МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Череповецкий государственный университет»

**Лабораторная работа № 4**

**«Источники света и материалы в библиотеке «OPENGL»»**

**Выполнил:**

студент гр. 1ИВТпб-01-31оп

Климов А.Г.

**Проверил:** преподаватель

Юдина О.В., доцент  
Отметка о зачете:

Череповец

2017 год

**Цель работы**

Изучить методы установки источников света и построения теней, приобрести практические навыки получения тени от любого объекта, а также управления общей освещенностью сцены.

**Задание**

Написать программу, демонстрирующую направленный источник света, управление общей освещенностью сцены, и построение тени от объекта на плоскость. Сцена может быть любой. Насколько это возможно, показать свойства материала объектов, влияние источника на формирование тени.

**Текст программы**

// Включаем необходимые библиотеки:

// OpelGL

#pragma comment(lib, "opengl32.lib")

#pragma comment(lib, "glu32.lib")

#pragma comment(lib, "glaux.lib")

float g\_LightPosition[4] = { 0, 1, 0, 1 }; // Позиция источника света

float g\_bLight = true; // Включен ли свет

float rotY = 0.0f; // Вращение по Y

float gg1 = 0;

float gg2 = 0;

// Наш главный хидер:

#include "main.h"

// В начале файла подключите хидер матричного класса:

#include "CMatrix.h"

// Переменные, хранящие угол вращения источника света:

float xRotation = 0.0f;

float yRotation = 0.0f;

// Эта матрица будет использоватся для хранения матрицы тени

CMatrix4x4 ShadowMatrix;

// Следующие переменные очень важны для функционирования теней. Первая - позиция

// источника света, вторая - нормаль поверхности, на которую отбрасывается тень.

CVector4 lightPos(1.5f, 2.3f, -2.0f, 1.0f);

CVector4 planeNormal(0.0f, 1.0f, 0.0f, 0.0f);

// Необходимые дескрипторы:

HWND g\_hWnd;

RECT g\_rRect;

HDC g\_hDC;

HGLRC g\_hRC;

HINSTANCE g\_hInstance;

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

//

// Функция, отрисовывающая куб

//

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

void DrawCube()

{

// Передний полигон

glBegin(GL\_QUADS);

glVertex3f(-0.2f, 0.0f, 0.0f);

glVertex3f(-0.2f, 0.5f, 0.0f);

glVertex3f(0.3f, 0.5f, 0.0f);

glVertex3f(0.3f, 0.0f, 0.0f);

glEnd();

// Задний полигон

glBegin(GL\_QUADS);

glVertex3f(-0.2f, 0.0f, -0.5f);

glVertex3f(-0.2f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.0f, -0.5f);

glEnd();

// Левый полигон

glBegin(GL\_QUADS);

glVertex3f(-0.2f, 0.0f, -0.5f);

glVertex3f(-0.2f, 0.5f, -0.5f);

glVertex3f(-0.2f, 0.5f, 0.0f);

glVertex3f(-0.2f, 0.0f, 0.0f);

glEnd();

// Правый полигон

glBegin(GL\_QUADS);

glVertex3f(0.3f, 0.0f, -0.5f);

glVertex3f(0.3f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, 0.0f);

glVertex3f(0.3f, 0.0f, 0.0f);

glEnd();

// Верхний полигон

glBegin(GL\_QUADS);

glVertex3f(-0.2f, 0.5f, 0.0f);

glVertex3f(-0.2f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, 0.0f);

glEnd();

// Нижний полигон

glBegin(GL\_QUADS);

glVertex3f(-0.2f, 0.0f, 0.0f);

glVertex3f(-0.2f, 0.0f, -0.5f);

glVertex3f(0.3f, 0.0f, -0.5f);

glVertex3f(0.3f, 0.0f, 0.0f);

glEnd();

}

void DrawCube1()

{

// Передний полигон

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.0f, 0.0f); //Красный

glVertex3f(-0.2f, 0.0f, 0.0f);

glVertex3f(-0.2f, 0.5f, 0.0f);

glVertex3f(0.3f, 0.5f, 0.0f);

glVertex3f(0.3f, 0.0f, 0.0f);

glEnd();

// Задний полигон

glBegin(GL\_QUADS);

glColor3f(0.0f, 0.0f, 1.0f); //Синий

glVertex3f(-0.2f, 0.0f, -0.5f);

glVertex3f(-0.2f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.0f, -0.5f);

glEnd();

