МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №8

по дисциплине: Компьютерная графика

тема: «текстуры»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Гибкин Ю. С.

Белгород 2020 г.

ImageLoader.h

#pragma once

#include <assert.h>

#include <fstream>

using namespace std;

//Represents an image

class Image {

public:

Image(char\* ps, int w, int h);

~Image();

char\* pixels;

int width;

int height;

};

//Reads a bitmap image from file.

Image\* loadBMP(const char\* filename);

GLuint loadTexture(Image\* image) {

GLuint\* textures = new GLuint[11];//

glGenTextures(12, textures); //make room for 12 texture//

glBindTexture(GL\_TEXTURE\_2D, \*textures);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, image->width, image->height,

0, GL\_RGB, GL\_UNSIGNED\_BYTE, image->pixels); // actual pixel data

return \*textures;

}

GLuint \_textureId1;

GLuint \_textureId2;

GLuint \_textureId3;

GLuint \_textureId4;

GLuint \_textureId5;

GLuint \_textureId6;

Image::Image(char\* ps, int w, int h) : pixels(ps), width(w), height(h) {}

Image::~Image() {

delete[] pixels;

}

namespace {

// Converts a four-character array to an integer, using little-endian form

int toInt(const char\* bytes) {

return (int)(

((unsigned char)bytes[3] << 24) |

((unsigned char)bytes[2] << 16) |

((unsigned char)bytes[1] << 8) |

(unsigned char)bytes[0]

);

}

//Converts a two-character array to a short, using little-endian form

short toShort(const char\* bytes) {

return (short)(((unsigned char)bytes[1] << 8) |

(unsigned char)bytes[0]);

}

//Reads the next four bytes as an integer, using little-endian form

int readInt(ifstream& input) {

char buffer[4];

input.read(buffer, 4);

return toInt(buffer);

}

//Reads the next two bytes as a short, using little-endian form

short readShort(ifstream& input) {

char buffer[2];

input.read(buffer, 2);

return toShort(buffer);

}

//Just like auto\_ptr, but for arrays

template<class T>

class auto\_array {

private:

T\* array;

mutable bool isReleased;

public:

explicit auto\_array(T\* array\_ = NULL) :

array(array\_), isReleased(false) {

}

auto\_array(const auto\_array<T>& aarray) {

array = aarray.array;

isReleased = aarray.isReleased;

aarray.isReleased = true;

}

~auto\_array() {

if (!isReleased && array != NULL) {

delete[] array;

}

}

T\* get() const {

return array;

}

T& operator\*() const {

return \*array;

}

void operator=(const auto\_array<T>& aarray) {

if (!isReleased && array != NULL) {

delete[] array;

}

array = aarray.array;

isReleased = aarray.isReleased;

aarray.isReleased = true;

}

T\* operator->() const {

return array;

}

T\* release() {

isReleased = true;

return array;

}

void reset(T\* array\_ = NULL) {

if (!isReleased && array != NULL) {

delete[] array;

}

array = array\_;

}

T\* operator+(int i) {

return array + i;

}

T& operator[](int i) {

return array[i];

}

};

}

Image\* loadBMP(const char\* filename) {

ifstream input;

input.open(filename, ifstream::binary);

assert(!input.fail() || !"Could not find file");

char buffer[2];

input.read(buffer, 2);

assert(buffer[0] == 'B' && buffer[1] == 'M' || !"Not a bitmap file");

input.ignore(8);

int dataOffset = readInt(input);

//Read the header

int headerSize = readInt(input);

int width;

int height;

switch (headerSize) {

case 40:

//V3

width = readInt(input);

height = readInt(input);

input.ignore(2);

assert(readShort(input) == 24 || !"Image is not 24 bits per pixel");

assert(readShort(input) == 0 || !"Image is compressed");

break;

case 12:

//OS/2 V1

width = readShort(input);

height = readShort(input);

input.ignore(2);

assert(readShort(input) == 24 || !"Image is not 24 bits per pixel");

break;

case 64:

//OS/2 V2

assert(!"Can't load OS/2 V2 bitmaps");

break;

case 108:

//Windows V4

assert(!"Can't load Windows V4 bitmaps");

break;

case 124:

