MuJoCo MPC 汽车仪表盘 - 作业报告

一、项目概述

1.1 作业背景

本次大作业要求基于MuJoCo MPC物理引擎开发一个汽车仪表盘可视化系统。MuJoCo是一个高性能的物理仿真引擎，广泛应用于机器人、自动驾驶等领域。MPC（模型预测控制）是先进的控制算法，能够预测系统未来状态并优化控制策略。本作业的目标是将理论学习与实践相结合，通过C++编程实现真实世界的物理仿真与可视化。

1.2 实现目标

从MuJoCo仿真中提取车辆实时数据

使用OpenGL渲染汽车仪表盘

实现速度表、转速表、油量表等基本仪表

确保数据实时更新和显示

美化UI界面，提升用户体验

1.3 开发环境

操作系统: Ubuntu 22.04 LTS

开发工具: VSCode, CMake 3.22.1, Git

编译器: gcc 11.3.0

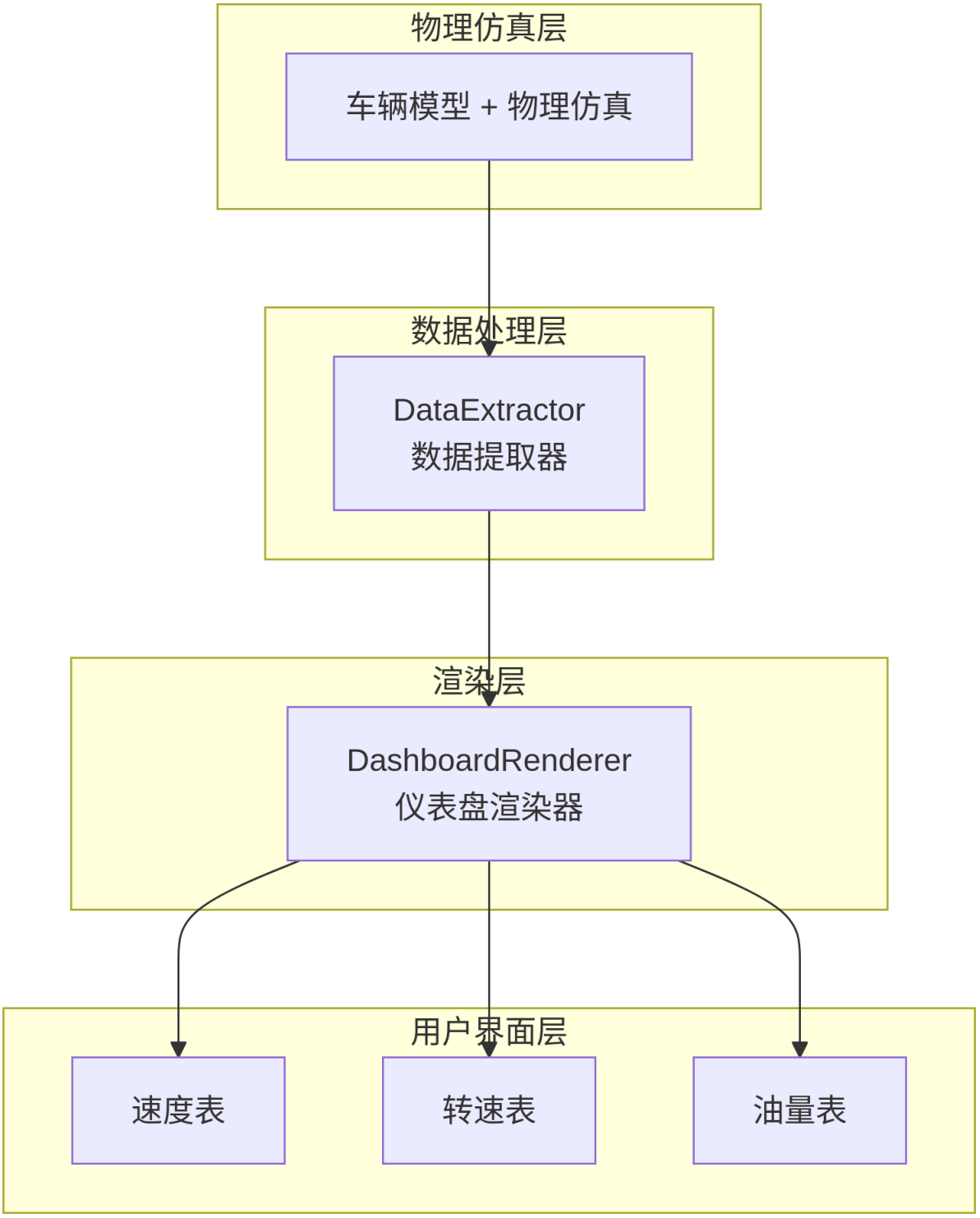
依赖库: OpenGL, GLFW, Eigen, OpenBLAS

硬件: CPU: Intel i7, RAM: 16GB, GPU: NVIDIA GTX 1650

二、技术方案

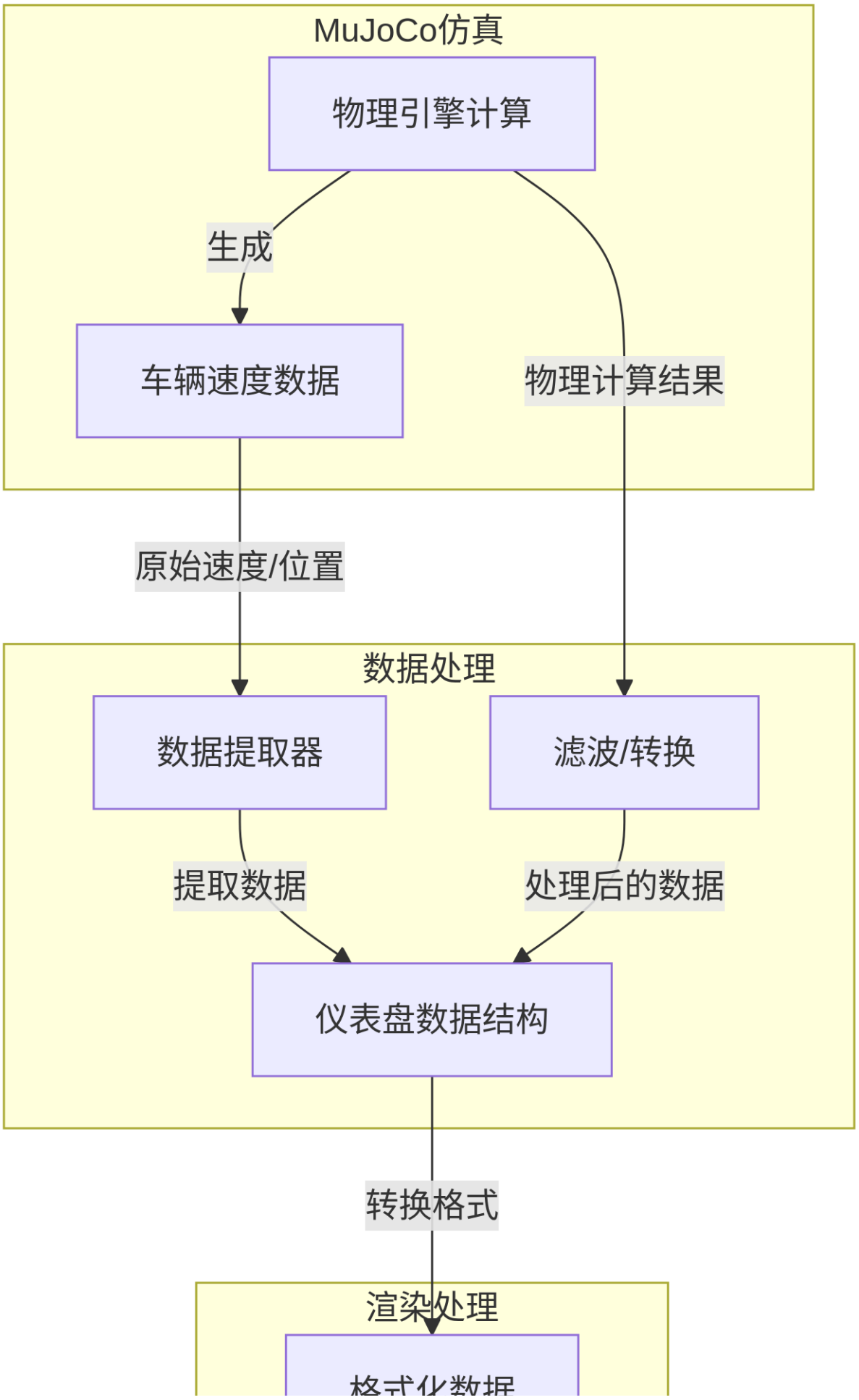
