计算机动画的算法与技术

课程项目：基于 GPU 的碰撞检测

Name ID

---

1 实验目的

利用 GPU 的并行运算优势实现高效的碰撞检测算法，并在此基础上实现刚体运动的模拟，制作一段动画.

2 方法

物体模拟分为两部分：刚体物理运算、碰撞检测与碰撞解决．接下来分别介绍.

2.1 刚体物理

刚体物理积分算法基于 [2].

在这个算法中，每个刚体由一系列球形粒子组成．每个粒子有如下属性，用小写字母表示:

- 固有属性：半径 r_i、质量 m_i、弹性 e_i、相对刚体重心的偏移位置 $\boldsymbol{u}_i$;
- 状态属性：位置 $\boldsymbol{x}_i$、速度 $\boldsymbol{v}_i$.

每个刚体有如下属性，用大写字母表示:

- 固有属性：质量 M_k、初始惯性矩 $\boldsymbol{I}_k(0)$;
- 状态属性：位置 $\boldsymbol{X}_k$、线速度 $\boldsymbol{V}_k$、加速度 $\boldsymbol{A}_k$、方向四元数 $\boldsymbol{Q}_k$、角动量 $\boldsymbol{L}_k$.

为简洁起见，接下来未说明时，不带下标的小写字母均表示某个粒子 i 的属性值，大写字母表示此粒子所属刚体 k 的属性值.

在每一步开始时，先计算每个粒子的位置和速度:

$$\begin{aligned}\boldsymbol{x} &= \boldsymbol{X} + \boldsymbol{Q}\boldsymbol{u}\boldsymbol{Q}^{-1} \\ \boldsymbol{v} &= \boldsymbol{V} + \boldsymbol{A} \times \left(\boldsymbol{Q}\boldsymbol{u}\boldsymbol{Q}^{-1}\right)\end{aligned}$$

接下来将所有粒子独立处理，计算每个粒子所受的合外力．这一部分的具体计算方法在**【碰撞解决】**一节中介绍，这里不作展开．得到每个粒子所受的合力 $\boldsymbol{f}$ 后，可计算每个刚体所受的合力与合力矩

$$\begin{aligned}\boldsymbol{F} &= \sum \boldsymbol{f} \\ \boldsymbol{T} &= \sum (\boldsymbol{x} - \boldsymbol{X}) \times \boldsymbol{f} \\ &= \sum \left(\boldsymbol{Q}\boldsymbol{u}\boldsymbol{Q}^{-1}\right) \times \boldsymbol{f}\end{aligned}$$

然后根据物理定律进行积分运算．首先考虑平移运动:

$$\begin{aligned}\mathrm{d}\boldsymbol{X} &= \boldsymbol{V}\mathrm{d}t \\ \mathrm{d}\boldsymbol{V} &= \boldsymbol{A}\mathrm{d}t\end{aligned}$$

这部分通过 Verlet 积分方法计算.

然后是旋转运动. 考虑角速度

$$\begin{aligned}\boldsymbol{\Omega} &= \boldsymbol{I}(t)^{-1}\boldsymbol{L} \\ &= \boldsymbol{Q}\boldsymbol{I}(0)^{-1}\boldsymbol{Q}^{-1}\boldsymbol{L}\end{aligned}$$

由此计算角动量与方向的变化量

$$\begin{aligned}\mathrm{d}\boldsymbol{L} &= \boldsymbol{T}\,\mathrm{d}t \\ \mathrm{d}\boldsymbol{Q} &= \left[\cos\frac{\|\boldsymbol{\Omega}\|\,\mathrm{d}t}{2} \quad \frac{\boldsymbol{\Omega}}{\|\boldsymbol{\Omega}\|}\sin\frac{\|\boldsymbol{\Omega}\|\,\mathrm{d}t}{2}\right]\end{aligned}$$

注意其中方向四元数的更新方式是作叉积 $\boldsymbol{Q}(t+\mathrm{d}t) = \mathrm{d}\boldsymbol{Q}(t) \times \boldsymbol{Q}(t)$. 这部分难以化为 Verlet 积分的形式，故采用前向 Euler 方法计算. 至此即完成了一个时间步长的积分运算.

