

厦门大学附属实验中学

研究性学习报告

课题名称：计算机模拟刚性球体系统的物理分析、算法设计和 OpenGL 可视化
课题组员：高二（3）班 朱宝林、尤比佳、林佳诚、张宏彬
指导老师：黄罗华
申报时间：2021 年 1 月 12 日

厦大附中研究性学习课题开题报告

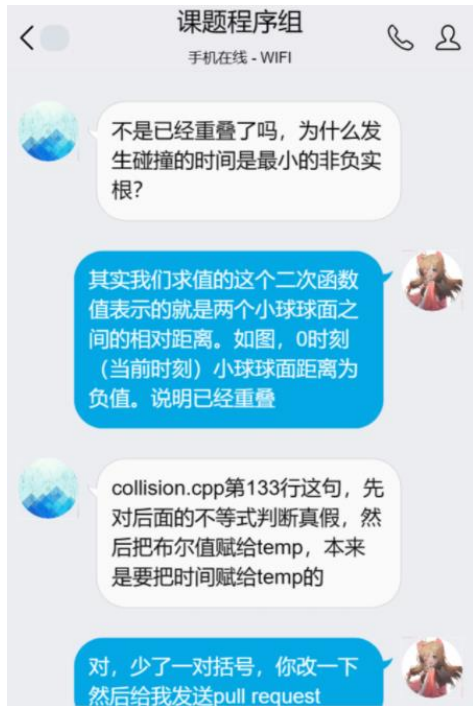
班级：高二（3）班 编号：_____

| | | | | |
|------------|---|----------|----------------------------------|------------------|
| 课题名称 | 计算机模拟刚性球体系统的物理分析、算法设计和 OpenGL 可视化 | | | |
| 导师 | 黄罗华 | 时间 | 2021 年 1 月 12 日至 2021 年 7 月 22 日 | |
| 课题组长 | 朱宝林 | 课题成员 | 尤比佳、林佳诚、张宏彬 | |
| 课题简要背景说明 | 现代科学研究常以数学、物理等基础学科作为理论基础，将计算机作为数值运算和可视化工具。在本次研究性学习中，我们选取物理学的经典模型——刚性球体碰撞系统为研究对象，研究在计算机上模拟该模型所需的程序设计、算法优化和可视化等内容，并将该模型应用到实际问题。 | | | |
| 可行性分析 | 关于刚性球体碰撞的研究材料丰富，课题组员有较好的物理、计算机知识水平，具备一定的编程能力 | | | |
| 课题研究的目的与意义 | 通过此次研究，我们希望深入探索小球碰撞这一经典模型，学习物理研究的基本方法；探索在计算机上模拟物理过程，学习使用优先队列、空间划分等算法优化程序流程、处理大量数据；学习使用算法分析的基本方法评估算法的性能优劣；学习使用 OpenGL 图形库将研究结果可视化；学习运用科学研究方法探索、分析并解决问题，实际体验科学研究过程。这将极大地培养我们的和科学研究能力，为我们将来从事科学研究工作打下坚实基础。 | | | |
| 主导课程 | 计算机科学与技术 | 相关课程 | 高中物理 | |
| 任务分工 | 物理组 | 林佳诚、张宏彬 | | |
| | 程序组 | 朱宝林、尤比佳 | | |
| 研究方法 | 实验法、文献研究法、模型方法、信息研究方法 | | | |
| 研究实施步骤 | 阶段 | 时间 | 主要任务 | 目标 |
| | 一 | 1 月 12 日 | 选择课题 | 确立课题 |
| | 二 | 1 月 17 日 | 开题报告 | 明确课题内容、学习范围 |
| | 三 | 2 月-5 月 | 研究资料 编写程序 | 完成学习任务 搭建程序框架 |
| | 四 | 5 月-7 月 | 算法探究和实验 | 完成算法探究和实验 |
| | 五 | 7 月 | 汇总整理 | 整理研究结果、资料 |
| | 六 | 7 月 22 日 | 结题报告(论文) | 填写结果报告 |
| 预期成果 | 研究论文、发布程序包和研究成果数据 | | | |
| 导师意见 | | | | |

厦大附中研究性学习过程日志 一

| | |
|------|---|
| 课题导师 | 黄罗华 |
| 课题名称 | 计算机模拟刚性球体系统的物理分析、算法设计和 OpenGL 可视化 |
| 课题成员 | 朱宝林、尤比佳、林佳诚、张宏彬 |
| 活动主题 | 讨论刚性球体的碰撞预测、检测和处理方案 |
| 活动过程 | <p>2021 年 4 月 11 日下午，课题组员在厦大附中科创教室集中，由组长安排活动任务。本次活动讨论刚性球体的碰撞预测、检测和处理方案。组长首先给成员发放阅读论文材料，涉及 3D 游戏中的碰撞检测算法、快速空间三角形对相交检测算法、少体硬球系统的动力学与统计研究等方面。组员浏览材料半小时后发言，总结材料内容并评价方法优劣。发言结束后小组成员共同讨论、拟定解决方案。最后分配任务，由物理组同学负责该部分论文，程序组同学按照物理部分论文进行程序实现。</p> |
| 主要收获 | <p>通过这次活动，我们培养了文献阅读和学习的能力，在小组讨论提高和他人进行学术交流、科学地分析评判的能力。我们也学习到了很多物理、计算机方面的专业知识。</p> |
| 成员签名 | 朱宝林、尤比佳、林佳诚、张宏彬 |
| 备注 | |

厦大附中研究性学习过程日志 二

| | |
|------|--|
| 课题导师 | 黄罗华 |
| 课题名称 | 计算机模拟刚性球体系统的物理分析、算法设计和 OpenGL 可视化 |
| 课题成员 | 朱宝林、尤比佳、林佳诚、张宏彬 |
| 活动主题 | 合作远程调试程序，解决程序运行错误 |
| 活动过程 | <p>2021 年 7 月 10 日下午，程序组员使用 TIM 远程共享屏幕合作调试程序。组员针对程序运行过程中出现的隔空碰撞、穿墙、忽略碰撞等问题，分工检查程序代码，讨论程序运行逻辑，找到 bug 并改正。程序组员使用 Github 管理代码库、进行协作。借助科技手段，我们得以便捷沟通、远程调试。左图是调试过程中成员的讨论记录。</p>  |
| 主要收获 | <p>通过这次活动，我们培养了使用计算机软件进行远程协作、合作调试代码的能力。熟练掌握编程辅助软件将有助于我们以后快速开发程序、在团队中与他人合作沟通。</p> |
| 成员签名 | 朱宝林、尤比佳、林佳诚、张宏彬 |
| 备注 | |

计算机模拟刚性球体系统的 物理分析、算法设计和 OpenGL 可视化

小组成员：朱宝林、尤比佳、林佳诚、张宏彬

指导老师：黄罗华

摘要：计算机科学技术在现代科学研究中起着不可替代的作用。本论文选取物理学中的经典问题——刚性球体碰撞模型作为研究对象，全面地研究了在计算机上模拟物理系统的各个步骤，包括物理分析、算法设计、数据结构和可视化实现。重点研究基于堆的优先队列、空间划分等优化算法的应用。最后使用程序模拟真实物理场景中的理想气体，展示计算机处理复杂问题的高效性。

关键词：计算机物理模拟 刚体力学 算法分析与设计 计算机图形学

第1章 绪论

1.1 研究背景

现代科学研究常以数学、物理等基础学科作为理论基础，将计算机作为数值运算和可视化工具。凭借强大的性能和高效的算法，计算机能够快速处理海量数据并得到准确的结果；利用可视化技术，研究人员得以实时观察数据情况，真实地对仿真过程进行分析，易于理解。计算机仿真模拟能够提高工作效率、减少经费开支、缩短研发周期，在工程技术领域被广泛应用。

刚性球体模型是物理学中的经典模型。当物体自身形状可以忽略时，常常将其简化为刚性球体或质点进行处理。借助刚性球体模型，物理学家对理想气体的微观运动进行了深入探索。刚体碰撞模型还在化学反应动态过程、元素相变、最密堆积问题、天体物理等领域具有广泛应用。

1.2 研究目的与意义

本论文选取刚性球体碰撞模型作为研究对象，全面地研究了在计算机上模拟物理系统的各个步骤，包括物理分析、算法设计、数据结构和可视化实现。重点研究基于堆的优先队列、空间划分等优化算法的应用。最后使用程序模拟真实物理场景中的理想气体，展示计算机处理复杂问题的高效性。

通过此次研究，我们希望深入探索小球碰撞这一经典模型，学习物理研究的基本方法；探索在计算机上模拟物理过程，学习使用优先队列、空间划分等算法优化程序流程、处理大量数据；学习使用算法分析的基本方法评估算法的性能优劣；学习使用 OpenGL 图形库将研究结果可视化；学习运用科学研究方法探索、分析并解决问题，实际体验科学研究过程。这将极大地培养我们的和科学研究能力，为我们将来从事科学研究工作打下坚实基础。

