Лабараторная работа 8

Клишевич Вадим, M3105

main.cpp

#include <ctime>  
#include <cstdlib>  
#include <fstream>  
#include "cube.h"  
  
//Размер.  
#define CUBE\_SIZE 13  
//Как часто таймер обновляется.  
#define TIMER 10  
  
//Позиция света относительно центра.  
GLfloat lightPos[] = {0, 100, 200, 0};  
//Проекции угла поворота на оси.  
int xRot = 24, yRot = 34, zRot = 0;  
//Отдаление.  
double translateZ = -100.0;  
  
std::vector<char> solution;  
  
//Создаём куб.  
cube cube1;  
//Флаг того, крутится куб сам, или нет (будет переключаться правой кнопкой мыши).  
int timerOn = 0;  
int it = 0;  
int tmpx = 0;  
int tmpy = 0;  
int grad = 10;  
  
//Отображение.  
void display()  
{  
 glPushMatrix(); //Сохранить текущие координаты(изменить текущие координаты).  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
 glColor3f(1, 0, 0);  
 glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos); //Источник света.  
 glTranslatef(0, 0, translateZ); //В глубь.  
 glRotatef(xRot, 1, 0, 0); //Вращение по оси X.  
 glRotatef(yRot, 0, 1, 0); //Вращение по оси Y.  
 glTranslatef(CUBE\_SIZE / -2.0, CUBE\_SIZE / -2.0, CUBE\_SIZE / -2.0); //Имитирование вращения камеры.  
 cube1.draw();  
 glPopMatrix();  
 glutSwapBuffers(); //Обменивает буфера, отображая окно целиком после того, как оно уже сформировано.  
}  
  
//Меняем размеры окна.  
void reshape(int w, int h)  
{  
 glMatrixMode(GL\_PROJECTION); //Говорит о том, что команды относятся к проекту.  
 glLoadIdentity(); // Считывает текущую матрицу.  
 //Настройка перспективы просмотра.  
 GLfloat fAspect = (GLfloat)w/(GLfloat)h;  
 GLfloat zNear = 1;  
 GLfloat zFar = 1000.0;  
 GLfloat aspect = fAspect;  
 GLfloat fH = tan( float(30 / 360.0f \* 3.14159f) ) \* zNear;  
 GLfloat fW = fH \* aspect;  
 glFrustum( -fW, fW, -fH, fH, zNear, zFar );  
  
 glMatrixMode(GL\_MODELVIEW); //Говорит о том, что работы будет теперь просмотром, а не проектом.  
 glLoadIdentity();  
}  
  
void init()  
{  
 glClearColor (0.3, 0.3, 0.3, 0.0); //Фон.  
 srand(time(0));  
 //Освещение.  
 float diffuseLight[] = {0.2, 0.2, 0.2, 1}; //Цвет объекта под рассеянным освещением.  
 float ambientLight[] = {0.9, 0.9, 0.9, 1.0}; //Окружающее освещение.  
 glMateriali(GL\_FRONT, GL\_SHININESS, 128); //Яркость материалов спереди.  
 //Свойства материала для освещения.  
 glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuseLight);  
 glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambientLight);  
 //Включить освещение.  
 glEnable(GL\_LIGHT0);  
 glEnable(GL\_COLOR\_MATERIAL);  
 glEnable(GL\_DEPTH\_TEST); //Автоматически сохранять значения глубины для всех прошедших тест фрагментов и отбрасывать не прошедшие.  
 std::ifstream F("../cube.in");  
 cube cube2(F, 13);  
 cube1 = cube2;  
 glutSwapBuffers();  
}  
  
void specialKeys(int key, int, int)  
{  
 if (key == GLUT\_KEY\_DOWN)  
 {  
 xRot -= grad;  
 if (xRot < 0)  
 xRot += 360;  
 if (xRot >= 360)  
 xRot -= 360;  
 glutPostRedisplay();//Вне таймера.  
 }  
  
 if (key == GLUT\_KEY\_UP)  
 {  
 xRot += grad;  
 if (xRot >= 360)  
 xRot -= 360;  
 glutPostRedisplay();  
 }  
  
 if (key == GLUT\_KEY\_RIGHT)  
 {  
 yRot += grad;  
 if (yRot >= 360)  
 yRot -= 360;  
 glutPostRedisplay();  
 }  
  
 if (key == GLUT\_KEY\_LEFT)  
 {  
 yRot -= grad;  
 if (yRot < 0)  
 yRot += 360;  
 glutPostRedisplay();  
 }  
  
 if (key == GLUT\_KEY\_HOME)  
 {  
 translateZ += 5;  
 glutPostRedisplay();  
 }  
  
