# An OpenGL-based Method for Magic Cube Implementation

计创 周子昕201730601080

# Abstract

To meet the increasing demand of displaying magic cubes in computers, this paper puts forward a method to simulate how magic cubes work using OpenGL framework. The result suggests that the method works smoothly, without distortion and disorder.

# Introduction

Magic cubes are widely used as both recreational toys and teaching aids. They play an essential role in many fields, like in video games and in professional contest display. Therefore, it is of great significance to simulate how magic cubes work using computer. In this work, an OpenGL-based method to implement magic cube is put forward.

# Related work

## Magic cube

In He[1]’s work, for the implementation of a rotating magic cube, the work is divided into four parts. Storing information of the magic cube in computer, manipulating it globally and locally and drawing a single unit of the magic cube. Some collision detection is taken into consideration to ensure the program runs without errors. In the following sections, each part is explored more deeply to gain a more complete understanding of the related work.

## Data Representation

He and Ben use a 3D array to represent the color of each small cube. The online user uses a struct in C++ to store the position a point. Also, another struct is used to store the vertices of a small cube. In order to save space, translation is used when creating 27 cubes. Comparing the two methods, we can see that one requires more space, which the other is harder to implement.

## Global Rotation

Implementing global rotation is to implement an arcball. By receiving information from user input (mouse action) and mapping the windows coordinate to the world’s coordinate, we can achieve rotating the whole magic cube. It resembles rotating a ball as shown in fig 3-1.

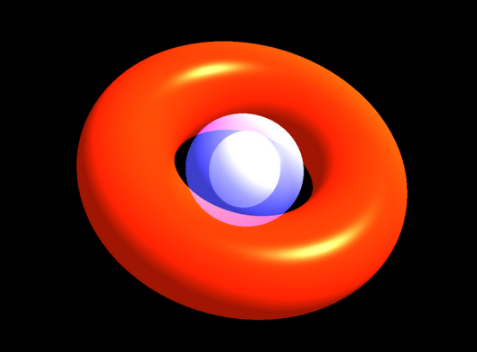


Fig 3-1

## Local Rotation

For local rotation, it needs to be considered in slices. He and Ben consider the rotation as times of 90 degrees. In this way, the rotation of each slice is simplified. All the operations of rotaion are composed of unit rotation. Then, each face of the cubes of this rotated slice is considered. The color and normal is changed when rotated. The online user use nine functions for each possible slice of the magic cube. To avoid collision, global variables are used to tell whether the magic cube can rotate in a particular direction.

## Drawing a cube

Both programmers draw the cubes using the vertices of the cubes, which is relatively easy to implement.

## Lighting

Lighting the scene includes setting the material of the cube and setting properties of the lights. Note that lighting may affect the color of the cubes displayed.

# Overview

From previous work, we can conclude that the implementation of magic cube is mainly separated into for parts: data representation, global rotation, local rotation and displaying a single cube. Taking the difficulty of implementation into consideration we decide to combine the two methods mentioned to represent data. For global rotation, simplification is used rather than arcball in this paper. For local rotation, the nine slices are handled separately just like the online user. And for drawing the cube, since the method of the online user is practical, we adopt this method. However, both methods discussed above has a drawback: when rotating, they use angles without restoring them. It could lead to bias after many rotations. In order to get rid of this problem, we make the program reset after each rotation to ensure no distortion.

# Technical sections

## Data Representation

A class is used to store one small cube, having three float numbers xa, ya and za to keep a record of the angle of each rotation. The u, b, l, r, f and ba variables represent the color of each face. Although it may seem to be a waste of space, it is easy for later implementation. Id of cubes are used to give the identity of the cubes, so that rotation after rotation can carry on.

