浙江大学

魔方模拟器中期文档

授课教师：	袁　昕
组　　长：	王伟杰
组　　员：	钟睿昕
	戴　卿
组　　别：	魔方模拟组
日　　期：	2023-7-11

1 项目介绍

1.1 项目说明

- 项目名称：魔方模拟器
- 开发小组名称：魔方模拟组
- 开发小组成员：王伟杰（组长）、戴卿、钟睿昕

1.2 项目背景

魔方模拟器是一个基于C++编程语言开发的应用程序，旨在提供一个实时、交互式的魔方解谜工具。在现实世界中，解开魔方可能需要花费大量的时间和精力，而通过模拟器，用户可以在计算机上模拟魔方的各种操作和解法，更加便捷的操作魔方，并高效地学习和掌握解谜技巧。该模拟器允许用户在图形界面中操作虚拟魔方，模拟真实的魔方操作过程和解法，通过使用计算机算法和交互界面，帮助用户更好地理解和掌握魔方的解法过程。

1.3 项目目标

完成该项目后，用户将拥有一个功能完善、易于使用的魔方模拟器应用程序。用户可以通过运行该程序，在计算机上模拟魔方的各种操作和解法过程，提高自己的解谜技巧，并学习魔方的解法。

该魔方模拟器的目标是为用户提供一个功能齐全、交互性强的魔方模拟平台，能够完成以下目标：

1. 提供魔方模拟器：开发一个魔方模拟器应用程序，实现在计算机上模拟魔方的各种操作和解谜过程。
2. 实时交互性：确保模拟器能够实时响应用户的操作，提供流畅的魔方操作体验，使用户感觉像是在操控真实的魔方一样。
3. 支持多种操作命令：实现支持各种操作命令，如旋转、视角移动、打乱、复原、存储文件等，以满足用户对魔方的灵活控制需求。
4. 提供解谜演示：能够根据用户的输入解谜步骤，展示魔方复原的过程，并支持动画效果，以帮助用户学习和理解解谜的技巧。
5. 支持求解算法：模拟器支持模仿复原通用算法，以满足用户的需求和挑战。

通过实现这些目标，该魔方模拟器将为魔方爱好者和初学者提供一个综合的学习和娱乐平台。用户可以通过模拟器的操作和演示功能，自由地探索和体验魔方解谜的乐趣，并逐步提高自己的解谜技巧。

1.4 项目用户

本项目面向对象为各层次水平魔方爱好者，包括魔方初学者、解谜爱好者、专业魔方选手等。

2 开发进度

第二轮迭代已完成

2.1 第一轮迭代

- [x] 正常显示魔方
- [x] 视角移动
- [x] 魔方旋转
- [x] 透明度改变

2.2 第二轮迭代

- [x] 随机打乱
- [x] 文件存储
- [x] 输入加载
- [x] 重置魔方

2.3 第三轮迭代

- [ ] 魔方复原算法
- [ ] 自动复原演示

3 技术难点

3.1 OpenGL使用

OpenGL是一个开放的图形库，用于渲染2D和3D图形。在魔方模拟器项目中，我们使用OpenGL来实现魔方的显示、视角移动和旋转功能，处理了坐标系转换、矩阵运算和视图投影变换等问题。

另外，处理用户输入事件是整个项目中一个重要的方面。在魔方模拟器中，我们响应了鼠标和键盘事件，以实现与用户交互的功能，如旋转魔方、移动视角等。为了正确处理这些事件，我们学习了OpenGL中鼠标和键盘事件的处理方式，并将其与魔方的操作相结合。

3.2 视角变换

- 处理坐标系转换：在OpenGL中存在多个坐标系，如模型坐标系、世界坐标系和观察坐标系。我们了解了各个坐标系之间的变换规律，这样才能正确地将魔方的几何信息从一个坐标系转换到另一个坐标系，并运用了矩阵乘法、平移、旋转和缩放等操作。
- 矩阵运算：在进行视角变换时会涉及到矩阵运算。视角变换是通过修改观察矩阵来实现的，该矩阵定义了观察者的位置、观察目标点和上方向。我们需要理解矩阵的乘法规则和矩阵变换的数学原理，以便于正确地计算和应用观察矩阵对于实现正确的视角变换。
- 视图投影变换：视图投影变换是将场景从三维空间投影到二维屏幕空间的过程。在OpenGL中，常用的投影方式是透视投影和正交投影。我们需要根据具体情况选择适当的投影方式，并正确计算和设置投影矩阵，来确保魔方在屏幕上正确显示，并具有适当的透视效果。

3.3 数据存储结构

在第二轮迭代中，文件存储是一个技术难点。为了实现随机打乱、输入加载和重置魔方等功能，我们设计了合适的数据存储结构，以便有效地保存和恢复魔方的状态。

- 定义了合适的序列化存储格式。我们将涉及到的魔方的各个面的颜色、坐标等信息转换为可存储的格式，并在序列化的过程中，考虑了文件的大小、读写效率和数据一致性等问题。
- 实现魔方状态的序列化和反序列化算法。序列化算法负责将魔方的状态转换为序列化数据，并将其写入文件中。反序列化算法负责从文件中读取序列化数据，并将其转换回魔方的状态。在实现这些算法时，我们考虑了数据的正确性、完整性和有效性。
- 将序列化和反序列化算法应用到文件存储的目标对象中。该步骤涉及到文件的打开、读写操作，以及与其他部分的集成。我们使文件存储功能能够正确地保存和加载魔方的状态，并且与其他功能模块协调工作。

3.4 魔方旋转算法

魔方的旋转是整个项目的核心功能之一。实现魔方旋转需要设计合适的旋转算法，以确保旋转操作正确、流畅，并且能够应对各种情况（如旋转方向、层级等）。旋转算法需要考虑魔方的数据结构、块的移动、颜色的变化等因素，同时还要确保旋转操作的效率和性能。

- 魔方的数据结构：我们采用的数据结构是使用三维数组表示魔方的6个面，每个面由9个小块组成。算法能够在数组中准确地找到每个小块的位置和颜色信息。
- 魔方块的移动：当旋转魔方时，相邻块之间会发生位置交换。该算法正确地跟踪块的位置，并进行适当的交换操作，以模拟旋转的效果，并且涉及到更新数组中块的位置和颜色信息。
- 处理颜色的变化：当旋转魔方时，颜色会随着块的移动而发生改变。算法在进行块的位置交换时，相应地更新颜色信息，以确保每个块始终具有正确的颜色。
- 旋转操作的效率和性能：魔方具有大量的块和可能的旋转操作，因此算法需要尽可能高效，以便在短时间内完成旋转，并且不会对整个程序的性能产生负面影响。我们采用了一些优化技术来改善算法，具体参考后文。

3.5 碰撞检测和边界处理

当魔方进行旋转或视角移动时，需要进行碰撞检测和边界处理，避免魔方超出屏幕或发生重叠等问题。为此，我们对魔方和视角的位置进行实时监测，在碰撞检测和边界处理过程中综合考虑了物体的几何形状、旋转状态和屏幕的尺寸等因素，并采取相应的措施来确保它们保持在合理的范围内。

