

}

矩阵计算完成后，就可以设置粒子系统的参数了：

var particleSystem = viewer.scene.primitives.add(new Cesium.ParticleSystem({

image : '../../SampleData/fire.png',

startScale : 1.0,

endScale : 4.0,

life : 1.0,

speed : 5.0,

width : 20,

height : 20,

rate : 5.0,

lifeTime : 16.0,

modelMatrix : computeModelMatrix(entity, Cesium.JulianDate.now()),

emitterModelMatrix : computeEmitterModelMatrix()

}));

效果如下：



粒子系统和粒子发射器的模型矩阵都可以随着时间改变。比如，我们想要动态移动发射器的位置，就可以只修改emitterModelMatrix，不修改modelMatrix。

这就是粒子系统的基本原理和操作