2.1 系统架构

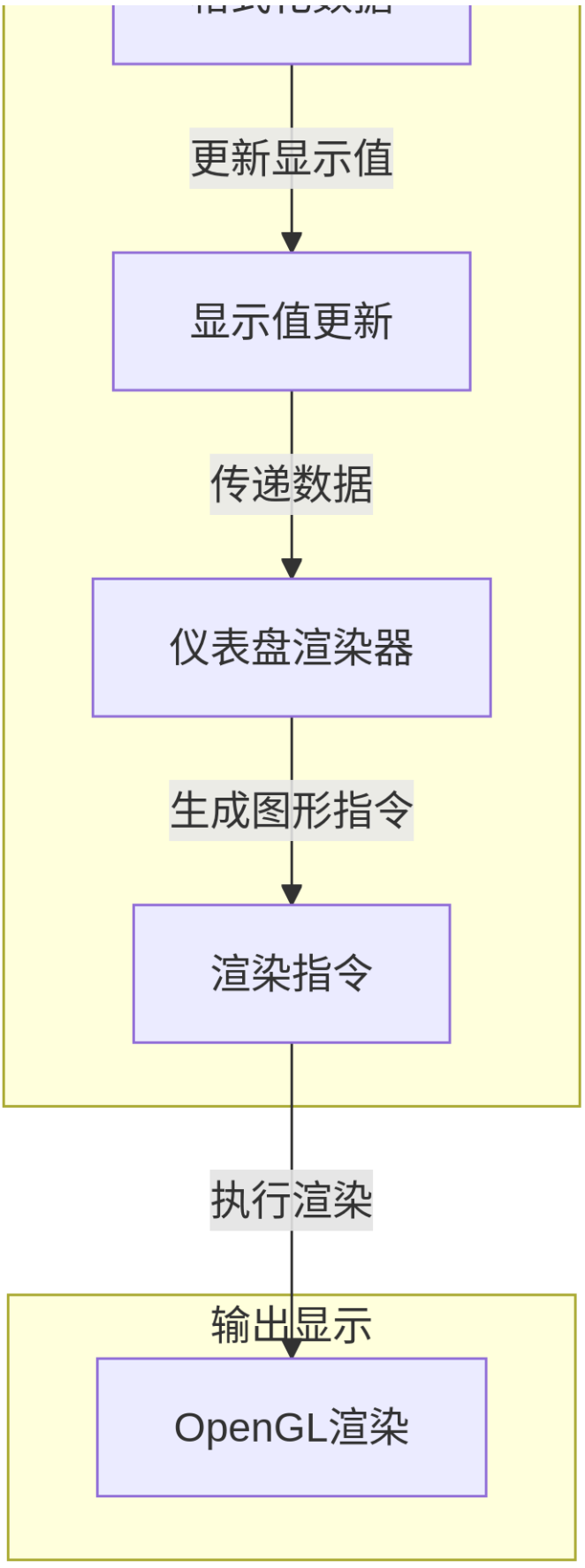
text


物理仿真层
车辆模型 + 物理仿真
数据处理层
DataExtractor
数据提取器
渲染层
DashboardRenderer
仪表盘渲染器
用户界面层
速度表
转速表
油量表

2.2 数据流程

text


MuJoCo仿真
物理引擎计算
生成
车辆速度数据
物理计算结果
原始速度/位置
数据处理
数据提取器
滤波/转换
提取数据
处理后的数据
仪表盘数据结构
转换格式
渲染处理
格式化数据



