Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: «Компьютерная графика»

Студент	Шлюков Алексей Павлович
Группа	РК6-74Б
Тип задания	Лабораторная работа №1-2
Название	«Знакомство с OpenGL»
Вариант лабораторной работы	1

Студент _____ **Шлюков А.П.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Витюков Ф.А.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2025 г.

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Цель выполнения лабораторной работы - ознакомление с синтаксисом базовых функций OpenGL

ЗАДАНИЕ

Разработать программу на языке программирования C++ результатом выполнения которой является вывод на экран геометрического объекта заданной формы (рис. 1), грани которого покрыты предоставленной текстурой (рис. 2). Для использования трёхмерной графики необходимо воспользоваться спецификацией OpenGL.

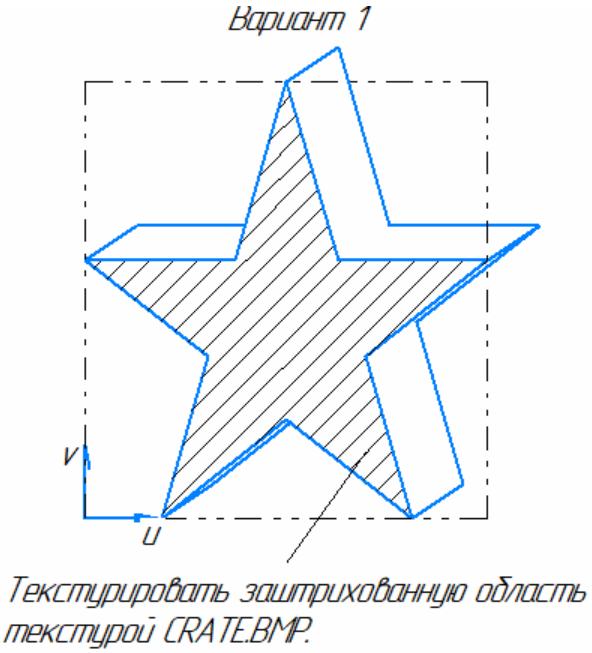

Рисунок 1 - форма геометрического объекта

Рисунок 2. Текстура для поверхностей объекта

ВВОДНАЯ ЧАСТЬ

Спецификация OpenGL предполагает создание трёхмерных геометрических моделей на основе геометрических примитивов: треугольников, четырёхугольников и многоугольников. Наиболее простым для отрисовки с использованием ресурсов встроенного GPU примитивом является полигон. Для построения геометрии, представленной на рисунке 1, необходимо представить её в виде набора полигонов. При этом следует использовать минимальное число примитивов.

Разбиение граней “звезды” на примитивы, представлено на рисунке 3.

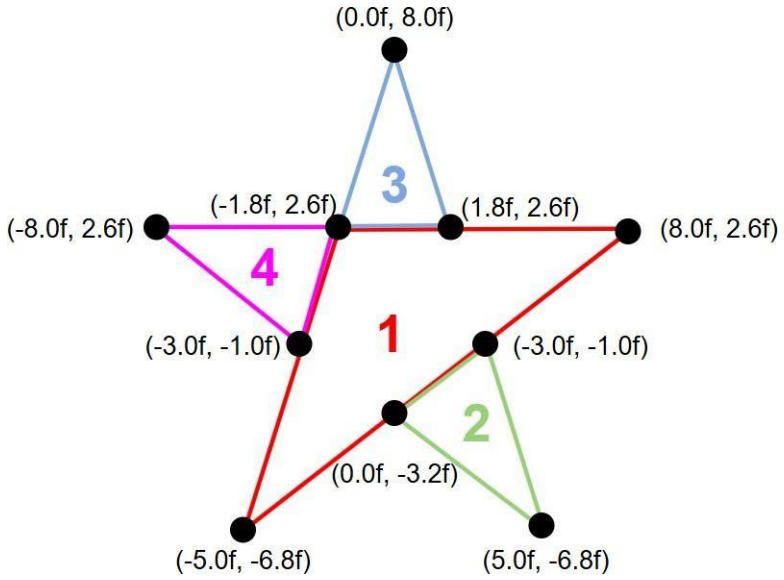

Рисунок 3. Разбиение передней грани

Для наложения текстуры (рис. 2) необходимо указать соответствующие (u, v) - координаты в каждой вершине геометрических примитивов, использованных при построении граней (рис. 3).