 if (key == GLUT\_KEY\_END)  
 {  
 translateZ -= 5;  
 glutPostRedisplay();  
 }  
  
}  
  
void keys(unsigned char key, int, int)  
{  
 glutSpecialFunc(specialKeys);  
 switch (key)  
 {  
 case 'r': {  
 cube1.small\_turn(1, grad);  
 break;  
 }  
 case 'L':  
 {  
 cube1.small\_turn(3, grad);  
 break;  
 }  
 case 'u':  
 {  
 cube1.small\_turn(4, grad);  
 break;  
 }  
 case 'D':  
 {  
 cube1.small\_turn(2, grad);  
 break;  
 }  
 case 'F':  
 {  
 cube1.small\_turn(0, grad);  
 break;  
 }  
 case 'b':  
 {  
 cube1.small\_turn(5, grad);  
 break;  
 }  
 case 'R': {  
 cube1.small\_turn(1, -grad);  
 break;  
 }  
 case 'l':  
 {  
 cube1.small\_turn(3, -grad);  
 break;  
 }  
 case 'U':  
 {  
 cube1.small\_turn(4, -grad);  
 break;  
 }  
 case 'd':  
 {  
 cube1.small\_turn(2, -grad);  
 break;  
 }  
 case 'f':  
 {  
 cube1.small\_turn(0, -grad);  
 break;  
 }  
 case 'B':  
 {  
 cube1.small\_turn(5, -grad);  
 break;  
 }  
 case 'y':  
 {  
 cube1.small\_turn(6, -grad);  
 break;  
 }  
 case 's':  
 {  
 cube\_sweep solver = cube1.solve();  
 solution = solver.solver();  
 break;  
 }  
 case 'g': {  
 cube\_sweep gen;  
 gen.generation();  
 cube cube2(gen, 13);  
 cube1 = cube2;  
 break;  
 }  
 case '[': {  
 cube\_sweep readed;  
 readed.read();  
 cube cube2(readed, 13);  
 cube1 = cube2;  
 break;  
 }  
 case ']': {  
 cube\_sweep written = cube1.solve();  
 written.write();  
 break;  
 }  
 default:  
 break;  
 }  
display();  
}  
  
void mouse(int key, int state, int x, int y)  
{  
 if (key == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)  
 {  
 timerOn = 1 - timerOn;  
 }  
  
 if (key == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)  
 {  
 tmpx = x;  
 tmpy = y;  
 glutPostRedisplay();  
 }  
 if (key == GLUT\_LEFT\_BUTTON && state == GLUT\_UP)  
 {  
 tmpx = -1;  
 tmpy = -1;  
 glutPostRedisplay();  
 }  
}  
  
void mousemove(int x, int y)  
{  
 if (tmpx > 0)  
 {  
 yRot += (x - tmpx);  
 tmpx = x;  
 if (yRot < 0)  
 yRot += 360;  
 if (yRot >= 360)  
 yRot -= 360;  
 glutPostRedisplay();  
 }  
 if (tmpy > 0)  
 {  
 xRot += (y - tmpy);  
 tmpy = y;  
 if (xRot < 0)  
 xRot += 360;  
 if (xRot >= 360)  
 xRot -= 360;  
 glutPostRedisplay();  
 }  
}  
  
void timer(int)  
{  
 glutTimerFunc(TIMER, timer, 0);  
 if (timerOn)  
 {  
 std::vector<char> comm = { 'r', 'l', 'u', 'd','f', 'b', 'R', 'L', 'U', 'D', 'F', 'B'};  
 // Если включен автоматический поворот, и смотрим  
 // если сейчас никакая грань не крутится, то начинаем крутить случайную,  
 // иначе крутим текущую.  
 if (cube1.rotated == -1)  
 keys(comm[rand() % 12], 0, 0);  
 else  
 if (cube1.angle[cube1.rotated] > 90)  
 cube1.small\_turn(cube1.rotated, -grad);  
 else  
 cube1.small\_turn(cube1.rotated, grad);  
 }  
 else  
 {  
 //СЕсли грань недокручено - докручиваем.  
 if (cube1.rotated != -1) {  
 if (cube1.angle[cube1.rotated]>90)  
 cube1.small\_turn(cube1.rotated, -grad);  
 else  
 cube1.small\_turn(cube1.rotated, grad);  
 }  
 //Сборка из вектора решений.  
 else if (it < solution.size()) {  
 keys(solution[it], 0, 0);  
 it++;  
 if (it == solution.size())  
 {  
 solution.clear();  
 it = 0;  
 }  
 }  
 }  
 display();  
}  
  
int main(int argc, char \*\*argv)  
{  
 glutInit(&argc, argv); //Инициализация GLUT.  
 //Для анимаций GLUT\_DOUBLE(в одном поворачиваем, во втором меняем)  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  
 glutInitWindowSize(800, 700);  
 glutInitWindowPosition(1, 1);  
 glutCreateWindow("Cube");  
 init();  
 glutDisplayFunc(display);  
 glutReshapeFunc(reshape);  
 glutKeyboardFunc(keys);  
 glutMouseFunc(mouse);  
 glutMotionFunc(mousemove);  
 glutTimerFunc(TIMER, timer, 0);  
 glutSpecialFunc(specialKeys);  
 //Основной цикл GLUT.  
 glutMainLoop();  
 return 0;  
}

cube.h

#ifndef LAB8\_CUBE\_H  
#define LAB8\_CUBE\_H  
  
#include "cube\_sweep.h"  
#include <GL/glut.h>  
#include <iostream>  
#include "small\_cube.h"  
#include <fstream>  
  
using namespace std;  
  
class cube {  
private:  
 small\_cube cubes[3][3][3];  
  
 double size;  
 int color[6];  
 cube\_sweep sweep; //Развёртка.  
  