2.2 碰撞检测

碰撞检测算法基于 [1]. 这一部分的目的是找出所有相交的粒子对.

这一算法的根基是扫描线（sweep-and-prune, SaP）方法. 选取主轴单位向量 $\boldsymbol{l}_z$，计算每个粒子的球心在轴上的坐标 $z_i = \boldsymbol{x}_i \cdot \boldsymbol{l}_z$，然后将所有坐标排序.

记所有粒子的最大半径为 R. 如此一来，对于某粒子 i，与之相交的粒子 j 在轴上的坐标 z_j 必然满足 $|z_i - z_j| < r_i + R$. 于是只需在排序后的序列中检查此范围内的粒子是否与 i 相交即可.

在此基础上引入了几处优化，接下来一一介绍.

2.2.1 并行化

扫描线方法可以在 GPU 上高度并行化. 每个粒子的 z_i 值计算是完全独立的，排序的过程也可以由并行的基数排序实现.

由于被排序的数值是浮点数形式，为了在此情况下应用基数排序，需要执行一些转换. 首先注意到，对于所有有限正数, IEEE 浮点数将幂（exponent）以无符号整数的形式置于高位，将有效数（significand）以固定精度的二进制分数形式置于低位，因此它们直接按位解释（bitcast）为整数时，大小关系不变. 而对于有限负数，由于符号位只表示 -1 的系数，故 bitcast 后的大小关系会颠倒. 另外，在升序排序时，所有负浮点数要置于所有正数之前，这与最高位（即浮点数符号位）的规定相反.

由此可以得出，在 bitcast 为无符号整数后，恢复原本浮点数大小关系的方法是：首先将符号位取反，此外若原本为负数，则将所有其余位也取反. 此后即可作为整数排序问题，进行基数排序.

基数排序本身可以通过将序列分段实现并行化，每个线程负责统计一个连续段内每个桶中元素的数量，所有统计结果由一个并行化前缀和算法合并后，再回到每个线程中将段内的元素重排，写入新数组的恰当位置. 这一部分的理论并行时间复杂度是 $O(\log n)$，其中 n 为序列长度. 不过实际上由于硬件性能有限，如此低的耗时并不现实，具体的线程数量也取决于硬件与元素数量进行调优.

2.2.2 主轴选择

通过主成分分析（PCA）方法选择主轴，可以使各粒子在轴上的投影更加分散，避免冗余计算．这即是求

$$\underset{\|\boldsymbol{l}_z\|=1}{\operatorname{argmax}}\ \operatorname{Var}_i\{\boldsymbol{l}_z\cdot\boldsymbol{x}_i\},$$

若记所有粒子的 3×1 位置向量水平拼接所得的 $3\times n$ 矩阵为 $\boldsymbol{X}$，则相当于求

$$\underset{\|\boldsymbol{l}_z\|=1}{\operatorname{argmax}}\ \boldsymbol{l}_z^{\mathrm{T}}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{l}_z,$$

即相当于求 3 阶协方差方阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 的主特征向量.

协方差矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 显然可以高度并行计算．主特征向量可以由幂迭代法（power iteration）计算，所需迭代次数非常少（约 10 次以内可基本收敛），顺序执行即可.

2.2.3 空间分割

在垂直于主轴的二维坐标系上，可以对粒子进行空间分割，以进一步排除大量冗余运算.

首先构建这样一个坐标系．任意选取一个不与 $\boldsymbol{l}_z$ 平行的单位向量，与 $\boldsymbol{l}_z$ 作叉积即可得一个垂直于 $\boldsymbol{l}_z$ 的向量 $\boldsymbol{l}_p$．计算 $\boldsymbol{l}_q=\boldsymbol{l}_z\times\boldsymbol{l}_p$，则以三个单位向量 $\boldsymbol{l}_p$, $\boldsymbol{l}_q$, $\boldsymbol{l}_z$ 为正方向形成一右手坐标系．在此坐标系下计算每个点的坐标 $p_i=\boldsymbol{x}_i\cdot\boldsymbol{l}_p$，$q_i=\boldsymbol{x}_i\cdot\boldsymbol{l}_q$．在 p–q 平面上进行空间分割.

