实验报告一

姓名：韩东时 学号：202311998063

环境配置

系统环境

- Windows 11
- 处理器架构：x86_64
- 编译器: MSVC（Visual Studio 2022）

依赖库安装

在 Windows 平台上使用 vcpkg 或手动安装 OpenGL 相关依赖库：

- GLFW：可在官网下载预编译库或使用 vcpkg 安装。
- GLEW：官网下载预编译库或使用 vcpkg 安装。

glad 配置

从 glad 的官方网站下载生成的 glad 源码包，将 `glad.h` 放入项目的 `include/glad` 目录中，`glad.c` 放入项目的 `src` 目录中。

项目环境搭建（Visual Studio）

在 Visual Studio 中创建新的控制台项目，项目配置如下：

- 将项目的 C++ 标准设置为 C++17。
- 在项目属性中设置附加包含目录为 `$(ProjectDir)include;$(ProjectDir)include/glad`。
- 将 `glad.c` 添加到项目的源文件中。
- 设置附加库目录指向 GLFW 和 GLEW 的预编译库所在位置。
- 在链接器输入中添加所需的库：`opengl32.lib;glew32.lib;glfw3.lib`。

说明

项目文件夹结构如下：

- `shaders` 文件夹存放着色器代码文件，在项目的调试或发布目录中手动拷贝该文件夹。
- `static` 文件夹存放纹理等资源文件，同样需要拷贝至程序运行目录。

在 Visual Studio 中设置工作目录为项目根目录，确保运行时能直接加载着色器与纹理文件。

编译 & 运行

在 Visual Studio 中直接使用 F5 运行项目，或使用 Ctrl+Shift+B 编译后手动运行生成的可执行文件。

任务实现

着色器加载器

为实现着色器文件的加载与编译，设计了一个 `ShaderLoader` 类，具体定义如下：

```
class ShaderLoader
{
private:
    static std::string readShaderFile(const std::string &filePath);
    static void checkShaderCompilation(unsigned int shader, const std::string
&type);
    static void checkProgramLinking(unsigned int program);

public:
    ShaderLoader();
    ~ShaderLoader();

    static unsigned int loadVertexShader(const std::string &filePath);
    static unsigned int loadFragmentShader(const std::string &filePath);
    static unsigned int compileShader(unsigned int type, const std::string
&source);

    static unsigned int createShaderProgram(const std::string &vertexPath, const
std::string &fragmentPath);

    static void deleteShader(unsigned int shader);
    static void deleteProgram(unsigned int program);
};
```

实现方法基于 OpenGL API，包括使用 `glCreateShader` 创建着色器对象，通过 `glShaderSource` 加载源代码，调用 `glCompileShader` 编译，使用 `glCreateProgram` 创建着色器程序，再通过 `glAttachShader` 和 `glLinkProgram` 链接着色器程序。

封装这些 API 可以便捷地加载和管理着色器代码文件。

任务 1

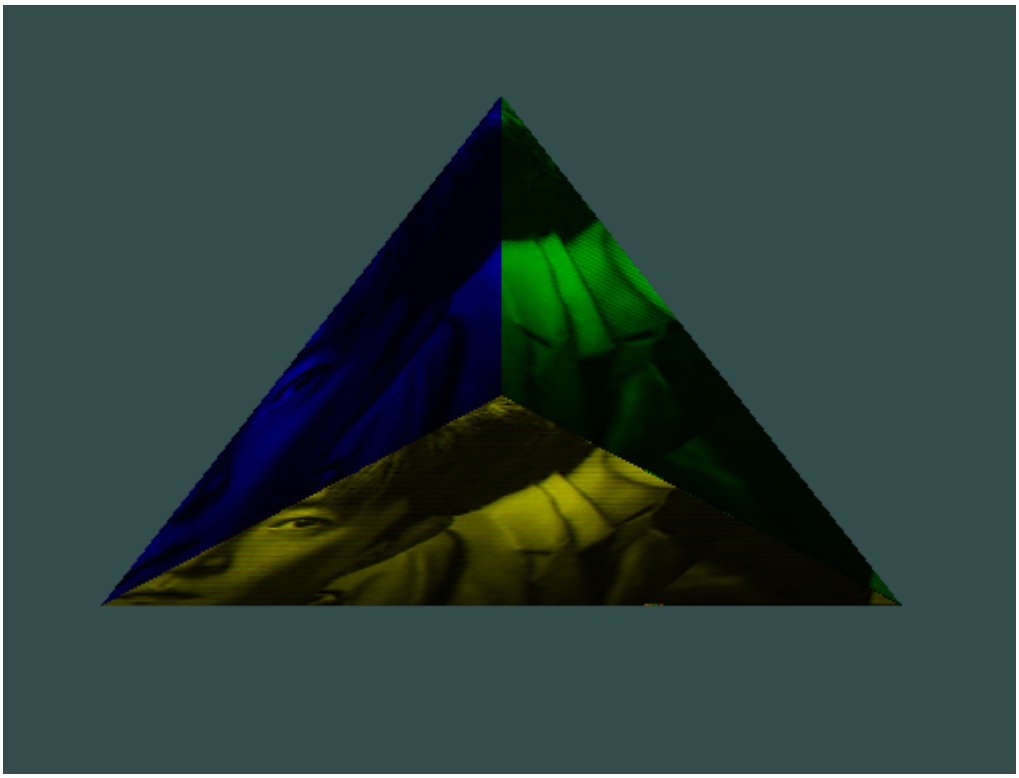
基于原本绘制三角形的基础代码，完成立方体的渲染任务（最基础的立方体为正四面体，包含四个顶点和四个三角面）。

任务 2

在任务 1 的基础上，对立方体的每个三角形面赋予不同颜色，要求至少一个三角面的三个顶点颜色互不相同。

任务 3

在任务 2 基础上，自行选择一张纹理图片，将其应用到渲染的立方体表面。



代码在压缩包中