//Windows V5

assert(!"Can't load Windows V5 bitmaps");

break;

default:

assert(!"Unknown bitmap format");

}

//Read the data

int bytesPerRow = ((width \* 3 + 3) / 4) \* 4 - (width \* 3 % 4);

int size = bytesPerRow \* height;

auto\_array<char> pixels(new char[size]);

input.seekg(dataOffset, ios\_base::beg);

input.read(pixels.get(), size);

//Get the data into the right format

auto\_array<char> pixels2(new char[width \* height \* 3]);

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

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

pixels2[3 \* (width \* y + x) + c] =

pixels[bytesPerRow \* y + 3 \* x + (2 - c)];

}

}

}

input.close();

return new Image(pixels2.release(), width, height);

}

void initRender() {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_BLEND); // turns on alpha blending

glEnable(GL\_NORMALIZE);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA); // to see the blending

Image\* image = loadBMP("assets/floor.bmp");

\_textureId1 = loadTexture(image);

image = loadBMP("assets/wall.bmp");

\_textureId2 = loadTexture(image);

image = loadBMP("assets/sky2.bmp");

\_textureId3 = loadTexture(image);

image = loadBMP("assets/glass.bmp");

\_textureId4 = loadTexture(image);

image = loadBMP("assets/wood.bmp");

\_textureId5 = loadTexture(image);

image = loadBMP("assets/lamp.bmp");

\_textureId6 = loadTexture(image);

delete image;

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // clears background colour and put alpha value as 1

}

Light.h

#pragma once

#include "glut.h"

static GLboolean lightSwitch = GL\_TRUE;

static int directionalLight = 0;

/\* Time varying or user-controled variables. \*/

static float lightAngle = 180.0;

static GLfloat lightPosition[4];

static GLfloat lightColor[] = { 1.0f, 1.0f, 0.8f }; // green-tinted

main

#include <cstring>

#include <Windows.h>

#include <time.h>

#include <iostream>

#include "glut.h"

#define STB\_IMAGE\_IMPLEMENTATION

#include "ImageLoader.h"

#include "Light.h"

int TIMER = 5;

bool cubeAutoRatote = false;

bool rotFlag = false;

unsigned int c[9] = { 0x00FF00, 0x0000FF, 0xFFFF00, 0xAAAAAA, 0xFF860D, 0xFF0000 };

// Cam config

int xRot = 25, yRot = 25, zRot = 0;

// Room config

float rs = CUBE\_SIZE \* 5; // size of room

float rw = CUBE\_SIZE \* 17; // Room width

float rh = CUBE\_SIZE \* 10; // Room height (to sky)

float wd = 0.01f; ; // element's padding

float wh = CUBE\_SIZE \* 15; // Wall's height

// Wall Lamp

float lampS = CUBE\_SIZE \* 4;// Lamp Size

float lampW = lampS \* 0.1; // Elements width

float lampPad = 1.0f; // Elements width

float bch = wh/2 - lampS/2; // Bottom corner height

float lampRX = rw - lampPad;// Coords on wall

// Floor config

static GLfloat floorVertices[4][3] = {

{ -rw, -rh, rw },

{ rw, -rh, rw },

{ rw, -rh, -rw },

{ -rw, -rh, -rw },

};

// Table config

float tw = rw \* 0.5; // width of table top from centre

float tl = rw \* 0.2; // length of table top from centre

float ttop = -rh \* 0.2;

float th = -rh \* 0.8; // height of table top from centre

float tt = CUBE\_SIZE \* 0.7; // thickness of table top

float ls = CUBE\_SIZE \* 0.4; // size of table leg

float tPadCenter = CUBE\_SIZE \* 2; // padding from center

float tlPadCenter = tPadCenter - ttop; // length padding from center

float min\_v = 0.0f;

float max\_v = 1.0f;

float wll = 0.01f;

void drawTableTop() {

glPushMatrix(); // 1 set where to start the current object transformation

glTranslatef(0.0f, -2.0f, 0.0f); // move downwards to lie on the floor

glBegin(GL\_QUADS);