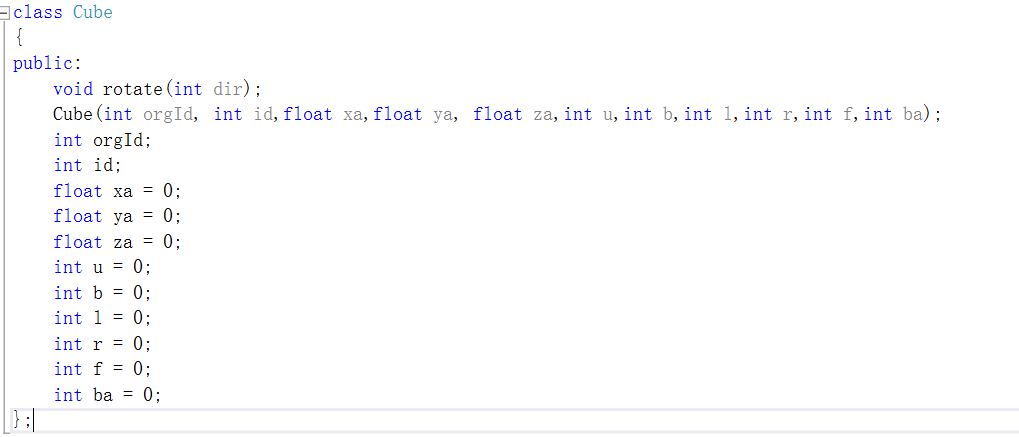


Fig 5-1

## Global Rotation

For simplicity and efficiency, we do not calculate the accurate angle of rotation. Instead, we use a number calculated by the distance of motion to represent rotation angle. Also, we check whether the angle is larger than 180 degrees. If it is, we decrease it by 360 degrees, ensuring the angle is in the range of (-180,180) degrees.

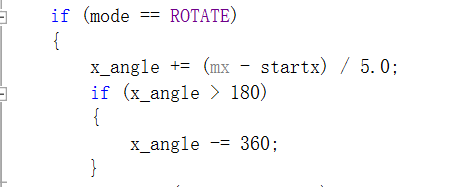


Fig 5-2

## Local Rotation

### Operation in one slice

This is one slice in a magic cube. All the operations happens in slices, and with 90 degrees as a basic unit.

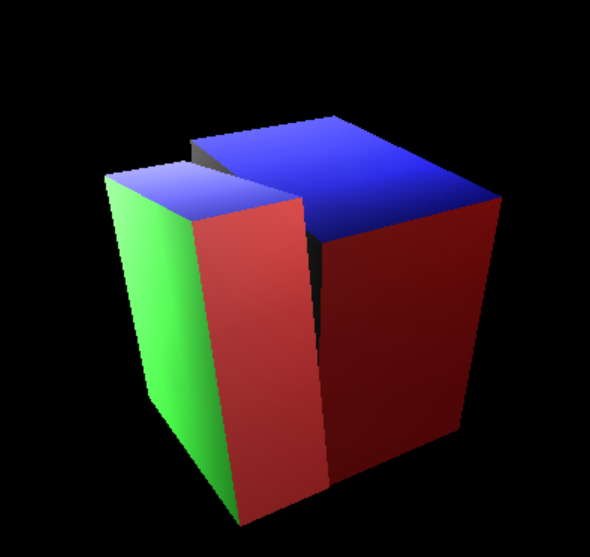


Fig 5-3

First, when mouse is pushed and moved, the increment angle of this slice is calculated and stored into the corresponding cubes’ attributes.

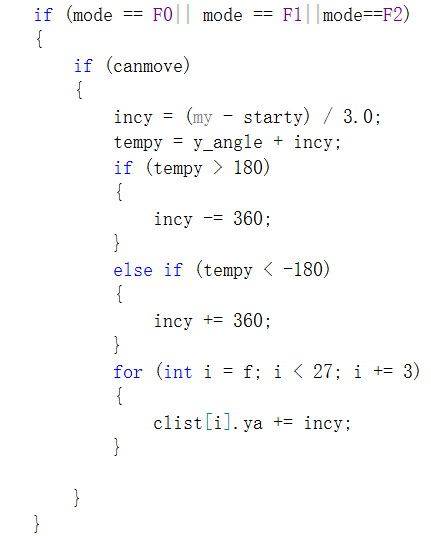


Fig 5-4

After this, when rotating, we simply draw the cubes using the rotation angles. But after each rotation, we require a format operation. This is to avoid collision between slices of different directions.

When key “F” is pressed, variable *mode* is set to be FORMAT and calculate the number of basic unit rotations.

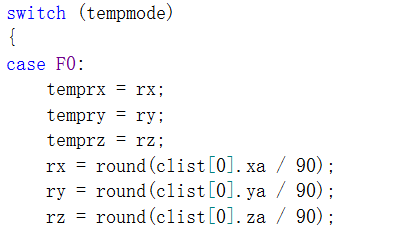


Fig 5-5

The positive and negative situations are handled differently. We only show negative situation here. For each rotation, we set the id of the cubes to be rotated 90 degrees. Also, the color of each face is rotated.

### Conflict-avoiding measures

As mentioned in last section, we require format operation after each rotation. In this way, no collision of different slices will happen.

## Drawing a cube

To draw a cube, we first rotate using the whole rotation angle.

