**第六章：粒子特效**

**绚丽的火焰与爆炸**

本章节主要介绍粒子特效设计的方法论，其中有相当的知识量是平台无关的；在本文中会以“爆炸”这个实际的例子为线索，进行详细的设计讲解，并最终使用GEiv实现它。

[为什么要使用”粒子”]

实现粒子特效的首要目的，是对一些环境效果进行模拟仿真，常见的环境效果，例如火焰、爆炸、雨、雪、雾等，都是无数微小的粒子以某些规律共同作用的结果。而对于计算机来讲，虽然没有足够的运算能力对每一个自然粒子进行抽象，但我们可以借鉴其原理，使用相对更少的粒子对这些自然现象进行模拟和仿真，以达到近似的效果。

[需要设计哪些内容]

[粒子属性]

首先需要设计的是单个粒子的属性，这里我们以粒子个体作为考虑的焦点，考虑的内容往往是粒子的共有属性，属性的内容可以是图形样式、大小、颜色等等。

[投射规律]

投射规律考虑粒子以何种方式投射到屏幕上，这里以粒子群为考虑的焦点，考虑的内容会涉及到实际的物理规律，例如粒子在空间中的角度分布、速度分布以及颗粒大小分布等情况。

[演变规律]

演变规律是抛射后的粒子随着时间变化的规律，它同样会涉及到物理规律的模拟，只不过这次是针对单个粒子的设计，例如速度、自旋角度、颜色等属性的变化规律。

[实例：爆炸]

爆炸特效在游戏中的使用相当广泛，属于经典的粒子系统。现在我们从零开始，设计一个爆炸的粒子特效。

在想要模拟爆炸前先来观察一个实际的爆炸例子：



从图片中我们能够概况一些基本的物理规律：

首先，在一个爆炸中，粒子的大小显然是不同的，而且，简单的想，粒子的大小与其质量成正比，所以粒子速度应该与其大小负相关，你可以看到颗粒状的小型碎片已经飞到了火焰之外的区域，这是动量守恒定律所确定的。

其次，在爆炸的中心，能量较高，呈现出亮白色；而在爆炸的外围，与空气接触后热量明显下降，火焰呈现出暗红色，在这个过程中，颜色也呈现出了明显的变化规律：亮白-》黄色-》红色-》暗红。

还有，速度的变化规律：在爆炸发生后，粒子的速度并不会一直不变，它还要受到空气阻力的作用，根据流体力学的相关内容，空气阻力与速度的平方成正比，与物体在运动方向的正投影面积成正比，所以其速度变化应该表现为某种受到阻尼的运动状态。

最后，在能量耗尽的暗红色区域，粒子逐渐消失，也就是说其颜色通道系数应该以某种非线性（先慢后快）的方式衰减。

[属性设计]

粒子图元：首先需要确定的问题，我们如何选择粒子的图形呢，使用点？圆形？方块？还是使用某种贴图呢……其实设计粒子的基本形态很值得一说，我们暂且使用圆形来设计，在最后您可以看到更改粒子形态对整体特效的影响。

粒子的颜色：由白到红，初始值使用白色。

粒子的大小：为了较为明确的产生大小两种粒子，我将使用一定的概率分布策略随机产生大小（详见投射设计部分）。

自旋角度：在圆周上均匀分布，由于一开始我们使用圆形作为图元，所以这个自旋这个属性不会显露出来。

通道：Alph初始值设置为1.0。

[投射设计]

产生：在我们给定爆炸点之后，假定粒子围绕着给定点进行+/-5位置浮动的随机的产生。

大小分布：以50%的概率产生6~35大小的粒子，否则产生6~24大小的粒子，这里只是一个简单方案，你也可以考虑使用高斯分布等。

速度方向分布：以产生点进行360度均匀分布。

速度大小分布：为了简化选择了恒定值，但是，空气阻力模型在演变中起到作用，故仍可观察到非常近似地模拟结果。

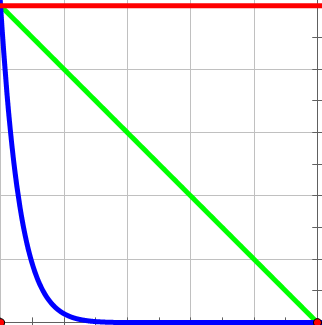
[演变设计]

