**第二章：动态绘制**

**让画面动起来！**

这篇文章主要介绍图元动画的基本实现。

[画面、动画与游戏]

在上一章节中，主要讲解了静态绘制的诸多方法。这些API终究是对OPENGL API的简单封装，这离游戏制作尚有一段距离。

什么是动画呢？动画是一系列画面的有序播放，当播放速度快于人眼视觉暂留时，我们便误以为画面中的物体在移动了，对于视频、动画来讲，这个播放速度大于24帧时，已经达到了不错的效果。

什么又是游戏呢？游戏是在玩家交互过程中变化的动画，对于计算机游戏来讲，这个交互过程主要依赖IO系统来完成。由于游戏对象根据IO的变化作出及时反应并来让玩家产生实在的控制感，24帧每秒的绘制速度虽然不会影响视觉暂留，但对于“控制感”来讲却会有一些滞后（lag），因此游戏普遍要求帧数在40以上，有些干脆同步在显示器绘制上，即所谓垂直同步。而Geiv的默认绘制速率被固定在60FPS。也就是说，前一章的静态绘制，实际上是每秒钟绘制60幅相同的画面而已。

[让画面动起来]

例子：

UESI UES = new R();

Obj rect = UES.creatObj(UESI.BGIndex);

rect.addGLRect("00FF00",0,0,200f,200f);//画一个矩形

rect.setGLFill(true);

rect.setPosition(CANExPos.POS\_Y\_CENTER); //放在Y轴中间位置

rect.show();

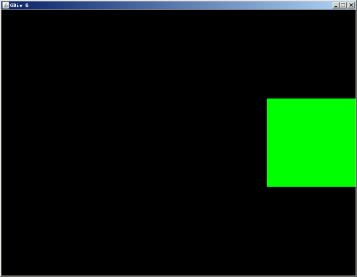
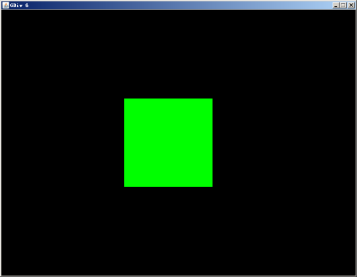
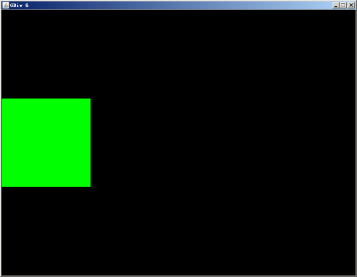
while(rect.getDx() + rect.getWidth() < UES.getScreenWidth()){

rect.setDx(rect.getDx() + 1f);//取出Dx，并加上1

UES.wait(3,17);//延时17ms，约为1/60

}

结果：



***但是上面的写法极其不科学！！！***

首先，引擎自己有自己的绘制线程，是基于Timer、及其准确的每秒钟执行60次绘制；而main方法相对于绘制来讲，是一个异步过程。延时17ms并不会准确的落在绘制线程的延时部分。

第二，线程总是有限的，在执行while循环时，整个main线程都在为这个动画提供服务，无法进行其他运算，如果此时需要5种不同的动画，为他们开辟5个新线程是及其不合理的，纵使引擎组建中有完善的同步锁保证数据安全，但我们也不应该提高无用的并发度。

[帧逻辑][interface geivcore.SerialTask]

SerialTask,字面意义是“串行任务”，这个名字起得不是很好（又是历史原因恩恩），实际的意义，是插入到绘制线程完成绘制之后的额外自定义逻辑。

例子：

//到了这里，我十分建议将类分开写，一个Main类已经满足不了需求了。

Main.java:

public class Main{

public static void main(String[] args) {

UESI UES = new R();

new MoveableRect(UES);

}

}

MoveableRect.java:

public class MoveableRect implements SerialTask{

Obj rect;

UESI UES;

public MoveableRect(UESI UES){

this.UES = UES;

rect = UES.creatObj(UESI.BGIndex);

rect.addGLRect("00FF00",0,0,200f,200f);

rect.setGLFill(true);

rect.setPosition(CANExPos.POS\_Y\_CENTER);

rect.show();

UES.addSerialTask(this);

}

@Override

public void Serial(int arg0) {

if(rect.getDx() + rect.getWidth() < UES.getScreenWidth()){

rect.setDx(rect.getDx() + 1f);

}

}

}

首先说SerialTask接口，它当中只有一个方法：

public void Serial(int arg0);

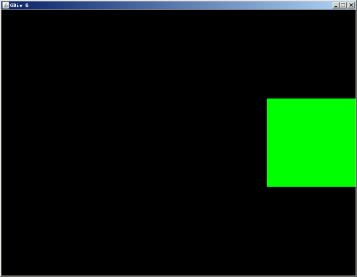
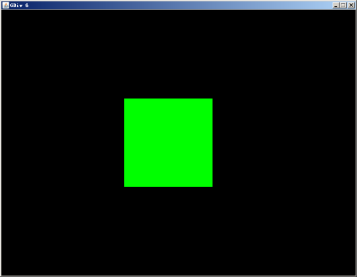
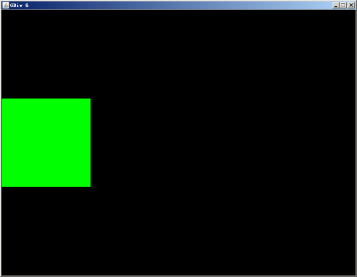
其中，参数是唯一的绘制时序信号，可用可不用。

之后MoveableRect类实现了Serial方法，在每次绘制过后，将rect的横坐标+1，也就是每秒60像素的速度。

最后，在MoveableRect的构造方法中，使用UESI为参数，并保留一个引用，在构造方法的最后部分，它调用了UES.addSerialTask(this);方法，将这个自定义逻辑（Serial的实现）插入了绘制线程的队列末尾。

这样，在每次绘制完成后，会精确的执行一次Serial方法，改变坐标，在下次绘制时图元便会移动一小段距离，依次类推，图元便移动了起来。

结果：



如果您观察仔细的话，应该能看出来，使用SerialTask实现的动画要比使用其他线程的异步调用相比平滑的多。

SerialTask的注意事项：

引擎的动态部分应该主要由SerialTask完成。SerialTask的Serial

方法中，能够书写的语句比较有限：这点非常重要，所有的Serial实现都要在绘制间隔里执行完毕（17ms），在一个游戏中，Serial的总数量可以成百上千，因此有如下限制：

1. 阻塞方法一定不要有，例如sleep，如果以前进行过awt编程，在onclick方法里调用阻塞方法会让整个绘制线程停止，这里的Serial也是一样。
2. 阻塞IO过程也一定不能有，总共17ms的执行实现，使用Serial调用阻塞IO时间来不及，诸如读取文件、网络访问等不适宜写入Serial。
3. 堆区分配请求过程不能过多，也就是说，尽量不要new对象，但是一些轻量化的数据对象还是可以接受的。

[总结]

SerialTask接口用来向绘制线程动态地添加一部分自定义运算逻辑，自定义类通过实现Serial方法，适当地进行图元变换以实现各种动态效果。

Serial尽可能只负责数值运算与类之间的消息传递，不要写费时间的东西。