1.3 研究内容

物理学：刚体力学、球体碰撞预测、检测、处理分析

计算机科学与技术：面向对象程序设计；算法分析；排序算法（基于堆的优先队列）；数据结构（空间划分）；计算机可视化（基于 OpenGL）

1.4 研究方法

实验法、文献研究法、模型方法、信息研究方法

第2章 物理分析

本章简要介绍本研究性学习所使用的物理模型：首先对刚性球体系统建模，然后分析系统内元素间相互作用过程，最后介绍该模型的具体应用场景。

2.1 刚性球体系统建模

我们需要建立模型包括：实体物理模型、状态物理模型和过程物理模型。

- 实体和状态物理模型：
 - 球：质量 m ，半径 r ，位置 \vec{p} ，速度 \vec{v}
 - 平面：位置（平面上一点） \vec{p} ，法向量 \vec{n}
 - 长方体容器：容器中心的位置 \vec{p} ，容器的长宽高 a, b, c ；容器壁和坐标系平面平行

系统中只有球体运动，球体与其它物体的相互作用（碰撞）可以分成三类问题：碰撞预测、检测和处理。所有碰撞均为弹性碰撞，不考虑摩擦和自旋。

物理过程的分析方法将决定编程模型：时间驱动模型依赖于碰撞检测，事件驱动模型依赖于碰撞预测。各类物体的碰撞检测都已有非常详尽的研究，如引文[3][5][6]，可以直接采用。本论文主要讨论事件驱动模型，使用碰撞预测分析。

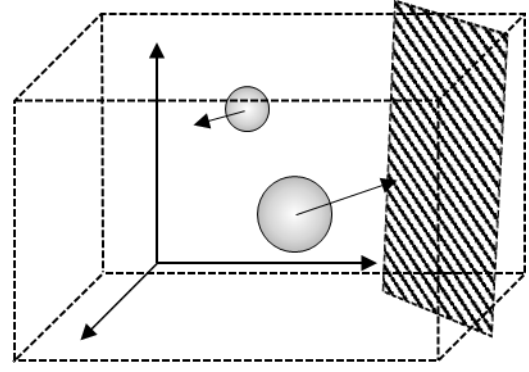


图 2.1 碰撞系统建模

2.2 碰撞预测

2.2.1 球-球

球体占有到球心一定距离内的空间，可以利用空间中两点距离公式确定两球是否发生相撞。

设空间中两球位置分别为 $\vec{p}_1 = (x_1, y_1, z_1)$ 和 $\vec{p}_2 = (x_2, y_2, z_2)$ ，两球速度为 $\vec{v}_1 = (v_{1x}, v_{1y}, v_{1z})$ 和 $\vec{v}_2 = (v_{2x}, v_{2y}, v_{2z})$ ，两球半径分别为 r_1, r_2 。

考虑两球球面距离 d 关于时间 t 的函数：

$$d = |(\vec{p}_1 + t\vec{v}_1) - (\vec{p}_2 + t\vec{v}_2)| - (r_1 + r_2) \quad (2.1)$$

设两球在 t 时刻相撞，则可令 $d = 0$ ， $\Delta \vec{v} = \vec{v}_1 - \vec{v}_2$ ， $\Delta \vec{p} = \vec{p}_1 - \vec{p}_2$ ，等价转换为一元二次方程：

$$|\Delta \vec{v}|^2 t^2 + 2(\Delta \vec{v} \cdot \Delta \vec{p})t + |\Delta \vec{p}|^2 - (r_1 + r_2)^2 = 0 \quad (2.2)$$

计算方程判别式

$$\Delta = 2\sqrt{(\Delta \vec{v} \cdot \Delta \vec{p})^2 - |\Delta \vec{v}|^2 [|\Delta \vec{p}|^2 - (r_1 + r_2)^2]} \quad (2.3)$$

若判别式 $\Delta \geq 0$ 得到方程两根：

$$t_1 = \frac{-2 \Delta v \cdot \Delta p + \sqrt{\Delta}}{2|\Delta v|^2} \quad t_2 = \frac{-2 \Delta v \cdot \Delta p - \sqrt{\Delta}}{2|\Delta v|^2} \quad (2.4)$$

对方程根的情况分类讨论，可知：

- ① 若方程两根均大于 0，则两球发生碰撞的时间是较小的一根 t_2
- ② 若方程出现负根，则两球可能重叠，或者背向运动，不会发生碰撞

我们可以通过函数绘图直观理解上文对根分布情况的讨论：

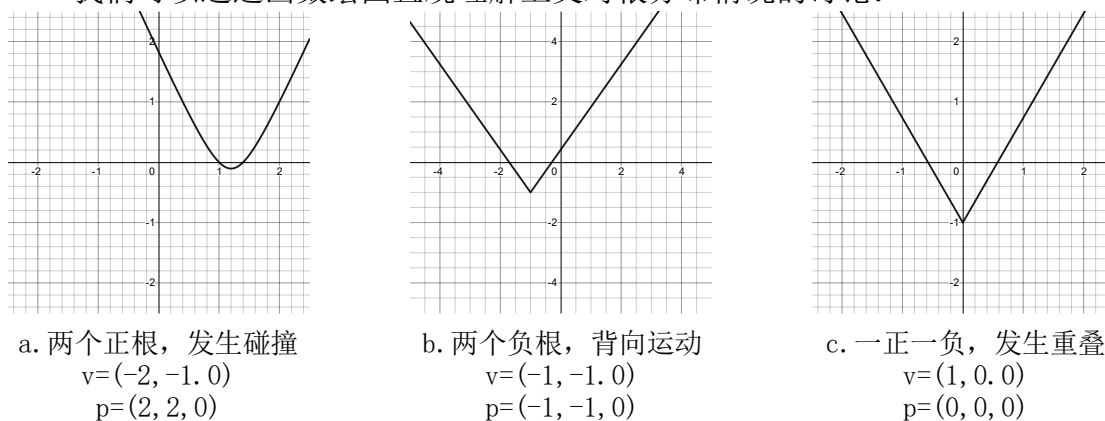


图 2.2 球面距离函数和碰撞时间方程根的分布

2.2.2 球-容器

只考虑球在容器内的情况。步骤如下：

- ① 根据球体速度方向确定会发生碰撞的容器壁
- ② 分别在 x, y, z 方向计算：碰撞时间 = (|容器壁位置 - 球位置分量| - 球半径) / 速度分量
- ③ 各容器壁碰撞时间中最小的即为球与容器的碰撞时间

2.3 碰撞处理

2.3.1 球-球

任何维度的球体碰撞均可转化为沿球心连线方向的一维碰撞，在其他方向上运动状态不变。这是因为刚性球体碰撞时，弹力的方向垂直于小球接触点的切面，从接触点指向球心。

假设某一时刻两球发生碰撞，取该时刻球体位置矢量相减并归一化得到球心连线的方向矢量

$$\vec{p} = \frac{\vec{p}_1 - \vec{p}_2}{|\vec{p}_1 - \vec{p}_2|} \quad (2.5)$$

将两球的速度与方向矢量点乘得到速度在球心连线方向的分量

$$v_{10} = \vec{v}_{10} \cdot \vec{p} \quad v_{20} = \vec{v}_{20} \cdot \vec{p} \quad (2.6)$$

在球心连线方向上（一维空间）处理球体碰撞，应用《力学》中联立动量、动能守恒方程得到的一维碰撞解

$$v_1 = \frac{(m_1 - m_2)v_{10} + 2m_2v_{20}}{m_1 + m_2} \quad v_2 = \frac{(m_2 - m_1)v_{20} + 2m_1v_{10}}{m_1 + m_2} \quad (2.7)$$

计算该方向上速度变化量，将变化量乘以方向向量转换到三维空间，再与原速度相加

$$\vec{v}_1 = \vec{v}_{10} + (v_1 - v_{10}) \cdot \vec{p} \quad \vec{v}_2 = \vec{v}_{20} + (v_2 - v_{20}) \cdot \vec{p} \quad (2.8)$$

从而完成了一次球体碰撞处理

2.3.2 球-平面或容器

球与平面的碰撞处理：将小球速度沿平面法向量方向的分量取反。

球与容器的碰撞处理：因为并不知道和哪个容器壁发生碰撞，所以首先由小球速度方向确定可能发生碰撞的容器壁，计算小球与容器壁的距离，取距离最小的方向，将小球在该方向的速度分量取反。

2.4 应用

2.4.1 理想气体

理想气体模型被广泛应用于研究气体性质。理想气体模型中，气体分子被理想化为质点或刚性小球，与其他分子无相互作用。理想气体分子在各个方向上的运动速率服从正态分布：

$$p(v_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(v_t - \mu)^2}{2\sigma^2}\right] (t = x, y, z) \quad (2.9)$$

热力学假设分子做热运动时任意时刻速度方向沿空间的任意反向都是等概率的，故速度分量的平均值 $\mu = \overline{v_x} = \overline{v_y} = \overline{v_z} = 0$ ，由文献[7]的推导得到方差 σ_x^2 的表达式，其中 k_B 是玻尔兹曼常数， T 是热力学温度， m 是粒子的质量

$$\sigma_x^2 = \frac{k_B T}{m} \quad (2.10)$$

在第6章我们将利用该结论为生成符合麦克斯韦分布律的模拟气体分子。

2.4.2 布朗运动

布朗运动是指微小粒子或颗粒在流体中做的无规则运动，在1827年由英国植物学家罗伯特·布朗在显微镜下观察到。然而，在1905年爱因斯坦提出相关理论前，世人对布朗运动一直存在误解。根据爱因斯坦的理论，布朗粒子每秒大约发生 10^{14} 次撞击，经典力学无法确定布朗粒子运动的路径。引文[10]使用计算机模拟的方法证明：少体硬球系统不能很好地模拟布朗运动。随着计算机运行速度的提升，从微观上模拟布朗运动成为可能。本论文所研究内容就是实现微观模拟布朗运动的基础。