速度衰减：

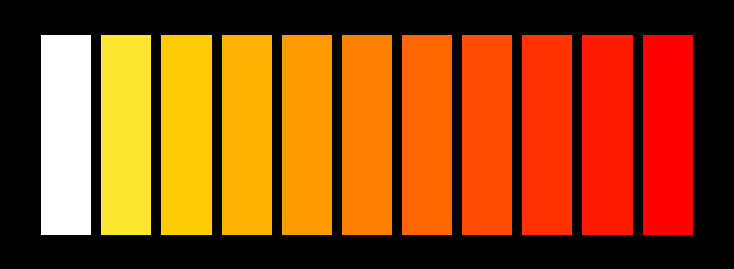


对于每一帧：v -= a\*w^2\*v;其中，w是粒子大小，a是衰减系数，v是当前速度，也就是说，速度进行阻尼衰减，并且大碎片的速度衰减的更快。

颜色衰减：



↑衰减时间图



↑衰减过程均匀抽样

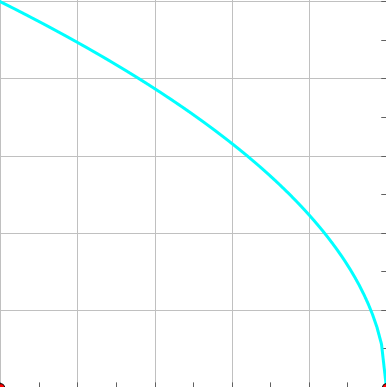
首先，RGB中的红色分量是不变的。

假如把时间t变量规格化到0~1之间。

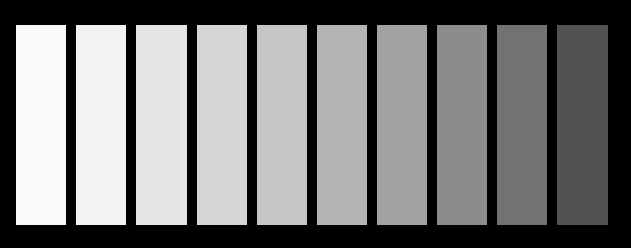
那么，蓝色分量应该最快衰减，因为爆炸主色调至少应该是一个暖色调。所以蓝色线使用的是t^16。

绿色分量暂时设置为伴随t的线性衰减，其实，G分量衰减速度应该依据大小而定，这里可以参考粒子特效的改进部分。

通道衰减：



↑衰减时间图



↑衰减过程均匀抽样

通道衰减过程先慢后快，这样，在特效开始的一段时间内，我们不会感到通道的变化，直到粒子快要消亡时才会有直观的视觉感受。

[编码实现]

接下来就是编码阶段了，我们也明确的看到，其实整个粒子特效的实现过程中，设计占了相当大的比例，在最后的阶段，只不过是要我们使用擅长的平台去实现罢了，其实很多软件开发都是这样的，编码只是个实现过程，不是什么高科技。

您可以到GitHub上找到本章中的例子。

在Geiv下，我们的粒子仅需要实现Individual接口，并使用个体的集群管理器进行管理即可（参阅第五章）。

ExpIndividual.java:

package com.geiv.test;

import engineextend.crowdcontroller.Individual;

import geivcore.UESI;

import geivcore.enginedata.obj.Obj;

import java.awt.Color;

import com.thrblock.util.REPR;

import com.thrblock.util.RandomSet;

public class ExpIndividual implements Individual {

UESI UES;

Obj Disp;

float STallms = 500;//这里设置了粒子从产生到消亡的总经历时间

float allms = STallms;

float Dms = 17;//这里设置了每一帧的时间，你也可以用1000/UES.getFPS这中方法在构造器里填充

float V = 4.5f;//运动的初始速度被固定为4.5像素每帧

float ax, ay;

float vx, vy;

float Theta;//自选角度，本例中暂时使用圆形，所以是看不出的

public ExpIndividual(UESI UES) {

this.UES = UES;