3.6 性能优化

此次实现的魔方模拟器包含复杂的3D功能，我们认为性能优化是至关重要的，尤其是在处理大规模的魔方或进行复杂的操作时，如果不进行性能优化，可能会导致性能瓶颈和卡顿现象，影响用户体验。因此为了提高性能，我们需要对渲染流程进行优化，减少不必要的绘制调用，并采用合适的渲染技术。

我们采用的优化技术是`VSync`技术。`VSync`通过协调渲染帧率与显示器刷新率之间的关系，可以避免画面撕裂（Tearing）现象，并提升渲染性能。画面撕裂是指在渲染过程中，显示器的刷新与渲染帧率不同步，导致画面出现不连续的情况。通过启用`VSync`，我们可以确保每一帧的渲染都与显示器的刷新率同步，消除画面撕裂，并提供更加平滑的视觉效果。

3.7 跨平台兼容性

我们希望在多个平台上运行魔方模拟器，例如Windows、Mac和Linux等，为此我们考虑跨平台兼容性的问题。不同平台上的图形库和输入处理方式可能有所不同，因此需要在设计和实现时进行适配，以确保在各个平台上能够正确运行和使用。

处理这个问题的难点是要确保在不同平台上正确调用`GLFW`库。`GLFW`是一个跨平台的库，但是在不同操作系统上需要链接不同的库文件。我们采用了Xmake项目管理工具来帮助自动适配各个平台的`GLFW`库，并合理地调用它们。

Xmake是一个跨平台的项目构建工具，它可以根据不同平台和编译器的要求，自动选择正确的链接文件和设置。通过使用Xmake，我们可以在项目配置文件中指定所需的依赖库，例如`GLFW`，并且Xmake会根据当前构建环境自动选择正确的库文件进行链接。这样，我们就可以避免手动管理不同平台的链接文件，减轻了跨平台开发的工作量。

使用Xmake之后，我们就可以为不同平台编写相应的配置代码，并根据需要包含或排除特定平台的库文件。这样在构建项目时，Xmake就会根据所选平台的配置自动完成正确的库链接。

我们合理配置和使用Xmake项目管理工具，解决了跨平台兼容性的问题，确保魔方模拟器能够在不同的操作系统上正确运行，并正确调用相应的`GLFW`库。因此我们可以更加专注于项目的逻辑和功能实现，而无需过多担心平台兼容性带来的困扰。

4 程序说明

魔方模拟器应用程序使用了MVVM架构，将Model、ViewModel和View进行了分离。Model是魔方的数据模型，ViewModel负责处理业务逻辑和与View的交互，View负责展示用户界面。在App层的代码中，进行了各个组件之间的初始化、绑定和显示操作，实现了整个应用程序的运行。

4.1 App层

1. 在构造函数和析构函数中，没有特定的操作，只是对`App`类进行了初始化和清理。
2. 在`init()`函数中，进行了应用程序的初始化操作：
    - 创建了`Cube`和`ViewModel`对象：通过`std::make_shared`函数创建了`Cube`类的对象`m_pCubeModel`和`ViewModel`类的对象`m_pViewModel`。
    - 初始化`ViewModel`对象：调用`m_pViewModel`的`init()`方法进行初始化操作。
    - 设置`CubeModel`：调用`m_pViewModel`的`setCubeModel()`方法，将`m_pCubeModel`设置为`ViewModel`的`CubeModel`。
    - 初始化主窗口：调用`m_mainWindow`的`init()`方法进行主窗口的初始化操作。如果初始化失败，则会返回false。
    - 进行绑定操作：
        - 绑定属性（properties）：调用`m_mainWindow`的`setCube()`、`setRotateMethod()`和`setRotateAngle()`方法，分别将`m_pViewModel`的`CubeData`、`RotateMethod`和`RotateAngle`属性绑定到主窗口上。

- 绑定命令（commands）：调用`m_mainWindow`的`setResetHandler()`、`setRandomHandler()`、`setRotateHandler()`、`setSaveHandler()`、`setLoadHandler()`、`setSolveHandler()`和`setStopHandler()`方法，将`m_pViewModel`的`ResetCmd`、`RandomCmd`、`RotateCmd`、`SaveCmd`、`LoadCmd`、`SolveCmd`和`StopCmd`命令绑定到主窗口上。
- 绑定通知（notifications）：调用`m_pViewModel`的`addObserver()`方法，将`m_mainWindow`的`MainWindowSink`对象作为观察者添加到`ViewModel`中；调用`m_mainWindow`的`addObserver()`方法，将`m_pViewModel`的`ViewModelSink`对象作为观察者添加到`m_mainWindow`中。

3. `Run()`函数用于运行应用程序：
   - 调用`m_mainWindow`的`show()`方法，显示主窗口。

4.2 Common层

- 定义了`INotification`、`Notification`基类以实现通知机制，`ICommandBase`作为命令控制的基类。
- 定义常用宏函数，以达到简化代码，增强复用性、可读性、可维护性的目的。
- 定义魔方模拟所需要的专用数据类型，如方块颜色、旋转方法等。
- 实现一些基本函数，如字符串转换大小写等。

4.3 Model层

Model层主要是Cube类的实现。

1. `Cube`类是一个魔方模拟器的主类，它包含了魔方模拟器的状态和操作方法。
2. 构造函数`Cube()`用于创建一个空的魔方对象。
3. `init()`方法用于初始化魔方对象，将其恢复到初始状态。
4. `getCubeData()`方法返回当前魔方的状态数据。
5. `randomCube()`方法用于随机打乱魔方。
6. `serialize()`和`deserialize()`方法用于将魔方对象序列化为字符串或从字符串中反序列化恢复魔方对象。
7. `doMethod()`方法根据传入的旋转方法对魔方进行旋转变换操作。
8. `s_charColorMap`和`s_colorCharMap`是字符和魔方颜色之间的映射关系，用于序列化和反序列化魔方状态数据。

4.4 Viewmodel层

1. `ViewModel.cpp/ViewModel.h`：

   这个类是`Notification`的继承类，相当于一个命令信号发出的“频道”，是通知的发送者，负责处理用户的命令并向View层发送通知。

   (a) `setCubeModel()`方法：通过该方法设置魔方模型，将一个`Cube`对象传递给`ViewModel`，以便`ViewModel`能够对其进行操作和管理。

(b) `getCubeData()`方法：获取当前魔方的数据。返回一个指向魔方数据的共享指针。

(c) `getRotateCmd()`、`getResetCmd()`、`getRandomCmd()`、`getSaveCmd()`、`getLoadCmd()`、`getSolveCmd()`和`getStopCmd()`方法：分别获取旋转命令、复位命令、随机打乱命令、保存命令、加载命令、求解命令和停止命令。这些命令将在View层触发，并在`ViewModel`中执行相应的操作。

(d) `m_pCube`、`m_pRSteps`属性：这些属性分别用于存储魔方模型和旋转方法序列。

(e) `m_pRotateCommand`、`m_pResetCommand`、`m_pRandomCommand`、`m_pSaveCommand`、`m_pLoadCommand`、`m_pSolveCommand`、`m_pRotEndCommand`和`m_pStopCommand`属性：这些属性用于存储旋转命令、复位命令、随机打乱命令、保存命令、加载命令、求解命令、完成旋转和停止命令的共享指针。