// bottom: normal pointing outwards

glTexCoord2f(min\_v, min\_v);

glVertex3f(-tw, ttop - tPadCenter, tl);

glTexCoord2f(max\_v, min\_v);

glVertex3f(-tw, ttop - tPadCenter, -tl);

glTexCoord2f(max\_v, max\_v);

glVertex3f(tw, ttop - tPadCenter, -tl);

glTexCoord2f(min\_v, max\_v);

glVertex3f(tw, ttop - tPadCenter, tl);

// Top:normal pointing outwards

// front:normal pointing outwards

glTexCoord2f(min\_v, min\_v);

glVertex3f(-tw, ttop + tt - tPadCenter, tl);

glTexCoord2f(max\_v, min\_v);

glVertex3f(-tw, ttop - tPadCenter, tl);

glTexCoord2f(max\_v, max\_v);

glVertex3f(tw, ttop - tPadCenter, tl);

glTexCoord2f(min\_v, max\_v);

glVertex3f(tw, ttop + tt - tPadCenter, tl);

// back:normal pointing outwards

glTexCoord2f(min\_v, min\_v);

glVertex3f(tw, ttop + tt - tPadCenter, -tl);

glTexCoord2f(max\_v, min\_v);

glVertex3f(tw, ttop - tPadCenter, -tl);

glTexCoord2f(max\_v, max\_v);

glVertex3f(-tw, ttop - tPadCenter, -tl);

glTexCoord2f(min\_v, max\_v);

glVertex3f(-tw, ttop + tt - tPadCenter, -tl);

// right:normal pointing outwards

glTexCoord2f(min\_v, min\_v);

glVertex3f(tw, ttop + tt - tPadCenter, tl);

glTexCoord2f(max\_v, min\_v);

glVertex3f(tw, ttop - tPadCenter, tl);

glTexCoord2f(max\_v, max\_v);

glVertex3f(tw, ttop - tPadCenter, -tl);

glTexCoord2f(min\_v, max\_v);

glVertex3f(tw, ttop + tt - tPadCenter, -tl);

// left:normal pointing outwards

glTexCoord2f(max\_v, max\_v);

glVertex3f(-tw, ttop + tt - tPadCenter, tl);

glTexCoord2f(min\_v, max\_v);

glVertex3f(-tw, ttop + tt - tPadCenter, -tl);

glTexCoord2f(min\_v, min\_v);

glVertex3f(-tw, ttop - tPadCenter, -tl);

glTexCoord2f(max\_v, min\_v);

glVertex3f(-tw, ttop - tPadCenter, tl);

// top

glTexCoord2f(min\_v, min\_v);

glVertex3f(-tw, ttop + tt - tPadCenter, tl);

glTexCoord2f(min\_v, max\_v);

glVertex3f(tw, ttop + tt - tPadCenter, tl);

glTexCoord2f(max\_v, max\_v);

glVertex3f(tw, ttop + tt - tPadCenter, -tl);

glTexCoord2f(max\_v, min\_v);

glVertex3f(-tw, ttop + tt - tPadCenter, -tl);

glEnd();

glPopMatrix();

}

void firstLeg() {

glPushMatrix(); // 1 set where to start the current object transformation

glTranslatef(0.0f, -2.0f, 0.0f); // move downwards to lie on the floor

glBegin(GL\_QUADS);

// front:normal pointing outwards

glNormal3f(0, 0, 1);

glTexCoord2f(min\_v, min\_v);

glVertex3f(-tw + ls, th - tlPadCenter, tl - ls);

glTexCoord2f(max\_v, min\_v);

glVertex3f(-tw + ls, -tlPadCenter, tl - ls);

glTexCoord2f(max\_v, max\_v);

glVertex3f(-tw + ls + ls, -tlPadCenter, tl - ls);

glTexCoord2f(min\_v, max\_v);

glVertex3f(-tw + ls + ls, th - tlPadCenter, tl - ls);

// back:normal pointing outwards

glNormal3f(0, 0, -1);

glTexCoord2f(1, 1);

glVertex3f(-tw + ls, th - tlPadCenter, tl - ls - ls);

glTexCoord2f(0, 1);

glVertex3f(-tw + ls + ls, th - tlPadCenter, tl - ls - ls);

glTexCoord2f(0, 0);

glVertex3f(-tw + ls + ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 0);

glVertex3f(-tw + ls, -tlPadCenter, tl - ls - ls);

