МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

Кафедра графики, конструирования и информационных технологий в
промышленном дизайне

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «КОМПЬЮТЕРНАЯ ГРАФИКА»

для обучающихся по направлению 54.03.01 «Дизайн»,
профиль «Промышленный дизайн» всех форм обучения

Воронеж 2021

УДК 658.512:621(07)
ББК 30.18:85.1:34.5я7
Составители: А.В. Кузовкин, А.П. Суворов, Ю.С. Золототрубова

Методические рекомендации по выполнению лабораторных работ по дисциплине «Компьютерная графика» для обучающихся по направлению 54.03.01 «Дизайн», профиль «Промышленный дизайн» всех форм обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост.: А.В. Кузовкин, А.П. Суворов, Ю.С. Золототрубова. – Воронеж: Изд-во ВГТУ, 2021. – 59 с.

Приводится описание выполнения лабораторных работ по курсу «Компьютерная графика» для студентов обучающихся по направлению 54.03.01 «Дизайн», профиль «Промышленный дизайн» всех форм обучения

УДК 658.512:621(07)
ББК 30.18:85.1:34.5я7

Рецензент - д.т.н., профессор Болдырев А.И.

Рекомендовано методическим семинаром кафедры ГКПД и методической комиссией ФИТКБ Воронежского государственного технического университета в качестве методических материалов

Лабораторная работа № 1. Подключение библиотек; контекст устройства, контекст воспроизведения; общий вид программы.

Цель работы.

Создание программы-заготовки для работы с библиотекой openGL.

Необходимые теоретические сведения.

Использование библиотеки OpenGL требует навыков программирования на C++, а также знания основ событийного программирования на уровне операционной системы Windows[1].

Для выполнения работы необходимо иметь представление об основных принципах формирования изображения на экране, о современных стандартах и интерфейсах программирования компьютерной графики ([7] – лекция 17), знать особенности представления цвета в цветовой модели RGB ([7] – лекция 4).

Основные понятия, используемые в данной лабораторной работе: контекст устройства, контекст воспроизведения, формат пиксела.

Контекст графического устройства (Device Context) указывает плоскость отображения, на которую осуществляется графический вывод: окно программы на экране дисплея, страница принтера или другое место, куда может быть направлен графический вывод. Если программа вызывает различные графические функции, такие как рисование точек, линий, фигур и др., необходимо указывать идентификатор контекста (hdc – handle of device context) и координаты. Смысл использования контекста устройства заключается в том, что вывод на различные устройства осуществляется одними и теми же функциями, изменяется лишь значение ***hDC***. «Контекст устройства является структурой, которая определяет комплект графических объектов и связанных с ними атрибутов и графические режимы, влияющие на вывод». При составлении программы необходимо получить это числовое значение перед рисованием, а после рисования – освободить контекст.

В OpenGL существует понятие *контекст воспроизведения* (контекст рендеринга), аналогичное понятию контекст устройства. Графическая система OpenGL также нуждается в ссылке на устройство, на которое будет осуществляться вывод. Это специальная ссылка на контекст воспроизведения – величина типа HGLRC (handle openGL rendering context, ссылка на контекст воспроизведения OpenGL) - ***hRC***.

Прежде чем получить контекст воспроизведения, сервер OpenGL должен получить детальные характеристики используемого оборудования. Эти характеристики хранятся в специальной структуре – описание формата пиксела. Формат пиксела определяет конфигурацию буфера цвета и вспомогательных буферов.

Порядок выполнения работы.

Программа, которая будет создана в результате выполнения данной лабораторной работы, должна осуществлять следующие действия: создавать пустое окно для графического вывода средствами openGL; устанавливать все необходимые параметры графического вывода; реагировать на нажатие клавиши <ESC> для закрытия окна и завершения работы. Для демонстрации графического вывода в конце лабораторной работы в окне будет построено некоторое изображение. Созданная в ходе выполнения лабораторной работы программа будет являться основой для выполнения всех последующих работ.

1. Создайте новое приложение в Visual C++.
2. Добавьте для сборки проекта библиотеки OpenGL. Для этого:
    - в меню Project/setting, выберите закладку LINK;

---

[1] Лабораторные работы практикума могут выполняться в среде визуального программирования Delphi. В Приложении 2 содержатся варианты минимальной программы OpenGL на Delphi.