Disp = UES.creatObj(UESI.XRIndex);//这里把图元产生在了XR层，前面的章节中介绍了该层次混合模式的特点。

Disp.addGLOval("FFFFFF",0,0,12,12,12);//画一个圆形

Disp.setGLFill(true);

Disp.setColor("FFFFFF");

Disp.setAlph(Disp.getTopDivIndex(), 1.0f);

allms = STallms;

}

@Override

public boolean isAvalible() {

return !Disp.isPrintable();//关于Individual请参考第五章的介绍

}

@Override

public void getUse(Object[] ARGS, float... FARGS) {

int Rad;//我们使用一定的分布方法产生Rad大小，RandomSet是内置的随机数发生器，其静态方法名称都比较好理解，就不在这里细细讲解了。

if (RandomSet.getRate(50)) {//以50%的概率返回布尔值true

Rad = RandomSet.getRandomNum(6, 35);//返回6~35随机数，均匀分布。

} else {

Rad = RandomSet.getRandomNum(6, 24);

}

Disp.setWidth(Rad);

Disp.setHeight(Rad);

//初始位置具有+/-5的浮动区域

Disp.setCentralX(FARGS[0] + RandomSet.getRandomNum(-5, 5));

Disp.setCentralY(FARGS[1] + RandomSet.getRandomNum(-5, 5));

//初始自选角度，0~360均匀分布。

Disp.setAngle(RandomSet.getRandomFloatIn\_1() \* 360);

//速度角，0~2PI均匀分布，使用弧度是为了方便调用Math下的三角函数。

Theta = (float) Math.PI \* 2 \* RandomSet.getRandomFloatIn\_1();

vx = V \* (float) Math.sin(Theta);//计算横纵向速度

vy = -V \* (float) Math.cos(Theta);

ax = -0.0003f \* (Disp.getWidth() \* Disp.getWidth()) \* vx;//计算加速度

ay = -0.0003f \* (Disp.getHeight() \* Disp.getHeight()) \* vy;

Disp.show();//显示到屏幕上（投射完成）

}

@Override

public void doStp(int clock) {

if (this.allms > Dms) {

allms -= Dms;//allms记录当前剩余存活期，使用这个变量是为了将存活期规格化到0~1之间。

//REPR是内置的变换工具，可以将一个规格化后的线性量转化为自定义的常用非线性量

//颜色变化

Disp.setColor(new Color(1.0f,

REPR.Rep\_POW\_1\_F(allms / STallms, Disp.getWidth() / 24), REPR.Rep\_POW\_F(allms / STallms, 16)));

//通道变化

Disp.setAlph(REPR.Rep\_POW\_1\_F(allms / STallms, Disp.getWidth() / 12));

//运算加速度

ax = -0.0003f \* (Disp.getWidth() \* Disp.getWidth()) \* vx;

ay = -0.0003f \* (Disp.getHeight() \* Disp.getHeight()) \* vy;

//运算速度

vx += ax;

vy += ay;

//运算位置

Disp.setDx(Disp.getDx() + vx);

Disp.setDy(Disp.getDy() + vy);

} else {

//声明周期结束后，将粒子资源回收

finish(Individual.SRC\_INNER);

}

}

@Override

public void finish(int src) {

Disp.hide();

//重置颜色与通道

Disp.setColor("FFFFFF");

Disp.setAlph(Disp.getTopDivIndex(), 1.0f);

//重置大小

Disp.setWidth(12);

Disp.setHeight(12);

//重置存活时间。

allms = STallms;

}

@Override

public void destroy() {

Disp.destroy();

}

}

Explosion.java:

package com.geiv.test;

import engineextend.crowdcontroller.CrowdController;

import geivcore.UESI;

public class Explosion{

UESI UES;

CrowdController cc;

public Explosion(UESI UES)

