**上海财经大学本科生毕业论文（设计）指导记录**

姓名： 张金豪 学号： 2018111481 专业： 计算机科学与技术

论文题目： 基于OpenGL的魔方模拟与复原

指导教师： 胡浩栋

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 联系方式 （面谈、电话、E-mail） | 主要内容 | 指导老师及 学生签名 |
|  |  | （对开题报告、确定题目的指导意见） |  |
|  |  | （对论文提纲的指导意见） |  |
|  |  | （对初稿的指导意见） |  |
|  |  | （对修改稿的指导意见） |  |
|  |  | （是否定稿、进入答辩以及其他指导意见） |  |

（指导意见如不够填写可附页）

年 月 日

基于OpenGL的3D魔方模拟与复原

摘 要

本文在充分了解魔方的构造原理之后，在visual studio2022的开发环境中，利用OpenGL函数库，利用空间坐标系对一个魔方进行了3D的模拟，并且利用在OpenGL中的点的变换赋予了魔方的随机旋转，随机打乱，以及重新开始的功能，在MFC的框架上，利用MFC设计了UI界面，界面有还原，打乱，以及复位的按钮，方便操作。在MFC的图形界面依旧可以用鼠标进行操作。魔方的求解算法有很多，千变万化，甚至有些算法可以在20步之内解出任何状态的魔方。

本文利用简单易懂的CFOP层序法，实现了魔方自动求解的过程，在还原的过程中魔方依旧可以利用鼠标转换视角。为方便观看，在MFC的交互界面中实现了一个计时器的功能。最后在程序中也提供了一个读取文本序列的接口，方便更好的理解魔方的绘制，构造与旋转。

本文的研究结论对于魔方爱好者模拟魔方的还原以及观看魔方求解有着一定的参考价值，也有利于理解机械臂，工业有关设备的模拟以及计算机视觉等其他相关领域。

关键词：OpenGL ，C++ ，魔方，MFC，CFOP

**The simulation and automatic restoration of 3D Rubik’s Cube based on OpenGL**

**Abstract**

OpenGL is the abbreviation of open graphics library. It is an interface widely used in the field of computer 3D drawing. OpenGL can be closely combined with visual studio and C ++, and the efficiency of OpenGL is very high. After fully understanding the construction principle of the Rubik's cube, this paper uses OpenGL function library to simulate a Rubik's cube in 3D in the development environment of visual studio 2022, and gives the functions of random rotation, random disturbance and restart of the Rubik's cube. On the framework of MFC, MFC is used to design the UI interface for convenient operation. The restoration algorithm of Rubik’s cube has been deeply studied in academic circles. The computer can solve the Rubik’s cube of any step within 20 steps.

In this paper, the simple and understandable CFOP sequence method is used to realize the automatic solution process of the Rubik’s cube. In the process of restoration, the mouse can still be used to change the perspective and angle of the Rubik’s cube. In order to facilitate viewing, a timing function is realized in MFC interaction. Finally, an interface for reading text sequence is also provided in the program to better understand the drawing, construction and rotation of Rubik’s cube.

The research conclusions of this paper have certain reference value for Rubik's cube enthusiasts to simulate the restoration of Rubik's Cube and watch the solving steps, and are also conducive to understanding other related fields such as robotic arms, simulation of industrial related equipment, and computer vision.

**Keywords:** OpenGL, Rubik’s cube, MFC, CFOP

目录

[一、序言 3](#_Toc99991134)

[二、正文 4](#_Toc99991135)

[2.1 魔方以及算法背景介绍 4](#_Toc99991136)

[2.1.1 魔方转动介绍 4](#_Toc99991137)

[2.1.2 还原算法介绍 4](#_Toc99991138)

[2.2所用工具介绍 4](#_Toc99991139)

[2.2.1 OpenGL 4](#_Toc99991140)

[2.2.2 MFC 5](#_Toc99991141)

[2.3 程序介绍-类 6](#_Toc99991142)

[2.3.1 MFC类 6](#_Toc99991143)

[2.3.2 OpenGL链接类 7](#_Toc99991144)

[2.3.3 魔方实现类 8](#_Toc99991145)

[2.3.3.1 cubecontrol类 8](#_Toc99991146)

[2.3.3.2 Cube类 9](#_Toc99991147)

[2.4 程序展示 11](#_Toc99991148)

[2.5 关键代码与难点 13](#_Toc99991149)

[三、参考文献 17](#_Toc99991150)

## 一、序言

本论文研究问题属计算机图形学的领域范围,由于二十一世纪是计算机技术时代,随着计算机技术的广泛普及,计算机图形学(computer graphics,简称CG)已成为计算机技术中最具挑战的研究领域之一,并正在蓬勃发展。简单地说,计算机图形学的主要内容就是研究人们怎样在电脑中显示图像、以及研究利用计算机进行图像的运算，处理和显示的相关原理与算法。虽然通常认为CG是指三维图形的处理，事实上也包括了二维图形及图像的处理[[1]](#footnote-1)。[[2]](#footnote-2)计算机图形学的应用包括： 游戏和电影，模拟和仿真，工业设计 (CAD/CAM)，可视化 (Tableau/ECharts)，虚拟现实和增强现实等领域。计算机图形学的目的是把现实图形作为二维图形显示在屏幕（像素点），图形学采用CPU计算，GPU加速；主要使用包括OpenGL，DirectX，Vulkan等开发工具。计算机图形学编程是基于着色器的，着色器编程学习起来比较困难，因为哪怕是再简单的东西进行绘制，也需要一系列特别复杂的过程并且从每个管线中传递下去图形数据才行。因此，GPU在计算机图形学编程中地位非凡。虽然计算机图形学学习困难，但是计算机图形学给生活带来的体验也是令人叹为观止的，像游戏中越来越逼真的画面，好莱坞中令人眼花缭乱的科技效果等，都基于计算机图形学的进步。