// right: normal pointing outwards

glNormal3f(1, 0, 0);

glTexCoord2f(0, 1);

glVertex3f(-tw + ls + ls, th - tlPadCenter, tl - ls);

glTexCoord2f(0, 0);

glVertex3f(-tw + ls + ls, -tlPadCenter, tl - ls);

glTexCoord2f(1, 0);

glVertex3f(-tw + ls + ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 1);

glVertex3f(-tw + ls + ls, th - tlPadCenter, tl - ls - ls);

// left:normal pointing outwards

glNormal3f(-1, 0, 0);

glTexCoord2f(1, 1);

glVertex3f(-tw + ls, th - tlPadCenter, tl - ls);

glTexCoord2f(0, 1);

glVertex3f(-tw + ls, th - tlPadCenter, tl - ls - ls);

glTexCoord2f(0, 0);

glVertex3f(-tw + ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 0);

glVertex3f(-tw + ls, -tlPadCenter, tl - ls);

glEnd();

glPopMatrix();

}

void secondLeg() {

glPushMatrix(); // 1 set where to start the current object transformation

glTranslatef(0.0f, -2.0f, 0.0f); // move downwards to lie on the floor

glBegin(GL\_QUADS);

// front:Normals pointing outwards

glNormal3f(0, 0, 1);

glTexCoord2f(1, 1);

glVertex3f(tw - ls, th - tlPadCenter, tl - ls);

glTexCoord2f(0, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, tl - ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, tl - ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls, -tlPadCenter, tl - ls);

// back:normals pointing outwards

glNormal3f(0, 0, -1);

glTexCoord2f(0, 1);

glVertex3f(tw - ls, th - tlPadCenter, tl - ls - ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, tl - ls - ls);

// left:normal pointing outwards

glNormal3f(-1, 0, 0);

glTexCoord2f(1, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, tl - ls);

glTexCoord2f(0, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, tl - ls - ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, tl - ls);

// right:normal pointing outwards

glNormal3f(1, 0, 0);

glTexCoord2f(0, 1);

glVertex3f(tw - ls, th - tlPadCenter, tl - ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls, -tlPadCenter, tl - ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls, -tlPadCenter, tl - ls - ls);

glTexCoord2f(1, 1);

glVertex3f(tw - ls, th - tlPadCenter, tl - ls - ls);

glEnd();

glPopMatrix();

}

void thirdLeg() {

glPushMatrix(); // 1 set where to start the current object transformation

glTranslatef(0.0f, -2.0f, 0.0f); // move downwards to lie on the floor

glBegin(GL\_QUADS);

// front:normal pointing outwards

glNormal3f(0, 0, 1);

glTexCoord2f(min\_v, min\_v);

glVertex3f(-tw + ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(max\_v, min\_v);

glVertex3f(-tw + ls, -tlPadCenter, -tl + ls);

glTexCoord2f(max\_v, max\_v);

glVertex3f(-tw + ls + ls, -tlPadCenter, -tl + ls);

glTexCoord2f(min\_v, max\_v);

glVertex3f(-tw + ls + ls, th - tlPadCenter, -tl + ls);

// back:normal pointing outwards

glNormal3f(0, 0, -1);

glTexCoord2f(1, 1);

glVertex3f(-tw + ls, th - tlPadCenter, -tl + ls + ls);

glTexCoord2f(0, 1);

glVertex3f(-tw + ls + ls, th - tlPadCenter, -tl + ls + ls);

glTexCoord2f(0, 0);

glVertex3f(-tw + ls + ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 0);

glVertex3f(-tw + ls, -tlPadCenter, -tl + ls + ls);

// left:normal pointing outwards

glNormal3f(-1, 0, 0);

glTexCoord2f(1, 1);

glVertex3f(-tw + ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(0, 1);

glVertex3f(-tw + ls, th - tlPadCenter, -tl + ls + ls);

glTexCoord2f(0, 0);

glVertex3f(-tw + ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 0);

glVertex3f(-tw + ls, -tlPadCenter, -tl + ls);