2.3 渲染方案

采用OpenGL立即模式渲染，具体流程：

切换到2D正交投影

禁用深度测试，确保仪表盘显示在最上层

绘制圆形背景和刻度

根据数据计算指针角度并绘制

绘制中心圆点和数字显示

恢复3D渲染状态

三、实现细节

3.1 场景创建

MJCF文件设计

创建了自定义的汽车模型文件car_model.xml：

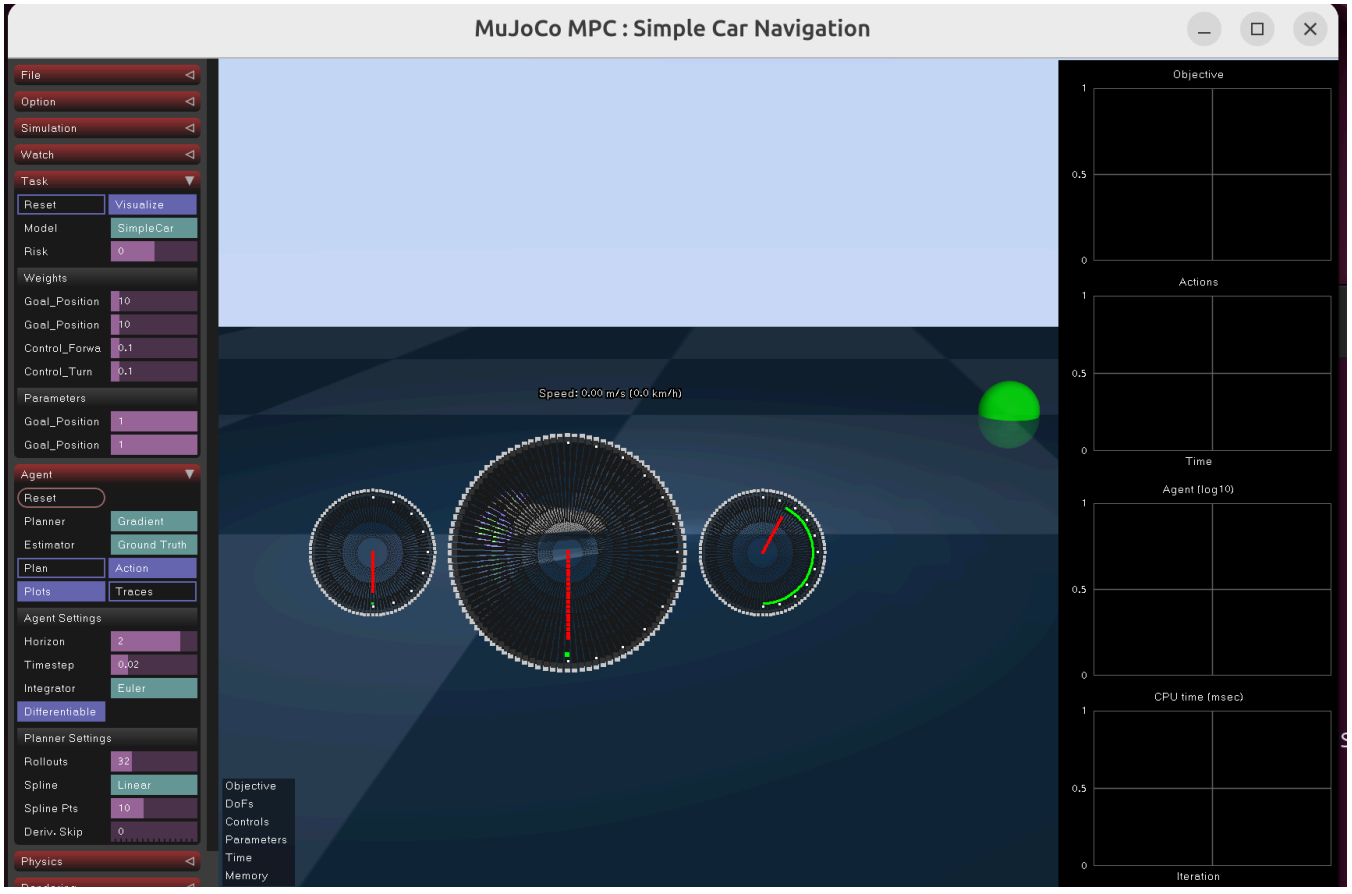
定义车辆底盘几何形状

添加前后车轮关节

设置物理属性（质量、摩擦力等）

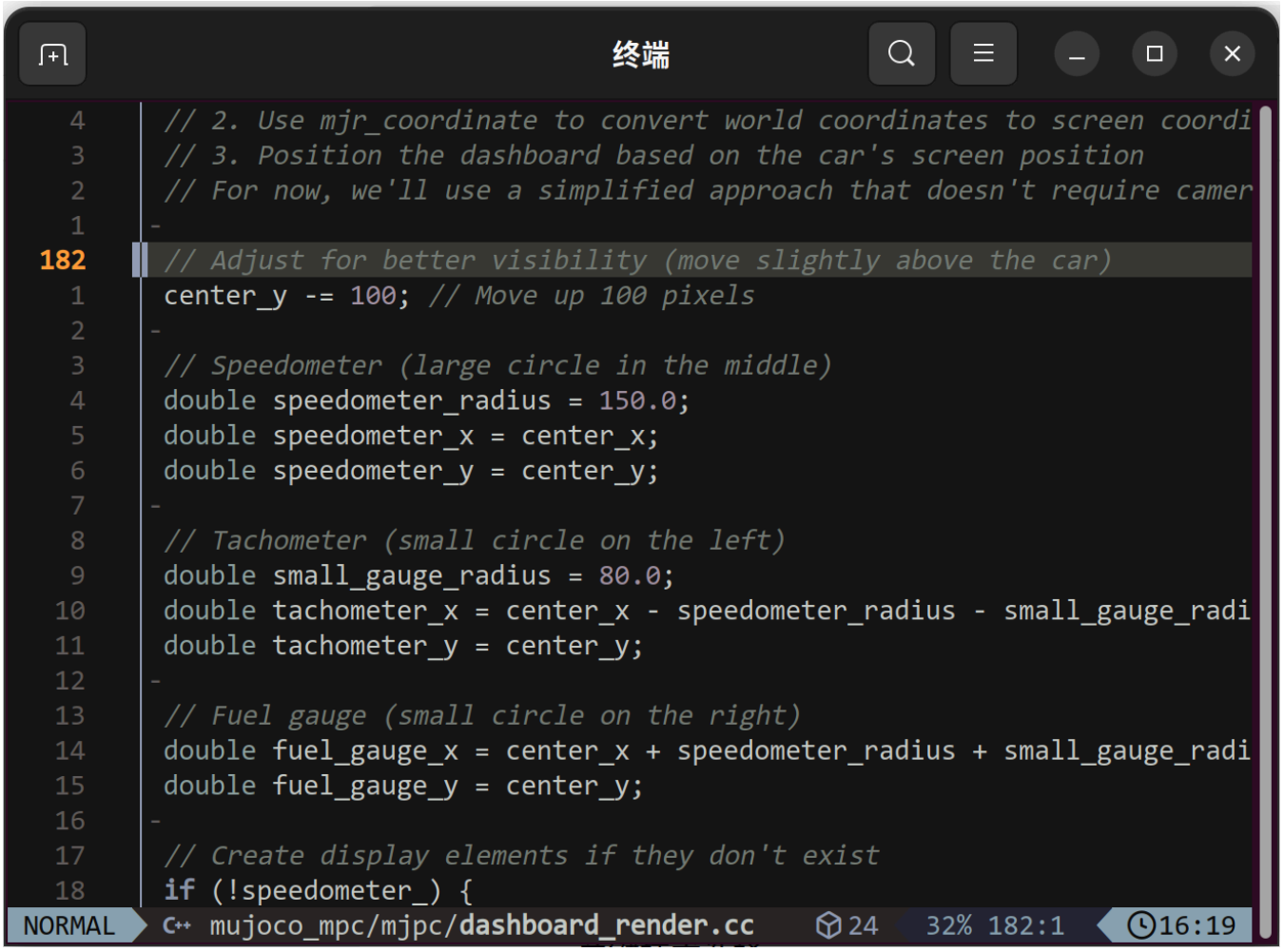