// Левый полигон

glBegin(GL\_QUADS);

glColor3f(1.0f, 1.0f, 0.0f); //Жёлтый

glVertex3f(-0.2f, 0.0f, -0.5f);

glVertex3f(-0.2f, 0.5f, -0.5f);

glVertex3f(-0.2f, 0.5f, 0.0f);

glVertex3f(-0.2f, 0.0f, 0.0f);

glEnd();

// Правый полигон

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.5f, 0.0f); //Оранжевый

glVertex3f(0.3f, 0.0f, -0.5f);

glVertex3f(0.3f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, 0.0f);

glVertex3f(0.3f, 0.0f, 0.0f);

glEnd();

// Верхний полигон

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.0f, 1.0f); //Фиолетовый

glVertex3f(-0.2f, 0.5f, 0.0f);

glVertex3f(-0.2f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, -0.5f);

glVertex3f(0.3f, 0.5f, 0.0f);

glEnd();

// Нижний полигон

glBegin(GL\_QUADS);

glColor3f(0.0, 1.0, 0.0); //Зелёный

glVertex3f(-0.2f, 0.0f, 0.0f);

glVertex3f(-0.2f, 0.0f, -0.5f);

glVertex3f(0.3f, 0.0f, -0.5f);

glVertex3f(0.3f, 0.0f, 0.0f);

glEnd();

}

void CreatePyramid(float x, float y, float z, int width, int height)

{

glBegin(GL\_TRIANGLES);

// Ниже мы добавим что-то новое: НОРМАЛИ. Значение, насколько сильно нужно

// осветить конкретный полигон, OpenGL рассчитывает из его нормалей. Что такоей нормаль?

// Нормаль - это направление полигона. Вы заметите, что мы присваиваем заднему полигону

// нормаль (0,1,-1). Это значит, что полигон направлен в обратную сторону по оси Z (внутрь

// экрана). Запомните, нормали - это не координаты, а только направления.

// Функция glNormal3f() позволяют нам указать нормаль для вершин, переданных за ней.

// Сейчас мы напишем нормали вручную, но вы можете вернутся к урокам камеры, где есть

// ничто иное, как функция для рассчета нормалей.

// Задняя сторона

glNormal3f(0, 1, -1); // Полигон направлен назад и вверх

glColor3ub(255, 0, 0); glVertex3f(x, y + height, z);

glColor3ub(0, 255, 255); glVertex3f(x - width, y - height, z - width);

glColor3ub(255, 0, 255); glVertex3f(x + width, y - height, z - width);

// Передняя сторона

glNormal3f(0, 1, 1);

glColor3ub(255, 0, 0); glVertex3f(x, y + height, z);

glColor3ub(0, 255, 255); glVertex3f(x + width, y - height, z + width);

glColor3ub(255, 0, 255); glVertex3f(x - width, y - height, z + width);

// Левая сторона

glNormal3f(-1, 1, 0);

glColor3ub(255, 0, 0); glVertex3f(x, y + height, z);

glColor3ub(255, 0, 255); glVertex3f(x - width, y - height, z + width);

glColor3ub(0, 255, 255); glVertex3f(x - width, y - height, z - width);

// Передняя правая сторона

glNormal3f(1, 1, 0);

glColor3ub(255, 0, 0); glVertex3f(x, y + height, z);

glColor3ub(255, 0, 255); glVertex3f(x + width, y - height, z - width);

glColor3ub(0, 255, 255); glVertex3f(x + width, y - height, z + width);

glEnd();

// Теперь отрендерим дно пирамиды

glBegin(GL\_QUADS);

// Эти вершины образуют дно пирамиды

glNormal3f(0, -1, 0);

glColor3ub(0, 0, 255); glVertex3f(x - width, y - height, z + width);

glColor3ub(0, 0, 255); glVertex3f(x + width, y - height, z + width);

glColor3ub(0, 0, 255); glVertex3f(x + width, y - height, z - width);

glColor3ub(0, 0, 255); glVertex3f(x - width, y - height, z - width);

glEnd();

}

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

//

// Функция вызывается каждый кадр и рендерит сцену

//

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

void RenderScene()

{

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

glLoadIdentity();

// Обновим позицию света ДО вызова glLookAt(), чтобы свет обновился

// корректно. Опустите функцию вниз, чтобы увидеть, что иначе случится.