// right: normal pointing outwards

glNormal3f(1, 0, 0);

glTexCoord2f(0, 1);

glVertex3f(-tw + ls + ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(0, 0);

glVertex3f(-tw + ls + ls, -tlPadCenter, -tl + ls);

glTexCoord2f(1, 0);

glVertex3f(-tw + ls + ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 1);

glVertex3f(-tw + ls + ls, th - tlPadCenter, -tl + ls + ls);

glEnd();

glPopMatrix();

}

void fourthLeg() {

glPushMatrix(); // 1 set where to start the current object transformation

glTranslatef(0.0f, -2.0f, 0.0f); // move downwards to lie on the floor

glBegin(GL\_QUADS);

// front:Normals pointing outwards

glNormal3f(0, 0, 1);

glTexCoord2f(1, 1);

glVertex3f(tw - ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(0, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, -tl + ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls, -tlPadCenter, -tl + ls);

// back:normals pointing outwards

glNormal3f(0, 0, -1);

glTexCoord2f(0, 1);

glVertex3f(tw - ls, th - tlPadCenter, -tl + ls + ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, -tl + ls + ls);

// left:normal pointing outwards

glNormal3f(-1, 0, 0);

glTexCoord2f(1, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(0, 1);

glVertex3f(tw - ls - ls, th - tlPadCenter, -tl + ls + ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls - ls, -tlPadCenter, -tl + ls);

// right:normal pointing outwards

glNormal3f(1, 0, 0);

glTexCoord2f(0, 1);

glVertex3f(tw - ls, th - tlPadCenter, -tl + ls);

glTexCoord2f(0, 0);

glVertex3f(tw - ls, -tlPadCenter, -tl + ls);

glTexCoord2f(1, 0);

glVertex3f(tw - ls, -tlPadCenter, -tl + ls + ls);

glTexCoord2f(1, 1);

glVertex3f(tw - ls, th - tlPadCenter, -tl + ls + ls);

glEnd();

glPopMatrix();

}

void drawTable() {

glEnable(GL\_TEXTURE\_2D);//enabling texture

glBindTexture(GL\_TEXTURE\_2D, \_textureId5);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

drawTableTop();

firstLeg();

secondLeg();

thirdLeg();

fourthLeg();

glDisable(GL\_TEXTURE\_2D);

}

void drawSky(int isNotMirrow = 1) {

glEnable(GL\_TEXTURE\_2D);//enabling texture

glBindTexture(GL\_TEXTURE\_2D, \_textureId3);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glEnable(GL\_BLEND);//turns on alpha blending

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glColor4f(1.0f, 1.0f, 1.0f, 0.5f);

glBegin(GL\_QUADS);//normals pointing inwards

glNormal3f(0, 1 \* isNotMirrow, 0);

glTexCoord2f(0, 0);

glVertex3f(-rw \* 6, rh \* 3, -rw);

glTexCoord2f(1, 0);

glVertex3f(rw \* 6, rh \* 3, -rw);

glTexCoord2f(1, 1);

glVertex3f(rw \* 6, rh \* 3, rw);

glTexCoord2f(0, 1);

glVertex3f(-rw \* 6, rh \* 3, rw);

glEnd();

glDisable(GL\_BLEND);

glDisable(GL\_TEXTURE\_2D);

}

void drawGlass(int isNotMirrow = 1) {

float pad = 0.05f;

glEnable(GL\_TEXTURE\_2D);//enabling texture

glBindTexture(GL\_TEXTURE\_2D, \_textureId4);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glEnable(GL\_BLEND);//turns on alpha blending

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glColor4f(1.0f, 1.0f, 1.0f, 0.5f);

glBegin(GL\_QUADS); //normals pointing outwards

glNormal3f(0, 1, 0);

//front

glTexCoord2f(0, 0);

glVertex3f(-rw + pad , -rh + pad, -rw + pad);

glTexCoord2f(1, 0);

glVertex3f(rw - pad, -rh + pad, -rw + pad);

glTexCoord2f(1, 1);

glVertex3f(rw - pad, rh - pad, -rw + pad);

glTexCoord2f(0, 1);

glVertex3f(-rw + pad, rh - pad, -rw + pad);