public:  
 cube() = default;  
  
 cube(cube const& a);  
  
 cube(cube\_sweep const& Cur, double \_size);  
  
 cube(std::ifstream& F, double size);  
  
 //Рисуем.  
 void draw();  
  
 //Поворот на 90 градусов грани.  
 void full\_turn(int idx, int sign);  
  
 //Крутит грань под номером idx на угол \_angle (в градусах).  
 void small\_turn(int idx, int \_angle);  
  
 //Поворот всего кубика на 90 градусов.  
 void swapper();  
  
 //Переводим кубик-рубик в развёртку.  
 cube\_sweep solve();  
  
 //Храним номер грани, которая в данный момент поварачивается, или -1 если ничего не поварачивается.  
 int rotated = -1;  
  
 //Храним угол поворота каждой грани.  
 int angle[7];  
  
private:  
 unsigned int \_correct\_color[6] = {0xFFFFFF, 0xCD853F, 0x0000FF, 0xFF0000, 0x00FF00, 0xFFFF00};  
 small\_cube tmp[4][4];  
 bool check[3][3][3]{true};  
 //Перевод цветов в числа, для облегчения сборки.  
 void switcher(std::vector<brink> &br, int a, int b, int c, int i, int j, int k);  
};  
  
#endif

cube.cpp

#include <cstring>  
#include "cube.h"  
  
  
cube::cube(cube const &a)  
 :size(a.size), rotated(a.rotated), sweep(a.sweep)  
{  
 for (int i = 0; i<3; i++)  
 for (int j = 0; j<3; j++)  
 for (int k = 0; k<3; k++)  
 cubes[i][j][k] = a.cubes[i][j][k];  
 for (int i = 0; i<6; i++) {  
 angle[i] = a.angle[i];  
 color[i] = a.color[i];  
 }  
 rotated = -1;  
}  
  
cube::cube(cube\_sweep const &Cur, double \_size)  
{  
 size = \_size;  
 for (auto& i : cubes)  
 for (auto& j : i)  
 for (auto& k : j) {  
 k.size = double(\_size) / 3 \* 0.95; //Маленькие кубики(размер).  
 }  
 int i, j, k;  
 size = \_size;  
 rotated = -1;  
 std::vector<brink> br = Cur.\_cube;  
 //Раскрашиваем кубик.  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 cubes[i][j][2].set(4, \_correct\_color[br[4][2-j][i]]);  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 cubes[i][j][0].set(2, \_correct\_color[br[2][j][i]]);  
 for (k = 0; k<3; k++)  
 for (j = 0; j<3; j++)  
 cubes[k][0][j].set(0, \_correct\_color[br[0][2-j][k]]);  
 for (k = 0; k<3; k++)  
 for (j = 0; j<3; j++)  
 cubes[k][2][j].set(5, \_correct\_color[br[5][2-j][2-k]]);  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 cubes[0][i][k].set(3, \_correct\_color[br[3][2-k][2-i]]);  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 cubes[2][i][k].set(1, \_correct\_color[br[1][2-k][i]]);  
 //Считываем цвета с развёртки.  
 for (int i = 0; i<6; i++) {  
 color[i] = sweep.\_cube[i][1][1];  
 angle[i] = 0;  
 }  
}  
  
cube::cube(std::ifstream &F, double size)  
{  
 cube\_sweep \_new(F);  
 sweep = \_new;  
 cube C(sweep, 13);  
 \*this = C;  
 rotated = -1;  
}  
  
void cube::draw()  
{  
 //check показывает, находится ли в состоянии покоя деталь с координатами (i, j, k).  
 memset(check, true, sizeof(check));  
 //Крутится ли грань.  
 if (rotated!=-1) {  
 glPushMatrix();  
 int i, j, k;  
 int tmp1 = 2;  
 if ((rotated==4) || (rotated==2))  
 tmp1 = 0;  
 if ((rotated==0) || (rotated==5))  
 tmp1 = 1;  
 if (rotated==6)  
 tmp1 = 3;  
 switch (tmp1) {  
 case 0: {  
 if (rotated==4)  
 k = 2;  
 else k = 0;  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 check[i][j][k] = false;  
 //Теперь нужно покрутить грань под номером current на угол rotate[current]  
 //относительно центра этой грани.  
 //Для этого сдвинемся к центру, покрутим, сдвинемся обратно.  
 glTranslated(size/2, size/2, 0); //Сдвигаемся к центру.  
 glRotatef(angle[rotated], 0, 0, 1); //Крутим.  
 glTranslated(-size/2, -size/2, 0); //Сдвигаемся обратно.  
 //Рисуем.  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 cubes[i][j][k].draw(size/3\*i, size/3\*j, size/3\*k);  
 break;  
 }  
 case 1: {  
 if (rotated==5)  
 j = 2;  
 else j = 0;  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 check[i][j][k] = false;  
 glTranslated(size/2, 0, size/2);  
 glRotatef(angle[rotated], 0, 1, 0);  
 glTranslated(-size/2, 0, -size/2);  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 cubes[i][j][k].draw(size/3\*i, size/3\*j, size/3\*k);  
 break;  
 }  
 case 2: {  
 if (rotated==1)  
 i = 2;  
 else i = 0;  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 check[i][j][k] = false;  
 glTranslated(0, size/2, size/2);  
 glRotatef(angle[rotated], 1, 0, 0);  
 glTranslated(0, -size/2, -size/2);  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 cubes[i][j][k].draw(size/3\*i, size/3\*j, size/3\*k);  
 break;  
 }  
 case 3: {  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 check[i][j][k] = false;  
 glTranslated(size/2, size/2, 0);  
 glRotatef(angle[rotated], 0, 0, 1);  
 glTranslated(-size/2, -size/2, 0);  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 cubes[i][j][k].draw(size/3\*i, size/3\*j, size/3\*k);  
 }  
 }  
 glPopMatrix();  
 }  
 for (int i = 0; i<3; i++)  
 for (int j = 0; j<3; j++)  
 for (int k = 0; k<3; k++)  
 if (check[i][j][k])  
 cubes[i][j][k].draw(size/3\*i, size/3\*j, size/3\*k);  
 //Теперь рисуем те детали, которые не поварачивались выше,  
 //они отмечены check[i][j][k] = true.  
 glPopMatrix();  
}  
  