(f) 友元类声明：`ViewModel`声明了一些友元类，以便实现命令和控制器的功能。

2. `commands`文件夹：

这个文件夹包含了各种命令（Command）的实现类，用于处理用户的不同操作。例如，`LoadCommand`用于加载魔方状态，`RandomCommand`用于随机打乱魔方，`RotateCommand`用于旋转魔方块，`SaveCommand`用于保存魔方状态等。每个命令类都实现了执行（`exec`）和设置参数（`setParameter`）方法，以便在模拟器中执行和撤销对应的操作。

4.5 View层

1. `MainWindow.cpp/MainWindow.h`：

- 这些文件集成了OpenGL图形库，并实现了创建主窗口、控制界面渲染、事件的绑定和控制等功能。
- `MainWindow`类负责创建和管理用户界面的主窗口的布局和显示。
- `MainWindow`类包含与界面相关的逻辑，如界面初始化、界面渲染控制等。

2. `envents`文件夹（`EventCallback.cpp/EventCallback.h`）：

- 这些文件包含了用于处理用户事件的回调函数。
- `EventCallback`类定义了一些回调函数，用于处理用户界面上的事件，例如鼠标拖动、命令输入等操作。
- 这些回调函数会调用ViewModel层暴露给View层的命令来处理用户的操作，并更新界面显示。

3. `RotAni`文件夹（`RotAni.cpp/RotAni.h`）：

魔方模拟器的旋转过渡动画控制器，它负责计算魔方旋转过程的中间状态，如旋转角度、旋转方法等。控制器在计算出中间状态后会将数据输出给`Renderer`对象，以实现对魔方旋转动画的渲染。

4. `render`文件夹（`Renderer.cpp/Renderer.h`）：

- 包含了与渲染相关的代码。
- `Renderer`类负责魔方的绘制和渲染，将魔方以图形的形式显示在界面上。
- `Renderer`类接收ViewModel层发送通知传递的魔方状态数据，并进行相应的绘制和渲染操作。

5. `sinks`文件夹：

- 这个文件夹包含了`MainWindowSink.cpp`和`MainWindowSink.h`文件。
- `MainWindowSink`类用于接收ViewModel层命令执行完毕后返回的通知。

5 协作情况

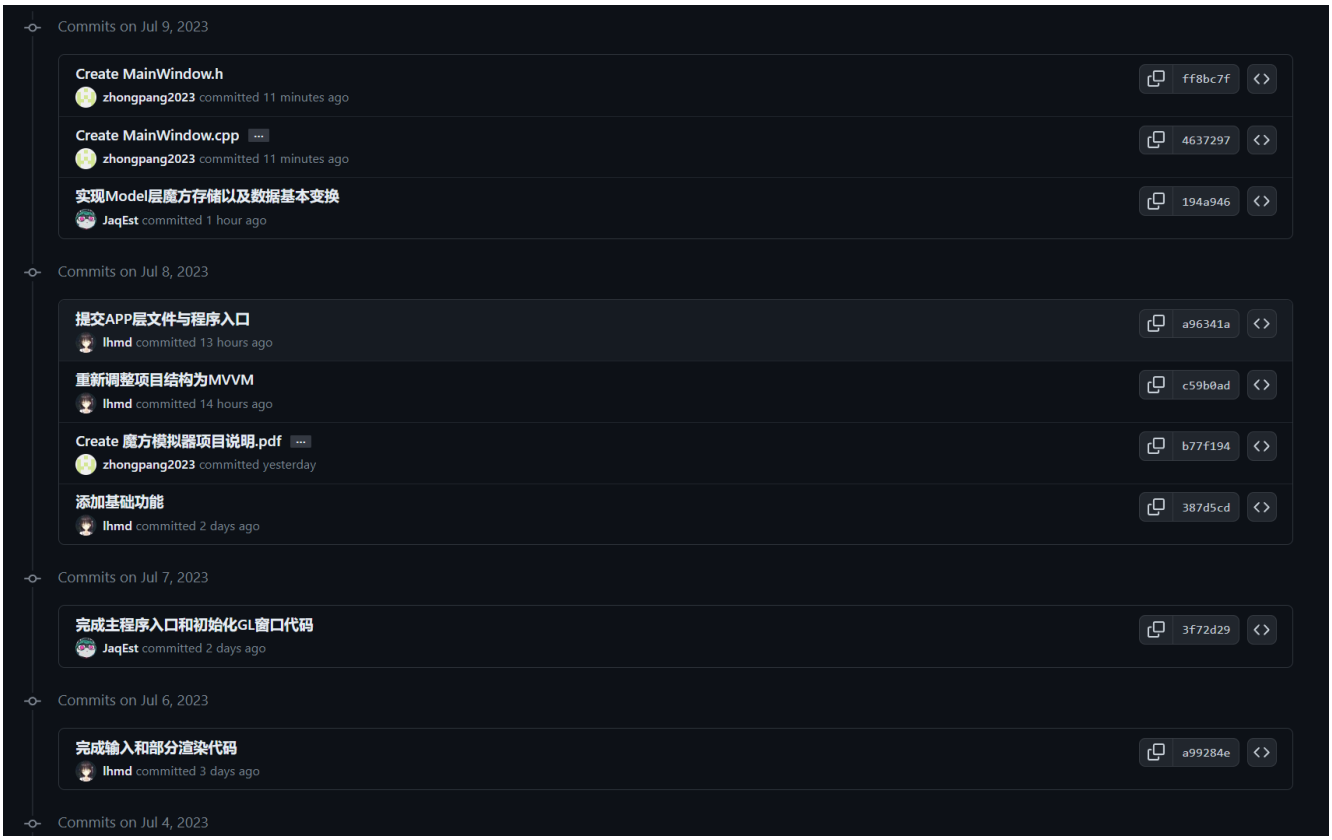
5.1 第一轮迭代

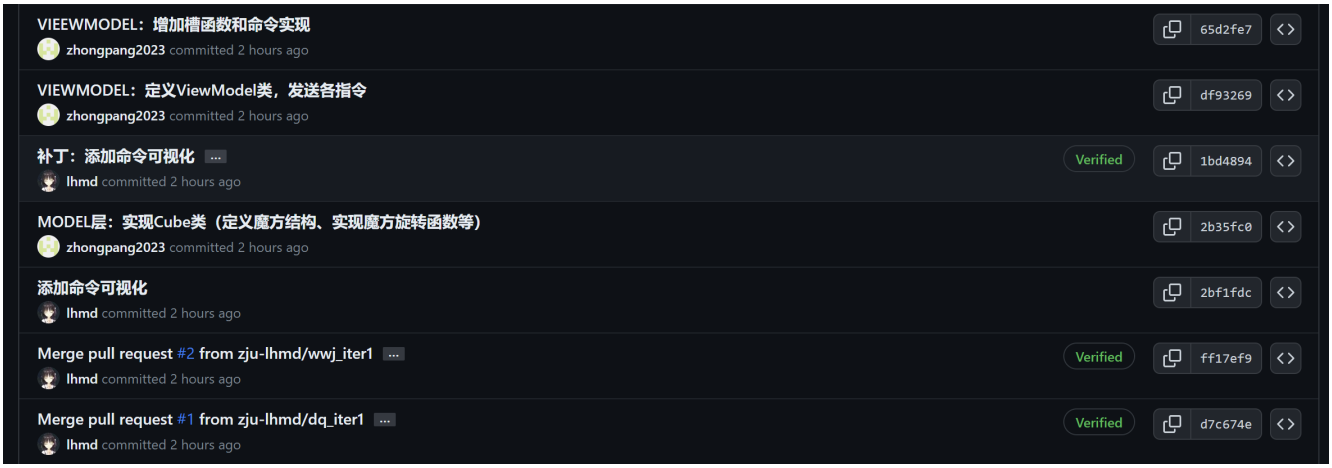
- 王伟杰：Common层+App层+xmake项目管理
- 戴卿：ViewModel层+Model层
- 钟睿昕：View层+文档撰写

