

README

1 概况与使用

概述

本次项目我实现了一个自己的有趣的黑洞系统，当然这个“黑洞”显然不那么物理，但是大体上拟合了黑洞的吸积、爆发、喷流等粒子效果（因为没有完全符合相对论尤其是光照肯定不是黑洞的光学特性），但我们要的可能只是一种视觉效果——至少有趣且震撼——单就这点而言，项目的意义或许已经实现了！在第一张黑洞的真实渲染图出现之前，为了便于读者观看和直观理解，各出版物及媒体展示的黑洞“假想图”都是光盘状模型，直到出现了第一张明确视界范围的黑洞真实图片。我们遵从方便的原则，沿用光盘的表示方法，虽然他没有那么物理，但是他很几何！

由于项目体量较大，我们可能只挑出来有特色的地方，或者更功利地，对应于得分点或者加分点的地方予以说明。还是和以前类似，此项目 make 时使用了 debug 编译模式，并且 cmakeLists 沿袭了过去两个作业的风格。经测试项目在 visual studio -debug 中可以正常运行。由于代码书写时在 time 和 gui 方面的考量不太严谨，所以这个程序实际上跑起来可能会有一个小的“bonus”(手动狗头)，这点读者可以自行探索，当然在正文的最后我也会揭秘；当然我们称之为 bonus 就是因为他虽然是一个小 bug，但是会带来非常有趣的特效，所以本人没有再做修改。

我们的文件结构树如下：

```
project/
  src/
    main.cpp
    particle_system.cpp    # 粒子系统
    camera.cpp
    shader.cpp             # 着色器管理
    gui.cpp
    CMakeLists
    script_parser.cpp      # 特效脚本解析
  shaders/
    particle.vs            # 粒子顶点着色器
    particle.fs            # 粒子片段着色器（包括光照计算）
    blackhole.vs           # 黑洞顶点着色器
    blackhole.fs           # 黑洞片段着色器
  scripts/                 # This is usually empty
  include/                 # pass
  out/                     # pass
  .vs/                     # pass
  build/                   # pass
```

```
third_party/    # pass
CMakeLists.txt
CMakeSettings.json
README.pdf      # 本次报告
```

当然本报告在撰写的过程中并不能保证项目是可以正常上传到教学网的, 所以**如果无法**上传到教学网, 本人已经将他 push 到 gitlab 上并且链接[在这里](#), 读者可以自行下载并且 make。

功能与操作

我们的项目包括对 12000 个 (default) 球体粒子的实时渲染, 所以运行时可能需要较大的内存。我们实现了比较物理的引力模拟: 牛顿引力 + 相对论, 以及完整的光照系统: 环境光 + 漫反射 + 镜面反射。项目还包括一个交互式相机控制系统和实时参数调节或者允许用户指定特效动画的 gui。

方便起见, 我们用枚举的方式罗列我们的 gui 参数界面, 也便于读者对系统的功能有一个初步的认识:

- 粒子参数 (Particle Parameters)
包括 Black Hole Mass: 黑洞质量 (100-10000); Particle Lifetime: 粒子寿命 (1-30 秒); Spiral Strength: 螺旋强度 (0-5); Turbulence Strength: 湍流强度 (0-2); Accretion Disk Radius: 吸积盘半径 (5-50); Particle Size: 粒子大小 (0.01-0.5); Color Intensity: 颜色强度 (0.5-5)
- 高级光照 (Advanced Lighting)
包括 Light Color: 光源颜色 (RGB); Light Intensity: 光照强度 (0-3); Directional Light: 切换点光源/方向光; Light Position/Direction: 光源位置或方向; 预设光源: 快速切换不同光源设置
- 特效控制 (Special Effects)
 - 伽马射线喷流: 允许启用/禁用喷流; 可以调节喷流强度、角度、速度; 允许设置喷流方向
 - 超新星爆炸: 可以手动触发爆炸效果; 可以调节爆炸强度
 - **事实上, 我们通过对黑洞质量, 吸积盘半径, 旋流强度等组合设置可以得到更多有趣的特效, 在拖动滑块的时候我们也可以观察到不同物理参数的过度动画, 也十分有趣**
- 相机控制 (Camera Control)
允许调整相机的观察中心位置 (Target); 允许调节相机到观察点的欧氏距离 (Radius); 允许调节相机的视场角 (FoV); 添加了 reset 按钮, 可以重置相机位姿 (用户视界); 支持显示相机的实时信息。
控制说明:
 - 右键拖拽: 围绕目标点旋转;
 - 鼠标滚轮: 缩放 (靠近/远离目标点);
 - 右键 + WASD: 移动目标点位置;

- 右键 +Q/E: 上下移动目标点;
- ESC: 程序退出
- 性能 (Performance)