2.4.3 行星环稳定性

1859年，麦克斯韦发表论文《论土星环运动的稳定性》(On the stability of the motion of Saturn's rings)，证明土星环是由大量独立的

小颗粒构成的。历史上对于行星环的研究主要有两种方法：一种把行星环当做 N 体系统，一种使用流体力学方法。如今，“流线表示法”等新的理论研究方法不断被提出，数值模拟技术也在飞速发展。刚体粒子模型已经被证明为是行星环不太恰当的近似，但它仍是重要的研究方法，引文[13][14]就是基于该模型展开分析。

第3章 程序设计和实现

在这一章，我们主要使用 UML(Unified Modeling Language，统一建模语言)来描述我们的程序设计过程。我们采取自顶向下的思路来设计程序，再用自底向上方法使用 C++语言开发程序。

3.1 面向对象分析

为了使用计算机实现对上述物理过程的模拟，并将其运用到更多场景中，我们使用面向对象的程序设计方法，将物理模型抽象为程序中的对象，将物理过程抽象为这些对象的操作。

从用户视角来看，我们设计的碰撞系统(CollisionSystem)对象具有输入数据、运行、输出数据的功能。在事件驱动的框架下，碰撞系统内包含物体(Object)类和事件(Event)类。物体类保存物体信息并提供物体相关操作(如碰撞处预测、处理

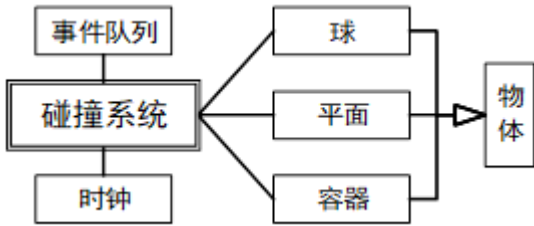


图 3.1 碰撞系统类图

等)。事件类保存碰撞事件的信息，并提供对事件的处理。图 3.1 展示了碰撞系统的类图，直线表示拥有关系，箭头表示继承关系。

3.2 API 设计

在实现数据类型前应当设计好 API(应用程序接口)。表 3.1-3.3 展示了 CollisionSystem 类、Event 类和 Ball 类的 API。其中，空间向量使用 GLM(OpenGL Mathematics)库的 vec3 类来表示，该库为 OpenGL 和计算机图形学量身定制，提供了对向量、矩阵等运算的支持。

表 3.1 CollisionSystem 类的 API

| CollisionSystem 类 | | |
|-------------------|--|---------|
| 返回类型 | 函数名和参数列表 | 说明 |
| | CollisionSystem(istream &is) | 从输入创建系统 |
| void | run(float t) | 运行指定时长 |
| void | reverse() | 反向运动 |
| float | ek() | 系统总动能 |
| istream & | operator>>(istream &, CollisionSystem &) | (友元) 输入 |
| ostream & | operator<<(ostream &, CollisionSystem &) | (友元) 输出 |

表 3.2 Event 类的 API

| Event 类 | | |
|-----------|---|----------|
| 返回类型 | 函数名和参数列表 | 说明 |
| | Event(shared_ptr<Ball>, shared_ptr<Object>, const float) | 创建事件 |
| float | t() | 返回事件发生时刻 |
| void | handle() | 处理事件 |
| bool | operator<(const Event &) | 比较事件发生先后 |
| ostream & | operator<<(ostream &, const Event &) | (友元) 输出 |

表 3.3 Ball 类的 API

| Ball 类 | | |
|--------------|--|----------|
| 返回类型 | 函数名和参数列表 | 说明 |
| | Ball(istream &) | 创建并递增计数器 |
| glm::vec3 | loc() | 位置 |
| glm::vec3 | v () | 速度 |
| float | r() | 半径 |
| float | m() | 质量 |
| unsigned int | cnt() | 碰撞次数 |
| Object_type | type() | 物体类型 |
| unsigned int | num() | 编号 |
| float | predict(Object &) | 预测碰撞 |
| void | bounce(Object &) | 处理碰撞 |
| bool | examine(Object &) | 检测重叠 |
| istream & | operator>>(istream &, Ball &) | (友元) 输入 |
| ostream & | operator<<(ostream &, const Ball &) | (友元) 输出 |
| ostream & | operator<<(ostream &, const vector<shared_ptr<Ball>> &) | 输出球队列 |

3.3 实现

按照上一节设计的 API，我们实现了碰撞系统和相关的类，它们的成员表示在图 3.2 中，其中箭头表示继承关系

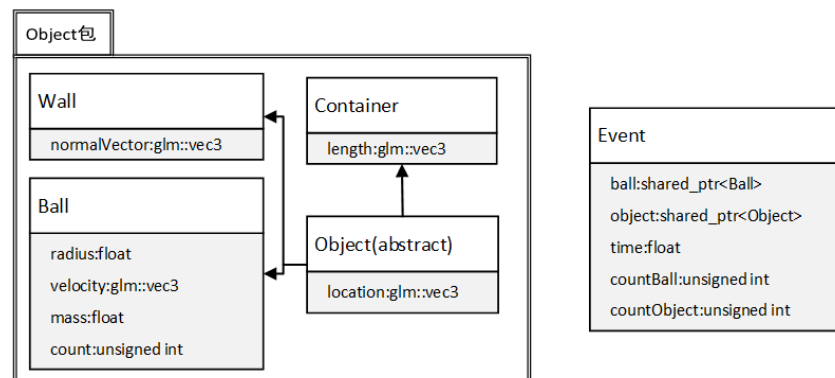


图 3.2 碰撞系统相关类的成员图

3.3.1 动态绑定和继承

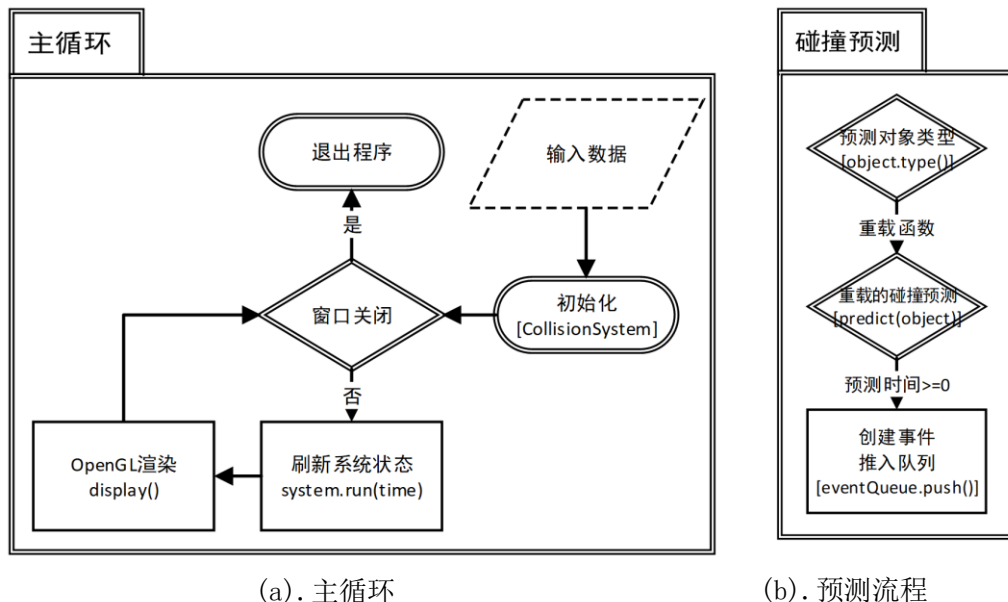
在实现 `Ball::bounce()` 函数时需要解决类继承的问题。在 `Event` 类型中，我们只储存第二碰撞物体的基类指针（如果在事件中为每一种物体创建指针备用，将使 `Event` 类和 `Object` 类继承体系不能相互独立，影响程序的扩展开发。也就是说：如果开发了新的 `Object` 子类型，那么 `Event` 的实现代码也要作出相应的更改）。在调用 `Event::handle()` 函数时，该函数无法确定调用 `Object::bounce()` 的哪个版本。最终，我们为 `Object` 类实现了一个统一的 `type` 接口，提供对象的具体子类型信息，然后使用动态绑定 `dynamic_cast` 来转换指针或应用到对应的子类型。部分代码如下

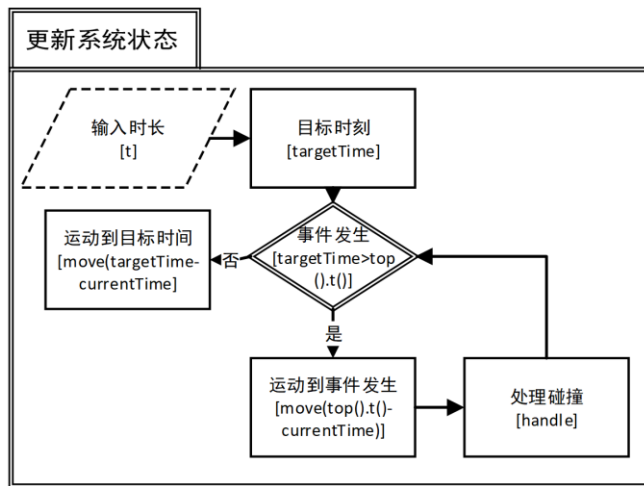
Object.cpp

```
void Ball::bounce(Object &object)
{
    switch (object.type()){
    case Object_type::BALL:
        {this->bounce(dynamic_cast<Ball &>(object));break;}
    case Object_type::WALL:
        {this->bounce(dynamic_cast<Wall &>(object));break;}
    case Object_type::CONTAINER:
        {this->bounce(dynamic_cast<Container &>(object));break;}
    }
}
```