////back

glTexCoord2f(0, 0);

glVertex3f(rw - pad, -rh + pad, rw - pad);

glTexCoord2f(1, 0);

glVertex3f(-rw + pad, -rh + pad, rw - pad);

glTexCoord2f(1, 1);

glVertex3f(-rw + pad, rh - pad, rw - pad);

glTexCoord2f(0, 1);

glVertex3f(rw - pad, rh - pad, rw - pad);

glEnd();

glDisable(GL\_BLEND);

glDisable(GL\_TEXTURE\_2D);

}

GLfloat lampLightPos[4] = { - lampRX + 0.3, 0.0, 0.0, .0 };

void drawWallLamp(int isNotMirrow = 1) {

glEnable(GL\_TEXTURE\_2D);//enabling texture

glBindTexture(GL\_TEXTURE\_2D, \_textureId6);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glBegin(GL\_QUADS); //normals pointing outwards

glNormal3f(0, -1 \* isNotMirrow, 0);

glTexCoord2f(0, 0);

glVertex3f(-lampRX, -lampS, -bch);

glTexCoord2f(1, 0);

glVertex3f(-lampRX, -lampS, bch);

glTexCoord2f(1, 1);

glVertex3f(-lampRX, lampS, bch);

glTexCoord2f(0, 1);

glVertex3f(-lampRX, lampS, -bch);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glPushMatrix();

glEnable(GL\_LIGHT1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lampLightPos);

GLfloat lampColor[] = { .0f, 1.0f, 0.8f };

glLightfv(GL\_LIGHT1, GL\_SHININESS, lampColor);

glLightf(GL\_LIGHT1, GL\_CONSTANT\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT1, GL\_LINEAR\_ATTENUATION, 0.2);

glLightf(GL\_LIGHT1, GL\_QUADRATIC\_ATTENUATION, 0.4);

glPopMatrix();

}

void drawWall(int isNotMirrow=1) { //enclosing the walls of the room{

glEnable(GL\_TEXTURE\_2D);//enabling texture

glBindTexture(GL\_TEXTURE\_2D, \_textureId2);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glBegin(GL\_QUADS); //normals pointing outwards

////left

glNormal3f(0, -1 \* isNotMirrow, 0);

glTexCoord2f(0, 0);

glVertex3f(-rw, -rh, -rw);

glTexCoord2f(5, 0);

glVertex3f(-rw, -rh, rw);

glTexCoord2f(5, 5);

glVertex3f(-rw, rh, rw);

glTexCoord2f(0, 5);

glVertex3f(-rw, rh, -rw);

////right

glTexCoord2f(0, 0);

glVertex3f(rw, -rh, rw);

glTexCoord2f(5, 0);

glVertex3f(rw, -rh, -rw);

glTexCoord2f(5, 5);

glVertex3f(rw, rh, -rw);

glTexCoord2f(0, 5);

glVertex3f(rw, rh, rw);

glEnd();

glDisable(GL\_TEXTURE\_2D);

}

static void drawFloor(int isNotMirrow = 1) {

glEnable(GL\_TEXTURE\_2D);//enabling texture

glBindTexture(GL\_TEXTURE\_2D, \_textureId1);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glBegin(GL\_QUADS);

glNormal3f(0, 1 \* isNotMirrow, 0);

glTexCoord2f(0, 0);

glVertex3fv(floorVertices[0]);

glTexCoord2f(15, 0);

glVertex3fv(floorVertices[1]);

glTexCoord2f(15, 15);

glVertex3fv(floorVertices[2]);

glTexCoord2f(0, 15);

glVertex3fv(floorVertices[3]);

glEnd();

glDisable(GL\_TEXTURE\_2D);

}

void drawInMirrow() {

drawSky(-1);

drawWall(-1);

drawWallLamp(-1);

drawFloor(-1);

drawTable();

cube.draw(-1);

}

void drawMirrors() {

GLint buffers = GL\_NONE; //get current color buffer from being drawn

glGetIntegerv(GL\_DRAW\_BUFFER, &buffers); // set the clear value

glClearStencil(0x00); // clear the stencil buffer

glEnable(GL\_STENCIL\_TEST);

glColorMask(0, 0, 0, 0); //Disable drawing colors to the screen

// always pass the stencil test

glStencilOp(GL\_REPLACE, GL\_REPLACE, GL\_REPLACE); // disable drawing to the color buffer

glStencilFunc(GL\_ALWAYS, 1, 0xffffffff); // set the operation to modify the stencil buffer