5.2 第二轮迭代

- 王伟杰：View层+文档撰写
- 戴卿：Common层+App层
- 钟睿昕：ViewModel层+Model层

5.3 部分代码提交记录





5.4 总结

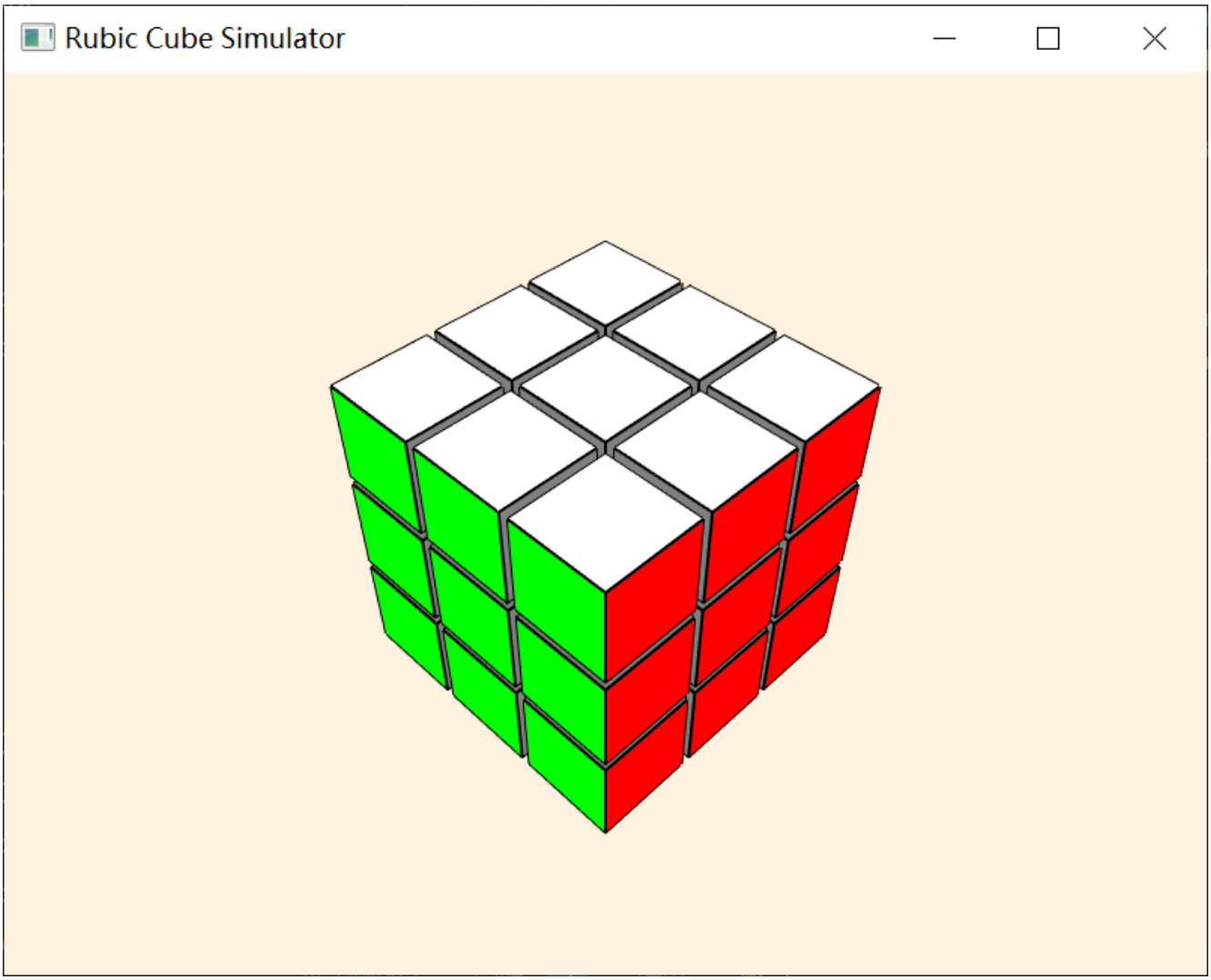
每一轮迭代中，我们组内的成员都有明确的分工，各自负责不同的任务模块。这样的分工可以确保每个成员可以专注于自己负责的部分，而不会产生冲突或干扰其他人的工作。我们使用GitHub作为项目管理工具，利用版本控制功能来管理代码的提交和合并。由于GitHub提供了分支管理和代码冲突解决的功能，组员的代码提交不会产生冲突，而且可以轻松地进行版本合并。

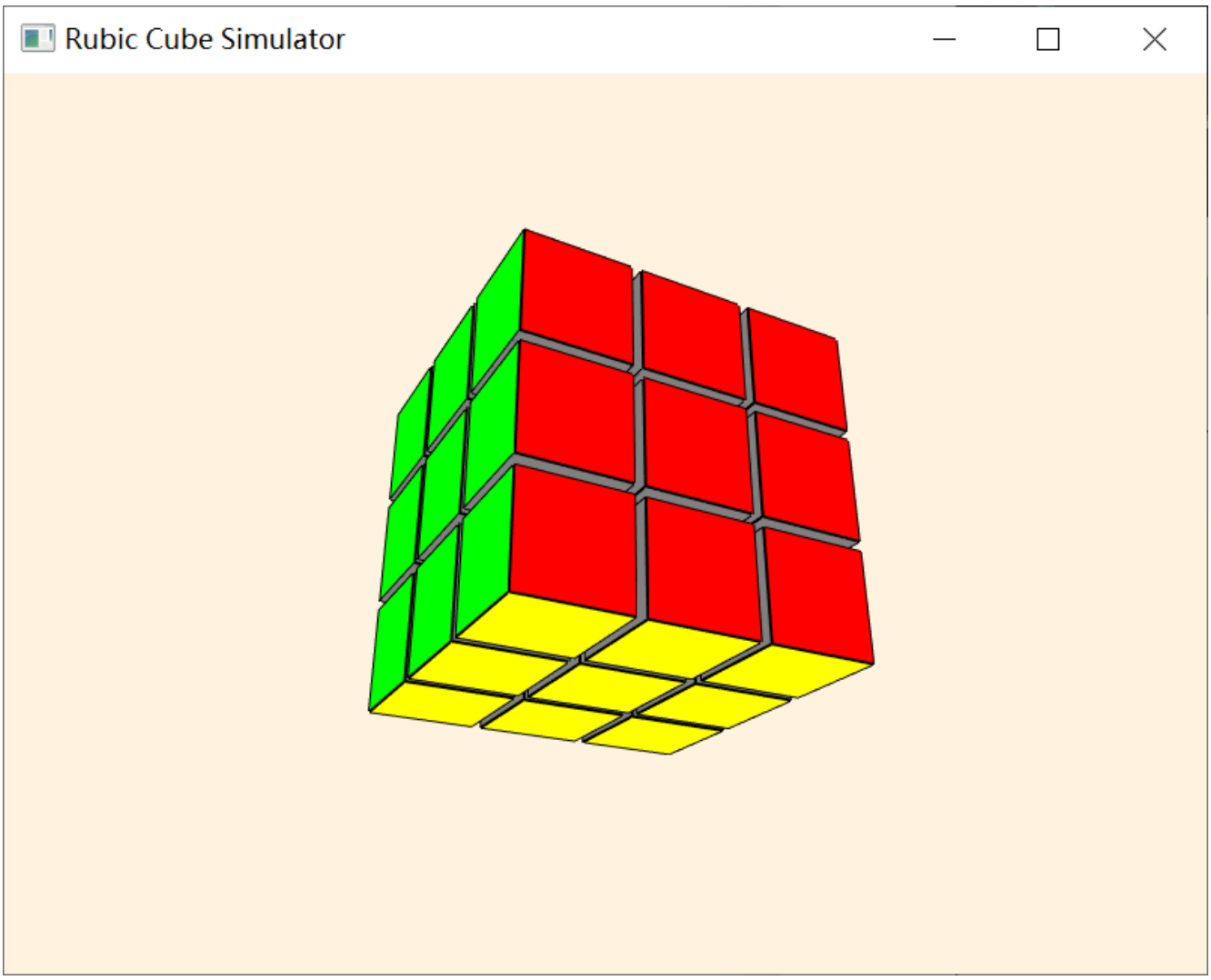
在项目中，组长负责协调整个项目的运行。组长负责监督项目进展，解决可能出现的问题，并协调组员之间的协作。通过组长的统一管理，我们可以确保项目按计划进行，并及时解决任何协作上的问题。