// Если источник не двигается, вызывать эту функцию заново не нужно.

glLightfv(GL\_LIGHT0, GL\_POSITION, g\_LightPosition);

//gluLookAt(0, 0, 6, 0, 0, 0, 0, 1, 0);

//CreatePyramid(0, 0, 0, 1, 1);

//SwapBuffers(g\_hDC);

glTranslatef(0.0f, -1.0f, -5.0f);

glRotatef(-yRotation, 1.0f, 0.0f, 0.0f);

glRotatef(-xRotation, 0.0f, 1.0f, 0.0f);

glColor3f(1.0f, 1.0f, 1.0f);

// Сначала мы отирсуем поверхность и забудем о ней. Затем я решил

// нарисовать тень. Чтобы сделать это, мы создадим матрицу тени, основываясь на

// позиции источника света и нормали поверхности. Создав эту матрицу, мы умножим её

// на текущую задействованную матрицу. Таким образом всё, что вы отрендерите далее,

// будет отрисовано плоским на поверхности, в точности как тень от обьекта. В принципе,

// ничего сложного, верно? Чтобы нарисовать что-то в качестве тени, вы должны закрасить

// обьект черным цветом и отключить текстуры. Я использовал glPush/PopMatrix, чтобы

// операции с тенью не затронули остальных обьектов сцены.

// Рисуем один большой обьект в качестве поверхности.

glColor3f(0.4f, 0.8f, 0.4f);

glBegin(GL\_QUADS);

glTexCoord2f(0.0, 0.0); glVertex3f(15.0f, -0.01f, 15.0f);

glTexCoord2f(1.0, 0.0); glVertex3f(-15.0f, -0.01f, 15.0f);

glTexCoord2f(1.0, 1.0); glVertex3f(-15.0f, -0.01f, -15.0f);

glTexCoord2f(0.0, 1.0); glVertex3f(15.0f, -0.01f, -15.0f);

glEnd();

// Создадим матрицу тени на основе нормали поверхности и позиции источника света

ShadowMatrix.CreateShadowMatrix(planeNormal, lightPos);

// Теперь рисуем тень...

glDisable(GL\_DEPTH\_TEST);

glPushMatrix();

// Одно то, что мы создали матрицу тени, ещё ничего не значит. Теперь нужно

// умножить её на текущую матрицу:

glMultMatrixf(ShadowMatrix.matrix);

glColor3f(0.0f, 0.0f, 0.0f);

// Нарисуем куб как обычно. Новая матрица сделает его плоским.

DrawCube();

glPopMatrix();

glEnable(GL\_DEPTH\_TEST);

// Теперь мы можем нарисовать обычный куб.

glColor3f(0.8f, 0.5f, 0.5f);

DrawCube1();

// Теперь нарисуем источник света

glColor3f(1.0f, 1.0f, 0.0f);

glTranslatef(lightPos.x, lightPos.y, lightPos.z);

GLUquadricObj \*pObj = gluNewQuadric();

gluSphere(pObj, 0.05f, 6, 6);

SwapBuffers(g\_hDC);

}

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

//

// Инициализирует игровое окно

//

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

void Init(HWND hWnd)

{

g\_hWnd = hWnd;

GetClientRect(g\_hWnd, &g\_rRect);

InitializeOpenGL(g\_rRect.right, g\_rRect.bottom);

}

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

//

// Главный цикл

//

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

WPARAM MainLoop()

{

MSG msg;

while (1)

{

if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE))

{

if (msg.message == WM\_QUIT)

break;

TranslateMessage(&msg);

DispatchMessage(&msg);

}

else

{

RenderScene();

}

}

DeInit();

return(msg.wParam);

}

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

//

// Обработка сообщений windows

//

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

LRESULT CALLBACK WinProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

LONG lRet = 0;

PAINTSTRUCT ps;

switch (uMsg)

{

case WM\_SIZE: // Если изменён размер окна

SizeOpenGLScreen(LOWORD(lParam), HIWORD(lParam));// LoWord=Width, HiWord=Height

GetClientRect(hWnd, &g\_rRect); // получаем window rectangle

break;

case WM\_PAINT: // Если нужно перерисовать сцену

BeginPaint(hWnd, &ps); // Иниц. paint struct

EndPaint(hWnd, &ps); // EndPaint, подчищаем

break;

case WM\_KEYDOWN: // Это сообщение означает, что нажата клавиша на клавиатуре.