设分割所用块的边长为 M；区域 $[aM,(a+1)M)\times[bM,(b+1)M)$ 记作块 (a,b)．一个半径为 r_i、坐标为 (p_i,q_i) 的粒子可能与这样的块 (a,b) 相交:

$$\left\lfloor\frac{p_i-r_i}{M}\right\rfloor\le a<\left\lceil\frac{p_i+r_i}{M}\right\rceil$$
$$\left\lfloor\frac{q_i-r_i}{M}\right\rfloor\le b<\left\lceil\frac{q_i+r_i}{M}\right\rceil$$

将原本的粒子在扫描线上用一系列新的记录替代：对于每个相交的块 (a,b)，放置一个记录点 $z'=z_i+N\cdot f(a,b)$．其中 N 是 z 坐标中的最大值与 R 之和，而 $f(a,b)$ 是块到整数的单射（不一定是满射）．这相当于将每个块赋予一个惟一编号，然后将不同块的记录点之间彻底分开．这里 f 采用基于 Cantor 配对函数的定义

$$f(a,b)=(2\pi(a,b)+\mathbb{1}[a<0])\cdot(2\cdot\mathbb{1}[y<0]-1)$$
$$\pi(a,b)=\frac{1}{2}(a+b)(a-b+1)+a$$

此番处理后，依照原始的扫描线方法进行排序与检查即可．由于块边界的记录点之间存在大的差异，在扫描线上检查时不会越过块边界，减少冗余运算.

存在一个细节问题．一个粒子可能会被分入多个块中，而若两个跨过块边界的粒子相交，处理不当时可能会将它们之间的碰撞计入多次．一个处理方式是将每个粒子在其相交的 (a,b) 最小的块中标记为“实”，在其余相交的块中标记为“虚”；此后，在扫描线上只从“实”粒子 i 出发检查碰撞，检查范围为 $(z_i-r_i,\ z_i)$ 内的“虚”粒子以及 $(z_i-r_i,\ z_i+r_i)$ 内的“虚”“实”粒子．这样一来，即避免了此类重复计入的问题.

2.3 碰撞解决

碰撞解决算法同样基于 [2].

对于两个相交的粒子 i, j，它们之间的相互作用力由一个带阻尼的弹簧模型近似，可分为三部分：排斥力（repulsive force）$\boldsymbol{f}_{\mathrm{s}}$、阻尼力（damping force）$\boldsymbol{f}_{\mathrm{d}}$、剪切力（shearing force）$\boldsymbol{f}_{\mathrm{t}}$. 具体而言，

$$
\begin{aligned}
\boldsymbol{f}_{\mathrm{s}} &= k_{\mathrm{s}} \cdot ((r_i + r_j) - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|) \cdot \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}, \\
\boldsymbol{f}_{\mathrm{d}} &= \eta \cdot (\boldsymbol{v}_j - \boldsymbol{v}_i), \\
\boldsymbol{f}_{\mathrm{t}} &= k_{\mathrm{t}} \cdot \left((\boldsymbol{v}_j - \boldsymbol{v}_i) - \left((\boldsymbol{v}_j - \boldsymbol{v}_i) \cdot \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} \right) \cdot \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} \right).
\end{aligned}
$$

地面对粒子的弹力也可类似由弹簧模型近似，包括排斥力与阻尼力，其中阻尼力的系数与粒子弹性 e_i 成正比. 摩擦力由 Coulomb 模型计算：动摩擦力 $\|\boldsymbol{f}_{\mathrm{fric}}\| = \mu \|\boldsymbol{f}_{\mathrm{norm}}\|$，最大静摩擦力等于动摩擦力.