void cube::full\_turn(int idx, int sign)  
{  
 if (sign==-1)  
 sign = 3;  
 int k = 0, i = 0, j = 0;  
 //sign задаётся в зависимости он направления.  
 //sign = -1, sign = 1  
 //если sign = -1, значит крутим 3 раза.  
 int tmp1 = 2;  
 if ((rotated==4) || (rotated==2))  
 tmp1 = 0;  
 if ((rotated==0) || (rotated==5))  
 tmp1 = 1;  
 switch (tmp1) {  
 case 0: {  
 for (int it = 0; it<sign; it++) {  
 //Копируем повёрнутую на 90 градусов верхнюю/нижнюю грань.  
 //В массив tmp, затем грани присваиваем tmp.  
 //И не забываем повернуть каждую деталь этой грани.  
 if (rotated==4)  
 k = 2;  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 tmp[j][2-i] = cubes[i][j][k];  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 tmp[i][j].XOY(), cubes[i][j][k] = tmp[i][j];  
 }  
 break;  
 }  
 case 1: {  
 for (int it = 0; it<sign; it++) {  
 if (rotated==5)  
 j = 2;  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 tmp[k][2-i] = cubes[i][j][k];  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 tmp[i][k].YOZ(), cubes[i][j][k] = tmp[i][k];  
 }  
 break;  
 }  
 case 2: {  
 for (int it = 0; it<sign; it++) {  
 if (rotated==1)  
 i = 2;  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 tmp[k][2-j] = cubes[i][j][k];  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 tmp[j][k].XOZ(), cubes[i][j][k] = tmp[j][k];  
 }  
 break;  
 }  
 }  
}  
  
void cube::small\_turn(int idx, int \_angle)  
{  
 //Мы пытаемся покрутить грань с номером idx,  
 //значит нужно проверить что другая грань уже не крутится.  
 if (rotated==-1 || rotated==idx) {  
 //Обновляем поворот.  
 angle[idx] += \_angle;  
 if (angle[idx]<0)  
 angle[idx] += 360;  
 if (angle[idx]%90!=0)  
 rotated = idx;  
 else {  
 //Если угол стал кратным 90, то поварачиваем на массиве.  
 if (rotated==6)  
 swapper();  
 if ((idx==0) || (idx==5)) {  
 if (angle[idx]==90)  
 full\_turn(idx, 1);  
 else  
 full\_turn(idx, -1);  
 }  
 else {  
 if (angle[idx]==90)  
 full\_turn(idx, -1);  
 else  
 full\_turn(idx, 1);  
 }  
 angle[idx] = 0;  
 rotated = -1;  
 }  
 }  
}  
  
void cube::swapper()  
{  
 small\_cube tmp1;  
 int i, j, k;  
 for (k = 0; k<3; k++) {  
 tmp1 = cubes[0][0][k];  
 cubes[0][0][k] = cubes[2][0][k];  
 cubes[2][0][k] = cubes[2][2][k];  
 cubes[2][2][k] = cubes[0][2][k];  
 cubes[0][2][k] = tmp1;  
 tmp1 = cubes[1][0][k];  
 cubes[1][0][k] = cubes[2][1][k];  
 cubes[2][1][k] = cubes[1][2][k];  
 cubes[1][2][k] = cubes[0][1][k];  
 cubes[0][1][k] = tmp1;  
 }  
 full\_turn(3, -1);  
 i = 0;  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 cubes[i][j][k].YOZ();  
 for (i = 1; i<3; i++)  
 for (j = 0; j<3; j++)  
 for (k = 0; k<3; k++)  
 cubes[i][j][k].XOY();  
 angle[6] = 0;  
 rotated = -1;  
 angle[6] = 0;  
 rotated = -1;  
}  
  
cube\_sweep cube::solve()  
{  
 int i, j, k;  
 std::vector<brink> br(6);  
 //Для каждой грани.  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 switcher(br, 4, 2- j, i, i, j, 2);  
 for (i = 0; i<3; i++)  
 for (j = 0; j<3; j++)  
 switcher(br, 2, j, i, i, j, 0);  
 for (k = 0; k<3; k++)  
 for (j = 0; j<3; j++)  
 switcher(br, 0, 2-j, k, k, 0, j);  
 for (k = 0; k<3; k++)  
 for (j = 0; j<3; j++)  
 switcher(br, 5, 2-j, 2-k, k, 2, j);  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 switcher(br, 3, 2- k, 2 - i, 0, i, k);  
 for (i = 0; i<3; i++)  
 for (k = 0; k<3; k++)  
 switcher(br, 1, 2 -k, i, 2, i, k);  
 sweep.\_cube = br;  
 return sweep;  
}  
  