包括显示当前的粒子数量以及设备的帧率信息。
- 一个省略: 笔者本来实现了 gui 的一个特效脚本的直接运行的选项, 就是在 scripts 文件夹下保留特定的.effect 文件, 他们存储了预先调好的系统的参数列表, 可以在交互时直接读取并且演示。但笔者事后发现这个功能是大而无当的, 因为你只有自己体验后才知道参数的搭配比例, 并且同一个特效不同人有不同的看法, 这也解释了为什么 scripts 文件夹是空的。

项目的运行效果大致如下: (报告不方便展示动态内容还请读者自行探索)

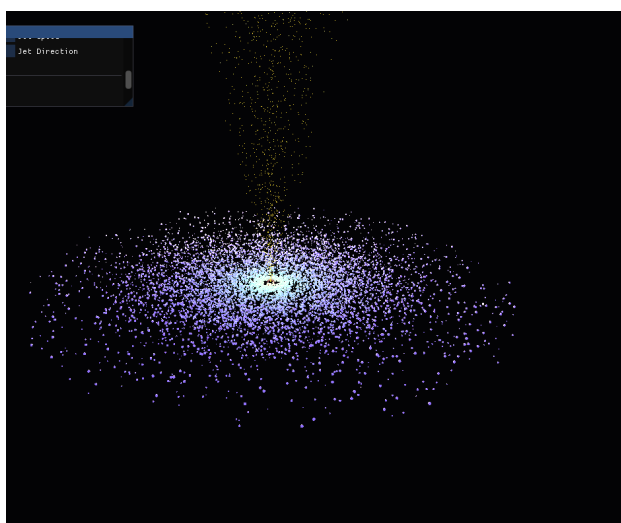


Figure 1: jet with default parameters

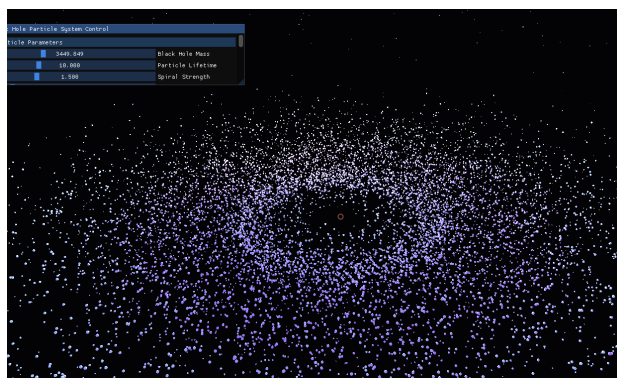


Figure 2: Silhouette during the transition process of a decreasing black hole radius

2 项目细节

2.1 粒子系统和物理模拟

项目采用了实例化技术处理大批量粒子, 将几何数据和实例数据分离从而实现高效渲染:

```
1 glEnableVertexAttribArray(2); // 位置
2 glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, sizeof(Particle), (void*)0);
3 glVertexAttribDivisor(2, 1); // 每个实例更新一次
```

我的当前粒子系统实现是为获得最佳实时视觉效果而设计的。它与更物理的“标准实现”之间存在明显的取舍, 主要体现在简化计算以增强视觉稳定性和可控性, 具体对照如下:

特性	我的实现（视觉优先）	标准实现（物理启发）	核心取舍 (Trade-off)
引力常数	我已省略 G 和 c , 仅使用经验缩放。	标准实现包含 G 和 c 作为明确常数。	取舍: 简化 vs. 可缩放性。我放弃了物理关联, 以获得更易调整的参数。
事件视界/吞噬	我使用固定的吞噬半径 $0.5f$ 。	标准实现使用随质量变化的史瓦西半径 R_s 。	取舍: 稳定可见 vs. 物理准确。我保证了粒子无论质量大小都快速消失, 视觉稳定。
相对论修正	我的修正因子是 $\text{relFactor} = 1.0 + 2.0/(r + 0.5f)$ 。	标准公式通常与 R_s 相关, 如 $\text{relFactor} = 1.0 + k \cdot R_s/r$ 。	取舍: 吞噬强度 vs. 公式关联。我的公式提供了强大的径向拉力, 以防止粒子被甩出。
螺旋吸积盘	我的螺旋力 $\mathbf{F}_{\text{spiral}}$ 仅基于固定的 \mathbf{Y} 轴和 $\text{params.spiralStrength}$ 。	标准实现将螺旋力与引力 $\mathbf{F}_{\text{gravity}}$ 关联起来。	取舍: 简易控制 vs. 动态响应。我的力是恒定的, 易于获得稳定的圆盘旋转。
阻力 (Drag)	我的实现未包含任何阻力或辐射项。	标准实现通常包含 $\mathbf{v}_{\text{new}} \propto \mathbf{v}_{\text{old}} \cdot (1 - \text{drag})$ 的阻力。	取舍: 运动自由 vs. 轨道衰减。我的粒子运动更“干净”, 但标准阻力更能模拟能量损耗和快速吸积。
潮汐力/寿命	我的衰减因子是 $\propto 1/(r^2 + \dots)$ 的经验形式。	标准实现倾向于使用更接近物理定律的 $1/r^3$ 形式; 而且事实上潮汐力并不直接影响粒子寿命。	取舍: 平滑衰减 vs. 物理定律。我的 $1/r^2$ 形式衰减更平滑, 但 $1/r^3$ 更符合真实潮汐力。