//this would be the stencil mask-->

glDisable(GL\_DEPTH\_TEST);

glBegin(GL\_QUADS);

//back wall

glNormal3f(0.0f, 0.0f, 1.0f);

glVertex3f(-rs, -wh, -rs);

glVertex3f(rs, -wh, -rs);

glVertex3f(rs, wh, -rs);

glVertex3f(-rs, wh, -rs);

glEnd();

glBegin(GL\_QUADS);

////Frontwall

glVertex3f(-rs, wh, rs);

glVertex3f(rs, wh, rs);

glVertex3f(rs, -wh, rs);

glVertex3f(-rs, -wh, rs);

glEnd();

////////////////////////////////////////////////

// reenable drawing to color buffer

glDrawBuffer((GLenum)buffers);

glColorMask(1, 1, 1, 1); //Enable drawing colors to the screen

glStencilFunc(GL\_EQUAL, 1, 0xffffffff); // draw only where the stencil buffer == 1

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_KEEP); // make stencil buffer read only

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // clear the color and depth buffers

glEnable(GL\_DEPTH\_TEST);

glPushMatrix(); // draw the mirror image

glScalef(1.0f, 1.0f, -1.0f);

for (int k = 1; k < 35; k += 2) {

glTranslatef(0.0f, 0.0f, k \* rw\*2);

drawInMirrow();

glTranslatef(0.0f, 0.0f, -(k+1) \* rw\*2);

drawInMirrow();

}

glPopMatrix();

glDisable(GL\_STENCIL\_TEST); // disable the stencil buffer

glDrawBuffer(GL\_NONE); // disable drawing to the color buffer

glDrawBuffer((GLenum)buffers);

}

void drawScene() {

drawMirrors();

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // clears background colour and put alpha value as 1

drawSky();

drawGlass();

drawWall();

drawWallLamp();

drawFloor();

drawTable();

}

void drawSunLight() {

lightAngle += 0.005f;

lightPosition[0] = 1.5 \* rw \* cos(lightAngle);

lightPosition[1] = 1.5 \* rw \* sin(lightAngle);

lightPosition[2] = -CUBE\_SIZE / 2;

lightPosition[3] = .0;

std::cout << lightPosition[0] << " " << lightPosition[1] << " " << lightPosition[2] << " " << std::endl;

glPushMatrix();

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor);

glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, 0.2);

glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, 0.4);

glPopMatrix();

}

void display()

{

glLoadIdentity(); // Reset the drawing perspective

glPushMatrix();

glEnable(GL\_LIGHTING);

glLightf(GL\_LIGHT0, GL\_SHININESS, 64);

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

glLightModelf(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, 1);

glEnable(GL\_NORMALIZE);

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

//glColor3f(1, 0, 0);

glTranslatef(0, 0, translateZ);

glRotatef(xRot, 1, 0, 0);

glRotatef(yRot, 0, 1, 0);

drawSunLight();

drawScene();

glTranslatef(CUBE\_SIZE / -2.0, CUBE\_SIZE / -2.0, CUBE\_SIZE / -2.0);

cube.draw();

glPushMatrix();

glDisable(GL\_LIGHTING);

glColor3f(1.0, 1.0, 0.0);

glTranslatef(lightPosition[0], lightPosition[1], lightPosition[2]);

glutSolidSphere(2\*CUBE\_SIZE, 75, 75);

glPopMatrix();

glPopMatrix();

glutSwapBuffers();

}

void init()

{

glEnable(GL\_NORMALIZE);

glClearColor(0.7, 0.7, 0.7, 0.7);

srand(time(0));

cube.clear(CUBE\_SIZE, c);

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(800, 800);

glutInitWindowPosition(1, 1);

glutCreateWindow("Rube Cube");

initRender(); //initialize rendering

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keys);

glutMouseFunc(mouse);

glutTimerFunc(TIMER, timer, 0);

glutSpecialFunc(specialKeys);

glutMainLoop();

return 0;

}