当今人工智能火热，而计算机视觉更是如日中天，计算机图形学与计算机视觉互为逆过程，计算机视觉是从图像中找到有用的信息，而计算机图形学是从一些信息中模拟出图像，计算机图形学中许多的算法，方法也对计算机视觉有着很重要的作用。可以这么理解，计算机图形学的发展，无论是对于其自身，还是对计算机视觉甚至对于显存的发展都作用重大。

一个小小的魔方，在进行旋转变换的时候就可以有4325亿亿种变化，当人在旋转魔方的时候都是根据旋转过后的每种情况进行穷举，制定不同的方案。本论文主要在研究如何利用OpenGL模拟出一个三维立体的魔方，以及如何实现魔方的渲染，旋转变换，以及复原等功能。在这个过程中，主要需要去解决魔方在三维坐标系中的构建，以及拧动魔方时魔方各个点坐标的转换，最大的难点还是在魔方还原过程中，对魔方每一面的情况进行穷举计算，这需要很大的代码量。

本文主要包括：上述的引言是本论文背景以及研究内容，目的以及问题的介绍，下文是魔方模拟算法以及还原算法的介绍，然后是魔方程序各种类的介绍，最后是魔方程序的展示。

# 二、正文

# 2.1 魔方以及算法背景介绍

# 2.1.1 魔方转动介绍

实际上，魔方不仅包括我们通常看到的3x3x3魔方，还包括机械装置，如中轴、弹簧、螺钉等。然而，当我们在计算机上模拟它时，我们只需要掌握27个立方体的特征以及由它们组成的完整立方体的特征。魔方旋转变换问题可以通过坐标变换来解决。因为魔方不同位置的小立方体特点不同，在程序设计中也需要拿出来考虑，首先对于魔方不同的块进行划分：位于魔方8个角的立方体称其为角块，角块的特点就是其三个面有颜色，在旋转变换的时候其以及还是会位于魔方8个角中的一个；位于魔方12条边上的小立方体称其为边块，每个块有2个颜色，在旋转变换的过程中其依旧是处于12条边的其中一条；位于每个面的中心的块称为中心块，只有一个颜色，旋转变换过后其依旧是某一面的中心；在魔方的正中央，这个小方框是没有任何意义的，可以无视它。

转动规则：首先将面对操作者的一层定义为Front，背对着操作者的一层定义为Back，在左边的层为Left，右边的层为Right，上层为Upper，下层为Down；我们用首字母的简写来表示，如F+表示前面一层向上旋转90度，F-表示前面一层向下旋转90度。

因为定义的是每次都是向上或者向下旋转90度，正如前面所提，旋转过程中，角块依旧是角块，边块依旧是边块，因此只需要循环交替的在每个块的下一个位置赋值，在对每个新位置上的块对其自身6个面的颜色做一次自我旋转（交替赋值），因此，魔方的一层的拧动就可以分解为一次平移操作和一次绕各块中心的旋转操作。这是魔方最基本的，也是在本程序中实现后续一系列复原方案的最重要的转换原理。

# 2.1.2 还原算法介绍

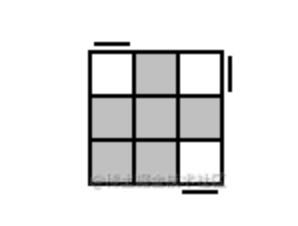
魔方在进行还原的过程中，有一个非常重要的特性，魔方的中心块以及各个面的中心块的相对位置不会改变，因此只需要关注角块和边块是否到达应有位置即可。由于我高中所用的魔方复原算法，包括大家所用的最广泛的算法是CFOP层序法，所以在这里所用的复原算法也是基于这个算法。首先将复原过程分为四步：

1.还原底部十字：这一步目标是还原四个底部边块，使得底部的出现十字，在这里使用的是贪心算法，每次都去寻找一个可以构成底面十字所需要的边块并且求出其到目标位置的一串移动步骤，在不影响其他十字边块的情况下归为，然后继续寻找下一个。

2.将底层二次恢复，在已实现1的情况下，将下两个层次全部恢复到原来的位置。

3.让最上面的平面恢复全色，再将顶部的边块重新排列，这样就能将整个大立方体恢复如初。

在还原过程中，除了复原第一层十字时没有特定的公式，其他层复原都可以通过特定的公式来解决，在还原过程中，由于公式复杂，在这里就不一一列举，举一个例子，例如顶层如下所示：