Разбиение боковой грани “звезды” на примитивы производится аналогичным образом и представлено на рисунке 4.

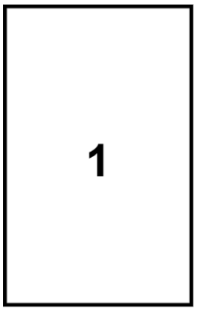

Рисунок 4. Разбиение боковой грани

РАЗБОР КОДА

Основная часть программы была предоставлена заранее. Основные изменения, необходимые для выполнения поставленной задачи, были внесены в работу функций DrawGLScene и LoadGLTextures.

Функция `GLvoid LoadGLTextures()` отвечает за загрузку картинки и её конвертирование в текстуру:

```
GLvoid LoadGLTextures()
{
        // Загрузка картинки
        AUX_RGBImageRec *texture1;
        texture1 = auxDIBImageLoad("Data/CRATE.bmp");
        // Создание текстуры
        glGenTextures(1, &texture[0]);
        glBindTexture(GL_TEXTURE_2D, texture[0]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, 3, texture1->sizeX, texture1-
>sizeY, 0, GL RGB, GL UNSIGNED BYTE, texture1->data);
}
```

Основная задача `glGenTexture` указать `OpenGL` на доступную память. В этом случае мы говорим `OpenGL`, что память доступна в `&texture[0]`. Затем мы создаем текстуру, и она будет сохранена в этой памяти. Далее, если мы привязываемся к памяти, в которой уже находиться текстура, мы говорим `OpenGL` захватить данные текстуры из этой области памяти.

За отрисовку объектов отвечает функция:

`GLvoid DrawGLScene(GLvoid);`

В этой функции для выполнения построения используются следующие функции:

`glLoadIdentity();` - сброс текущего положения и настроек матрицы к default

`glTranslatef (GLfloat x, GLfloat y, GLfloat z);` - перемещение точки рисования в новое положение с изменениями по x, y, z соответственно

`glBegin (GLenum mode)` - рисование объектов по точкам с диапазоном значений от 0 до 1

`glVertex3f (GLfloat x, GLfloat y, GLfloat z)` - установка точки с координатами (x, y, z) относительно текущей системы координат

Функция `InitGL(GLsizei Width, GLsizei Height)` - инициализация состояния OpenGL после создания окна

```
GLvoid InitGL(GLsizei Width, GLsizei Height)  // Вызвать после создания
окна GL
{
        LoadGLTextures(); // Загрузка текстур
        glEnable(GL_TEXTURE_2D); // Разрешение наложение текстуры
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        // Очистка экрана в черный цвет
        glClearDepth(1.0);              // Разрешить очистку буфера глубины
        glDepthFunc(GL_LESS);  // Тип теста глубины
        glEnable(GL_DEPTH_TEST);// разрешить тест глубины
        glShadeModel(GL_SMOOTH);// разрешить плавное цветовое сглаживание
        glMatrixMode(GL_PROJECTION);// Выбор матрицы проекции
        glLoadIdentity();               // Сброс матрицы проекции
        gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);
        // Вычислить соотношение геометрических размеров для окна
        glMatrixMode(GL_MODELVIEW);// Выбор матрицы просмотра модели
}
```

Функция устанавливает начальные параметры рендеринга. Включение 2D-текстур (`glEnable(GL_TEXTURE_2D)`) позволяет использовать координаты текстур при отрисовке. `glClearColor` задаёт цвет, в который будет заполняться цветовой буфер при вызове `glClear`. Настройки глубины (`glClearDepth, glDepthFunc, glEnable(GL_DEPTH_TEST)`) конфигурируют Z-буфер и алгоритм отбора видимых фрагментов (меньший по глубине фрагмент скрывает больший). `glShadeModel(GL_SMOOTH)` задаёт интерполяцию цветов между вершинами (`Gouraud shading`). Блок работы с матрицами отвечает за настройку проекционной матрицы: сначала переключаются на матрицу проекции, сбрасывают её, затем оказывается перспективная проекция через `gluPerspective` с указанным полем зрения, соотношением сторон и ближней/дальней плоскостями обрезания. В завершение переключаются на матрицу моделирования/вида (`GL_MODELVIEW`) для последующих трансформаций сцены.