采用MVVM架构编写的工程使得我们不必等待对方的代码完成，而可以直接编写和测试各自的部分，极大提高了协作效率。

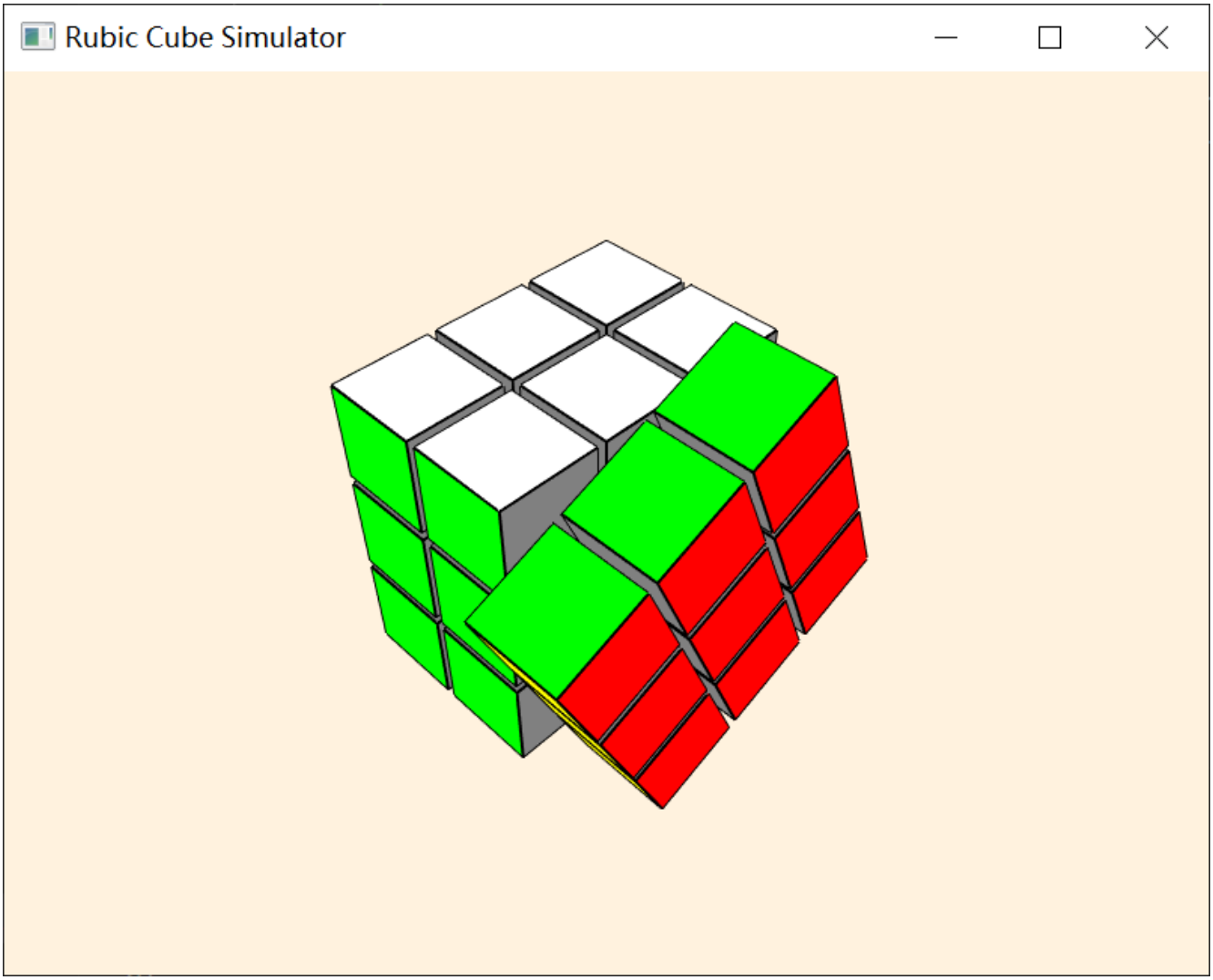
6　部分效果图

- 显示：

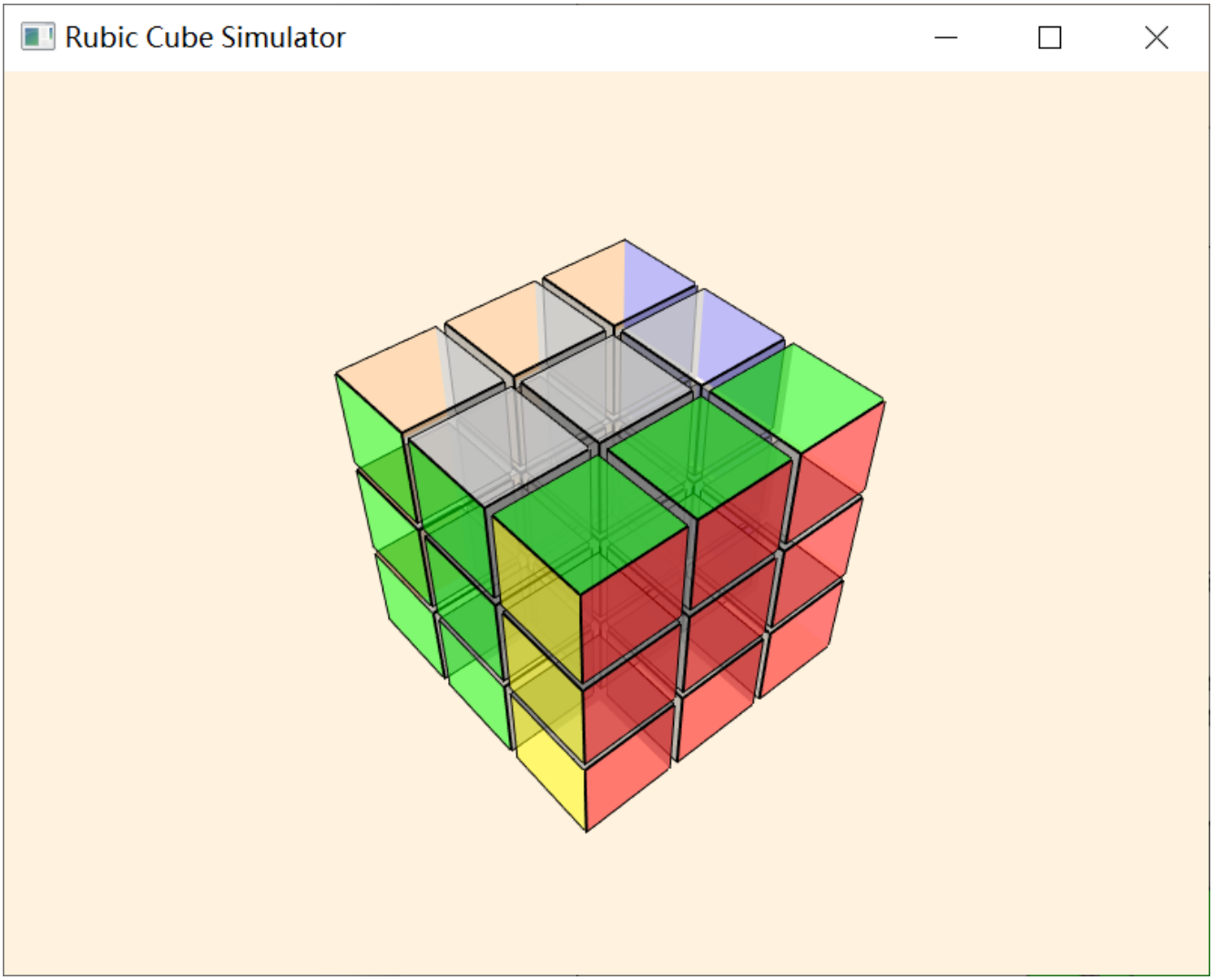



Rubic Cube Simulator

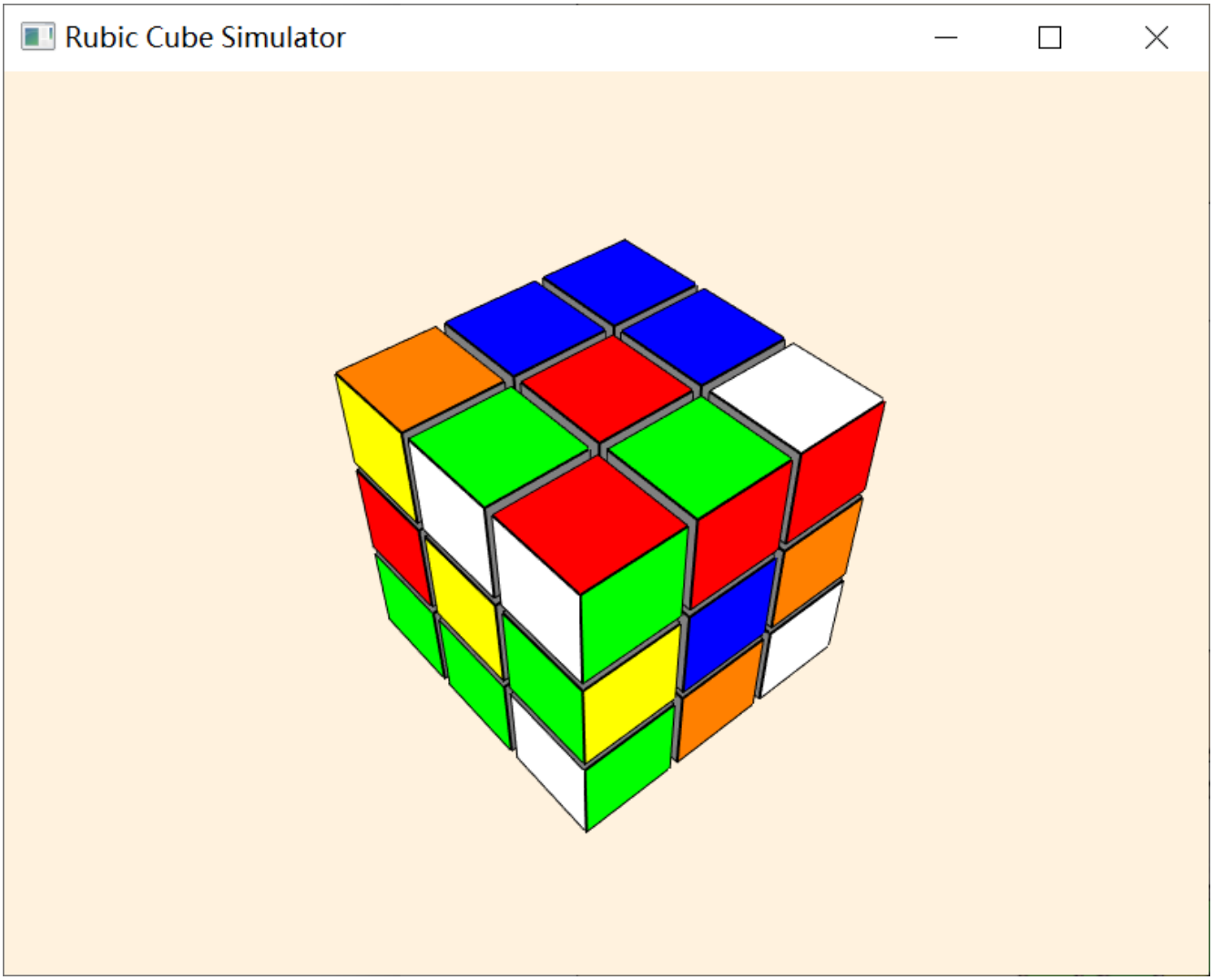
- 旋转



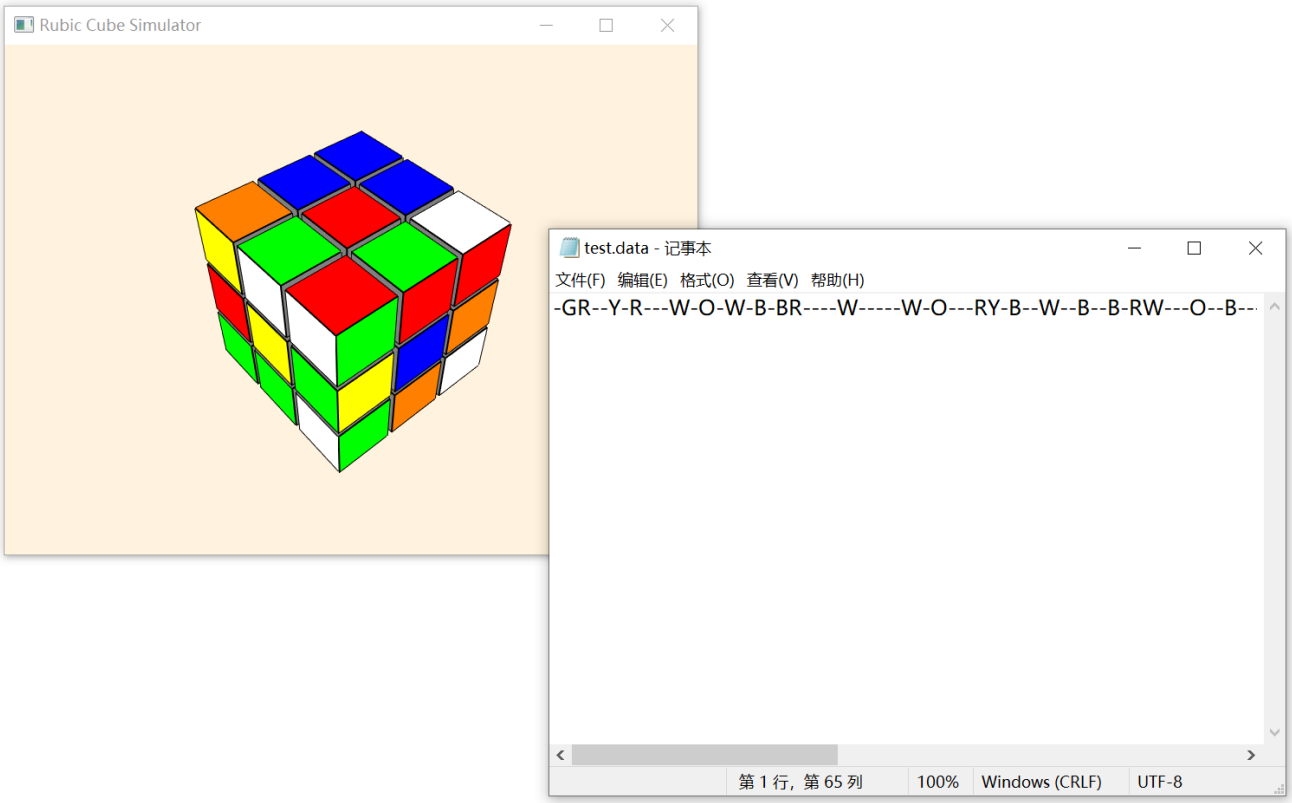
- 透明度变换



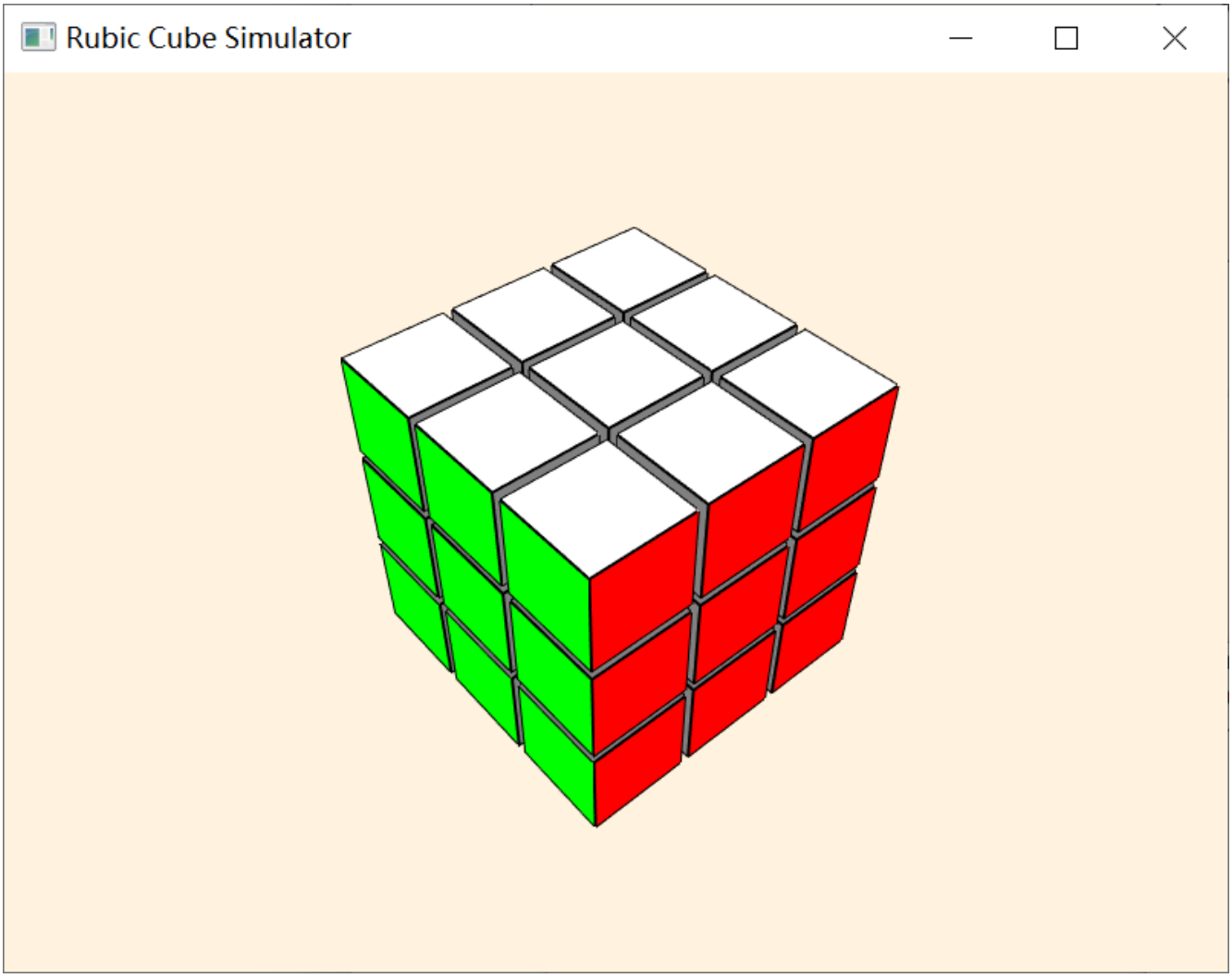
- 随机打乱



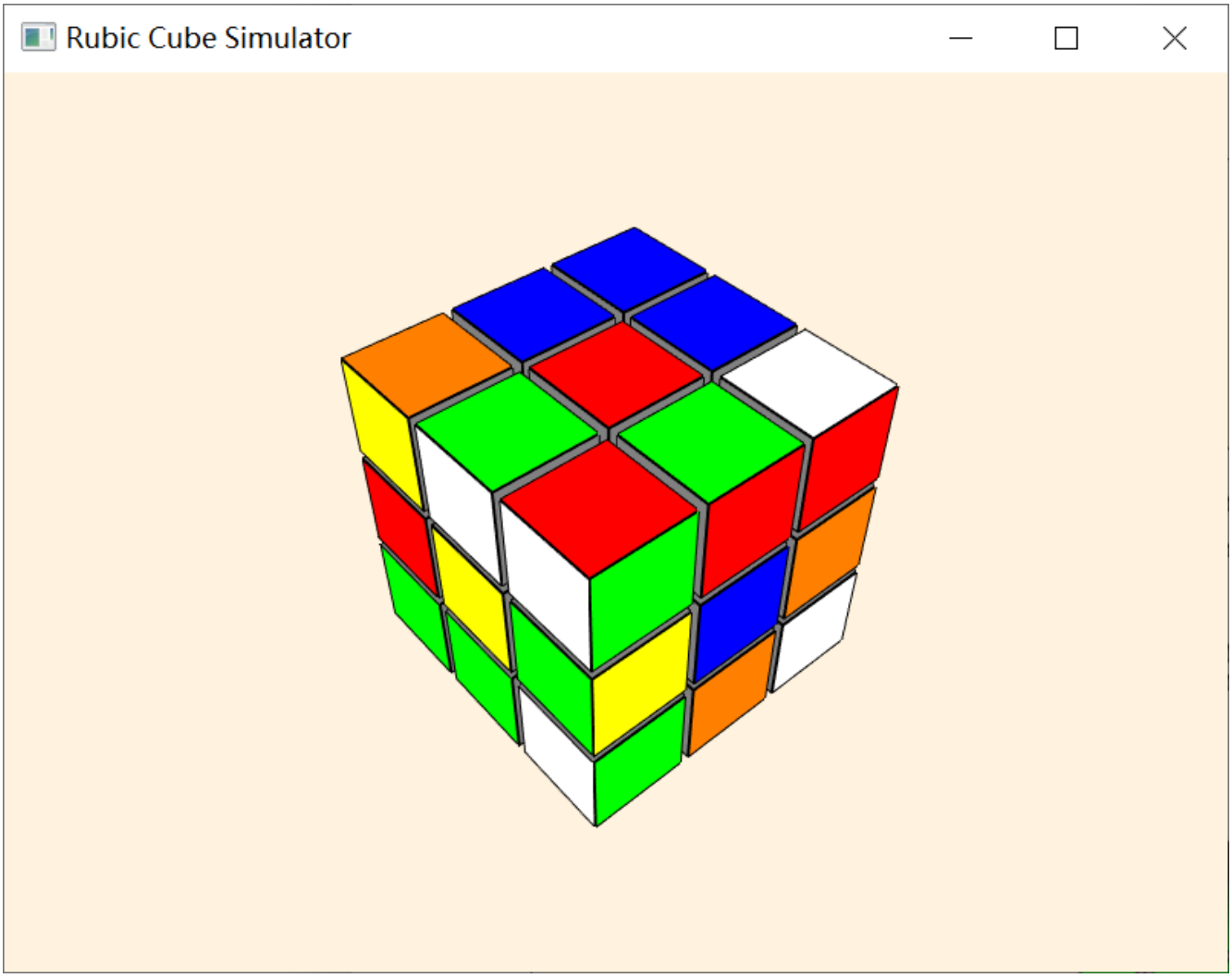
- 文件存储


Rubic Cube Simulator
test.data - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-GR--Y-R---W-O-W-B-BR----W-----W-O---RY-B--W--B--B-RW---O--B--
第 1 行，第 65 列
100%
Windows (CRLF)
UTF-8

- 重置魔方



- 加载文件