则可以通过U+L+U-R-U+L-U-R+这样的一个公式进行还原，其他公式也如此。

# 2.2所用工具介绍

# 2.2.1 OpenGL

OpenGL是图形硬件的软件接口。OpenGL是一个用于图形设备的软体界面.这个接口包含了数以百计的程式及功能，可以让程式设定物件及动作，以产生高品质的影像，特别是立体影像。大部分 OpenGL要求包括框架缓冲的显卡。大多数OpenGL需要图形硬件包含一个帧缓冲区。许多OpenGL调用都是关于绘制对象的，比如点、线、多边形和位图，但是一些绘制方法依赖于帧缓冲区的存在。此外，一些OpenGL特别关注帧缓冲区操作。

程序员视角：OpenGL是一组允许指定2D或3D几何对象的命令，以及控制如何将这些对象渲染到帧缓冲区的命令。在大多数情况下，OpenGL提供即时模式接口，这意味着指定一个对象会导致绘制该对象。一个使用OpenGL的典型程序首先调用在帧缓冲区中打开一个窗口，程序将在其中绘制。然后，分配它以分配GL上下文并将其与窗口关联。一旦分配了GL上下文，程序员就可以自由地发出OpenGL命令。一些调用用于绘制简单的几何对象，如点、线段和多边形，而另一些调用则影响这些基本体的渲染，包括它们的照明或着色方式，以及它们如何从用户的二维或三维模型空间映射到二维屏幕。对直接影响也有要求。

实现者视角：

OpenGL是一系列指令，这些指令会对图形的硬件运行产生一定的作用。如果硬件只包含一个可寻址的framebuffer，那么OpenGL必须几乎完全在主机CPU上实现。更典型的是，图形硬件可能包含不同程度的图形加速，从能够呈现二维线和多边形的光栅子系统到能够转换和计算几何数据的复杂浮点处理器。OpenGL实现者的任务是提供CPU软件接口，同时在CPU和图形硬件之间划分每个OpenGL命令的工作。为了在执行OpenGL调用时获得最佳性能，这种划分必须根据可用的图形硬件进行调整。

OpenGL维护了大量的状态信息。这个状态控制对象如何绘制到framebuffer中。有些状态对用户来说是直接可用的:他或她可以通过调用来获取它的值。然而，其中的一些，只有通过它对被画对象的影响才可见。这个规范的主要目标之一是使OpenGL的状态信息显式，阐明它是如何变化的，并指出它的影响是什么。

在该程序中，使用OpenGL完成魔方的仿真；同时，OpenGL具有使用键盘和鼠标进行交互的功能。这个函数使用起来非常方便，所以我们在这个程序中使用这个功能，用鼠标左键控制立方体角度的旋转，用鼠标右键控制立方体的哪一层旋转；当然，OpenGL可以使用键盘来控制交互的功能，我们在这个程序中没有使用过。

# 2.2.2 MFC

[[3]](#footnote-3)MFC是一个框架，是用C++语言对Windows程序设计中的相关的概念进行面向对象的封装。如果偏重形象简单的理解，可以把MFC理解为向导加代码加资源，当然向导在某些时候也是可有可无的。

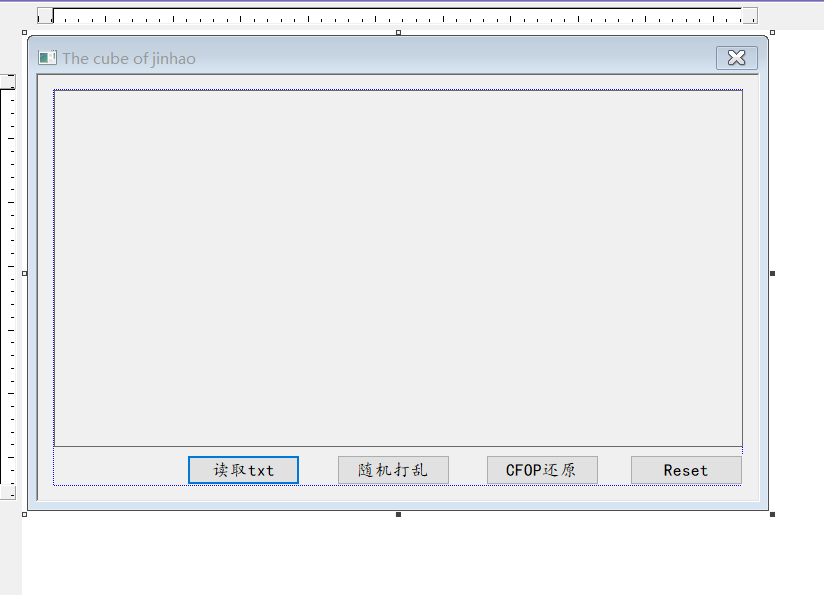
Microsoft 基础类 (MFC) 库为许多 Win32 和 COM API 提供了面向对象的包装器。 它可用于创建非常简单的桌面应用程序，但当需要开发具有多个控件的更复杂的用户界面时，它最为有用。MFC引用包括类、全局函数、全局变量和巨集组成微软的基本类库。MFC引用一般不会对继承或继承性操作进行说明。 每个类包含的各个层次结构图对于定位基类很有用。每个类的文档包括类概述、按类别分类的成员摘要以及成员函数、重载运算符和数据成员的主题。 公共和受保护的类成员仅在它们通常用于应用程序或派生类时才被记录。

