《科研与工程中的C++编程》课程报告

胡泳欢 | 3180101964 | 20-Jul-2019

目录

1 设计思路

按照需求分析所确定的，我们的项目大致需要分为 ***数理计算（Common & Model）***、***绘制（View）*** 与 ***消息机制（View Model & Command）*** 这几个部分。

1.1 数理计算 (Common & Model)

Common层提供程序所要用到的基础数学物理对象。

向量（Vec）：二元浮点数对。支持相加、数乘、内积、外积、取模、取夹角、旋转等几乎所有常用的向量运算，用于科学计算。除了表示向量外，亦用来表示空间上的点，视为从原点出发的一束向量。
线段（Segment）：由两个点（向量）表示。
多边形（Poly）：由顶点的集合表示，顶点按逆时针排序。支持判断相交、求交点与交线等操作。
刚体（RigidBody）：在多边形的基础上，增加了质量、角速度、速度等物理属性。**碰撞** 在其中作为一个成员方法出现。

Model层基于Common层，参考主流的物理引擎，高度封装产生了“物理空间”PhysicsSpace类。上层架构只需要与物理空间交互即可，而不需要（也不允许）操作单独的刚体。物理空间支持**操作力场**、 **添加刚体**、 **删除刚体**、**模拟时间流逝** 。
另有一个单独的Model类作为消息接口区，负责接收ViewModel的指令并提供反馈。

各类的方法及成员可以参见/doc/static class diagram中的静态结构图。

1.2 绘制 (View)

注：我们实际上对View层发生了错误理解，使得最终做出的模型是”伪MVVM“模型。见"错误与反思"一章。

1.3 消息机制 (View Model)

在我们的设计中，各层间的通讯通过命令进行。我们将通知处理为命令的“换皮”存在，也即是自底层发向上层的命令。

Command作为一个纯虚类存在，对某一条具体的命令再将其具体实现。用户与UI发生交互后，命令由Window层传给ViewModel层，ViewModel再将向Model发送相应的命令，由Model进行对数据进行运算、更改。之后，Model向ViewModel发送反馈，再由ViewModel通知View进行绘制。数据作为命令的参数被传递，而没有使用正统MVVM的数据绑定。

注：实际编码中，ViewModel与View发生了耦合。见"错误与反思"一章。

2 分工任务的解决方案

我实际负责的是Common层与Model层的开发，以及持续集成的部署。

2.1 碰撞算法

目标：能够以$O(n^2)$（其中n为碰撞两刚体的边数中的较大者）的效率处理二维平面中两个刚体的完全弹性碰撞。

理想的物理世界里，刚体是不可形变的，一个刚体在与另一刚体接触后会立刻受到$|f| \to +\infty$的回复力，碰撞瞬间完成。现实世界里，物体在与另一物体接触后，会获得与形变程度呈正相关的回复力，发生“接触”、“形变”、“恢复”的过程。鉴于程序模拟的“时间”是离散流逝的，我们只能在检测到一个刚体插入另一刚体后，再施加一个大小足够大（但不能是无穷，否则碰撞时间$\to 0$，超出浮点数表示范围）的回复力。

表记入射刚体为a，受射刚体为b，我们拥有如下16个参量（向量视为2个参量）：

$$M_a, M_b, \overrightarrow{V_a}, \overrightarrow{V_b}, I_a, I_b, w_a, w_b, \overrightarrow{Ca}, \overrightarrow{C_b}, \vec{O}, \vec{f}$$

其中，$M, \vec{V}, I, w$是a, b的物理属性，分别表示质量、速度、转动惯量和角速度。$\vec{O}$表示两刚体的交点、$\vec{C}$表示刚体的匀质质心，由Poly类提供方法计算。$\vec{f}$是回复力，方向为接触面的法向、指向受射刚体外侧，大小为常量。

设有未知量碰撞时间dt，对a，有：

$$\Delta\overrightarrow{V_a} = \frac{\vec{f}}{m_a}dt$$

$$\Delta w_a = \frac{\vec{f} \times (\overrightarrow{C_a} - \vec{O})}{I_a}$$

其中$\times$是向量叉乘。对b，是类似的，但注意将$\vec{f}$代换为$-\vec{f}$。

能量守恒：

$$M_a|\overrightarrow{V_a}|^2 + I_a w_a^2 + M_b|\overrightarrow{V_b}|^2 + I_b w_b^2 = M_a|\overrightarrow{V_a} + \Delta\overrightarrow{V_a}|^2 + I_a w_a^2 + M_b|\overrightarrow{V_b} + \Delta\overrightarrow{V_b}|^2 + I_b w_b^2$$