Функция `DrawGLScene()` - формирование и рендеринг геометрии сцены, управление анимацией

```
1  GLvoid DrawGLScene(GLvoid)
 2{
 3        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 4        glLoadIdentity();
 5        glTranslatef(0.0f, 0.0f, -70.0f);
 6        glRotatef(xrot+30, 1.0f, 0.0f, 0.0f);  // Вращение по оси X
 7        glRotatef(yrot-30, 0.0f, 1.0f, 0.0f); // Вращение по оси Y
 8        glRotatef(zrot, 0.0f, 0.0f, 1.0f);   // Вращение по оси Z
 9        glBindTexture(GL_TEXTURE_2D, texture[0]);
10
11        // передняя грань
12        glBegin(GL_TRIANGLES);
13
14        // треугольник 1
15        glTexCoord2f(0.1875f, 0.0f); glVertex3f(-5.0f, -6.8f, 0.0f);
16        ...
17        glEnd();
18
19        // задняя грань
20        glBegin(GL_TRIANGLES);
21        // треугольник 1
22        glVertex3f(-5.0f, -6.8f, -6.0f);
23        ...
24        glEnd();
25
26        // боковые грани
27        glBegin(GL_QUADS);
28        ...
29        glEnd();
30
31        xrot+=1.3f;
32        yrot += 1.2f;
33        zrot += 1.4f;
34        Sleep(16);
35}
```

Сброс буферов цвета и глубины (`glClear`) обеспечивает готовность к отрисовке нового кадра. `glLoadIdentity` возвращает текущую матрицу моделирования/вида к единичной, после чего применяются трансформации: перенос сцены вдоль оси Z для установки камеры (`glTranslatef`) и последовательные вращения вокруг осей X, Y и Z (`glRotatef`). Такие преобразования формируют модельно-видовую матрицу, влияющую на все последующие вершины. `glBindTexture` делает ранее созданную текстуру активной; далее последующие вершины и фрагменты будут получать значения текстурных координат. Геометрия задаётся сериями примитивов: передняя и

задняя грани собираются из треугольников (GL_TRIANGLES), боковые из четырёхугольников (GL_QUADS). Для передней грани явно указываются и координаты вершин (glVertex3f), и соответствующие текстурные координаты (glTexCoord2f), что обеспечивает корректное отображение выбранной части текстуры на соответствующих полигонах. После определения всех примитивов инкрементируются углы вращения (xrot, yrot, zrot) для анимации; вызов Sleep(16) даёт простой способ ограничить частоту обновления.