{

this.UES = UES;

cc = new CrowdController(UES, true);

for(int i = 0;i < 512;i++)//装入了512个粒子资源

{

cc.addIndividual(new ExpIndividual(UES));

}

}

public void doEffect(float dx,float dy) {

for(int i = 0;i < 128;i++)//当每次调用时，分配128个粒子资源，同时也意味着，您可以同时在屏幕上产生4个异步的爆炸特效。

{

cc.getAvailible().getUse(null,dx,dy);

}

}

public void forceClose() {

cc.finishAllInd();

}

}

Main.java:

package com.geiv.test;

import geivcore.R;

import geivcore.UESI;

public class Main{

public static void main(String[] args) {

UESI UES = new R();

Explosion exp = new Explosion(UES);

for(;;){

exp.doEffect(400,300); //产生一个爆炸

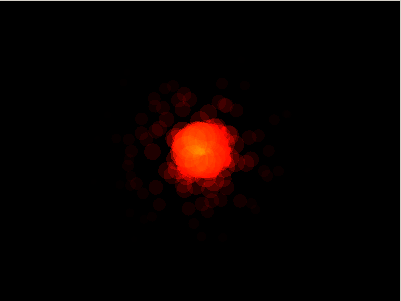
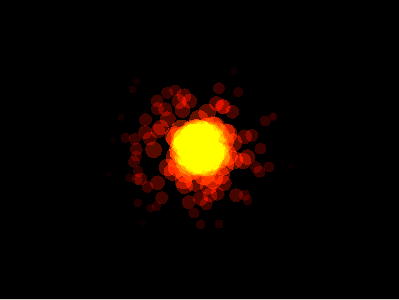
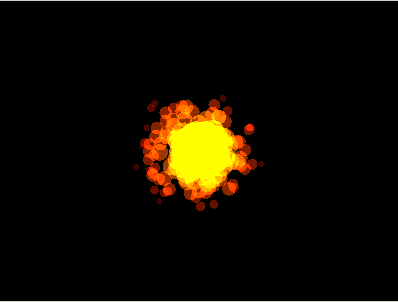
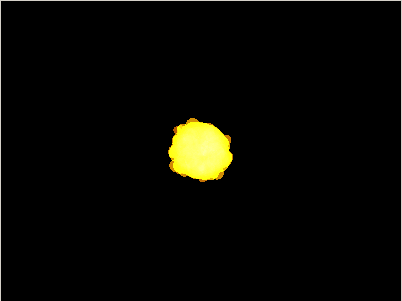
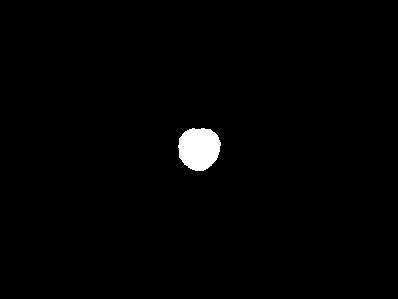
UES.wait(3,1000); //延时1秒

}

}

}

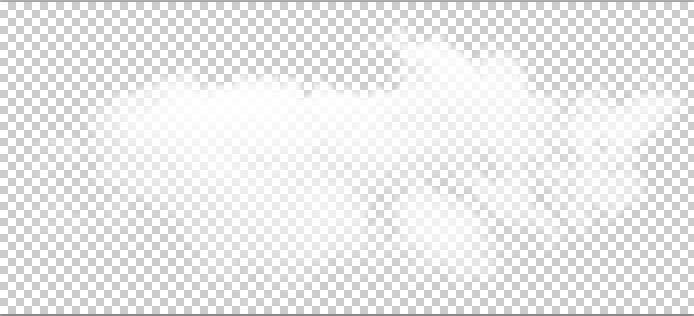
执行效果：



[粒子特效的改进]