在本程序中，考虑到魔方大多数是由键盘控制的，而键盘控制需要一个非常长的说明文档，用来表示转动哪一层，转动的角度，以及进行什么样的功能，在这个过程中，操作非常的繁琐，所以决定基于Visual Studio的开发环境，利用MFC做一个比较简单的交互对话框，对话框主要是基于MFC向导生成的，没有对向导生成的代码做太大的改变，只是改变了对话框的按钮文案，以及对话框的标题。希望能够利用MFC做一个可以不读说明文档直接上手的魔方。

# 2.3 程序介绍-类

## MFC类

（1）MFC类

这个类是用来在Visual Studio2022（下称VS）中用于创建MFC程序生成的主文件，在这个文件中包括其他项目特定的标头，包括resource.h文件等，并且在这个文件中声明了MFC应用程序类，下图为MFC设计的UI界面：

（2）MFCDlg类

这个类通过MFC应用程序向导创建的一个对话框类，这个类用来定义应用程序的主对话框的发生的事件以及行为，对话框的模板包含在oglMFCCube.rc中，当然这个rc文件可以通过VS2022用图形界面直接编辑。

在MFCDlg的.h头文件中，我们定义了下面的函数：

首先是开启旋转和复原线程：

static UINT CtrlThreadFunc(LPVOID pParam); //开启旋转线程

static UINT AutoThreadFunc(LPVOID pParam); //开启复原线程

然后是生成消息映射函数

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

afx\_msg void OnSize(UINT nType, int cx, int cy);

最后是生成的事件响应函数

afx\_msg void OnBnClickedRandom();//随机打乱

afx\_msg void OnBnClickedReturn();//还原

afx\_msg void OnBnClickedReset();//重新恢复原来颜色

声明了如下变量：

CCubeWnd m\_Cube;

在.cpp文件中，首先建立了一系列消息映射，随后实现了CtrlThreadFunc、AutoThreadFunc两个线程函数用以在BnClick事件中调用。

在 OnBnClicked中，有三个按键，这三个按键对应三个事件，这三个键的作用是：

void CoglMFCCubeDlg::OnBnClickedRandom()//随机打乱

void CoglMFCCubeDlg::OnBnClickedReturn()//复原

void CoglMFCCubeDlg::OnBnClickedReset()//复位

### 2.3.2 OpenGL链接类

（1）CubeWnd类

这个类是由VS2022创建的MFC类，用来实现一些基本的窗口变量和需要MFC进行调用的函数将会在这个类中进行声明以及实现，通过这个类的声明和实现，我们可以对MFC和OpenGL的逻辑进行分离封装，在渲染的时候只需要在消息函数中调用实现好的Init函数就可以实现。

在。H头文件中，

public:

int InitGL();

virtual void Reset();

void BottomCrossSolve(); //底面十字

void BottomCornerSolve(); //底面角块

void SecondSolve(); //中棱归位

void TopCrossFaceSolve(); //顶面十字

void TopCrossPosSolve(); //顶角面位

void TopLastSolve(); //收尾

protected:

virtual void SaveCube(); //保存当前魔方

virtual void RestoreCube(); //粘贴保存的魔方

private:

void RotateSide(); //鼠标左键旋转

void ReSizeGLScene( int width, int height); //OpenGL的ReSize设置

BOOL SetPixelformat(HDC hdc); //OpenGL的像素等设置

void Get3DCoord(Coord3D &lCoord, CPoint point); //获取三维坐标

void CenterPosSolve(); //中心块归位，白为底，前为蓝，作为魔方还原的基础参考

void SecondMethod(int Pos); //第二层的固定转法

void TopCrossFace(); //顶层十字面的转法

void TopCrossPos(); //顶层十字位的转法

void TopConer2Rotate(char Face); //顶层同侧对角旋转

void TopConer3Change(char Face); //顶层三角交换、旋转

int FindYellow(); //找顶层角块顶面为黄色的块

void DisplayTimer();

定义了一个结构体，用于表示三维坐标系：

struct Coord3D

{

double x;

double y;

double z;

};

声明了一些变量：

CCubeControl m\_3DCube;

CPoint m\_LastPointR; //用于鼠标右键移动旋转视图

float m\_RotateX; //用于3D视图旋转

float m\_RotateY; //用于3D视图旋转

HDC m\_hDC; //本窗口hDC

HGLRC m\_hRC; //用于OpenGL的hRc

Coord3D LdwCoord; //鼠标左键按下时的三维坐标

Coord3D LupCoord; //鼠标左键抬起时的三维坐标

int timeCounter; //计时器

同样，在这个部分的.cpp中，先建立了消息映射，随后，借助Oncreate和Oncreat消息映射函数实现了OpenGL与MFC的链接；

然后补全了OpenGL中绑定鼠标事件的函数，用来实现左键的转动魔方和右键的改变视图，在这些函数中包括：获取鼠标左键按下的位置，这个位置对应一个三维坐标；获取鼠标右键抬起时的位置，这个位置也对应一个三维坐标；按住右键实现旋转功能；当鼠标悬浮于魔方上面将鼠标需修改为手型。

在后面，实现了所有的魔方处理函数，在这些函数中主要是繁复的条件判断与魔方复原公式的程序化实现，实现难度主要在算法上，使用了很多if/else，存在一定量的冗余以及优化可能性。

## 2.3.3 魔方实现类

## 2.3.3.1 cubecontrol类

这个类是cube类的上层实现，主要是借助cube类与三维数组模拟一个真正的三维魔方。在.h文件中，声明了如下变量：

CCube \*m\_Cubes[3][3][3];