我的视觉优先模型是为了实现“粒子不容易被甩出去”和“形成稳定圆盘”的目标，所以修正以换取稳定性和可控性。虽然这牺牲了严格的物理准确性，但对于实时图形和视觉效果而言，是一个理想的选择。（这一点，我在对比实验中写了更加物理的方法，但确实没有什么视觉效果，所以也不多做报告。笔者可以根据我的上述对比图表自行修改出一个更物理的写法对照效果。）

```
1 void ParticleSystem::update(float deltaTime, const glm::vec3& cameraPosition) {
2     std::random_device rd;
3     std::mt19937 gen(rd());
4     std::uniform_real_distribution<float> dist(0.0f, 1.0f);
5     std::uniform_real_distribution<float> distColor(0.5f, 1.0f);
6
7     if (explosionActive) {
8         updateExplosionParticles(deltaTime);
9         explosionTimer -= deltaTime;
10        if (explosionTimer <= 0.0f) {
11            explosionActive = false;
12        }
13    }
14
15    if (params.enableJet) {
```

```

16     updateJetParticles(deltaTime);
17 }
18
19 for (auto& p : particles) {
20     if (p.life <= 0.0f && p.type == 0) { // 只重置正常粒子
21         resetParticle(p, gen, dist, distColor);
22         continue;
23     }
24
25     // 计算到黑洞的距离
26     float distance = glm::length(p.position);
27
28     if (distance < 0.5f && p.type == 0) {
29         // 粒子被黑洞吞噬
30         p.life = 0.0f;
31         continue;
32     }
33
34     glm::vec3 gravityDir = glm::normalize(-p.position);
35
36     // 牛顿引力 + 相对论经验修正
37     float gravity = params.blackHoleMass / (distance * distance);
38     float relFactor = 1.0f + 2.0f / (distance + 0.5f);
39
40     // 螺旋吸积效应
41     glm::vec3 spiralForce = glm::cross(gravityDir, glm::vec3(0.0f, 1.0f, 0.0f)) * params.
        spiralStrength;
42
43     // 湍流效应
44     glm::vec3 turbulence = glm::vec3(
45         (dist(gen) - 0.5f) * 2.0f,
46         (dist(gen) - 0.5f) * 2.0f,
47         (dist(gen) - 0.5f) * 2.0f
48     ) * params.turbulenceStrength;
49
50     glm::vec3 acceleration = gravityDir * gravity * relFactor + spiralForce + turbulence;
51
52     // 只有正常粒子受黑洞引力影响
53     if (p.type == 0) {
54         p.velocity += acceleration * deltaTime;
55     }
56
57     // 速度限制
58     float speed = glm::length(p.velocity);
59     if (speed > 100.0f) {
60         p.velocity = glm::normalize(p.velocity) * 100.0f;
61     }
62
63     p.position += p.velocity * deltaTime;
64
65     if (p.type == 0) {
66         // 正常粒子受潮汐力影响
67         float tidalFactor = 1.0f + 5.0f / (distance * distance + 0.1f);
68         p.life -= deltaTime * tidalFactor;
69     }
70     else if (p.type == 1 || p.type == 2) {
71         // 特效粒子固定寿命衰减

```

```

72     p.life -= deltaTime;
73 }
74
75 if (p.type == 0) {
76     // 正常粒子颜色
77     float speedFactor = glm::length(p.velocity) / 50.0f;
78     float energyRelease = 1.0f / (distance + 0.5f);
79
80     p.color.r = 0.5f + energyRelease * 0.5f;
81     p.color.g = 0.3f + speedFactor * 0.5f;
82     p.color.b = 1.0f - energyRelease * 0.3f;
83 }
84 else if (p.type == 1) {
85     // 喷流粒子 - 黄色/橙色
86     p.color = glm::vec3(1.0f, 0.8f, 0.2f);
87 }
88 else if (p.type == 2) {
89     // 爆炸粒子 - 红色/橙色
90     float lifeRatio = p.life / 3.0f;
91     p.color = glm::vec3(1.0f, 0.5f * lifeRatio, 0.1f * lifeRatio);
92 }
93
94 p.color = glm::clamp(p.color, 0.0f, 1.0f);
95 }
96
97 updateBuffers();
98 }