void cube::switcher(std::vector<brink> &br, int a, int b, int c, int i, int j, int k)  
{  
 switch (cubes[i][j][k].\_color[a])  
 {  
 case 0xFFFFFF:{br[a][b][c] = 0;  
 break;  
 }  
 case 0xCD853F:{br[a][b][c] = 1;  
 break;  
 }  
 case 0x000000FF:{br[a][b][c] = 2;  
 break;  
 }  
 case 0xFF0000:{br[a][b][c] = 3;  
 break;  
 }  
 case 0x00FF00:{br[a][b][c] = 4;  
 break;  
 }  
 case 0xFFFF00:{br[a][b][c] = 5;  
 break;  
 }  
 }  
}

small\_cube.h

#ifndef LAB8\_SMALL\_CUBE\_H  
#define LAB8\_SMALL\_CUBE\_H  
  
#include <iostream>  
#include <vector>  
#include <GL/glut.h>  
  
//Маленькие кубики.  
class small\_cube {  
public:  
 unsigned char \_rgb[3]; //Для того, чтобы покрасить кубики в с помощью функций "gl".  
 unsigned int \_color[6]; //Цвета для метода "set", который мы вызываем в других файлах.(верх, низ, впереди, сзади, лево, право)  
 double size; //Размер кубика.  
  
 small\_cube();  
  
 //Меням цвета кубиков при повороте в разных осях.  
 void XOY();  
  
 void XOZ();  
  
 void YOZ();  
  
  
 //Записываем цвета в массив с помощью сдвига(8(char) и 32(int)).  
 unsigned char \*RGB\_(unsigned int a);  
  
 //Рисование маленьких кубиков со смещением.  
 void draw(double x, double y, double z);  
  
 //Рисуем.  
 void draw();  
  
 //Меняем цвета.  
 void set(int ind, unsigned int color);  
};  
  
#endif

small\_cube.cpp

#include "small\_cube.h"  
  
small\_cube::small\_cube()  
{  
 size = 0.0;  
 //По умолчанию черный цвет.  
 for (unsigned int & i : \_color)  
 i = 0;  
}  
  
void small\_cube::XOY()  
{  
 unsigned int tmp = \_color[5];  
 \_color[5] = \_color[3];  
 \_color[3] = \_color[0];  
 \_color[0] = \_color[1];  
 \_color[1] = tmp;  
}  
  
void small\_cube::XOZ()  
{  
 unsigned int tmp = \_color[0];  
 \_color[0] = \_color[2];  
 \_color[2] = \_color[5];  
 \_color[5] = \_color[4];  
 \_color[4] = tmp;  
}  
  
void small\_cube::YOZ()  
{  
 unsigned int tmp = \_color[1];  
 \_color[1] = \_color[4];  
 \_color[4] = \_color[3];  
 \_color[3] = \_color[2];  
 \_color[2] = tmp;  
}  
  
unsigned char \*small\_cube::RGB\_(unsigned int a)  
{  
 \_rgb[0] = \_color[a] >> 16;  
 \_rgb[1] = \_color[a] >> 8;  
 \_rgb[2] = \_color[a];  
 return \_rgb;  
}  
  
void small\_cube::draw(double x, double y, double z)  
{  
 glPushMatrix(); //Сохранение текущих координат.  
 glTranslated(x, y, z); //Смещение относительно центра, где рисуем кубик.  
 glBegin(GL\_QUADS); //Неделимая часть того, из чего состоит кубик(квадраты).  
 glColor3ubv(RGB\_(4)); //Красим стороны(матрицы) куба.  
 glNormal3f(0, 0, 1); //Задаём нормаль(сторона наложения цвета).  
 glVertex3f(size, size, size); //Первая вершина.  
 glVertex3f(0, size, size); //Вторая вершина.  
 glVertex3f(0, 0, size); //Третья вершина.  
 glVertex3f(size, 0, size); //Четвертая вершина.  
 glColor3ubv(RGB\_(3));  
 glNormal3f(-1, 0, 0);  
 glVertex3f(0, size, size);  
 glVertex3f(0, size, 0);  
 glVertex3f(0, 0, 0);  
 glVertex3f(0, 0, size);  
 glColor3ubv(RGB\_(0));  
 glNormal3f(0, -1, 0);  
 glVertex3f(size, 0, size);  
 glVertex3f(0, 0, size);  
 glVertex3f(0, 0, 0);  
 glVertex3f(size, 0, 0);  
 glColor3ubv(RGB\_(1));  
 glNormal3f(1, 0, 0);  
 glVertex3f(size, size, 0);  
 glVertex3f(size, size, size);  
 glVertex3f(size, 0, size);  
 glVertex3f(size, 0, 0);  
 glColor3ubv(RGB\_(5));  
 glNormal3f(0, 1, 0);  
 glVertex3f(size, size, 0);  
 glVertex3f(0, size, 0);  
 glVertex3f(0, size, size);  
 glVertex3f(size, size, size);  
 glColor3ubv(RGB\_(2));  
 glNormal3f(0, 0, -1);  
 glVertex3f(size, 0, 0);  
 glVertex3f(0, 0, 0);  
 glVertex3f(0, size, 0);  
 glVertex3f(size, size, 0);  
 glEnd();  
 glPopMatrix(); //Восстанавливаем координаты центра.  
}  
  