CCube \*m\_SavedCubes[3][3][3];

float m\_RotateX[3], m\_RotateY[3], m\_RotateZ[3];

float m\_CubeX, m\_CubeY, m\_CubeZ; //旋转角度的控制

定义了如下函数：

public:

void Render();

void RotateX(unsigned int block, int degrees);

void RotateY(unsigned int block, int degrees);

void RotateZ(unsigned int block, int degrees);

BOOL IsSameFace(int Face, int color); //Face面颜色是否全部为color

BOOL IsRestore(); //魔方是否已经还原

void Reset();

void Save();

void Restore();

int GetFaceColor(int Face, int Pos); //获取Pos位置的方块Face面的颜色

int GetFace(int color); //颜色为color的中心块所在的面

int GetCurPos(int OrgPos); //输入为原始位置，输出为当前位置

int GetOrgPos(int CurPos); //输入为当前位置，输出为原始位置

private:

void InitDraw(); //位置及颜色信息的初始化

int FindCenterPos(int color); //返回当前所在位置，输入为某个中心块颜色，中心块主要用于该面定色判断

在这个类的cpp中，我们首先实现了cubecontrol的构造函数和析构函数：**构造函数用于为类的对象进行初始化，并且如果没有定义构造函数，那么就会生成一个默认的构造函数。而析构函数**用来实现与构造函数相反的操作：释放对象使用的资源，并且销毁非static的成员。包括CCubeControl::CCubeControl(void)和CCubeControl::~CCubeControl(void)。

随后实现了Initdraw()和Render()函数，他们实现了3D魔方的渲染；下面为Render()函数伪代码：

Void CCubecontrol::Render(){

//旋转角度在本函数内部控制，旋转函数改变的是m\_ROtare

//通过循环函数循环绘制cube

//绘制过程中调用Cube的Render函数渲染}

随后，又实现了FindCenterPos()和GetFace()函数用以实现代码复用；

最后，实现了RotateX、RotateY、RotateZ函数块和IsSameFace、IsRestore、GetCurPos、GetOrgPos、GetFaceColor函数块，这两个函数块都为CCubeWnd类提供了服务，其中前者是为了在窗口中借助公式进行三维的渲染而写；后者则为自动复原代码提供了一些判定函数，从而使得代码重用性提高。

## 2.3.3.2 Cube类

这个类定义了一个魔方小块，每个魔方小块有六个面，每个面都有颜色，但是在实际情况下，一个魔方小块至多能有三个面有颜色（角块）。类内也定义了关于OpenGL光照参数的一些变量，除此之外，为了便于cubecontrol类的调用，cube类实现了面数组（由中心方块确定）的定义。

在这个类中声明了如下变量：

float m\_Sides[6][3]; //总体魔方的颜色

int Position; //方块位置，共27个

int FaceColor[6]; //小块六个面的颜色

//魔方小块边长

float m\_SideLength;

float m\_RoundWidth;

int m\_RoundSteps;

//光照参量

float m\_SideMaterial[4];

float m\_Shininess[1];

float m\_BevelDiffuse[4];

float m\_BevelMaterial[4];

float m\_BevelShininess[1];

该类定义了如下函数：

void Render();

void RotateFaceFW(int Axis); //正向

void RotateFaceRV(int Axis); //反向

void RotateFaceMe(int a, int b, int c, int d); //旋转面，进行颜色赋值

void MakeRoundedSide();

void MakeCornerSphere();

在这个类的cpp文件中，首先定义了M\_PI，这个常量是为了让最终绘制的魔方具有更好的视觉效果：#define M\_PI 3.1415926535897932f。

随后，实现了Cube的Render()函数，这里画的是立方体6个面，颜色配置是根据m\_Sides来确定的，因为设置了材质和光照，所以需要设置每个面法向量。

接着，实现了SetFaceColor()函数，代码非常简单，通过Position变量确定27个魔方小块的面颜色，例如：

if(Position == 1)

{

FaceColor[0] = 0;

FaceColor[1] = 0;

FaceColor[2] = WHITE;

FaceColor[3] = GREEN;

FaceColor[4] = ORANGE;

FaceColor[5] = 0;

}

这个小块有三个颜色，是橙色，绿色，白色三个面的交界角块。

而后又实现了MakeRoundedSide()、MakeCornerSphere()两个函数来画圆边和圆角以实现更真实的视觉效果。

接下来，我们实现了RotateFaceMe()、RotateFaceFW()、RotateFaceRV()三个函数以实现绕轴旋转，部分伪代码如下：

void CCube::RotateFaceMe(int a, int b, int c, int d) //面旋转，进行颜色赋值

{//利用中间值保存某个面的颜色

//颜色赋值

}

void CCube::RotateFaceFW(int Axis) //绕轴正方向旋转，即角度为正

{ switch (Axis) //1为x轴，2为y轴，3为z轴 //选择轴，1为x轴，2为y轴，3为z轴

{//绕X轴旋转，调用RotateFaceMe(3, 2, 1, 0); /

break;

// 绕Y轴旋转，调用RotateFaceMe(5, 1, 4, 3);

break;