其中$||$是向量取模。

解出dt是一个12项除24项构成的分式，再反代回上式，即可求出碰撞后的速度及角速度。

理论上，此时工作已经完成了。但注意到时间是离散流逝的，这会引发所有碰撞模型都要解决的***连续碰撞问题***：速度改变后，位移作为速度对时间的积分却还未改变。入射刚体还来不及离开，就会开始下一轮的碰撞检测。其结果是相碰的刚体会经历一阵诡异的振荡才能相互分离或共速。

为了解决问题，我们引入了碰撞冷却机制。两刚体间的碰撞在5个离散的时间点内不会重复计算，除非入射刚体又与其他刚体发生了碰撞。这个简单的机制很好地解决了问题。

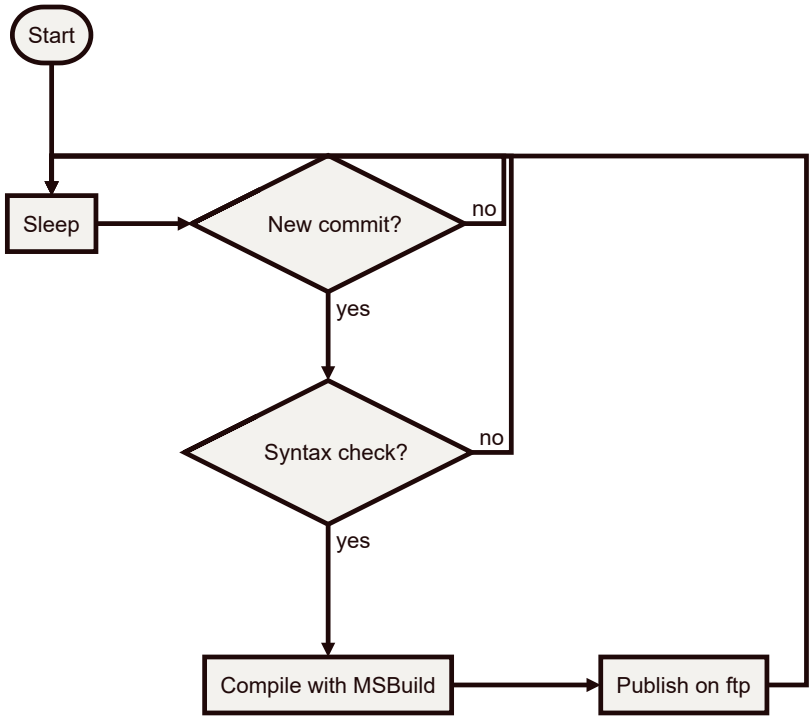
2.2 物理空间

物理空间类PhysicsSpace的主要成员是`std::vector<RigidBody>`类型的`m_RigidBodySet`，即刚体的集合。同时，它还提供若干的接口，与命令相一一对应。

注意PhysicsSpace本身不处理消息，另有单独的Model类作为其消息接口区，其有一个PhysicsSpace类的成员`physicsSpace`。Model类在接受命令、更改完数据后（具体表现为调用`physicsSpace.方法()`），会通知ViewModel更新绘制。

2.3 持续集成

我们的持续集成通过一台阿里云VPS实现。VPS上部署有.bat脚本，以自动从GitHub抓取最新版本、执行语法检查并自动编译部署。其运行逻辑如下：*（需要markdown阅读器支持flowchart）*



最终生成的Release是以当前时间命名的.zip压缩包，其中包含以Release模式编译的程序、由脚本生成的commit日志与编译日志，储存于ftp服务器上。

3 运行效果图

4 错误与反思

我们最大的错误发生在理解错了View层的内容。我们的View实际上相当于fltk衍生而来的图形库扩展，只不过是提供了适用于我们的Common层的数据结构的接口。这样一来，实际上绘制的逻辑很大一部分是由ViewModel承担的，这也导致View与ViewModel发生了耦合，使MVVM退化成了MVC。

但其实只要微小的改动，就可以解除这种耦合：在View层设立自己的消息接口区，将ViewModel中所有调用View提供的方法的代码复制到View层即可。如此一来，ViewModel就成为了一个真正的**“消息的交换机”**。

5 课程建议

总体而言，作为大一的同学，我在课程中高强度、高速度地学习到了大量前沿知识。相比于我校落后于时代的CS本科课程设置，这门课提供了许多接近于科研与工业界开发一线的技术内容。我想，虽然我们在课程中遇到了不少的挫折与困难，甚至可能成绩不会很好看，能够如此高密度地学习知识与方法是令我们获益匪浅的。

此外，我建议可以**将课程的时间略微延长**，还可以**配备助教**，应当能让大家的学习效果更加显著。