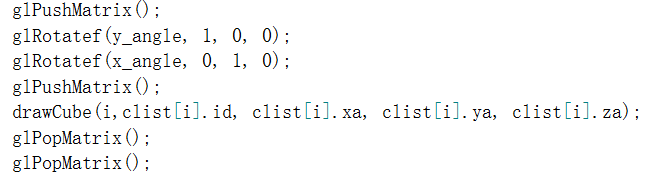
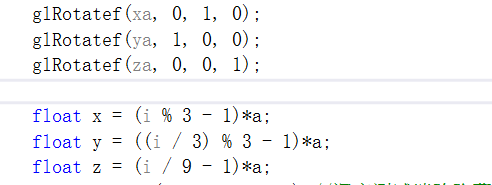


Fig 5-6

Then, we rotate using its local angle and calculate the position of this cube using index i.



Color of each face is decided using value stored in the cube class.

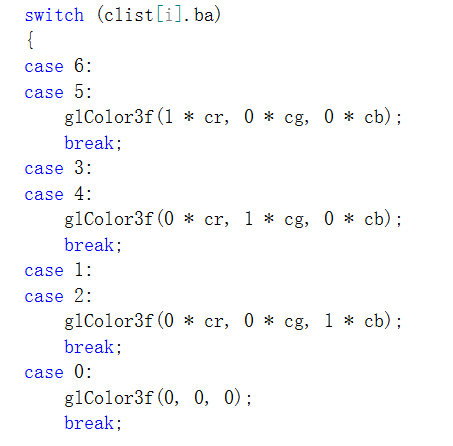


Fig 5-7

We draw quads for each face of the cube, defining the vertices of each face in sequence.

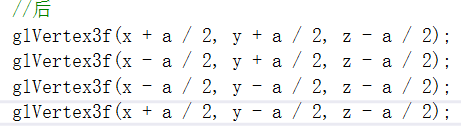


Fig 5-8

## Lighting

Two lights are utilized for lighting. One is point light, the other is spot light. The position of point light can be change by keyboard.

## Operations on keyboard

### Rotating whole magic cube

Key R

### Formatting the magic cube

Key F

### Rotating each slice

Front and back from left to right: key q w e

Left and right from back to front: key a s d

Clockwise and counterclockwise from bottom to top: key z x c

### Changing position of light

X increase and decrease: key u and i

Y increase and decrease: key j and k

Z increase and decrease: key n and m

# Results & discussion

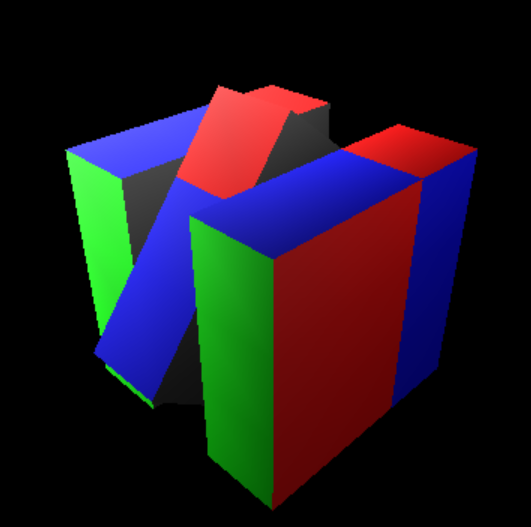


Fig 6-1

As demonstrated in the video, the program works well. However, we can see some drawbacks. First, there are some small cubes that behave abnormally in terms of lighting. They show a different color as the other cubes on the same surface. This may be caused by the edges of the cells.

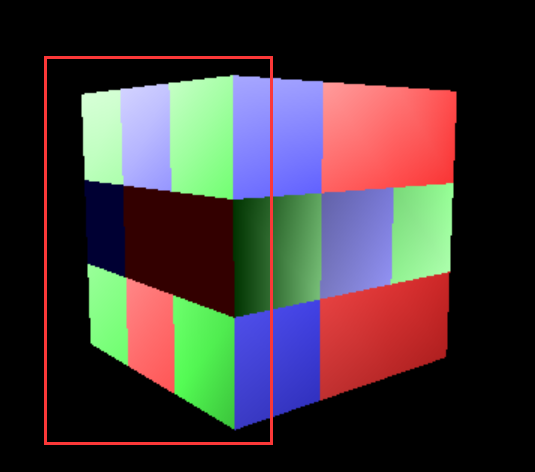


Fig 6-2

# Conclusions

In this paper, we put forward a practical and efficient method to simulate the working of a magic cube. Although working smoothly, there are still some room for this method to be improved. In future work, redundancy in code can be reduced, the deficiency of lighting can be got over and the texture mapping can be implemented.

Reference

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(07):893-895.
2. 网络用户.基于OpenGL的3D旋转魔方实现. https://wenku.baidu.com/view/a8f6a522777f5acfa1c7aa00b52acfc788eb9f1c.html