// Сама клавиша передаётся в параметре wParam

switch (wParam)

{

case VK\_ESCAPE: // Если нажат ESCAPE

PostQuitMessage(0); // Выходим

break;

case VK\_UP:

lightPos.z -= 0.05f;

break;

case VK\_DOWN:

lightPos.z += 0.05f;

break;

case VK\_LEFT:

lightPos.x -= 0.05f;

break;

case VK\_RIGHT:

lightPos.x += 0.05f;

break;

case VK\_F1:

xRotation -= 5;

break;

case VK\_F2:

xRotation += 5;

break;

case VK\_F3:

yRotation -= 5;

break;

case VK\_F4:

yRotation += 5;

break;

case VK\_F5:

yRotation = 0; xRotation = 0;

break;

case VK\_ADD: // Если нажата ПЛЮС

g\_LightPosition[1] += 0.1f; // Увеличиваем значение Y

// Убедимся, что не превысили 5

if (g\_LightPosition[1] > 5) g\_LightPosition[1] = 5;

break;

case VK\_SUBTRACT: // Если МИНУС

g\_LightPosition[1] -= 0.1f;// Уменьшим значение Y

// Убедиммся, что оно не меньше -5

if (g\_LightPosition[1] < -5) g\_LightPosition[1] = -5;

break;

case 'L':

g\_bLight = !g\_bLight; // Выключим свет

if (g\_bLight) // Включим свет

glEnable(GL\_LIGHTING);

else

glDisable(GL\_LIGHTING); // Выключим свет

break;

}

break;

case WM\_CLOSE: // Если окно было закрыто

PostQuitMessage(0); // Выходим

break;

default: // Return по умолчанию

lRet = DefWindowProc(hWnd, uMsg, wParam, lParam);

break;

}

return lRet;

}

**Контрольные вопросы**

1. *От чего зависит расчет общей освещенностью объекта?*

Расчет освещенности любого объекта зависит от двух основных составляющих: материала, из которого состоит объект и света, падающего на объект. Материал описывает характеристики поверхности, на которую падает свет. Свет, испускаемый источниками, подразделяется на три составляющих: фоновый, диффузный и отражаемый.

1. *Почему источник света можно рассматривать как три источника, расположенных в одной точке?*

Свет, испускаемый источниками, подразделяется на три составляющих: фоновый, диффузный и отражаемый. Фоновая составляющая определяет общую яркость источника света. Диффузная составляющая задает характеристики рассеиваемости света во всех направлениях. Отражающая составляющая определяет интенсивность отражения данного света от поверхности предметов. Поэтому источник света можно рассматривать как три разных источника расположенных в одной точке.

1. *Назовите характеристики материала.*

Для каждого объекта задается материал, из которого он состоит. Материал имеет аналогичные характеристики, как и источники, но только в смысле отражения, а также дополнительный параметр - собственное излучение.

1. *Сколько источников света поддерживает OpenGL?*

Источники света пронумерованы и для каждого сопоставлена константа (GL\_LIGHT0, GL\_LIGHT1, GL\_LIGHT2…). Чаще всего поддерживается до восьми источников света.

1. *Какой командой задаются параметры источника света?*

glLightfv

Источник света GL\_LIGHT0 имеет особенные настройки по умолчанию. По умолчанию учет освещения отключен. Поэтому требуется включить его командой glEnable(GL\_LIGHTING). Для установки общей освещенности используется функция glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient); где ambient - это массив, содержащий составляющие цвета в общей освещенности. Для правильного расчета освещенности необходимо задавать вектора нормалей для каждой грани командой glNormal.

Алгоритм работы с источниками света;

1. Командой (glEnable(GL\_LIGHTING)) разрешается расчет освещения.
2. Команда glEnable(GL\_COLOR\_MATERIAL) разрешает применение материалов.
3. Задается материалы поверхностей (glMaterialfv) для фонового, диффузного и отражаемого цвета.
4. Разрешается первый источник света (glEnable(GL\_LIGHT1)).
5. Задаются параметры источника света (glLightfv) для фоного, диффузного и отражаемого цвета, а также положение источника.
6. *Как построить тень с помощью OpenGL?*

Для реализации тени требуется построить проекцию объекта на плоскость, куда падает тень. Для этого необходимо найти точки пересечения лучей исходящих из источника света и проходящих через вершины граней. Пусть имеем О - центр источника света, А - точку объекта и плоскость заданную уравнением N\*P+d=0, на которой строится тень. Тогда точка пересечения P находится следующим образом:

t=-(N\*O+d) / (N\*(A-O))

P=O+(A-O)\*t

При построении теней с помощью OpenGL необходимо построить плоскость, на которую падает тень с включенными источниками света, затем отключить освещенность, построить тень, а затем включить освещенность и построить объект.