// 绕Z轴旋转，调用RotateFaceMe(5, 2, 4, 0);

break;}}

void CCube::RotateFaceRV(int Axis) //绕轴反方向旋转，即角度为负

{ switch (Axis) //1为x轴，2为y轴，3为z轴 //选择轴，1为x轴，2为y轴，3为z轴

{//绕X轴旋转，调用RotateFaceMe(0, 1, 2 ,3); /

break;

// 绕Y轴旋转，调用RotateFaceMe(3, 4, 1, 5);

break;

// 绕Z轴旋转，调用RotateFaceMe(0, 4, 2, 5);

break;}}

最后，实现了Reset、Copy、SetColor函数以实现一些复位和定义操作。

# 程序展示

# 

图1.初始化结果

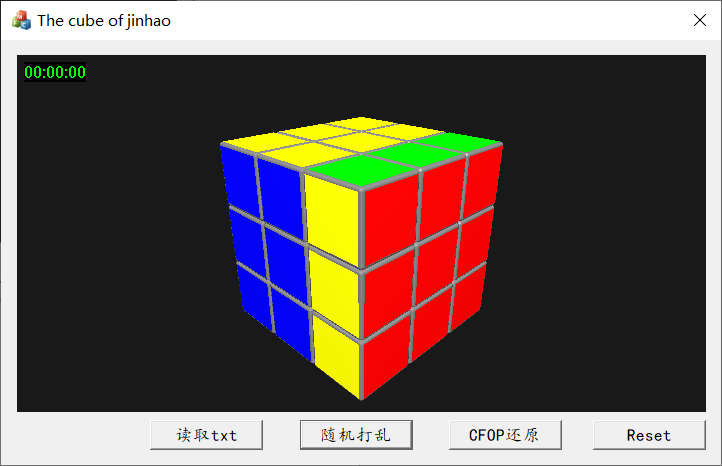


图2.右键可以用来实现视图旋转、左键用来实现魔方某一层的拧动

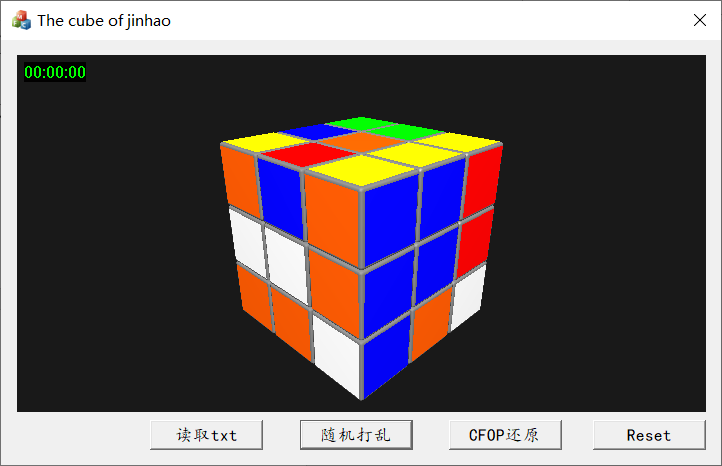


图3.点击打乱按钮之后，可以进行魔方的随机打乱

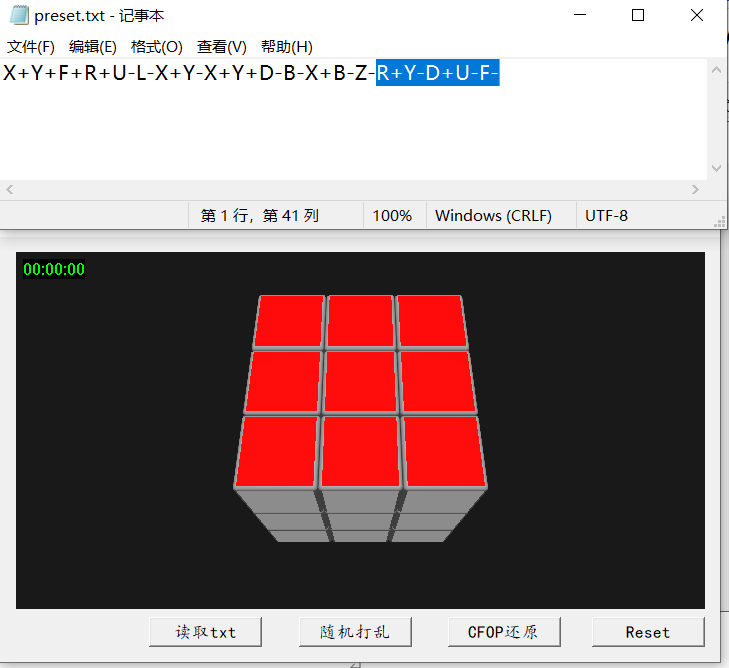


图4.点击读取txt后可以读取文本序列

点击复位魔方颜色重新回到六个全色，点击还原魔方可以基于CFOP算法进行还原。

视频演示放置在代码文件里来展示。

## 2.5 关键代码与难点

（1）在MFC中嵌入OpenGL

当程序中有事件发生时，MFC会调用特定的消息（以WM\_为前缀），例如OnPaint，OnSize，OnCreate等。可以通过在属性窗口中找到消息按钮添加消息事件。通过属性->消息窗口下找到WM\_PAINT事件。然后再下拉列表中选择<Add>OnPaint。注意到消息函数同时加到CubeWnd.h和CubeWnd.cpp中。但是，与用户添加的函数不同的是，这个函数有"afx\_msg"的前缀，还会在cpp文件的顶部加一个调用。