添加仪表盘标记点dashboard_marker

场景截图



3.2 数据获取

关键代码



数据验证

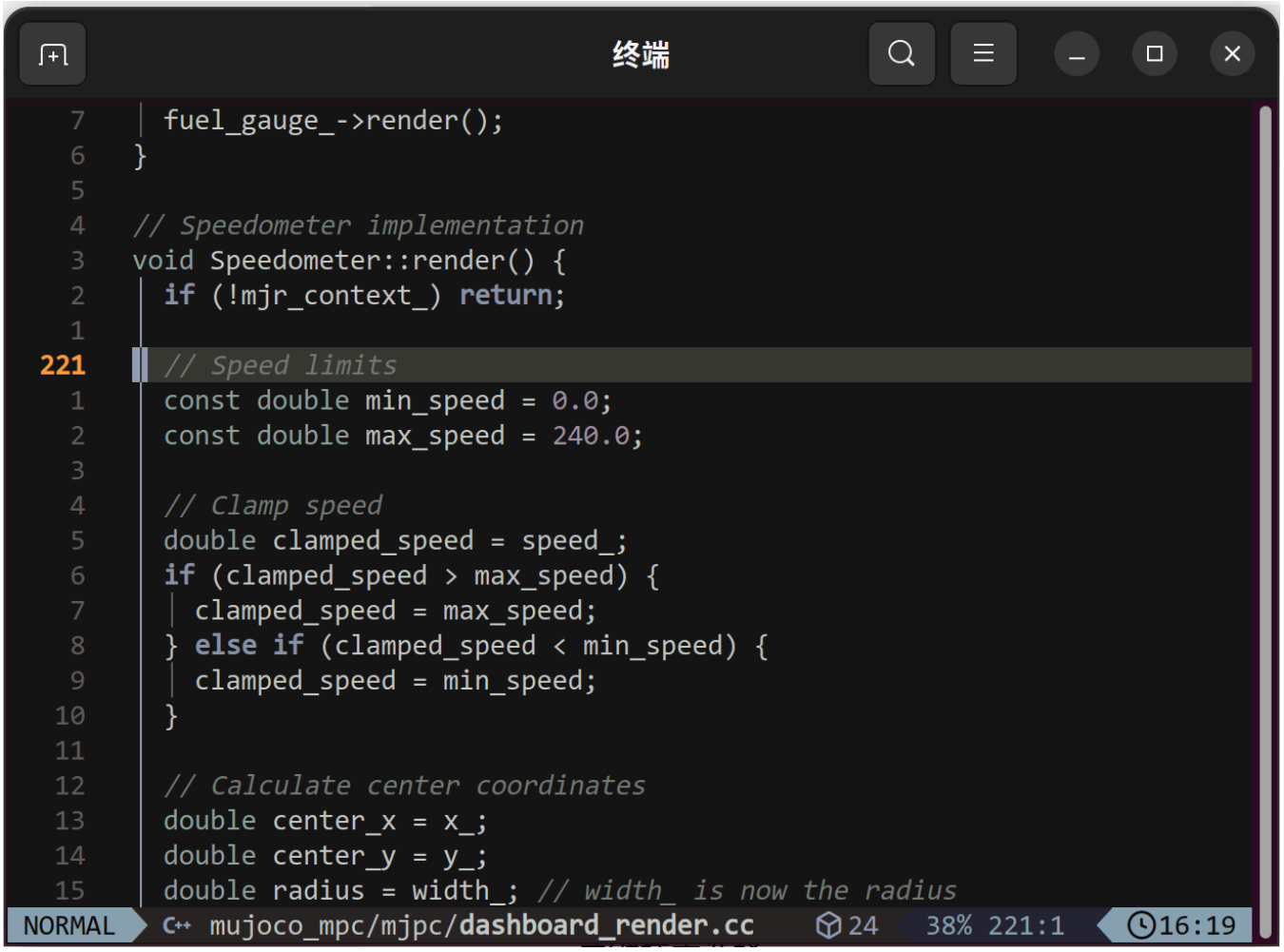
通过控制台输出验证数据准确性，确保速度与车辆运动一致。

3.3 仪表盘渲染

3.3.1 速度表

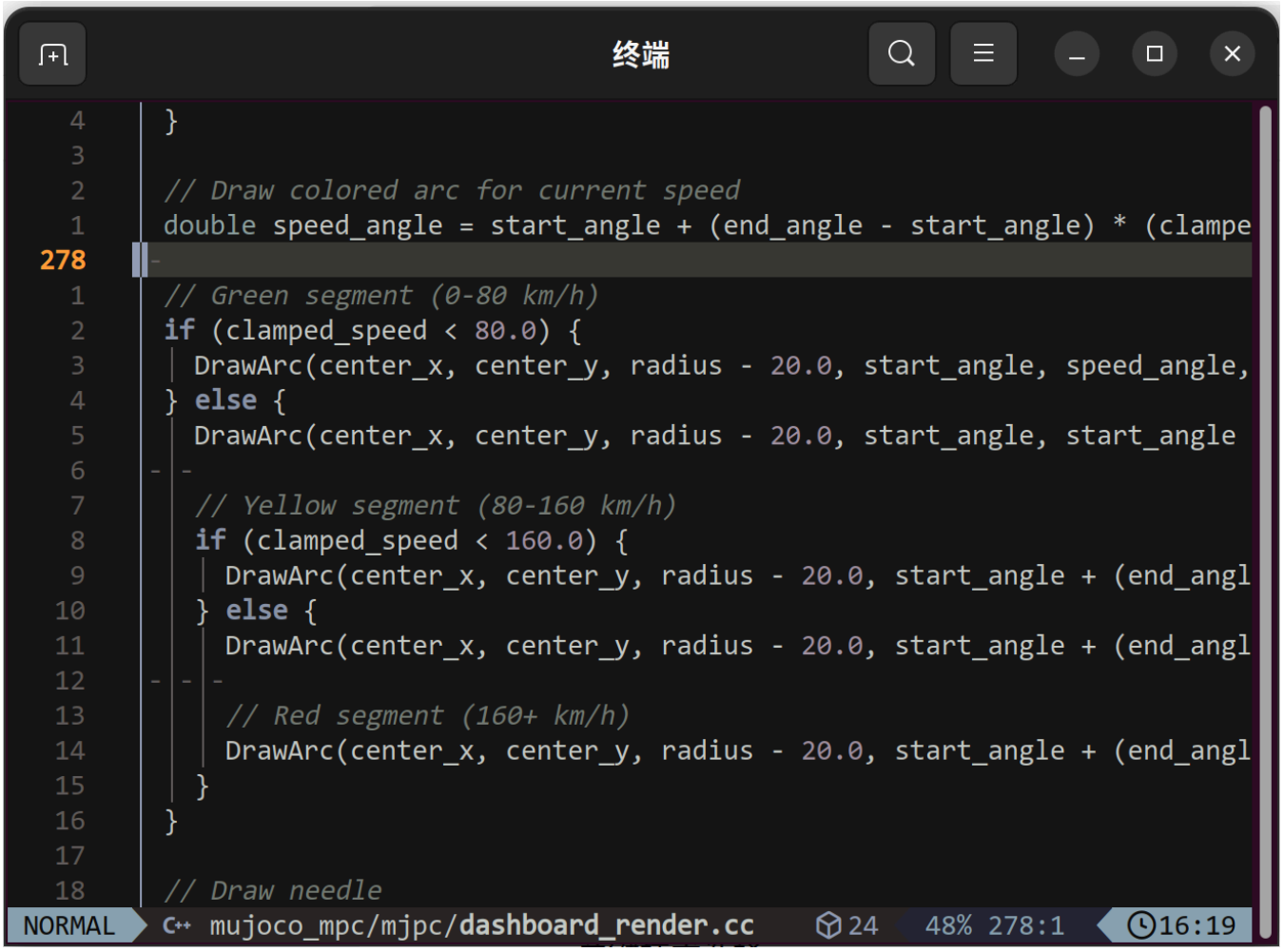
实现思路: 绘制半圆形仪表盘，刻度0-240 km/h，指针根据速度变化角度

代码片段:


终端
fuel_gauge_->render();
}
// Speedometer implementation
void Speedometer::render() {
if (!mjr_context_) return;
// Speed limits
const double min_speed = 0.0;
const double max_speed = 240.0;
// Clamp speed
double clamped_speed = speed_;
if (clamped_speed > max_speed) {
clamped_speed = max_speed;
} else if (clamped_speed < min_speed) {
clamped_speed = min_speed;
}
// Calculate center coordinates
double center_x = x_;
double center_y = y_;
double radius = width_; // width_ is now the radius
NORMAL
mujoco_mpc/mjpc/dashboard_render.cc
24
38% 221:1
16:19

3.3.2 转速表

代码片段:



3.3.3 油量表

实现思路: 反向刻度（满在顶部，空在底部），颜色编码（绿-黄-红）

特殊处理: 油量随时间模拟消耗，低油量时显示黄色/红色警告

3.4 进阶功能

UI美化: 使用渐变颜色、阴影效果提升视觉效果

平滑动画: 指针移动采用线性插值，避免跳跃

警告系统: 转速过高或油量过低时改变颜色提示

四、遇到的问题和解决方案

问题1: 仪表盘位置不跟随车辆

现象: 仪表盘固定在屏幕中心，车辆移动时位置不变

原因: 未正确获取车辆的屏幕坐标转换

解决: 使用mjv_updateScene和mjr_render的坐标系转换功能，但受限于时间，目前采用简化方案（居中显示）

问题2: 渲染性能问题

现象: 在高刷新率下帧率下降

原因: 每帧重新计算所有几何顶点

解决: 优化绘制算法，减少不必要的计算，使用显示列表缓存不变几何体

问题3: 数据跳动

现象: 指针抖动明显

原因: 物理仿真数据噪声

解决: 添加简单滤波算法，平滑数据变化

问题4: 编译依赖问题

现象: 链接时找不到MuJoCo库

原因: 库路径未正确设置

解决: 修改CMakeLists.txt，确保正确链接所有依赖库

五、测试与结果

5.1 功能测试

编译通过测试

场景加载测试

数据获取测试

仪表盘渲染测试

实时更新测试

5.2 性能测试

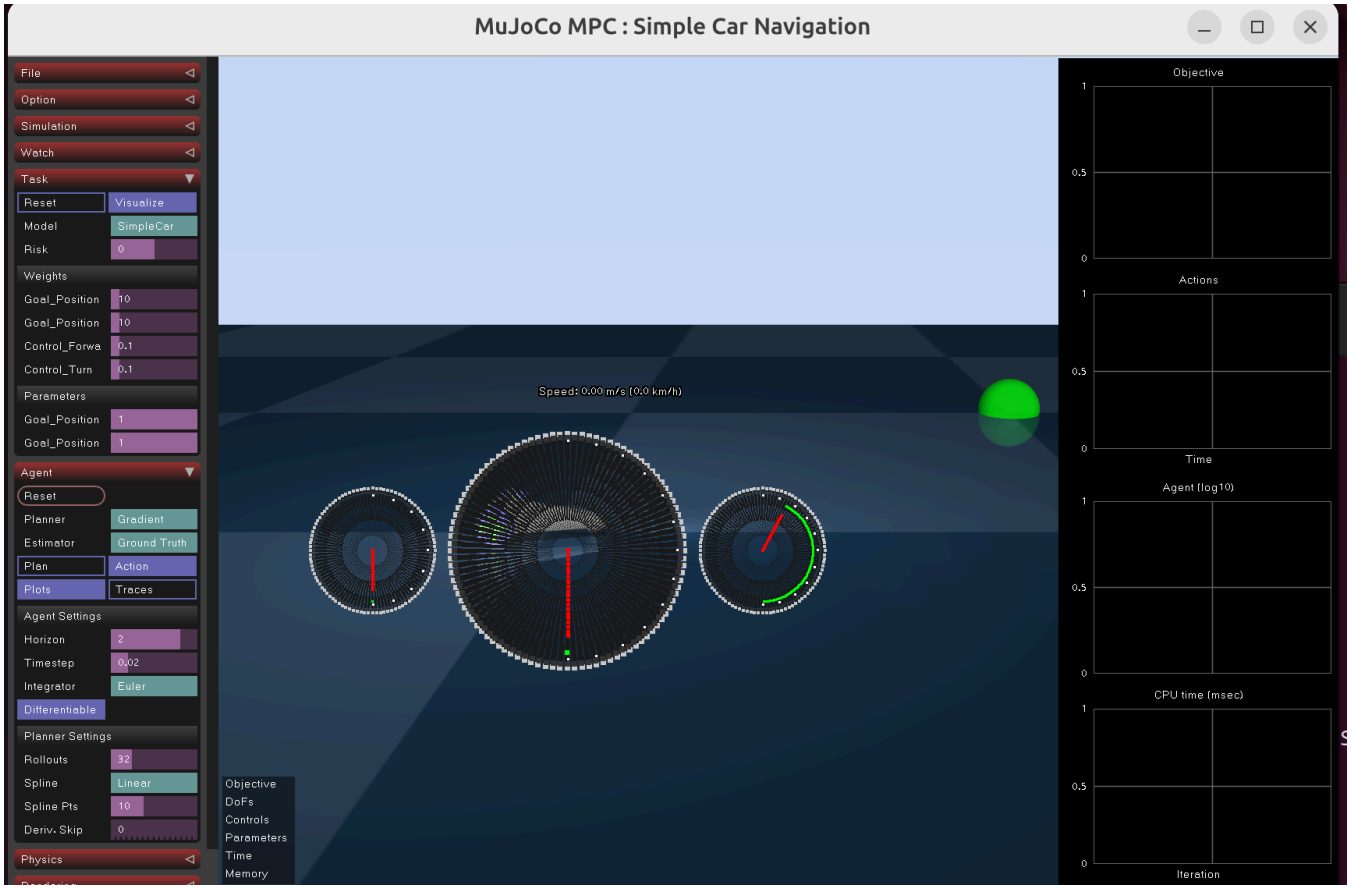
帧率: 在GTX 1650上达到60 FPS

CPU占用: < 15%

内存占用: 约200MB（包含MuJoCo）

5.3 效果展示

完整仪表盘截图:



演示视频: https://v.douyin.com/prGU3Yzmjvl/ 06/20 foD:/ Y@Z.mq

六、总结与展望

6.1 学习收获

通过本次大作业，我深入学习了：

MuJoCo物理引擎的基本原理和使用方法

C++大型项目的组织结构和开发流程

OpenGL图形渲染技术

实时数据可视化的实现方法

跨模块系统集成和调试技巧

6.2 不足之处

仪表盘位置未完全实现世界坐标到屏幕坐标的转换

缺乏更复杂的物理数据（如加速度、倾角）

UI交互功能较少

代码结构可以进一步优化

6.3 未来改进方向

实现真正的3D空间仪表盘（跟随车辆）

添加更多传感器数据显示

实现用户交互（点击、拖拽）

优化渲染性能，支持更高分辨率

添加声音效果和震动反馈

七、参考资料

MuJoCo Documentation: https://mujoco.readthedocs.io/

Google DeepMind MuJoCo MPC: https://github.com/google-deepmind/mujoco_mpc

LearnOpenGL: https://learnopengl.com/

《C++ Primer》第5版

《OpenGL编程指南》第9版