void small\_cube::draw()  
{  
 draw(0.0, 0.0, 0.0);  
}  
  
void small\_cube::set(int ind, unsigned int color)  
{  
 \_color[ind] = color;  
}

cube\_sweep.h

#ifndef LAB8\_CUBE\_SWEEP\_H  
#define LAB8\_CUBE\_SWEEP\_H  
  
#include <random>  
#include <ctime>  
#include <vector>  
#include <iostream>  
#include <string>  
#include "brink.h"  
  
using namespace std;  
  
//Обработка ошибок для проверки на корректность кубика.  
class cube\_exeption  
{  
public:  
 explicit cube\_exeption(std::string s);  
 std::string get\_error();  
private:  
 std::string \_error;  
};  
  
//Развёртка кубика.  
class cube\_sweep  
{  
 friend class cube\_solver;  
public:  
 cube\_sweep();  
 cube\_sweep(cube\_sweep const &obj);  
 void read(string filename = "../cube.out");  
 void write(string filename = "../cube.out");  
 explicit cube\_sweep(std::vector<brink> const & v); //Массив из 6 граней.  
 explicit cube\_sweep(std::ifstream &F);  
 void generation();  
 bool is\_correct(); //Проверка на то, собрался ли последний уровень.  
 std::vector<char> solver();  
 void print();  
 std::vector<char> answer();  
  
 //Вертим грани.  
 void R();  
 void R1();  
 void L();  
 void L1();  
 void U();  
 void U1();  
 void F();  
 void F1();  
 void D();  
 void D1();  
 void B();  
 void B1();  
 //Поворот всего куба.  
 void y();  
  
 bool operator ==(cube\_sweep const & a) const;  
 std::vector<char> \_ans;  
 std::vector<brink> \_cube;  
};  
  
#endif

cube\_sweep.cpp

#include <iostream>  
#include <random>  
#include <ctime>  
#include <vector>  
#include <string>  
#include <fstream>  
#include "brink.h"  
#include "cube\_sweep.h"  
#include "cube\_solver.h"  
  
cube\_exeption::cube\_exeption(std::string s)  
 : \_error(std::move(s)) {}  
  
std::string cube\_exeption:: get\_error()  
{  
 return \_error;  
}  
  
cube\_sweep::cube\_sweep()  
{  
 brink a;  
 for (int i = 0; i < 6; i++)  
 {  
 a.create(i);  
 \_cube.push\_back(a);  
 }  
}  
  
  
cube\_sweep::cube\_sweep(cube\_sweep const &obj)  
{  
 \_cube = obj.\_cube;  
}  
  
cube\_sweep::cube\_sweep(std::vector<brink> const & v)  
{  
 \_cube = v;  
}  
  
//Записываем данные в развёртку.  
cube\_sweep::cube\_sweep(std::ifstream &F)  
{  
 try {  
 \_cube.resize(6);  
 std::string s;  
 for (int i = 0; i<3; i++) {  
 getline(F,s);  
 if (s.length()!=6)  
 throw cube\_exeption("Incorrect string length");  
 for (int j = 0; j<3; j++) {  
 switch (s[j+3]) {  
 case 'W':\_cube[4][i][j] = 0;  
 break;  
 case 'O':\_cube[4][i][j] = 1;  
 break;  
 case 'B':\_cube[4][i][j] = 2;  
 break;  
 case 'R':\_cube[4][i][j] = 3;  
 break;  
 case 'G':\_cube[4][i][j] = 4;  
 break;  
 case 'Y':\_cube[4][i][j] = 5;  
 break;  
 default:throw cube\_exeption("Incorrect color");  
 }  
 }  
 }  
 for (int i = 0; i<3; i++) {  
 getline(F, s);  
 if (s.length()!=12)  
 throw cube\_exeption("Incorrect string length");  
 std::vector<int> n\_brick{3, 0, 1, 5};  
 for (int k = 0; k<4; k++)  
 for (int j = 0; j<3; j++) {  
 switch (s[j+k\*3]) {  
 case 'W':\_cube[n\_brick[k]][i][j] = 0;  
 break;  
 case 'O':\_cube[n\_brick[k]][i][j] = 1;  
 break;  
 case 'B':\_cube[n\_brick[k]][i][j] = 2;  
 break;  
 case 'R':\_cube[n\_brick[k]][i][j] = 3;  
 break;  
 case 'G':\_cube[n\_brick[k]][i][j] = 4;  
 break;  
 case 'Y':\_cube[n\_brick[k]][i][j] = 5;  
 break;  
 default:throw cube\_exeption("Incorrect color");  
 }  
 }  
 }  
 for (int i = 0; i < 3; i++) {  
 getline(F, s);  
 if (s.length() != 6)  
 throw cube\_exeption("Incorrect string length");  
 for (int j = 0; j < 3; j++) {  
 switch (s[j+3]) {  
 case 'W':\_cube[2][i][j] = 0;  
 break;  
 case 'O':\_cube[2][i][j] = 1;  
 break;  
 case 'B':\_cube[2][i][j] = 2;  
 break;  
 case 'R':\_cube[2][i][j] = 3;  
 break;  
 case 'G':\_cube[2][i][j] = 4;  
 break;  
 case 'Y':\_cube[2][i][j] = 5;  
 break;  
 default:throw cube\_exeption("Incorrect color");  
 }  
 }  
 }  
 }  
 catch (cube\_exeption &ex) {  
 std::cerr << ex.get\_error() << std::endl;  
 \_cube.resize(0);  
 }  
}  
  