将上述所有力合计后即可按照前述方法进行积分.

3 结果

3.1 效率

测试环境为 MacBook Pro (Early 2015)，GPU 为 Intel Iris Graphics 6100，这与 [1] 中实验所使用的 NVIDIA Tesla C1060 性能较为接近.

测试场景中放置大量刚体，各自由八个粒子组成，反复测量一步模拟（`step()`，包括碰撞检测、碰撞解决与积分）的耗时. 所得结果与 [1] 汇报的结果对比如下.

粒子数量	本实验 (ms)	[1] (ms)
16 000	3.9	3
128 000	12	11
960 000	87	161

虽然由于测试环境和虚拟场景的设定都不尽相同，数据不完全具有可比性，但是可以看出二者达到了类似的效率，也即与 CPU 处理相比有显著优势.

3.2 问题

程序存在的一个问题是在相同的初始设定下，多次执行程序所得的物体运动方式不同. 经检查，其原因是多个线程计算浮点数之和时累加顺序不固定，而 IEEE 浮点数运算并不满足交换律，导致结果的微小差异不断累积，形成蝴蝶效应.

虽然此问题可以通过略微改写循环形式，使 Taichi 固定运算顺序来解决，但一方面对性能可能有细微影响，另一方面这样的差异是可以接受的，毕竟其本质是浮点数精度的限制，而不是结果错误，因此没有针对此问题修改程序.

附录 A　程序细节

环境与运行方式

程序基于“太极”（Taichi）并行计算框架编写，此框架基于 Python 语言．因此，程序要求系统装有支持 Taichi 的 Python（实验中为 Python 3.9.5）．

推荐使用 Python 虚拟环境来运行程序．将源代码目录作为工作目录，执行命令

```
python3 -m venv ti                  # 创建虚拟环境
source ti/bin/activate              # 激活虚拟环境
pip3 install -r requirements.txt    # 安装依赖项
```

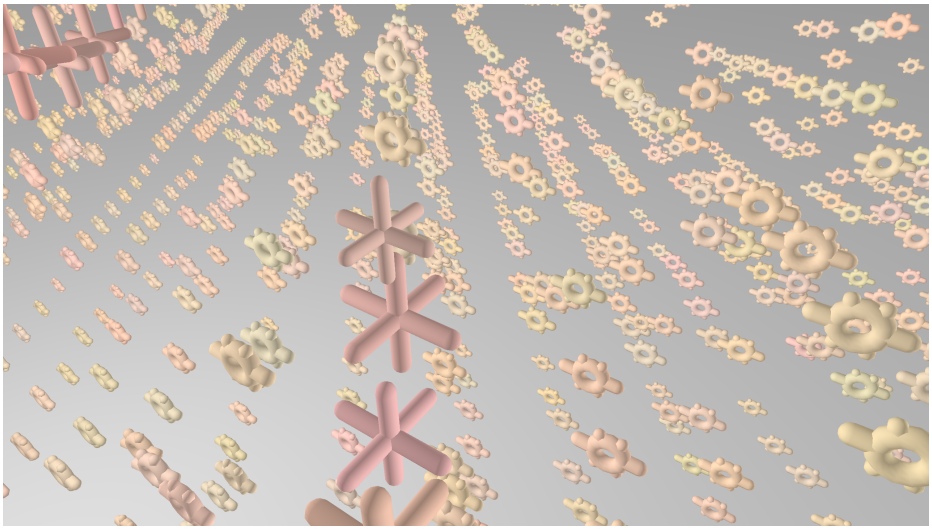
此后即可在同一目录下选择一个程序运行：

```
python3 sim1-spheres.py     # 球体场景
python3 sim2-molecules.py   # 不规则物体场景
```

Taichi 对不同 OS 与 GPU 的支持可能要求不同的配置方式．若出现问题，可依次尝试设置环境变量 `TI_BACKEND` 为以下值之一，重新运行：

```
export TI_BACKEND=gpu
export TI_BACKEND=opengl
export TI_BACKEND=vulkan
export TI_BACKEND=cpu       # 无 GPU 加速
```