```

2.2 光照与真实感渲染

我在片段着色器中实现了 pong 光照模型：

```

1 // 环境光分量
2 float ambientStrength = 0.3;
3 vec3 ambient = ambientStrength * lightColor;
4
5 // 漫反射分量
6 float diff = max(dot(norm, lightDirCalc), 0.0);
7 vec3 diffuse = diff * lightColor;
8
9 // 镜面反射分量
10 vec3 reflectDir = reflect(-lightDirCalc, norm);
11 float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
12 vec3 specular = specularStrength * spec * lightColor;

```

承接上文用法介绍部分，我们这里实现了点光源和方向光的自由切换，但这是出于方便的“视觉模型”，事实上，他并不物理（能量不守恒），只能说在视觉相似度和效率与真实渲染间做的让步，因为这个项目要想很物理不是小机器能渲染出来的，也不容易实时。

同时，我们模拟了多普勒效应和能量释放过程，当然只是量纲上的近似：

```

1 float speedFactor = glm::length(p.velocity) / 50.0f;
2 float energyRelease = 1.0f / (distance + 0.5f);
3 float dopplerFactor = 1.0 - speedFactor * 0.8;
4
5 p.color.r = 0.5f + energyRelease * 0.5f * dopplerFactor;

```

```

6 p.color.g = 0.3f + speedFactor * 0.5f;
7 p.color.b = 1.0f - energyRelease * 0.3f * (1 - dopplerFactor);

```

2.3 特效系统实现

通过控制立体角（锥形放射区域）与喷流粒子属性，实现了类似 gamma 射线喷流的效果：

```

1 // 喷流方向计算（锥形随机分布）
2 float angle = params.jetAngle * M_PI / 180.0f;
3 float randomAngle = dist(gen) * angle;
4 glm::mat4 rotationMatrix = glm::rotate(glm::mat4(1.0f), randomAngle, axis);
5 glm::vec3 randomDir = glm::vec3(rotationMatrix * glm::vec4(jetDir, 1.0f));

```

同时利用球面均匀分布采样和能量衰减效应（粒子颜色随生命周期衰减）做出了简单的爆炸特效：

```

1 // 球面均匀随机方向
2 float theta = dist(gen) * 2.0f * M_PI;
3 float phi = acos(2.0f * dist(gen) - 1.0f);
4 p.velocity = glm::vec3(sin(phi)*cos(theta), sin(phi)*sin(theta), cos(phi))
5                 * params.explosionStrength;

```

2.4 其他细节

我们沿袭了过去作业的技巧，实现了 imgui 的交互，一个交互的实时轨道相机系统（是欧拉数写的，因为涉及可视化欧拉角的问题，没有直接利用四元数，所以垂直旋转会有死锁。但是我觉得本次项目可视化的目标不应该纠结在这个方面，因此并未调整），还有大量粒子渲染时的内存访问优化等问题，这些内容都是必要而基本的，这里不过多展开了，读者可以进一步查阅源码。

3 写在最后

不知读者是否注意到，如果在程序运行期间我们拖动窗口或者开启新的任务（视图界面）时，我们会看到许多粒子好像从盘心“喷出”，但是喷出的粒子数量每次貌似都是差不多的。这就是我们上文说的“bonus”。我们并没有修改完善他，是因为他的视觉效果也挺有趣：这也是这个项目的初心所在。我们或许并不需要确保所有的实现都是严谨的，因为这非常困难甚至不可能；一些不影响整体效果的小小的 bug 其实是可爱的，他们有时候更能激起我们的热情和兴趣。

具体而言，这个现象是时间步长“deltaTime”的 delay 导致的，这个 delay 正是相邻 frame 的差值。我们粒子系统的更新依赖 deltaTime，在这个步长跨度太大时，引力的计算变得不稳定：

```

1 float gravity = params.blackHoleMass / (distance * distance);
2 glm::vec3 acceleration = gravityDir * gravity * relFactor + spiralForce + turbulence;
3 p.velocity += acceleration * deltaTime;

```

可以看到随着 deltaTime 的增大，粒子的速度增大，导致他们看似会从黑洞的中心“涌出”，由于我们已经限制了 deltaTime 的阈值为 0.1f，但是这个阈值对于粒子系统而言还是很大，有趣的是，正是固定的阈值才保证了被抛出粒子的最大数量基本不变。这事实上是数值积分不稳定的体现，主要是因为我们系统在实现考虑了这样的取舍：我们选择牺牲一定的精确性获得更简单和直观的代码，但好在，他们对整个系统的效果而言，是基本无害的！**有时候那些‘不完美’的意外发现反而能为科学可视化增添独特的魅力！**