结题文档

软件 03 陈启乾 2020012385

说明程序运行环境，以及项目和代码依赖的编程环境

本机程序运行环境

1. Windows 11
2. Vulkan 图形后端

理论可以在所有操作系统，以及任何支持 Vulkan/Metal/OpenGL/DX 的图形后端上运行。

编程环境及配置方法

1. Rust 1.74.1：可以从 Rust 官网下载 https://www.rust-lang.org/zh-CN/learn/get-started
2. 其他依赖包以 `cargo.toml`
3. 在项目根目录下运行 `cargo run --release` 即可编译运行

各个程序模块之间的逻辑关系

各个模块均位于 `src` 目录下，具体介绍如下：

Camera 模块

维护 `position`, `yaw` 和 `pitch`。

在用户输入（拖拽界面/滚轮）的时候，更新 `yaw` 和 `pitch`；在用户输入（按键）的时候，更新 `position`。

在渲染前将 `position`, `yaw` 和 `pitch` 转换为 `view` 矩阵，作为 Uniform Buffer 传入渲染管线。

Light 模块

维护 `position` 和 `color`，分别表示光源的位置和颜色。

在渲染前将 `position` 和 `color` 转换为 `light` 矩阵，作为 Uniform Buffer 传入渲染管线。

Framework 模块

主要是与 Event Loop 相关的代码，创建窗口以及后端实例，处理窗口事件。

当 `Event::RedrawRequested` 事件发生时，会调用 `State::update` 更新状态，再调用 `State::render` 渲染。

当 `Event::WindowEvent` 事件发生时，根据具体事件，调用 Camera 和 Light 的方法，更新摄像机和光源的状态。

Resources 模块

从文件加载模型、纹理到 Rust 对象的模块。

Texture 模块

维护纹理相关功能，包括：

- 创建 Depth Texture
- 从 `image::DynamicImage` 创建 Texture, Sampler, ImageView，并且写入 Texture Buffer

Model 模块

维护模型相关功能，包括：

- 将 `obj` 文件解析为 `Model` 对象
- 将 `obj` 文件内的 Mesh 转化为 `Mesh` 对象，将 `mtl` 文件内的 Material 转化为 `Material` 对象
- 将 `Mesh` 对象转换为 Vertex Buffer 和 Index Buffer
- 维护 `draw` 方法，绘制 Model（包括实例化绘制）

同时还维护了光源相关的绘制方法，包括 `draw_light_model` 和 `draw_light_model_instance`。

Instance 模块

这里主要维护实例化绘制的 Buffer 以及信息，会在每次绘制所有小球前将 Instance 的 Vertex Buffer 更新；以及接收从主模块传入的小球信息，将其转换为 Instance 的 Vertex Buffer。

Main 模块

这里主要维护主模块的状态，会在主事件循环中被更新以及维护，包括：

1. 渲染物体的管线：`render_pipeline`
2. 渲染光源的管线：`light_render_pipeline`
3. 模型：`obj_model`
4. 深度纹理：`depth_texture`
5. 摄像机的状态：`camera_state`
6. 光源的状态：`light_state`
7. 实例化绘制的状态：`instance_state`
8. 计算模块的状态：`compute_state`
9. 上一次更新 FPS 的时间：`last_fps_update`

在创建(`State::New`) 时候，我们分别递归创建以上各个状态，然后将其传入 `State` 对象中。

在更新(`State::Update`) 时候，我们会调用以上各个状态的更新函数，包括更新摄像机的状态，更新光源的状态，计算碰撞检测的结果，并且用碰撞检测的结果去更新实例化绘制的状态。

在绘制(`State::Render`) 时候，我们会调用以上各个状态的绘制函数，包括绘制物体，绘制光源，绘制实例化绘制的小球。

除此之外

Compute 模块

这里主要维护碰撞检测和处理相关的功能。

程序运行的主要流程

1. 初始化
2. 计算与渲染
    - 碰撞检测
    - 更新数据
    - 渲染

简要说明各个功能的演示方法

运行程序后，即可看到许多球体在空间中运动，同时有一个光源在球体上方运动。

1. 使用 WASD 控制摄像机的移动
2. 使用鼠标在画面中拖动控制摄像机的视角。
3. 使用鼠标滚轮可以调整画面的缩放。

在窗口左上角会显示当前的渲染帧率，以 Frames Per Second(FPS) 为单位。

参考文献或引用代码出处

1. Learn WGPU: https://github.com/jinleili/learn-wgpu-zh/