- в строке "Object/Library Modules" добавьте строку "OpenGL32.lib GLu32.lib GLaux.lib"
- для завершения щелкните левой клавишей мыши по кнопке OK.

3. Введите в текст кода следующие строки (они сообщают компилятору какие библиотечные файлы следует использовать):

```
#include <windows.h>          // Заголовочный файл для Windows
#include <gl\gl.h>            // Заголовочный файл для библиотеки OpenGL32
#include <gl\glu.h>           // Заголовочный файл для библиотеки GLu32
#include <gl\glaux.h>         // Заголовочный файл для библиотеки GLaux
```

4. Инициализируйте все переменные, которые будут использованы в программе. Первые две строки устанавливают контекст воспроизведения (рендеринга) и контекст устройства. Контекст рендеринга OpenGL определен как ***hRC*** и связывает вызовы OpenGL с окном Windows. Для того чтобы рисовать в окне, необходимо создать контекст устройства Windows, который определен как ***hDC***. Контекст устройства связывает окно с GDI. Контекст воспроизведения связывает OpenGL с контекстом устройства.

```
static HGLRC hRC;             // Постоянный контекст рендеринга
static HDC hDC;               // Приватный контекст устройства GDI
```

5. Объявите массив для отслеживания нажатия клавиш на клавиатуре (указанный ниже способ позволяет отслеживать нажатие нескольких клавиш одновременно).

```
 BOOL   keys[256];            // Массив для процедуры обработки клавиатуры
```

6. В следующей секции кода будут произведены все настройки для OpenGL Эта процедура может быть вызвана только после того как будет создано окно OpenGL. Установим цвет, которым будет очищен экран. Все значения могут быть в диапазоне от 0.0f до 1.0f, при этом 0.0 самый темный, а 1.0 самый светлый. Первые три параметра определяют цвет в модели RGB: первый - интенсивность красного, второй – зеленого, третий – синего. Наибольшее значение – 1.0f, является самым ярким значением данного цвета. Последний параметр – альфа-значение (прозрачность) – пока будет равен 0.0f.

```
GLvoid InitGL(GLsizei Width, GLsizei Height)  //Вызвать после создания окна GL
{
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);          // Очистка экрана в черный цвет
}
```

7. Следующая секция кода – функция масштабирования сцены, вызываемая OpenGL всякий раз, когда изменяется размер окна. Даже если размеры окна не изменяются (например, в полноэкранном режиме), эта процедура все равно должна быть вызвана хотя бы один раз (обычно во время запуска программы), т.к. сцена масштабируется, основываясь на ширине и высоте отображаемого окна.

```
   GLvoid ReSizeGLScene(GLsizei Width, GLsizei Height)
{
        if (Height==0)         // Предотвращение деления на ноль,
                               //если окно слишком мало
                Height=1;
        glViewport(0, 0, Width, Height);
                // Сброс текущей области вывода и перспективных преобразований
}
```

8. Следующая секция предназначена для рисования сцены. Впоследствии код будет добавляться именно в эту секцию программы. Пока запишем в этой секции только команду для очистки экрана цветом, который мы определили выше. Команды рисования будут следовать за ней.

```
GLvoid DrawGLScene(GLvoid)
{
        glClear(GL_COLOR_BUFFER_BIT);  // очистка экрана
}
```

9. Одна из важнейших секций кода устанавливает параметры окна Windows, формат пикселя, обрабатывает сообщения при изменении размеров окна, при нажатии на клавиши, и закрытии программы.

Первые четыре строки делают следующее: переменная ***hWnd*** – является указателем на окно. Переменная ***message*** – сообщения, передаваемые программе системой. Переменные ***wParam*** и ***lParam*** содержат информацию, которая посылается вместе с сообщением, например такую как ширина и высота окна.

```
LRESULT CALLBACK WndProc(      HWND    hWnd,
                               UINT    message,
                               WPARAM  wParam,
                               LPARAM  lParam)
```

Код между скобками устанавливает *формат пикселей*. Формат пикселя определяет то, как OpenGL будет выводить в окно. Большая часть кода игнорируется, но тем не менее необходима.

```
{
        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 трафарета
        0,                              // Нет вспомогательных буферов
        PFD_MAIN_PLANE,                 // Главный слой рисования
        0,                              // Резерв
        0, 0, 0                         // Маски слоя игнорируются
        };
```