在OnPaint函数内部，只有小部分自动生成的代码需要更改。自己渲染（或者说是绘制）OpenGL窗口的方式与MFC不同，不同在OpenGL是通过定时器来绘制的。为了以防意外，我们必须在下面设置一个特定的帧速率。因为视窗要以计时器呈现，所以 OnPaint功能无法像原来那样被调用。所以，我们必须添加一个新的行来取代原来那个进行调用。在CubeWnd..cpp文件中的 OnPaint函数中添加上代码：

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

glTranslated(0.0,0.0,-7.0);

glRotatef(m\_RotateX,1,0,0);

glRotatef(m\_RotateY,0,1,0);

m\_3DCube.Render(); //调用函数，绘制3D魔方

glFlush();

SwapBuffers(m\_hDC);

DisplayTimer();

同理，创建OnCreate函数并在其中添加上代码：InitGL()便实现了在MFC中嵌入OpenGL

(2)初始化OpenGL与像素格式设置

HDC m\_hDC; //本窗口hDC

HGLRC m\_hRC; //用于OpenGL的hRc

void ReSizeGLScene( int width, int height); //OpenGL的ReSize设置

BOOL SetPixelformat(HDC hdc); //OpenGL的像素等设置

int InitGL();

随后，首先着手实现SetPixelformat()函数，定义一个”PIXELFORMATDESCRIPTOR”类型的对象pfd，并借助windgi中的ChoosePixelFormat和SetPixelFormat来实现功能。

（3）线程控制

由于渲染处理的计算量太大，导致程序经常出现假死的现象，因为程序只有一个线程，因此解决的办法就是用两个线程，一个线程用于随机打乱，一个线程用于复原。

查阅资料后发现，在MFC中把线程分为两类，一类为界面线程，一类为工作线程。两者的区别在于界面线程能够处理消息响应，而工作线程则不能。对于该项目来说，只要把计算的过程放到一个工作线程里来进行就可以了。

（4）控件随着对话框大小变化

要让对话框的控件随着对话框的大小的变化进行自动调整，是一件比较困难的事情，需要在WM\_SIZE的响应函数中对所有的控件用MoveWindow()或者SetWindowpos()设置其变化比例，程序比较繁琐。在项目中，在CubeDlg中引入了EasySIze库：

#include "EasySize.h"

并通过宏定义DECLARE\_EASYSIZE声明添加EASYSIZE。

随后，添加事件响应函数：

BEGIN\_EASYSIZE\_MAP(CoglMFCCubeDlg)

EASYSIZE(IDC\_FRAME, ES\_BORDER, ES\_BORDER, ES\_BORDER, ES\_BORDER, 0) //对控件IDC\_FRAME设置，响应EASYSIZE

EASYSIZE(IDC\_RETURN, ES\_BORDER, ES\_KEEPSIZE, ES\_KEEPSIZE, ES\_BORDER, 0)

EASYSIZE(IDC\_RESET, ES\_BORDER, ES\_KEEPSIZE, ES\_KEEPSIZE, ES\_BORDER, 0)

EASYSIZE(IDC\_RANDOM, ES\_BORDER, ES\_KEEPSIZE, ES\_KEEPSIZE, ES\_BORDER, 0)

END\_EASYSIZE\_MAP

EasySize用一些宏封装了内部的实现机制，使用起来方便快捷，帮助我们解决了控件大小随之变化的问题。

（5）CubeControl::RotateXYZ()

该三个方法分别以X/Y/Z为法向量旋转。block是所在的层，共三层，0,1,2分别对应魔方的左中右，方向与X/Y/Z一致，遵从OpenGL的右手原则。旋转函数实际控制的是m\_Rotate，是在Render()中实际作用，这样才有旋转的动画效果

函数需要控制的是角度达到90后，清除角度，对转动的方块赋值、面的旋转(赋值)，以RatateX为例，旋转代码如下：

void CCubeControl::RotateX(unsigned int block, int degrees)

{ m\_RotateX[block] += degrees;

if (degrees < 0 && m\_RotateX[block] == -90) //旋转角度达到90度就要更新位置信息

{m\_RotateX[block] = 0;//旋转90度后，必须清除角度，否则在其他轴旋转时会再次旋转，魔方模型就错误了

CCube \*tempcube = m\_Cubes[block][0][0];

m\_Cubes[block][0][0] = m\_Cubes[block][2][0];

m\_Cubes[block][2][0] = m\_Cubes[block][2][2];

m\_Cubes[block][2][2] = m\_Cubes[block][0][2];

m\_Cubes[block][0][2] = tempcube;

tempcube = m\_Cubes[block][1][0];

m\_Cubes[block][1][0] = m\_Cubes[block][2][1];

m\_Cubes[block][2][1] = m\_Cubes[block][1][2];

m\_Cubes[block][1][2] = m\_Cubes[block][0][1];

m\_Cubes[block][0][1] = tempcube;

for (int j = 0; j < 3; j++)

{for (int k = 0; k < 3; k++)

{m\_Cubes[block][j][k]->RotateFaceRV(1); //-90度对应反向旋转面

}

}

}

else if (degrees > 0 && m\_RotateX[block] == 90) //正向

{ m\_RotateX[block] = 0;