3.4 碰撞系统运行流程图

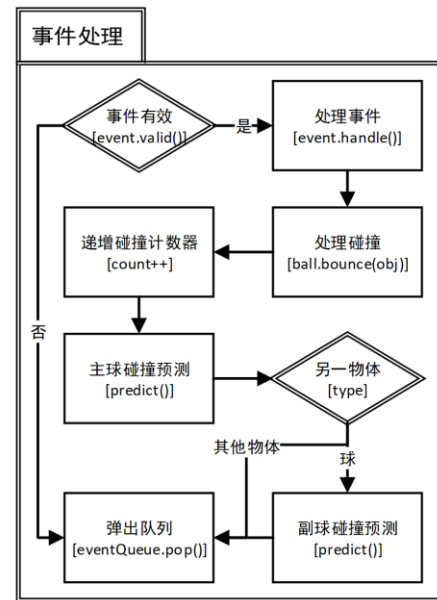
图 3.4 详细地展示了碰撞系统程序的运行流程。总共分为 4 个模块：主循环，更新循环，处理模块，预测模块。后 3 个模块都由 `CollisionSystem` 类管理，对用户隐藏。





(c). 更新循环

图 3.4 程序流程图



(d). 处理循环

第4章 算法探究和分析

本章通过实验探究模拟碰撞程序中涉及到的几种重要算法。

4.1 算法探究 1：时间和事件驱动方法

本节通过实验探究程序运行时间关于输入小球个数的增长模型，以此来比较时间驱动和事件驱动方法模拟的性能，并给出粗略的证明。

4.1.1 数据生成

使用 STL 标准库随机数引擎，我们生成了这样 10 组数据用于测试，每组 3 份：球体质量 1kg，半径 0.5m，间距 0.5m，均匀填充在正方体容器中，容器壁与最外层球体间隔 1m。球体速度沿坐标轴方向分量服从正态分布 $N(0,1)$ 。球体分布可视化如图 3.1。

表 4.1 测试数据特征

| 组别 | 小球个数 | 容器边长 | 数密度 |
|----|-------|------|------|
| 1 | 8 | 4.5 | 0.09 |
| 2 | 64 | 7.5 | 0.15 |
| 3 | 125 | 9 | 0.17 |
| 4 | 512 | 13.5 | 0.21 |
| 5 | 1331 | 18 | 0.23 |
| 6 | 4096 | 25.5 | 0.25 |
| 7 | 8000 | 31.5 | 0.26 |
| 8 | 10648 | 34.5 | 0.26 |
| 9 | 15625 | 39 | 0.26 |

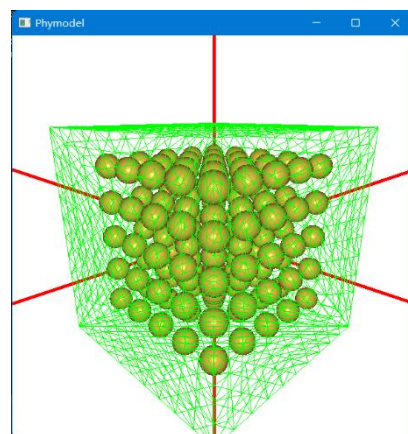


图 4.1 球体分布可视化

4.1.2 实验过程及结果

实验中我们使用 C++11 新特性来测量函数的运行时间，该方法比使用 C 库 `time()` 函数更加精确，部分代码如下

main.cpp

```
#include <chrono>
using namespace chrono;
auto start = system_clock::now();
system.run(10.0f);
auto end = system_clock::now();
auto duration = duration_cast<milliseconds>(end - start);
cout << duration.count() << endl;
```

程序输出 CollisionSystem 模拟运行 10s 所用的时间。为了尽可能排除其他因素影响，实验中程序在相同状态的同一台电脑上运行，每份数据运行两次，取运行时间平均值，再对每组数据取平均，最终得到折线图 4.2。

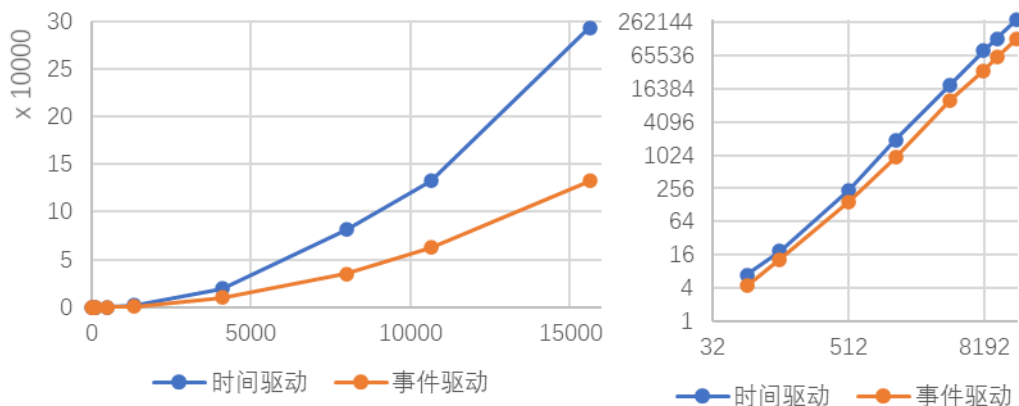


图 4.2 程序运行时间随问题规模的增长图像（右图为对数坐标）
横轴为小球个数，纵轴为运行时间（单位 ms）

从折线图中可以看出，事件驱动所需时间通常是时间驱动的一半。实验证明：事件驱动方法优于时间驱动方法。但对数坐标系显示出两种驱动模型的增长量级都在平方级别左右，仍有很大改进空间。

4.1.3 简要证明

对时间驱动模型，程序运行时间主要取决于步长。步长太短将导致运行时间过长，步长太长会导致模拟准确性下降。最终我们选择 $1/60s$ 作步长，这恰好符合一般显示器开启垂直同步后的帧率，便于绘制 OpenGL 可视化动画。

步长和球体数量将决定时间驱动的计算总量。假设步长为 Δt ，小球个数为 n ，则模拟单位时间调用碰撞检测函数 $\frac{n^2}{\Delta t}$ 次，调用移动函数 n 次，调用碰撞处理函数的次数随系统运动剧烈程度而变化，但总在 $\frac{n \lg n}{\Delta t}$ 到 $\frac{n}{\Delta t}$ 次以内。由此我们可以得到时间驱动模型运行时间的上界

$$T(n) = O(n^2) \quad (4.1)$$

对于事件驱动模型，从代码中可以看出，其在每次处理碰撞时最多需要 n 次优先队列操作，每次优先队列操作时间为对数级别，因此事件驱动的运行时间上界为

$$T(n) = O(n \lg n) \quad (4.2)$$

4.2 算法探究 2：浮点运算误差

计算机进行浮点数运算时将不可避免地产生误差。在实际生产中，浮点误差累积可能造成严重后果。本节采用回退的方法来测量浮点运算产生的误差。

4.2.1 实验方案

使用实验 4.1 中的第 4 组共 3 份数据进行实验，设置时间驱动步长为 $1/3600s$ 。程序使碰撞系统运动指定时间 t ，随后令系统内所有小球速度反向，运行 t 时间，再令小球速度反向，输出所有小球的位置和速度和系统总碰撞次数。如果没有运算误差，小球本应回到初始状态。通过测量小球与初始状态的差异，可以评估浮点运算误差产生的影响。又因为位置和速度的量级存在差异，我们将这两个量分开评估：

$$\text{位置误差 } \Delta \vec{p} = \frac{\sum |\vec{p}_0 - \vec{p}|}{n} \quad \text{速度误差 } \Delta \vec{v} = \frac{\sum |\vec{v}_0 - \vec{v}|}{n} \quad (1.7)$$

4.2.2 实验结果和分析

实验详细数据保存在附件 result_4.2.xlsx。以下是统计分析的结果：

| 运行时间 | 1S | 6S | 11S | 16S |
|-----------|----------|----------|----------|----------|
| 碰撞次数 | 1250 | 8534 | 15824 | 22843 |
| 事件驱动 位置误差 | 0 | 2.01E-06 | 1.401931 | 4.657029 |
| 时间驱动 位置误差 | 0.001345 | 2.677461 | 5.292433 | 6.270913 |
| 事件驱动 速度误差 | 3.91E-12 | 6.35E-06 | 2.012043 | 2.216703 |
| 时间驱动 速度误差 | 0.002285 | 2.122623 | 2.211876 | 2.219462 |