若正确运行，应当出现一个展示物体动画的窗口．



在此窗口中按下以下按键，可以为所有粒子施加不同种方向的外力，此时地面也改变颜色．如果多个按键同时按下，则为多种外力叠加．

- 左箭头键：x 方向，朝向原点；
- 上箭头键：y 方向，朝向原点；
- 空格键：旋转方向，使物体绕原点公转．

组织结构

两份程序的结构一致，球体场景由不规则物体场景简化而来．以下均由不规则场景的程序（`sim2-molecules.py`）为例，并在假设 Taichi 通过 GPU（而非 CPU 多线程）加速程序的前提下作出分析与解释．

程序首先定义一系列 Taichi 域，也即在 CPU 与 GPU 之间交换的数据．其中重要的域罗列如下：

- `x0`, `m`, `elas`, `body`: 每个粒子相对刚体重心的位置、粒子的质量、弹性，以及所属的刚体序号．
- `bodyPos`, `bodyVel`, `bodyAng`...: 每个刚体的位置、速度、角速度等状态，详见源码注释．
- `fSum`, `tSum`: 每个刚体在当前时刻受到的合力以及合力矩．
- `projPos`, `projIdx`: 每一个位置代表一个粒子在空间划分后的投影坐标，以及其序号．

然后定义了一系列 Taichi 的函数与核，它们即是在 GPU 上执行的子过程，其区别在于核可以由 Python 域的程序直接调用，而函数不可以．其中重要的子过程罗列如下：

- 模拟相关
  - `init()`: 构造物体形态与参数，预计算重心与惯性张量，设定位置，赋予初速度．
  - `step()`: 执行一步模拟．其中两个重要的子过程：
    * `colliResp(i, j)`: 若物体 `i` 与 `j` 接触，则计算它们之间的相互作用力，并将结果计入全局域 `fSum`（力）与 `tSum`（力矩）．
    * `sortProj(N)`: 将全局域 `projPos` 的前 `N` 个值排为升序，同时将全局域 `projIdx` 作为标签随排序过程一起移动．
- 显示相关
  - `buildMesh()`: 构造用于显示物体的网格模型．
  - `updateMesh()`: 根据物体的位置与方向，更新网格模型．