7 心得

7.1 王伟杰

学习这个课程之前，我对C++的项目管理以及各种软件架构模式没有任何了解，之前在其他课程上和组员合作完成项目时也体会到了很多合作开发的困难。在学习这门课时，我不仅在课堂上学到了很多团队开发和程序框架相关的知识，还亲身体验了团队项目工程的合作开发。在我们完成MVVM项目框架的时候，我深刻体会到了这个模式的便捷之处，每个人都只专注于自己的模块，不需要等待其他人的代码，而且在多次迭代的过程中，项目成员也可以对整个项目的代码进行一遍完整的开发，让大家都能学到很多东西。

在第一轮迭代中，我负责了Common层和App层以及xmake项目管理的任务。通过这个阶段的工作，我对C++和MVVM模式有了更深入的了解。在第二轮迭代中，我负责了View层的开发以及文档的撰写。这使我能够进一步加深对用户界面和文档编写的理解。通过GitHub的版本控制和分支管理，我能够与其他组员无缝协作，确保代码的顺利合并和项目的进展。

7.2 戴卿

之前我对项目管理和软件架构模式有一定的了解，但在实际的团队开发过程中还存在一些不足。通过参与这个课程的学习和项目实践，我对团队协作和软件架构有了更深入的认识和体验。

在第一轮迭代中，我负责了ViewModel层和Model层的任务。通过这个阶段的工作，我学会了如何合理地管理数据和处理业务逻辑，以及如何与其他层进行良好的交互。在第二轮迭代中，我转变了角色，负责了Common层和App层的开发工作。通过这个过程，我进一步提高了对C++编程语言的理解和应用能力。

参与这个项目让我更深入地了解了MVVM架构模式，并在实践中加以应用。通过模块化的开发和清晰的分工，我们能够更加高效地开发和测试各自的模块，大大提高了工作效率。同时，我也学到了如何撰写清晰而详细的文档，以便更好地与团队成员进行沟通和交流。

7.3 钟睿昕

项目中我担任了View层和文档撰写的任务，并在第二轮迭代中负责了ViewModel层和Model层的开发。通过参与这个项目，我收获了许多宝贵的经验和知识。

在第一轮迭代中，我致力于实现View层的功能，并负责了文档的撰写。通过开发View层，我深入理解了用户界面设计和实现的方法。我学会了如何与用户进行交互，并将魔方的状态以清晰的方式展示给用户。同时，通过撰写文档，我提高了自己的表达能力，并将项目的思路和功能清晰地记录下来，便于团队成员之间的沟通和协作。在第二轮迭代中，我转变了角色，负责了ViewModel层和Model层的开发工作。这个阶段使我更加熟练地运用了C++语言，并加深了对其语法和语言特性的理解。例如，我学会了如何定义纯虚类，以及如何进行指针的转化等技巧。这些知识的掌握使我能够更加灵活地编写代码，并提高了我的编程能力。

7.4 总结