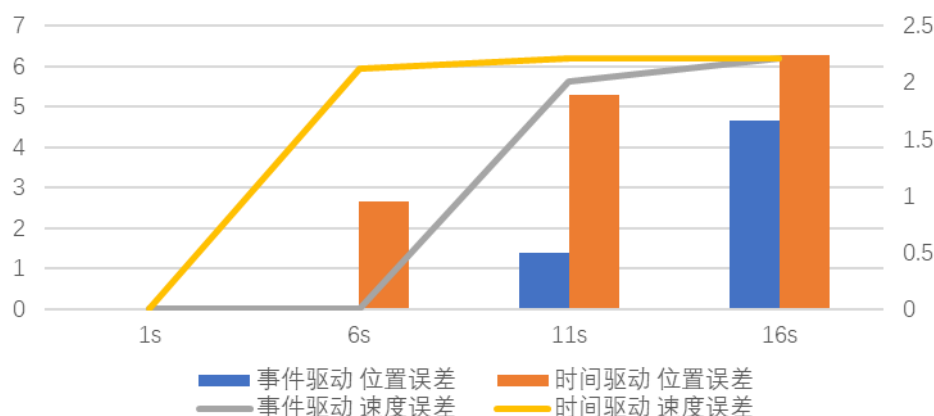


图 4.3 平均误差统计图

本质上，浮点数计算误差主要发生在碰撞处理。数据量较大时，系统单位时间内碰撞次数不变，因此总误差也随运行时间增长。

图 4.3 是组合统计图。左纵轴和柱形图表示位置误差，右纵轴和折线图表示速度误差，横轴表示运行时间。不论看位置还是看速度，事件驱动的精确度总是高于时间驱动。运行 1s 时，两者都能保持位置、速度无误差。运行 11s-16s 时，两者的速度误差增长均放缓。这是因为碰撞处理遵循能量守恒定律，系统总动能不变，速度变化也会被限制在一定范围内。

在 6s 时，时间驱动模拟的小球位置平均偏离了近 1m。选取 6s 时的位置误差数据进行统计，得到散点图 4.4

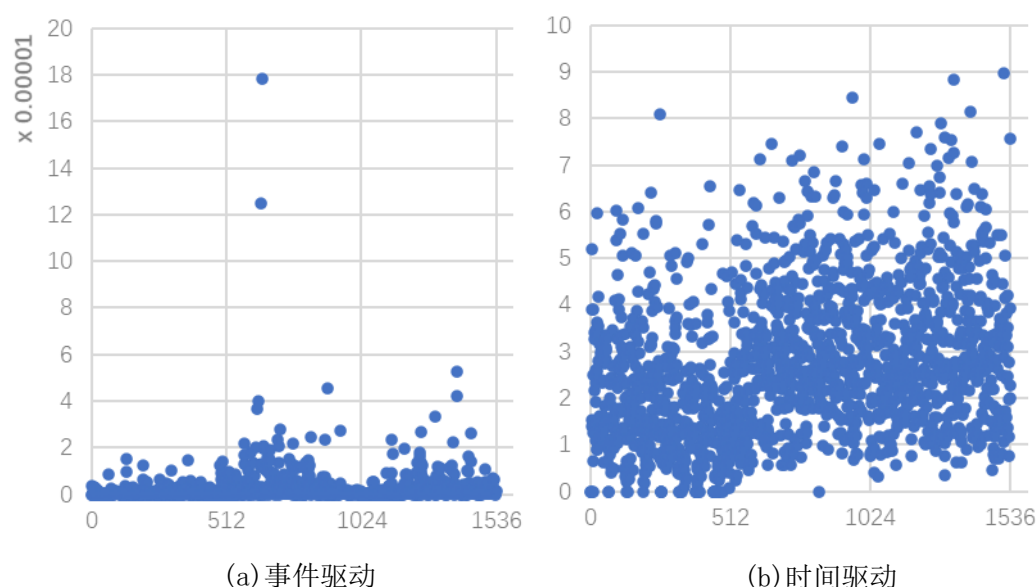


图 4.4 6s 位置误差散点图

从散点图 4.4 可以看出，事件驱动下的小球最大位置误差仅为 0.00018，而时间驱动平均误差就达到了 2.7。事件驱动产生误差的平均值和方差均远远小于时间驱动。

然而，如果时间大于 16s，事件驱动产生的误差也很严重。图 4.5 展示了事件驱动下时间回退后小球的位置。6s 时基本复原，10s 时有不少球发生偏移，16s 时系统已经严重偏离原位置。测量更长时间后的误差已经失去了意义。

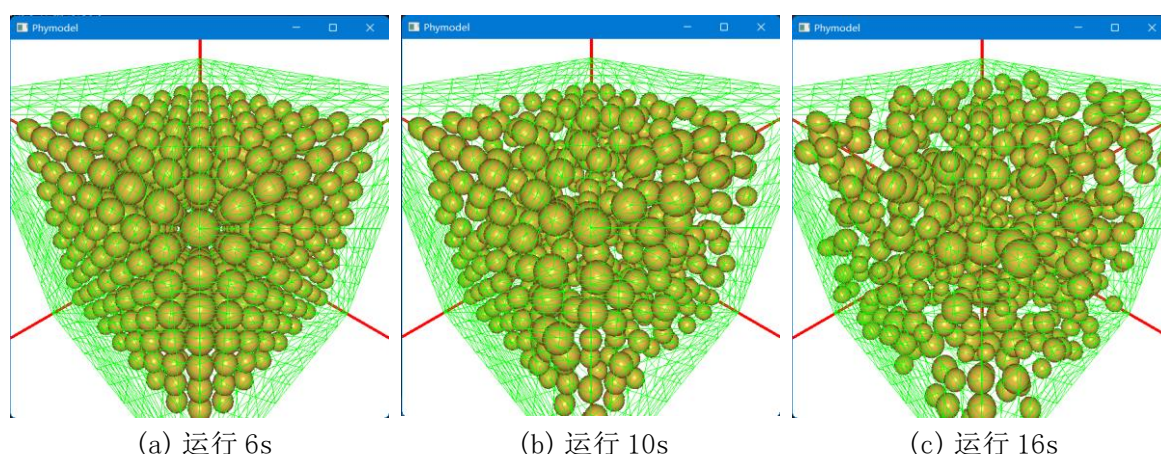


图 4.5 回溯位置可视化

4.3 算法探究 3: 索引优先队列

在图 4.6 中我们看到, 程序使用的内存随运行时间增加而快速增大。图中测试小球数量为 15625, 内存使用量随时间增加而快速增长, 3min 后使用内存接近 200MB, 优先队列长度达到 2124875 (未开启 OpenGL 且设置了队列时间上限)。通过索引优先队列, 可以保证优先队列的长度最多与小球数量呈线性关系。

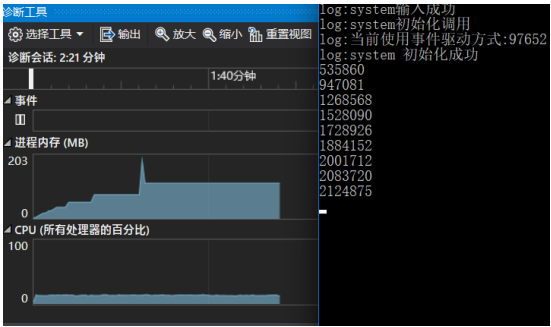


图 4.6 Visual Studio 调试界面

4.3.1 算法简介

索引优先队列的思想是：使用数组下标表示对象，当需要更新对象的值时可以用下标快速找到对象，更新后对象在队列中的位置可能发生变化，以保证整个队列仍然是一个优先队列。索引优先队列使得我们能够访问、更改队列元素。如此，只需要创建一个长度和小球数量相等的队列，仅储存该小球最近的碰撞，省去了无效事件占用的大部分空间。

依照该思想，我们设计了一个简易的索引优先队列。我们并不将其写成 C++ 容器模板，而是作为一个事件管理类，替换系统中原有的 STL 优先队列。

表 4.2 索引优先队列 API

| class Event_mgr | | |
|-----------------|----------------|-----------|
| 返回类型 | 函数名和参数列表 | 说明 |
| | Event_mgr(int) | 设置容量 |
| void | push(Event &) | 推入队列，自动维护 |
| void | pop() | 弹出队首事件 |
| Event & | top() | 返回队首事件 |
| bool | contains(int) | 是否包含某一元素 |

4.3.2 算法实现

为了实现快速索引，我们创建两个数组 pq 和 events。在 events 中储存 Event 对象，对象的下标为事件中第一个球 (ball1) 的编号，events[i] 代表小球 i 即将发生的事件。pq 存储的是 events 的下标，我们把 pq 当作优先队列，但在上浮和下沉操作中比较 pq 中值对应的 events 数组中事件的时间。比如，我们更新小球 1 相关的事件，只需要更新 events[1]，然后去维护 pq。

但我们只能遍历 pq 才能找到元素值 1 的位置，再通过下标快速上浮下沉。为了快速找到 pq 中元素值对应的下标，我们额外设置数组 qp，储存 pq 中元素值在 pq 数组中的下标。更新与小球 i 相关的事件的操作可以简化为以下步骤：

- events[i] = new event

- 通过 $qp[i]$ 的值 j 知道数组 pq 中值为 i 的下标为 j （找到小球 i 对应的事件在优先队列中的位置 j ）
- 对 $pq[j]$ 进行上浮或下沉操作
- 在交换 pq 数组中元素值同时交换 qp 对应元素的值