最后是 Python 域的一系列过程，这也是程序核心逻辑的起始点，具体在下一节解释．

逻辑流程

程序第 755–756 行调用 `init()` 与 `buildMesh()` 两个核，完成全局初始设定．

接下来是一些创建窗口、录制数据等工作，较为繁琐，若有兴趣请参见注释．

第 806 行起定义的 `render()` 函数按照当前的物体状态，将三维场景渲染到窗口上，并且若设定进行录制，则一并将物体状态记录下来．

第 837 行起的循环在每一帧执行 1 / 60 秒的模拟，即 10 次 `step()` 调用，然后调用 `render()` 渲染图像．

在第 327 行起定义的 `step()` 核内部，流程与**【方法】**一节中所述一致，即

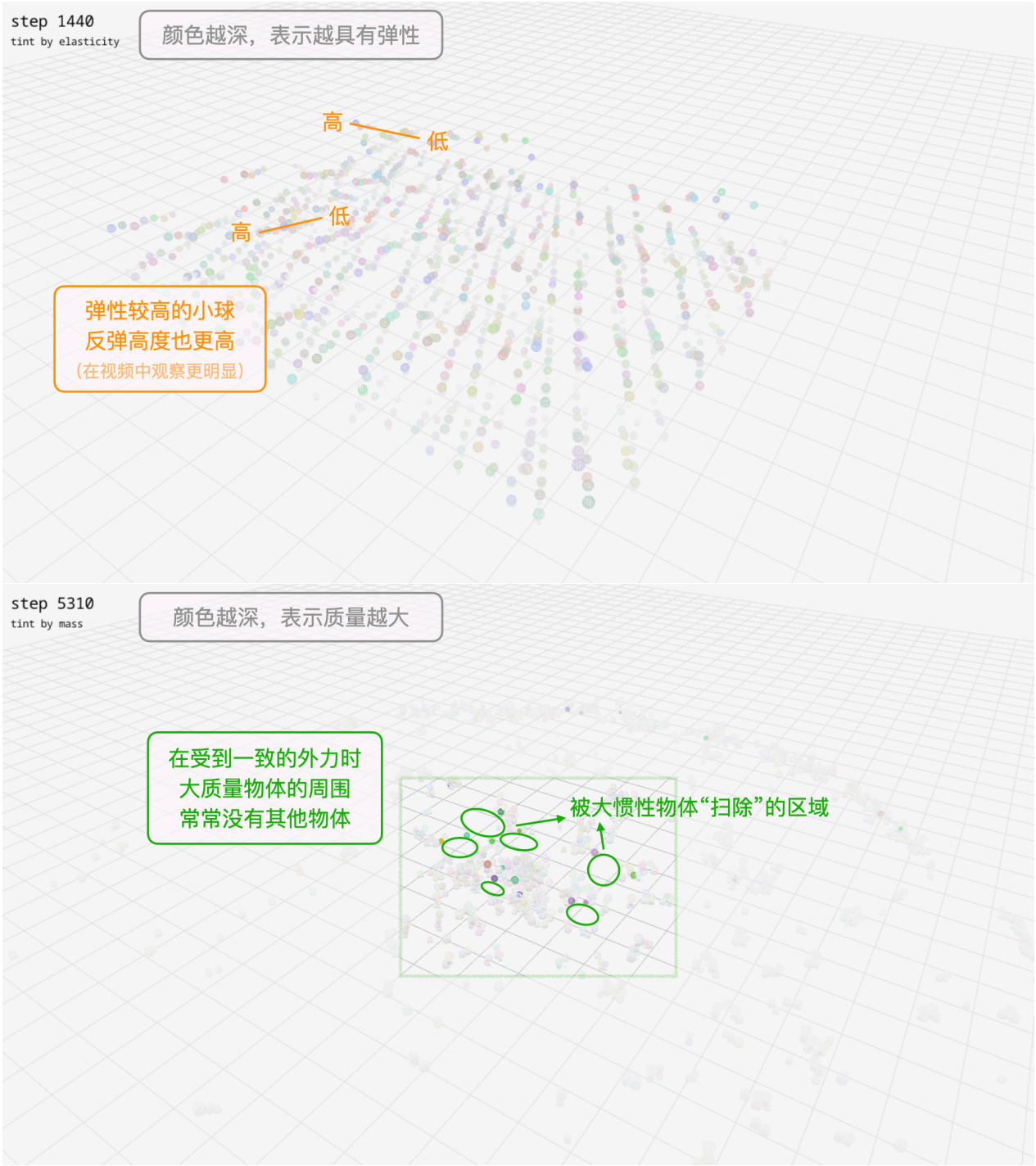
- 计算粒子位置与速度（328 行起），
- 执行 PCA 并选取投影轴（344 行起），
- 进行空间划分、计算投影坐标并排序（376 行起），
- 寻找相交粒子对并计算互作用力（411 行起），

- 计入重力、地面弹力与摩擦力，以及通过输入控制的额外拉力（440 行起）.
- 计算积分并更新刚体状态（492 行起）.

具体实现方式与前文的描述一致，不再赘述.

附录 B　演示视频

演示视频见附件 `demo.mp4`. 视频依次展示了球体场景与不规则物体场景的视频录像. 其中地面颜色的变化指示施加的不同外力. 另外，视频也包括了线框示意图，按照弹性与质量突出显示一部分物体，以更清晰地展示这两种属性对物体运动的影响. 具体如下.