Функция WndProc(...) - обработчик оконных сообщений и инициализация контекста рендеринга.

```
LRESULT CALLBACK WndProc(
                                HWND    hWnd,
                                UINT    message,
                                WPARAM wParam,
                                LPARAM lParam)
{
        RECT    Screen;         // используется позднее для размеров окна
        GLuint  PixelFormat;
        static  PIXELFORMATDESCRIPTOR pfd=
        {
                sizeof(PIXELFORMATDESCRIPTOR), // Размер этой структуры
                1,                              // Номер версии (?)
                PFD_DRAW_TO_WINDOW |// Формат для Окна
                PFD_SUPPORT_OPENGL |// Формат для OpenGL
                PFD_DOUBLEBUFFER,// Формат для двойного буфера
                PFD_TYPE_RGBA, // Требуется RGBA формат
                16,                             // Выбор 16 бит глубины
цвета
                0, 0, 0, 0, 0, 0,// Игнорирование цветовых битов (?)
                0,                              // нет буфера прозрачности
                0,                              // Сдвиговый бит
игнорируется (?)
                0,                              // Нет буфера аккумуляции
                0, 0, 0, 0,             // Биты аккумуляции игнорируются (?)
                16,                             // 16 битный Z-буфер (буфер
глубины)
                0,                              // Нет буфера траффарета
                0,                              // Нет вспомогательных
буферов (?)
                PFD_MAIN_PLANE,// Главный слой рисования
                0,                              // Резерв (?)
                0, 0, 0                 // Маски слоя игнорируются (?)
        };
        switch (message)        // Тип сообщения
        {
                case WM_CREATE:
```

```
                hDC = GetDC(hWnd);      // Получить контекст устройства для
окна
                PixelFormat = ChoosePixelFormat(hDC, &pfd);
                        // Найти ближайшее совпадение для нашего формата
пикселов
                if (!PixelFormat)
                {
                        MessageBox(0,"Can't Find A Suitable
PixelFormat.","Error",MB_OK|MB_ICONERROR);
                        PostQuitMessage(0);
                        // Это сообщение говорит, что программа должна
завершится
                        break;  // Предтовращение повтора кода
                }

                if(!SetPixelFormat(hDC,PixelFormat,&pfd))
                {
                        MessageBox(0,"Can't Set The
PixelFormat.","Error",MB_OK|MB_ICONERROR);
                        PostQuitMessage(0);
                        break;
                }
                hRC = wglCreateContext(hDC);
                if(!hRC)
                {
                        MessageBox(0,"Can't Create A GL Rendering
Context.","Error",MB_OK|MB_ICONERROR);
                        PostQuitMessage(0);
                        break;
                }
                if(!wglMakeCurrent(hDC, hRC))
                {
                        MessageBox(0,"Can't activate
GLRC.","Error",MB_OK|MB_ICONERROR);
                        PostQuitMessage(0);
                        break;
                }
                GetClientRect(hWnd, &Screen);
                InitGL(Screen.right, Screen.bottom);
                break;

                case WM_DESTROY:
                case WM_CLOSE:
                ChangeDisplaySettings(NULL, 0);

                wglMakeCurrent(hDC,NULL);
                wglDeleteContext(hRC);
                ReleaseDC(hWnd,hDC);

                PostQuitMessage(0);
                break;

                case WM_KEYDOWN:
                keys[wParam] = TRUE;
                break;

                case WM_KEYUP:
                keys[wParam] = FALSE;
                break;
```

```
                case WM_SIZE:
                ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));
                break;

                default:
        return (DefWindowProc(hWnd, message, wParam, lParam));
        }
        return (0);
}
```

WndProc реализует стандартную петлю обработки оконных сообщений Windows и отвечает за создание и уничтожение контекста визуализации OpenGL. При сообщении WM_CREATE происходит: получение контекста устройства (GetDC), выбор подходящего формата пикселя (ChoosePixelFormat) на основе PIXELFORMATDESCRIPTOR, установка формата (SetPixelFormat), создание рендерингового контекста OpenGL (wglCreateContext) и привязка его к поточному контексту устройства (wglMakeCurrent). После успешной настройки вызывается InitGL с размерами клиентской области. При закрытии окна реализована корректная очистка ресурсов: снятие контекста (wglMakeCurrent(..., NULL)), удаление контекста (wglDeleteContext) и освобождение DC. Обработчики WM_KEYDOWN/WM_KEYUP обновляют булев массив keys[] для обработки ввода; WM_SIZE вызывает функцию, пересчитывающую проекцию.