不妨模拟一次实际的操作：更新小球 6 的事件

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------|---|-----|-----|-----|-----|-----|-----|-----|
| pq | | 7 | 3 | 1 | 2 | 5 | 4 | 6 |
| qp | | 3 | 4 | 2 | 6 | 5 | 7 | 1 |
| events | | 3.3 | 4.2 | 2.3 | 9.7 | 8.5 | INF | 1.2 |

a. 原索引优先队列

| | | | | | | | | |
|--------|--|-----|-----|-----|-----|-----|-----|-----|
| pq | | 7 | 3 | 1 | 2 | 5 | 4 | 6 |
| qp | | 3 | 4 | 2 | 6 | 5 | 7 | 1 |
| events | | 3.3 | 4.2 | 2.3 | 9.7 | 8.5 | 0.5 | 1.2 |

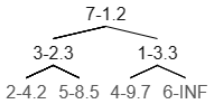
b. 更新 $events[6]$ ，通过 $qp[6]$ 快速得到 $pq[7]$

| | | | | | | | | |
|--------|--|-----|-----|-----|-----|-----|-----|-----|
| pq | | 7 | 3 | 1 | 2 | 5 | 4 | 6 |
| qp | | 3 | 4 | 2 | 6 | 5 | 7 | 1 |
| events | | 3.3 | 4.2 | 2.3 | 9.7 | 8.5 | 0.5 | 1.2 |

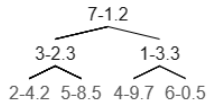
c. $pq[7]$ 两次上浮， $qp[6]$ 随之交换（第二次为虚线）

| | | | | | | | | |
|--------|--|-----|-----|-----|-----|-----|-----|-----|
| pq | | 6 | 3 | 7 | 2 | 5 | 4 | 1 |
| qp | | 7 | 4 | 2 | 6 | 5 | 1 | 3 |
| events | | 3.3 | 4.2 | 2.3 | 9.7 | 8.5 | 0.5 | 1.2 |

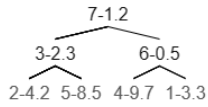
d. 完成排序



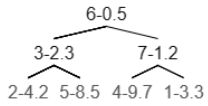
A. 原二叉堆



B. 更新元素



C. 第一次上浮



D. 第二次上浮

图 4.7 索引优先队列操作可视化

4.3.3 实验结果

使用索引优先队列后，程序使用内存空间大幅缩小，小球数量为 15625 时仅使用不到 4MB 内存，比原方法减小了 50 倍；优先队列长度不大于小球数量。且因为自动处理了无效事件，程序运行速度得到了一定提升，能够同时运行 OpenGL 可视化，帧率在 3-4 帧每秒。

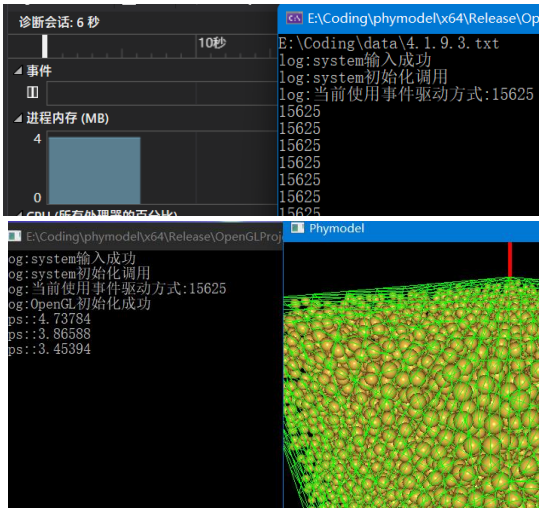


图 4.8 索引队列内存使用和 OpenGL 帧率

4.4 算法探究 4：空间划分

在 4.3 节中，我们使用索引优先队列对程序使用空间进行了优化。但其运行速度仍然不够理想。本节简要介绍如何使用空间划分提高程序运行速度，不进行实验。

空间划分是游戏程序设计中重要的优化思想，在场景管理、渲染、物理等方面应用广泛。统计发现，随着球体数目的增加，程序运行时间主要花费在碰撞预测上。通过空间划分可以避免远距离物体的碰撞预测，从而大幅节省时间。比如在实验 4.1 的第九组数据中，每次碰撞后 2 个小球一共要进行 31250 次碰撞预测。而如果使用空间划分，最多只需要与两球相邻的共 34 个网格中的最多 918 个小球进行 1458 次预测。空间划分也会带来额外的时间开销，需要在每次 move 操作时监测小球在网格间的移动，不过监测移动花费的时间较少。

空间划分方法主要有以下几种：四叉树/八叉树、层次包围盒树、BSP 树、k-d 树等。当空间中物体形状各异、大小不同时，使用树状结构递归划分空间是最佳选择。但考虑到后期实验使用球体规格相同，我们只需要采用最简单的均匀划分方法即可。均匀划分空间的方法将容器划分为大小相等的网格

(Grid)，网格中包含指向在网格中的球的指针，每个球也附带指向自己所在网格的指针。均匀划分空间使得我们可以通过下标运算快速找到网格位置、判断球体所属网格、监视球体运动。从上面的描述可以知道，空间划分将碰撞处理的时间复杂度限定在线性级别，其常数因子与粒子的数密度和网格的大小呈正相关。

第5章 OpenGL 可视化

5.1 模型构造

本节介绍在 OpenGL 中绘制网格平面和球体的一些细节。

5.1.1 绘制思路

如果系统中物体数量增多，那么采用传统方法为每一个物体建立模型、计算顶点、向 OpenGL 传输数据进行渲染就会耗费大量时间。事实上，可以只用一个 C++ 调用就告诉 OpenGL 渲染一个对象的多个副本，这种方法在 OpenGL 中称为实例化。

按照实例化的思想，我们只需要创建一个位于(0,0,0)，法向量 $n = (0,0,1)$ 的平面模型，然后告诉 OpenGL 使用实例化方法绘制它的多个副本。对于每个副本，改变其位置和法向量，就得到我们所需要的平面。

5.1.2 计算旋转矩阵

在 Wall 类中，我们储存平面的位置和法向量 normalVector，但在 OpenGL 中储存模型的顶点。因此，需要根据每个平面的位置和法向量计算如何移动顶点，也就是计算对应的平移、旋转矩阵。

已知旋转前法向量为 z ，旋转后法向量为 p ，可以推出向量间夹角 θ 和旋转轴 k ：

$$\theta = \arccos(\vec{a} \cdot \vec{b}) \quad \vec{k} = \vec{a} \times \vec{b} \quad (5.1)$$

在三维空间中旋转一个向量可以使用罗德里格旋转公式。应用该公式可以得到如下关系：

$$\vec{a} = \vec{b} \cos \theta + (\vec{k} \times \vec{b}) \sin \theta + \vec{k}(\vec{k} \cdot \vec{b})(1 - \cos \theta) \quad (5.2)$$

如果用 K 表示叉乘中的反对称矩阵

$$\mathbf{K} = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix} \quad (5.3)$$

则(5.2)式又可表示为

$$\vec{a} = \mathbf{R}\vec{b} \quad (5.4)$$

其中:

$$\mathbf{R} = \exp(\theta \mathbf{K}) \quad (5.5)$$

C++实现如下

main.cpp

```
glm::mat4 buildRotate(glm::vec3 vectorBefore, glm::vec3
vectorAfter)
{
    glm::vec3 rotationAxis;
    float rotationAngle;
    glm::mat4 rotationMatrix;
    rotationAxis = glm::cross(vectorBefore, vectorAfter);
    rotationAngle = acos(glm::dot(glm::normalize(vectorBefore),
glm::normalize(vectorAfter)));
    rotationMatrix = glm::rotate(glm::mat4(1.0f), rotationAngle,
rotationAxis);
    return rotationMatrix;
}
```

5.1.3 曲面细分着色器

为了方便观察,我们不使用填充整个平面,而是绘制网格。在平面模型中只记录了正方形平面的四个顶点,不应当采用计算网格顶点、逐一绘制线图元的方法。这项工作可以交给 OpenGL 中的曲面细分着色器,它可以生成并操控大量网格形式的三角形来渲染复杂的表面和形状。这里我们使用其最基础的功能:生成三角形网格。只需要配置曲面细分参数,指定图元类型和曲面细分级别即可:

tcs_plane.glsl

```
layout (triangles, equal_spacing, cw) in;
void main(void)
{
    gl_Position = (gl_TessCoord.x * gl_in[0].gl_Position) +
                  (gl_TessCoord.y * gl_in[1].gl_Position) +
```

```
(gl_TessCoord.z * gl_in[2].gl_Position); }
```