学习这门短学期课程之前，我们对C++的项目管理和软件架构模式都没有太多的了解。然而，在课程的学习过程中，我们不仅在课堂上获得了大量的理论知识，还有机会亲自参与了一个真实的团队项目开发，从中学到了许多宝贵的经验。

首先，通过这门课程，我们学习到了不同的项目管理工具和方法。我们采用了GitHub作为项目管理工具，利用其版本控制和分支管理功能，使得团队成员可以并行开发各自的模块，并轻松地解决代码冲突和合并问题。这种协作方式大大提高了我们的工作效率，使得每个成员都能专注于自己负责的部分，而不会互相干扰或产生冲突。

其次，我们学习了MVVM（Model-View-ViewModel）软件架构模式，并将其应用到项目开发中。MVVM模式将用户界面（View）与数据和逻辑（Model）分离，并通过ViewModel来连接二者。这种模式的使用使得我们可以独立地开发和测试各个模块，而不需要等待其他成员的代码完成。这种并行开发的方式极大地提高了我们的协作效率，同时也增加了项目的稳定性和可维护性。

在多次迭代的过程中，我们有机会对整个项目的代码进行一遍完整的开发。这使得我们能够更好地理解整个项目的结构和功能，并在每次迭代中不断改进和优化代码。这种全局的开发经历让我们受益匪浅，不仅加深了对C++编程语言和OpenGL图形库的理解，还锻炼了我们的问题解决能力和团队协作能力。