Функция WinMain(...) - регистрация окна, переключение в полноэкранный режим и основной цикл приложения.

```
int WINAPI WinMain(
                HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR lpCmdLine,
                int nCmdShow)
{
        MSG             msg;    // Структура сообщения Windows
        WNDCLASS        wc; // Структура класса Windows для установки типа
окна
        HWND            hWnd;// Сохранение дискриптора окна

        wc.style                        = CS HREDRAW | CS VREDRAW |
CS_OWNDC;
```

```
        wc.lpfnWndProc          = (WNDPROC) WndProc;
        wc.cbClsExtra           = 0;
        wc.cbWndExtra           = 0;
        wc.hInstance            = hInstance;
        wc.hIcon                        = NULL;
        wc.hCursor                      = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground        = NULL;
        wc.lpszMenuName         = NULL;
        wc.lpszClassName        = "OpenGL WinClass";

        if(!RegisterClass(&wc))
        {
        MessageBox(0,"Failed To Register The Window
Class.","Error",MB_OK|MB_ICONERROR);
        return FALSE;
        }

        hWnd = CreateWindow(
        "OpenGL WinClass",
        "Jeff Molofee's GL Code Tutorial ... NeHe '99",       // Заголовок
вверху окна

        WS_POPUP |
        WS_CLIPCHILDREN |
        WS_CLIPSIBLINGS,

        0, 0,                   // Позиция окна на экране
        640, 480,               // Ширина и высота окна

        NULL,
        NULL,
        hInstance,
        NULL);

        if(!hWnd)
        {
        MessageBox(0,"Window Creation Error.","Error",MB_OK|MB_ICONERROR);
                return FALSE;
        }

        DEVMODE dmScreenSettings;                       // Режим работы

        memset(&dmScreenSettings, 0, sizeof(DEVMODE));        // Очистка
для хранения установок
        dmScreenSettings.dmSize         = sizeof(DEVMODE);              //
Размер структуры Devmode
        dmScreenSettings.dmPelsWidth    = 640;                  // Ширина
экрана
        dmScreenSettings.dmPelsHeight = 480;                    // Высота
экрана
        dmScreenSettings.dmFields       = DM_PELSWIDTH | DM_PELSHEIGHT;
        // Режим Пиксела
        ChangeDisplaySettings(&dmScreenSettings, CDS_FULLSCREEN);
        // Переключение в полный экран

        ShowWindow(hWnd, SW_SHOW);
        UpdateWindow(hWnd);
        SetFocus(hWnd);
```

```
        while (1)
        {
                // Обработка всех сообщений
                while (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
                {
                        if (GetMessage(&msg, NULL, 0, 0))
                        {
                                TranslateMessage(&msg);
                                DispatchMessage(&msg);
                        }
                        else
                        {
                                return TRUE;
                        }
                }

                DrawGLScene();                          // Нарисовать сцену
                SwapBuffers(hDC);                       // Переключить буфер
экрана
                if (keys[VK_ESCAPE]) SendMessage(hWnd,WM_CLOSE,0,0); //
Если ESC - выйти
        }
}
```

Функция WinMain является точкой входа Windows-приложения и выполняет регистрацию класса окна (WNDCLASS), создание окна (CreateWindow) и инициализацию полноэкранного режима через ChangeDisplaySettings. В процессе инициализации задаются параметры класса окна (стиль CS_HREDRAW | CS_VREDRAW | CS_OWNDC, указатель на WndProc, курсор и имя класса); при ошибке регистрации или создания окна отображается диалог об ошибке и выполнение завершается. После показа окна (ShowWindow, UpdateWindow, SetFocus) запускается главный цикл приложения, реализующий модель «обработка сообщений + рендер»: при наличии сообщений выполняется их извлечение и обработка (PeekMessage/GetMessage, TranslateMessage, DispatchMessage), а в свободное от сообщений время вызываются DrawGLScene() и SwapBuffers(hDC) для отрисовки кадра и демонстрации буфера. Выход из программы инициируется получением WM_QUIT или нажатием клавиши ESC, которое проверяется через массив keys[] и приводит к отправке WM_CLOSE.

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

В результате работы программы на экране отображается геометрическая фигура заданной формы (рис. 5).

Рисунок 5. Результат работы программы

ВЫВОДЫ

В ходе выполнения лабораторной работы было проведено знакомство со спецификацией OpenGL, которая представляет собой универсальный программный интерфейс для создания приложений, работающих с двухмерной и трехмерной компьютерной графикой. Была также разработана программа на языке C++, которая в результате своей работы выводит на экран геометрический объект определенной формы (см. рис. 1), грани которого покрыты предоставленной текстурой (см. рис. 2).