图 5.1 展示了使用曲面细分着色器绘制的平面和容器：

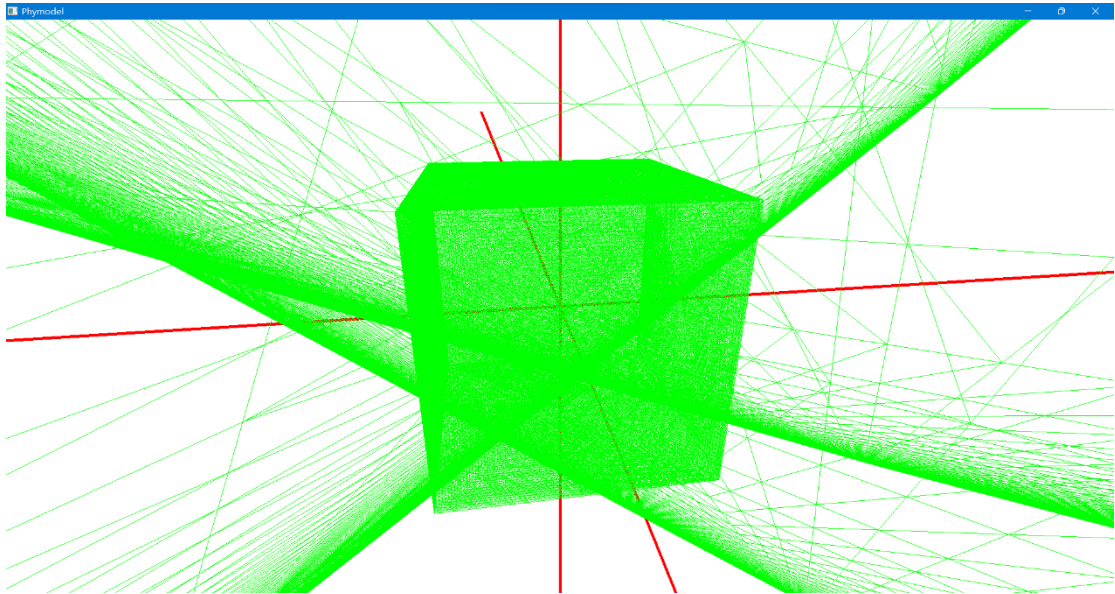


图 5.1 使用曲面细分着色器绘制的平面和容器

5.1.4 数据传输优化

我们尝试使用以下方法绘制大量小球：

1. 使用 for 循环为每个小球重复调用绘制命令、传递统一变量

main.cpp

```
for(auto const Ball& ball;system.Balls){  
    lMat = glm::translate(glm::mat4(0.0f),ball.loc());//计算相关矩阵  
    glUniformMatrix4fv(...);//传递统一变量  
    glDrawArrays(...);//绘制
```

这一方法效率极低，每次管线上只有一个小球的数据通过，不能发挥显卡并行处理的硬件优势。使用该方法绘制小球数量超过 100 后帧率明显下降。

2. 使用实例化和统一变量数组

使用统一变量数组，只需要传递一次数据即可。

vs_sphere.glsl

```
uniform vec3 locations[100];  
void main() {gl_Position = locations[gl_InstanceID];}
```

然而在 OpenGL 着色器中，统一变量大小具有严格的限制。经过尝试，当统一变量数组大小超过 100 后，GLSL 编译报错。该方法无法绘制大量小球。

3. 使用实例化和 OpenGL 缓存

使用缓存对象才是在 OpenGL 中传递数据的最佳方式。缓冲对象是由 OpenGL 维护的一块内存区域，并提供了多种方式创建、更新缓存数据。

vs_sphere.glsl

```
layout (location = 0) in vec3 vertPos;
layout (location = 1) in vec3 vertNormal;
layout (location = 2) in float scales;
layout (location = 3) in vec3 model;
```

main.cpp

```
//VBO3:model
std::vector<glm::vec3> models; //vec3
for (auto const &i : balls)
    models.push_back(i->loc());
glBindBuffer(GL_ARRAY_BUFFER, sphereVbo[3]); //传送模型位置
glBufferData(GL_ARRAY_BUFFER, models.size() * sizeof(glm::vec3),
&models[0], GL_DYNAMIC_DRAW); //刷新缓冲区
glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(3);
glVertexAttribDivisor(3, 1); //配置实例化更新数据
//draw
glEnable(GL_CULL_FACE); //背面剔除，加快速度
glFrontFace(GL_CCW);
glDepthFunc(GL_LEQUAL);
glDrawArraysInstanced(GL_TRIANGLES, 0, mySphere.getNumIndices(),
balls.size());
glDisable(GL_CULL_FACE);
```

考虑到在 C++ 程序和 OpenGL 间传递数据效率较低，且 GPU 浮点运算性能比 CPU 好，我们将小球的位置和大小数据写入 OpenGL 缓冲区，由顶点着色器计算平移、缩放矩阵。这样减少了传输的数据量，加快了计算速率。经过测试，使用该方法可以在使用 Blinn-Phong 光照效果时流畅显示 20000 以上个小球。

5.2 动画循环设计

由于使用了 OpenGL 动画，动画的绘制方式是双缓冲，逐帧绘制。为了使模拟尽可能贴近实际，考虑绘制和事件处理时间的关系就尤为重要。根据《Game Programming Pattern》对游戏循环的论述，我们采用“追逐时间”的方法设计动画循环，如图 5.2 所示。

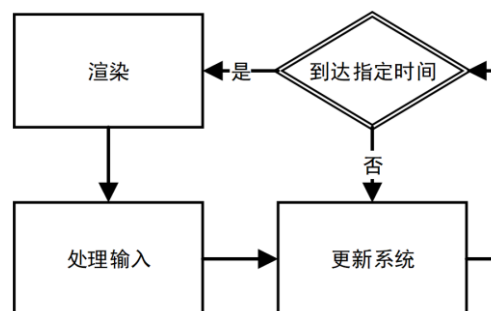


图 5.2 动画循环设计

第6章 模拟实验：理想气体

本章我们将在具体的模拟实验中检验碰撞系统的有效性。

6.1 实验方案

使用构建好的物理系统，模拟 $0.027\mu\text{m}^3$ 体积内理想气体分子的运动，监测其宏观状态（温度和压强）是否稳定，以此来验证碰撞系统的有效性。

6.1.1 实验数据

查阅 CRC 手册可以获得经过精确测量的氧气分子数据。以标准状态（273.15K, 1atm）下的数据为基础，根据 2.3 节中的热力学分析，可以得到以下参数：

| 名称 | 数据 |
|------------|--|
| 氧气分子直径 | $\Phi = 0.346\text{nm}$ |
| 氧气分子质量 | $m = 5.3 \times 10^{-26}\text{kg}$ |
| 阿伏伽德罗常数 | $N_A = 6.022\,140\,76 \times 10^{23}\text{mol}^{-1}$ |
| 玻尔兹曼常数 | $k_B = 1.380649 \times 10^{-23}\text{J} \cdot \text{K}^{-1}$ |
| 理想气体摩尔体积 | $V_m = 22.71095464 \times 10^{-3}\text{m}^3 \cdot \text{mol}^{-1}$ |
| 标况下氧气分子间距 | $d = \sqrt[3]{\frac{V_m}{N_A}} = 3.353473 \times 10^{-9}\text{m}$ |
| 气体分子速率分布方差 | $\sigma_x^2 = \frac{k_B T}{m} = 2.667499 \times 10^2$ |
| 每条边上气体分子个数 | 89 |

根据上表数据编写 C++程序生成 $0.027\mu\text{m}^3$ 体积正方体容器和其中均匀分布的 704969 个气体分子。使用 Excel 对生成的气体分子进行统计，可以得到速率分布直方图。由图可以看出，生成的气体分子很好地符合麦克斯韦分布律。

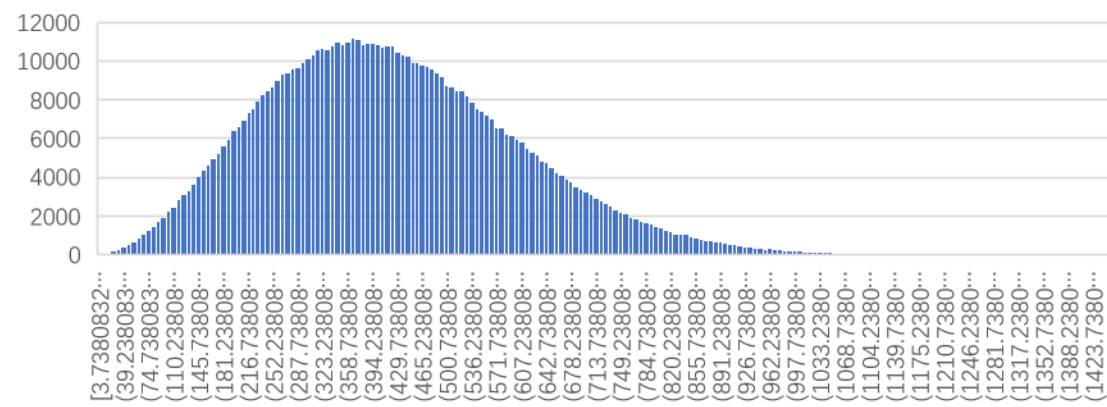


图 6.1 气体分子速率分布直方图

6.1.2 参数测量

由热力学定律可知, $T = \frac{2\overline{\epsilon_k}}{2k_B}$ 。Ball 类提供了返回动能的函数, 只需遍历系统 Ball 队列对粒子动能求和即可算得系统温度。同理可以通过统计粒子与容器壁碰撞的动量该变量来计算压强。

6.2 实验过程

由于 OpenGL 透视投影近裁剪平面的限制, 我们无法对微米至纳米尺度的粒子和容器进行可视化。程序每个循环模拟系统 1 微秒运动, 并输出系统温度和压强。总共运行 20 微秒后停止, 整理数据。