“一堆圆形一点儿也不像嘛”这是我同学看到程序后的第一句评价，的确，从粒子的行为模式上来讲，是有类似爆炸的性质了，不过一个爆炸也不能只让圆形组成不是吗？

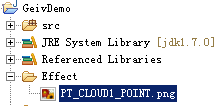
在属性设计时，我提到了关于粒子图元选择的问题，对于爆炸这个特效，显然均匀的圆形（或者其他图形）不是一种好的图元构成，我们需要一个形状并不均匀，甚至伴有随机性的图形来替换这个圆，于是笔者想到了“云”这个东西。



↑由于云是白色的，所以为了展示，把PS的衬底一起截下来了。

云本来是与爆炸毫不相干的东西，选择它是由它的图形性质决定的：边缘渐变、具有随机性、在颜色通道上也不均匀。而且，加上我们之前定义的自旋随机分布，加入自旋角的云看起来和彼此具有更大的差异。

为了使用云这个素材，先把它放在项目目录里：



之后找到ExpIndividual类，找到它的图元绘制部分：

我们把

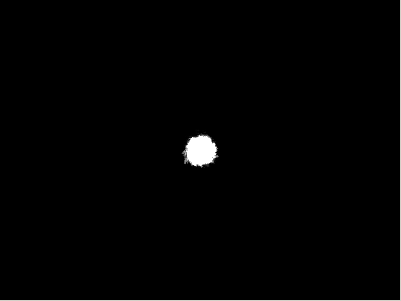
disp.addGLOval("FFFFFF",0,0,12,12,12);

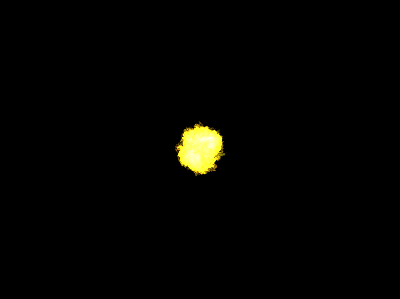
disp.setGLFill(true);

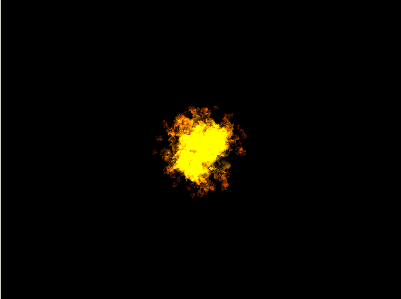
这两行改为：

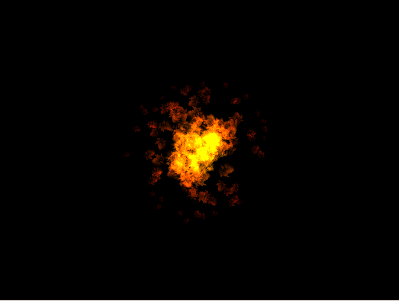
disp.addGLImage(0, 0, 12, 12, ".\\Effect\\PT\_CLOUD1\_POINT.png");

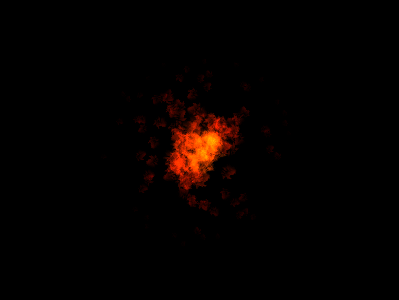
经过改进的特效：

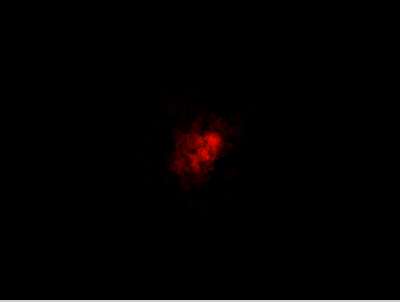












[总结]

本章介绍了粒子特效设计的基本步骤，即属性、投射、演化三部分。

粒子特效是对自然的模拟，因此在设计时要充分地考虑到物理因素，这样会得到更好的仿真结果。

最后，恰当地选择粒子图元可以得到更好的结果，而粒子图元的选择与图形的性质有关。