CCube \*tempcube = m\_Cubes[block][0][2];

m\_Cubes[block][0][2] = m\_Cubes[block][2][2];

m\_Cubes[block][2][2] = m\_Cubes[block][2][0];

m\_Cubes[block][2][0] = m\_Cubes[block][0][0];

m\_Cubes[block][0][0] = tempcube;

tempcube = m\_Cubes[block][0][1];

m\_Cubes[block][0][1] = m\_Cubes[block][1][2];

m\_Cubes[block][1][2] = m\_Cubes[block][2][1];

m\_Cubes[block][2][1] = m\_Cubes[block][1][0];

m\_Cubes[block][1][0] = tempcube;

for (int j = 0; j < 3; j++)

{

for (int k = 0; k < 3; k++)

{

m\_Cubes[block][j][k]->RotateFaceFW(1); //90度对应正向旋转面

}

}

}

if ( (m\_RotateX[0] == m\_RotateX[1] ||

m\_RotateX[0] == m\_RotateX[1]-90 ||

m\_RotateX[0] == m\_RotateX[1]+90) &&

(m\_RotateX[1] == m\_RotateX[2] ||

m\_RotateX[1] == m\_RotateX[2]-90 ||

m\_RotateX[1] == m\_RotateX[2]+90))

{

m\_CubeX += m\_RotateX[0];

m\_RotateX[0] = m\_RotateX[1] = m\_RotateX[2] = 0; //旋转角度复位为0

}

}

(6)Cube::Render()

函数的功能是画一个倒边、倒角的立方体，并配置颜色光照；通过glBegin(GL\_QUADS);代码选择四边形后，根据m\_Sides来确定颜色配置，依次绘制立方体的六个面。因设置了材质和光照，所以需要设置每个面法向量，因为有边角倒圆，在法向量方向最大，所以需要加上倒角宽度m\_RoundWidth，伪代码如下：

void CCube::Render() //画一个倒边、倒角的立方体，并配置颜色光照

{/\* 开始绘制6个面 \*/

glMaterialfv(GL\_FRONT, GL\_SPECULAR, m\_SideMaterial);

glMaterialfv(GL\_FRONT, GL\_SHININESS, m\_Shininess);

glBegin(GL\_QUADS); //选择四边形

//右面：矩形面绘制，则法向量应为(1, 0, 0)即x正方形

glColor3fv(m\_Sides[5]);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, m\_Sides[5]);

glNormal3f(1, 0, 0); //法向量设置为x正方向

//顶面：矩形面绘制，则法向量应为(0, 1, 0)即y正方向

//前面：矩形面绘制，则法向量应为(0, 0, 1)即z正方向

//后面：矩形面绘制，则法向量应为(0, 0, -1)即z反方向

//底面：矩形面绘制，则法向量应为(0, -1, 0)即y反方向

//左面：矩形面绘制，则法向量应为(-1, 0, 0)即x反方向

glEnd();

/\* 绘制8个倒角 \*/

//上、右、前三面交角

//上、左、前三面交角

//下、左、前三面交角

//下、右、前三面交角

//上、右、后三面交角

//上、左、后三面交角

//下、左、后三面交角

//下、右、后三面交角

/\* 绘制倒边，即12条棱 \*/

//上、右两面交棱

//上、左两面交棱

//下、左两面交棱

//下、右两面交棱

//前、右两面交棱

//前、左两面交棱

//后、左两面交棱

//后、右两面交棱

//后、上两面交棱

//前、上两面交棱

//前、下两面交棱

//后、下两面交棱

}

（7）Cube::MakeRoundedside()

如前所述，MakeRoundedSide()及MakeCornerSphere()是相对不那么重要的代码，主要为了视觉效果呈现：用圆或圆锥曲面在两个相邻曲面之间形成平滑过渡，称之为倒圆角。代码以及伪代码在这里不进行解读。

（8）Cube::RotateFaceMe（）

该部分的三个函数的基础便是RotateFaceMe()函数，用以在面旋转的过程中，进行颜色赋值。借助创建的TempFace对象，实现了旋转后的中心面颜色索引的更替。

void CCube::RotateFaceMe(int a, int b, int c, int d)

{

int TempFace = FaceColor[a];

FaceColor[a] = FaceColor[b];

FaceColor[b] = FaceColor[c];

FaceColor[c] = FaceColor[d];

FaceColor[d] = TempFace;

for(int i=0; i<6; i++)

SetColor(i, FaceColor[i]);

}

# 三、参考文献

[1].何智勇,贲可荣.基于OpenGL的魔方自动求解算法与实现[J].哈尔滨工业大学学报， 2004年第七期:1~3页

[2].贲可荣:计算机求解魔方算法[J].计算技术与自动化，1992年第3期 :31~37页

[3].Mark Segal & Kurt Akeley: “The OpenGL Graphics System: A Specification”[M], 2004:2-30

[4]. WOO M．OpenGL Programming Guide ［M］．Third Edition． ［s．l．］：Addison-Wesley，1999:1-10．

1. 匡天君,滕远道,王乘,徐明毅，基于OpenGL和MFC的三维图形开发 [↑](#footnote-ref-1)
2. 刘利刚，什么是计算机图形学 [↑](#footnote-ref-2)
3. 微软官方文档，https://docs.microsoft.com/en-us/cpp/mfc/mfc-desktop-applications?view=msvc-170 [↑](#footnote-ref-3)