6.3 实验结果

组合统计图 6.2 展示了模拟运行 20 微秒内理想气体的宏观状态量变化。

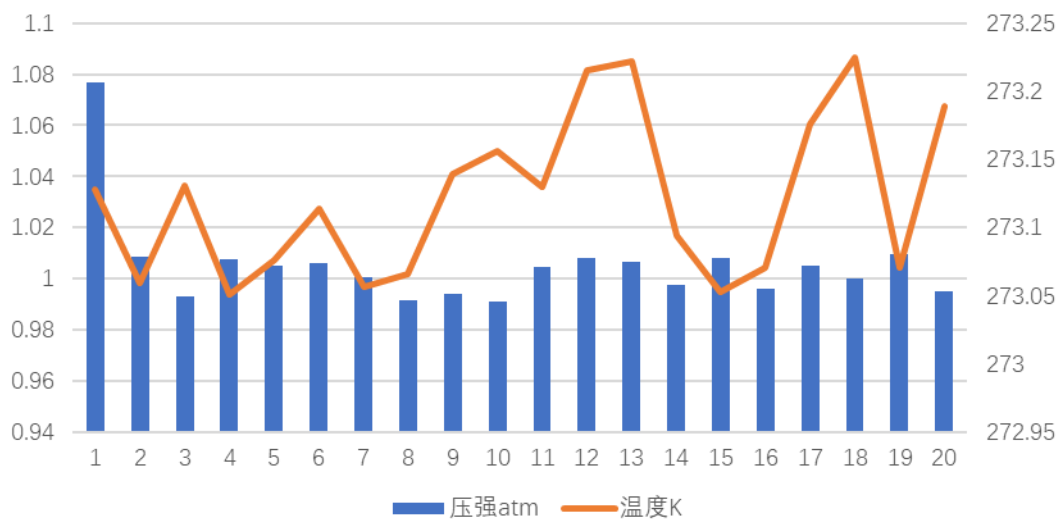


图 6.2 极短时间间隔内理想气体的宏观状态量变化
(柱形图和左坐标轴为压强, 折线图和右坐标轴为温度)

因为微观状态下的理想气体内部并不稳定, 且测量时间间隔极短, 所以宏观状态量不可能恒定不变。图中理想气体状态量的波动在实验误差范围内, 因此实验是成功的, 碰撞程序的稳定性得到了充分的验证。

第7章 成果应用和感悟反思

事实上, 第六章的理想气体模拟实验就是一次成功的成果应用。在程序设计过程中, 我们充分考虑到了程序的扩展性, 严格遵守面向对象程序设计原则, 规范应用程序接口, 为增量开发做好准备。若在源码基础上进行改进, 该程序还可以应用于物理学的其它方面; 添加相互作用力后甚至可以用于社会学集群模型的研究, 这在参考资料[25][26]中有所体现。可见模型思维的跨学科作用。

在为期半年的研究性学习中, 我们阅读了大量物理、计算机方面的书籍、论文, 深入学习了刚体力学、计算机程序设计、算法分析、数据结构、计算机

图形学方面的内容，并掌握了一定的技能，能够科学地思考问题、设计方案、解决问题。通过小组合作，我们增进了同学之间的友谊，在协作学习、研究讨论中锻炼了合作、沟通能力。

在本次研究性学习中，我们大量采取实验和分析的研究方法，体现科学研究的严谨客观。物理组员认真查找资料，思考物理分析方法，为程序编写做铺垫；程序组员总计编写程序代码超过 2000 行，相当于计算机系普通大一学生的期末作业。但由于篇幅限制，我们未能对论文中谈及的其它优秀算法进行实验评估，考虑根据输入规模选择不同算法来使程序运行效率达到最高。

本次研究性学习程序代码和实验数据在 Github 上开源，仓库地址为：<https://github.com/Tiandians/Phymodel>。在 Release 页面发布了各个实验的编译后程序和输入数据，也发布了所有实验的数据和结果集合包。该实验可以在任何支持 OpenGL4.0 及以上的计算机上复现，也可以直接使用打包的原始实验结果数据进行分析。

第8章 参考文献

第一章

[1]王振泽. 计算机仿真技术的发展及应用[J]. 电子技术与软件工程, 2017(4):170.

第二章

[2]舒幼生. 力学:物理类[M]. 北京大学出版社:北京, 2005.

[3]姜那. 3D 游戏中碰撞检测算法的研究[D]. 吉林农业大学, 2014.

[4]周游, 刘程, 涂灿辉, 刘华, 郑才龙. 二维完全弹性碰撞的理论研究[J]. 物理通报, 2017, {4} (12):46-50.

[5]谷宁. 基于刚体特性的物理仿真引擎的设计与实现[D]. 东北师范大学, 2009.

[6]李乾. 虚拟三维场景中刚体碰撞响应仿真研究与应用[D]. 北京化工大学, 2019.

[7]王裕平. Maxwell 速率分布的推导

[EB/OL]. <https://wenku.baidu.com/view/d4a07478cfc789eb172dc8f6.html>, 2014-06-21.

[8]爱因斯坦. 爱因斯坦文集 第2卷[M]. 商务印书馆:北京, 2017:.

[9]丛伟. 深度探究“布朗运动”演示教具的设计与实现[D]. 哈尔滨:哈尔滨师范大学, 2020.

[10]付文玉, 侯锡苗, 贺丽霞, 等. 少体硬球系统的动力学与统计研究[J]. 物理学报, 2005, 第6期:P2552-2556.

[11]School of Mathematics and Statistics University of St Andrews, Scotland. James Clerk Maxwell on the nature of Saturn's rings[EB/OL]. https://mathshistory.st-andrews.ac.uk/Extras/Maxwell_Saturn/, 2006-03.

[12]James Clerk Maxwell Foundation. Saturn's Rings[DB/OL].

https://www.clerkmaxwellfoundation.org/Saturn-s_Rings.pdf, 2015.

[13]周济林, 孙义燧. 行星环动力学[J]. 天文学进展, 1996, 第2期:P130-138.

[14]陈道汉. 行星环(II)[J]. 天文学进展, 1984, 第2期:P102-110.

[15]张立健. 高中生物理模型建构能力培养的策略研究[D]. 福建师范大学, 2006.

第三章

[15]Robert Sedgewick, Kevin Wayne. 算法（第4版）[M]. 人民邮电出版社:北京, 2012.

第四章

[16]Nystrom, Robert. Game Programming Patterns[M]. Ganever Benning:England, 2014.

[17]KillerAery. 空间划分的数据结构(二叉树/八叉树/BVH树/BSP树/k-d树) [EB/OL].

<https://www.cnblogs.com/KillerAery/p/10878367.html>, 2019-05-26.

[18>nullzx. 索引优先队列的工作原理与简易实现 [EB/OL].

<https://www.cnblogs.com/nullzx/p/6624731.html>, 2017-03-2.

第五章

[19]Graham Sellers, Richard S. Wright, Jr., etc. OpenGL 超级宝典（第7版）[M]. 人民邮电出版社:北京, 2020.

[20]John Kessenich, Graham Sellers, Dave Shreiner. OpenGL 编程指南（原书第9版）[M]. 机械工业出版社:北京, 2017.

[21]Gordon V. Scott, Clevenger John. 计算机图形学编程：使用 OpenGL 和 C++[M]. 人民邮电出版社:北京, 2020.

[22]Wikipedia. Rodrigues' rotation formula[DB/OL].

https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula, 2021-06-05.

[23]闪电的蓝熊猫. 从0开始的OpenGL学习（二十六）-实例数组

[EB/OL]. <https://www.jianshu.com/p/571be8c027db>, 2017-12-10.

第六章

[24]CRC Press. Handbook of Chemistry and Physics [DB/OL].

<https://hbcpc.chemnetbase.com/faces/contents/ContentsSearch.xhtml>, 2021-06-05.

[24]National Institute of Standards and Technology. Physical Reference Data

[DB/OL]. <https://www.nist.gov/pml/productsservices/physical-reference-data>, 2021-06-05.

第七章

[25]Ele 实验室. 计算机模拟鱼儿结群行为下的群体策略 [EB/OL].

<https://www.bilibili.com/video/BV1tq4ylj7TW>, 2021-05-24.

[26]Helbing, D., Farkas, I. & Vicsek, T. Simulating dynamical features of escape panic. Nature 407, 487 - 490 (2000). <https://doi.org/10.1038/35035023>

研究性学习课题评价表

| | |
|--------|--|
| 课题导师 | 黄罗华 |
| 课题名称 | 计算机模拟刚性球体系统的物理分析、算法设计和 OpenGL 可视化 |
| 课题成员 | 朱宝林、尤比佳、林佳诚、张宏彬 |
| 小组自评 | <p>在为期半年的研究性学习中，我们阅读了大量物理、计算机方面的书籍、论文，深入学习了刚体力学、计算机程序设计、算法分析、数据结构、计算机图形学方面的内容，并掌握了一定的技能，能够科学地思考问题、设计方案、解决问题。在小组合作中，我们增进了同学之间的友谊，在协作学习、研究讨论中锻炼了合作、沟通能力。</p> <p>通过本次研究性学习，我们深刻认识到计算机科学技术在现代科学研究中的重要作用。组员能够运用学科交叉的思路拓展视野，联系不同领域的知识，这将极大促进我们今后的学习生活。</p> <p>小组成员签名： 朱宝林、尤比佳、林佳诚、张宏彬</p> <p style="text-align: right;">日期：2021 年 7 月 22 日</p> |
| 指导教师评价 | <p>指导教师签名：</p> <p style="text-align: right;">日期：</p> |