//Генерация рандомного кубика.  
void cube\_sweep::generation()  
{  
 std::mt19937 gen(time(0));  
 std::uniform\_int\_distribution<int> v(0, 3);  
 int num = v(gen);  
 for (int i = 0; i < num; i++)  
 R();  
 num = v(gen);  
 for (int i = 0; i < num; i++)  
 U();  
 num = v(gen);  
 for (int i = 0; i < num; i++)  
 L();  
 num = v(gen);  
 for (int i = 0; i < num; i++)  
 F();  
 num = v(gen);  
 for (int i = 0; i < num; i++)  
 D();  
 num = v(gen);  
 for (int i = 0; i < num; i++)  
 B();  
}  
  
//Сборка последнего слоя.  
bool cube\_sweep:: is\_correct()  
{  
 cube\_sweep full;  
 if (full == \*this)  
 return true;  
 cube\_solver solve(this);  
 return solve.\_solve();  
}  
  
//Проверка последнего слоя на сборку.  
std::vector<char> cube\_sweep::solver()  
{  
 cube\_solver solve(this);  
 try {  
 if (!solve.\_solve())  
 throw cube\_exeption("Incorrect state");  
 return \_ans;  
 }  
 catch (cube\_exeption ex)  
 {  
 std::cerr << ex.get\_error();  
 exit(1);  
 }  
}  
  
void cube\_sweep::print()  
{  
 try {  
 if (\_cube.empty())  
 throw cube\_exeption("Cube is empty");  
 std::string s = "WOBRGY";  
 for (int i = 0; i < 3; i++) {  
 std::cout << " ";  
 for (int j = 0; j < 3; j++)  
 std:: cout << s[\_cube[4][i][j]];  
 std::cout<< std:: endl;  
 }  
 for (int i = 0; i < 3; i++)  
 {  
 for (int j = 0; j < 3; j++)  
 std::cout << s[\_cube[3][i][j]];  
 for (int j = 0; j < 3; j++)  
 std::cout << s[\_cube[0][i][j]];  
 for (int j = 0; j < 3; j++)  
 std::cout << s[\_cube[1][i][j]];  
 for (int j = 0; j < 3; j++)  
 std::cout << s[\_cube[5][i][j]];  
 std::cout << std::endl;  
 }  
 for (int i = 0; i < 3; i++) {  
 std::cout << " ";  
 for (int j = 0; j < 3; j++)  
 std:: cout << s[\_cube[2][i][j]];  
 std::cout<< std:: endl;  
 }  
 }  
 catch (cube\_exeption ex)  
 {  
 std::cerr << ex.get\_error() << std::endl;  
 }  
}  
  
std::vector<char> cube\_sweep::answer()  
{  
 return \_ans;  
}  
  
void cube\_sweep::R()  
{  
 \_ans.push\_back('R');  
 std::vector<int> tmp = {\_cube[0][0][2], \_cube[0][1][2], \_cube[0][2][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][i][2] = \_cube[2][i][2];  
 \_cube[2][i][2] = \_cube[5][2 - i][0];  
 \_cube[5][2 - i][0] = \_cube[4][i][2];  
 \_cube[4][i][2] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[1].right\_wise();  
}  
  
void cube\_sweep::R1()  
{  
 \_ans.push\_back('r');  
 std::vector<int> tmp = {\_cube[0][0][2], \_cube[0][1][2], \_cube[0][2][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][i][2] = \_cube[4][i][2];  
 \_cube[4][i][2] = \_cube[5][2 - i][0];  
 \_cube[5][2 - i][0] = \_cube[2][i][2];  
 \_cube[2][i][2] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[1].left\_wise();  
}  
  
void cube\_sweep::L()  
{  
 \_ans.push\_back('L');  
 std::vector<int> tmp = {\_cube[0][0][0], \_cube[0][1][0], \_cube[0][2][0]};  
 tmp.push\_back(\_cube[0][0][0]);  
 tmp.push\_back(\_cube[0][1][0]);  
 tmp.push\_back(\_cube[0][2][0]);  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][i][0] = \_cube[4][i][0];  
 \_cube[4][i][0] = \_cube[5][2 - i][2];  
 \_cube[5][2 - i][2] = \_cube[2][i][0];  
 \_cube[2][i][0] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[3].right\_wise();  
}  
  
void cube\_sweep::L1()  
{  
 \_ans.push\_back('l');  
 std::vector<int> tmp = {\_cube[0][0][0], \_cube[0][1][0], \_cube[0][2][0]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][i][0] = \_cube[2][i][0];  
 \_cube[2][i][0] = \_cube[5][2 - i][2];  
 \_cube[5][2 - i][2] = \_cube[4][i][0];  
 \_cube[4][i][0] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[3].left\_wise();  
}  
  