附录 C　录制与回放线框图

在运行模拟程序时，设置环境变量 `REC`，可以使程序录制回放数据．具体格式为

```
REC=<n_steps>,<file>
```

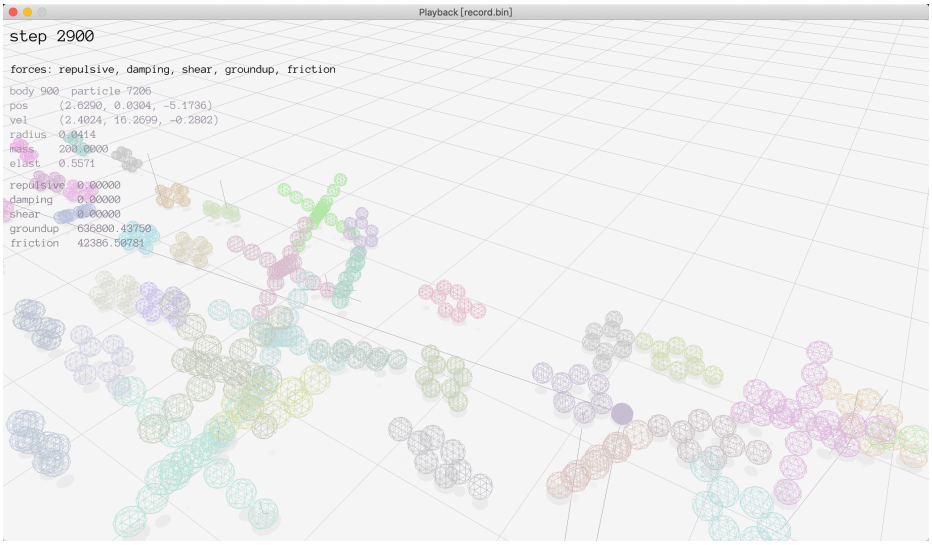
其中 `<n_steps>` 为十进制正整数，表示要录制的步数（每帧 10 步，每秒 60 帧即 600 步），`<file>` 为存储的数据文件名称，此部分可以省略，默认为 `record.bin`．例如以下命令行：

- `REC=600,sim1.bin python3 sim1-spheres.py` 用于录制 1 秒的球体动画至文件 `sim1.bin`；
- `REC=6000 python3 sim2-molecules.py` 用于录制 10 秒的复杂物体动画至文件 `record.bin`．

源码目录下所附的 `playback.c` 是回放程序．将此源文件与图形库 raylib 一同编译并链接以获得可执行文件 `playback`．

运行方式为 `./playback <file>`，其中 `<file>` 为存储的数据文件名称，若省略则同样默认为 `record.bin`．运行后可使用左右按键连续播放，也可使用上下按键逐步播放；用左 Shift 键调整播放速度．另外，可用 W、A、S、D、Q、Z 六个按键在三维空间中移动，并用 E、R 两个按键调整摄像机视角．按下数字 0 键可以切换着色方式，如按照质量、按照弹性系数赋予深浅颜色．

若取消 Python 源文件末尾倒数第 8 至倒数第 6 行的三个注释符号，则录制的文件额外包括速度、受力情况、接触情况等详细数据．在编译 `playback.c` 时额外定义符号 `RECDEBUG` 为 1（GCC 命令行参数 `-DRECDEBUG=1`），即可播放此类记录．此时按下数字 1 至 5 键可以切换五种不同力（排斥力、阻尼力、剪切力、地面弹力、摩擦力）分别是否由粒子出发的直线段指示方向与大小．在开发过程中，此功能用于调试物体运动的问题，如下图展示了地面作用力过大的错误情形．



参考文献

[1] **Real-time Collision Culling of a Million Bodies on Graphics Processing Units**. Fuchang Liu, Takahiro Harada, Youngeun Lee, Young J. Kim. ACM SIGGRAPH 2010, DOI:10.1145/1882261.1866180.

[2] **Real-Time Rigid Body Simulation on GPUs**. Takahiro Harada. GPU Gems 3 Ch. 29, NVIDIA.

[3] **Improved GPU Sorting**. Peter Kipfer, Rüdiger Westermann. GPU Gems 2 Ch. 46, NVIDIA.