Следующая секция предназначена для обработки системных сообщений: выход из программы, нажатие клавиш, перемещение окна и т.д., каждая секция "***case***" обрабатывает свой тип сообщения.

```
        switch (message)        // Тип сообщения
        {
```

WM_CREATE указывает программе, что сообщение должно быть создано. Сначала следует запросить ***DC*** (контекст устройства) для окна – без него рисование в окне невозможно. Затем запрашивается формат пикселя. Компьютер будет выбирать формат, который полностью совпадает или наиболее близок к запрашиваемому формату[2].

```
        case WM_CREATE:
           hDC = GetDC(hWnd);  // Получить контекст устройства для окна
           PixelFormat = ChoosePixelFormat(hDC, &pfd);
                 // Найти ближайшее совпадение для формата пикселов
```

Если подходящий формат пикселя не найден, будет выведено сообщение об ошибке.

```
        if (!PixelFormat)
                {
                        MessageBox(0,
                                "Не найден подходящий формат пиксела.",
                                "Ошибка",MB_OK|MB_ICONERROR);
```
[3]

---

[2] В программе на С++ указатель записан в виде &pfd, в программе на Delphi следует записать @pfd.

[3] В Delphi вместо символа «|» используется «or».

```
            PostQuitMessage(0);
      // Это сообщение говорит, что программа должна завершится
            break;  // Предотвращение повтора кода
      }
```

Если подходящий формат найден, компьютер будет пытаться установить формат пиксела для контекста устройства. Если формат пикселя не может быть установлен по какой-то причине, появится сообщение об ошибке, что формат пикселя не установлен.

```
if(!SetPixelFormat(hDC,PixelFormat,&pfd))
      {
            MessageBox(0,"Формат пиксела не установлен.",
                        "Ошибка",MB_OK|MB_ICONERROR);
            PostQuitMessage(0);
            break;
      }
```

Если код записан, как показано выше, будет создан контекст устройства (***DC***), и установлен подходящий формат пикселя.

Далее следует создать Контекст Рендеринга (***RC***), для этого OpenGL использует ***DC***. Функция ***wglCreateContext*** захватывает Контекст Рендеринга и сохраняет его в переменной ***hRC***. Если по какой-то причине Контекст Рендеринга недоступен, должно появиться сообщение об ошибке.

```
hRC = wglCreateContext(hDC);
if(!hRC)
      {
      MessageBox(0,"Контекст воспроизведения не создан.",
                  "Ошибка",MB_OK|MB_ICONERROR);
      PostQuitMessage(0);
      break;
      }
```

Необходимо сделать активным Контекст Рендеринга, для того чтобы можно было рисовать в окне средствами OpenGL. Если по какой-либо причине это невозможно, должно появиться сообщение об ошибке.

```
if(!wglMakeCurrent(hDC, hRC))
      {
      MessageBox(0,"Невозможно активизировать GLRC.",
                  "Ошибка",MB_OK|MB_ICONERROR);
      PostQuitMessage(0);
      break;
      }
```

10. Создадим область рисования OpenGL. С помощью функции ***GetClientRect*** можно определить ширину и высоту окна. После того как ширина и высота окна получены, инициализируем экран OpenGL. Это достигается при помощи вызова функции ***InitGL*** с шириной и высотой окна в качестве параметров.

```
      GetClientRect(hWnd, &Screen);
      InitGL(Screen.right, Screen.bottom);
      break;
```

11. Следующий фрагмент кода необходим для уничтожения окна. Он использует сообщения ***WM_DESTROY*** и ***WM_CLOSE***. Программа будет посылать это сообщение при выходе из программы по нажатию ALT-F4 или при ошибке (PostQuitMessage(0)).

Функция ***ChangeDisplaySettings(NULL,0)*** восстанавливает разрешение рабочего стола (делая его таким, каким оно было до перехода в полноэкранный режим).

Функция ***ReleaseDC(hWnd,hDC)*** уничтожает контекст устройства окна.