void cube\_sweep::U()  
{  
 \_ans.push\_back('U');  
 std::vector<int> tmp = {\_cube[0][0][0], \_cube[0][0][1], \_cube[0][0][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][0][i] = \_cube[1][0][i];  
 \_cube[1][0][i] = \_cube[5][0][i];  
 \_cube[5][0][i] = \_cube[3][0][i];  
 \_cube[3][0][i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[4].right\_wise();  
}  
  
void cube\_sweep::U1()  
{  
 \_ans.push\_back('u');  
 std::vector<int> tmp = {\_cube[0][0][0], \_cube[0][0][1], \_cube[0][0][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][0][i] = \_cube[3][0][i];  
 \_cube[3][0][i] = \_cube[5][0][i];  
 \_cube[5][0][i] = \_cube[1][0][i];  
 \_cube[1][0][i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[4].left\_wise();  
}  
  
void cube\_sweep::F()  
{  
 \_ans.push\_back('F');  
 std::vector<int> tmp = {\_cube[3][0][2], \_cube[3][1][2], \_cube[3][2][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[3][i][2] = \_cube[2][0][i];  
 \_cube[2][0][i] = \_cube[1][2 - i][0];  
 \_cube[1][2 - i][0] = \_cube[4][2][2 - i];  
 \_cube[4][2][2 - i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[0].right\_wise();  
}  
  
void cube\_sweep::F1()  
{  
 \_ans.push\_back('f');  
 std::vector<int> tmp = {\_cube[3][0][2], \_cube[3][1][2], \_cube[3][2][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[3][i][2] = \_cube[4][2][2 - i];  
 \_cube[4][2][2 - i] = \_cube[1][2 - i][0];  
 \_cube[1][2 - i][0] = \_cube[2][0][i];  
 \_cube[2][0][i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[0].left\_wise();  
}  
  
void cube\_sweep::D()  
{  
 \_ans.push\_back('D');  
 std::vector<int> tmp = {\_cube[0][2][0], \_cube[0][2][1], \_cube[0][2][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][2][i] = \_cube[3][2][i];  
 \_cube[3][2][i] = \_cube[5][2][i];  
 \_cube[5][2][i] = \_cube[1][2][i];  
 \_cube[1][2][i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[2].right\_wise();  
}  
  
void cube\_sweep::D1()  
{  
 \_ans.push\_back('d');  
 std::vector<int> tmp = {\_cube[0][2][0], \_cube[0][2][1], \_cube[0][2][2]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[0][2][i] = \_cube[1][2][i];  
 \_cube[1][2][i] = \_cube[5][2][i];  
 \_cube[5][2][i] = \_cube[3][2][i];  
 \_cube[3][2][i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[2].left\_wise();  
}  
  
void cube\_sweep::B()  
{  
 \_ans.push\_back('B');  
 std::vector<int> tmp = {\_cube[3][0][0], \_cube[3][1][0], \_cube[3][2][0]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[3][i][0] = \_cube[4][0][2 - i];  
 \_cube[4][0][2 - i] = \_cube[1][2 - i][2];  
 \_cube[1][2 - i][2] = \_cube[2][2][i];  
 \_cube[2][2][i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[5].right\_wise();  
}  
  
void cube\_sweep::B1()  
{  
 \_ans.push\_back('b');  
 std::vector<int> tmp = {\_cube[3][0][0], \_cube[3][1][0], \_cube[3][2][0]};  
 for (int i = 0; i < 3; i++) {  
 \_cube[3][i][0] = \_cube[2][2][i];  
 \_cube[2][2][i] = \_cube[1][2 - i][2];  
 \_cube[1][2 - i][2] = \_cube[4][0][2 - i];  
 \_cube[4][0][2 - i] = tmp[i];  
 }  
 tmp.resize(0);  
 \_cube[5].left\_wise();  
}  
  
void cube\_sweep::y()  
{  
 \_ans.push\_back('y');  
 brink \_brink;  
 \_brink = \_cube[0];  
 \_cube[0] = \_cube[1];  
 \_cube[1] = \_cube[5];  
 \_cube[5] = \_cube[3];  
 \_cube[3] = \_brink;  
 \_cube[4].right\_wise();  
 \_cube[2].left\_wise();  
}  
  
bool cube\_sweep:: operator ==(cube\_sweep const&a) const  
{  
 return a.\_cube == \_cube;  
}  
  
void cube\_sweep::read(string filename) {  
 ifstream fin(filename);  
 for (int i = 0; i < 6; i++) {  
 for (int j = 0; j < 3; j++) {  
 for (int k = 0; k < 3; k++) {  
 fin >> \_cube[i][j][k];  
 }  
 }  
 }  
 fin.close();  
}  
  
void cube\_sweep::write(string filename) {  
 ofstream fout(filename);  
 for (int i = 0; i < 6; i++) {  
 for (int j = 0; j < 3; j++) {  
 for (int k = 0; k < 3; k++) {  
 fout << \_cube[i][j][k] << " ";  
 }  
 fout << endl;  
 }  
 fout << endl;  
 }  
 fout.close();  
}