Перечисленные действия уничтожают окно OpenGL.

```
      case WM_DESTROY:
      case WM_CLOSE:
      ChangeDisplaySettings(NULL, 0);
      wglMakeCurrent(hDC,NULL);
      wglDeleteContext(hRC);
      ReleaseDC(hWnd,hDC);
```

```
                PostQuitMessage(0);
                break;
```

12. Опишем обработку сообщений, возникающих при нажатии клавиш.

Сообщение ***WM_KEYDOWN*** возникает всякий раз при нажатии клавиши. Клавиша, которая была нажата, сохраняется в переменной ***wParam***. При нажатии клавиши элемент массива, соответствующий коду нажатой клавиши, принимает значение ***TRUE***.

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

Сообщение ***WM_KEYUP*** вызывается всякий раз при отпускании клавиши. Клавиша, которая отжата, также сохраняется в переменной ***wParam***. При отпускании клавиши соответствующий элемент массива принимает значение ***FALSE***.

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

13. В завершении программы следует обработать изменение размеров окна. Даже при запуске программы в полноэкранном режиме этот код необходим, без него экран OpenGL не появится.

Всякий раз сообщение ***WM_SIZE*** посылается Windows с двумя параметрами - новая ширина, и новая высота экрана. Эти параметры сохранены в ***LOWORD(lParam)*** и ***HIWORD(lParam)***. Вызов функции ***ReSizeGLScene*** изменяет размеры экрана, т.е. передает высоту и ширину в эту секцию кода.

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

Следующий код необходим для обработки Windows всех поступивших сообщений и завершения процедуры.

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

14. Следующая процедура необходима для создания и регистрации окна Windows.

```
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 дескриптора окна
```

Флаги стиля ***CS_HREDRAW*** и ***CS_VREDRAW*** служат для перерисовки окна при его перемещении. ***CS_OWNDC*** создает скрытый ***DC*** для окна, т.е. ***DC*** не может использоваться совместно нескольким приложениями. ***WndProc*** - процедура, которая перехватывает сообщения для программы. Свойство ***hIcon*** установлено равным нулю, т.е. иконка окна не нужна; для мыши используется стандартный указатель. Фоновый цвет не имеет значения (мы установим его в GL). Если меню в этом окне не требуется, то установим его значение в ***NULL.*** Имя класса – это любое имя.

```
        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";
```

Если при регистрации класса произошла ошибка, появится соответствующее сообщение.

```
        if(!RegisterClass(&wc))
```

```
        {
        MessageBox(0,"Ошибка регистрации класса окна.",
                        "Ошибка",MB_OK|MB_ICONERROR);
        return FALSE;
        }
```

Создадим окно. Однако OpenGL будет вызвана только после того как будет послано сообщение ***WM_CREATE***. Флаги ***WS_CLIPCHILDREN*** и ***WS_CLIPSIBLINGS*** требуются для OpenGL и должны быть добавлены именно здесь.

```
        hWnd = CreateWindow(
        "OpenGL WinClass",
        "Это минимальная программа OpenGL",   // 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);
```

Далее следует обычная проверка на ошибки. Если окно не было создано по какой-то причине, сообщение об ошибке следует вывести на экран. При этом генерируется окно с сообщением об ошибке и предложением завершить программу.

```
        if(!hWnd)
        {
        MessageBox(0,"Ошибка создания окна.","Ошибка",MB_OK|MB_ICONERROR);
                return FALSE;
        }
```

Для того, чтобы можно было перейти в полноэкранный режим необходимо следовать правилу: ширина и высота в полноэкранном режиме должна совпадать с шириной и высотой, которые установлены для окна вывода.

```
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
```

Функция ***ShowWindow*** показывает созданное окно.

Функция ***UpdateWindow*** обновляет окно, ***SetFocus*** делает окно активным, и вызывает ***wglMakeCurrent(hDC,hRC)*** чтобы убедиться, что Контекст рендеринга не освобожден.

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

Для того чтобы программа не завершала свою работу сразу же после отрисовки изображения, создадим бесконечный цикл. Для выхода из цикла можно использовать нажатие клавиши ESC. При этом программе будет отправлено сообщение о выходе, и она прервется.

```
        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;
                }
        }
```

Функция ***DrawGLScene*** вызывает ту часть программы, которая фактически рисует объекты OpenGL. Пока оставим эту часть секции пустой, все что будет сделано - очистка экрана черным цветом.

SwapBuffers(hDC) очень важная команда. Мы имеем окно с установленной двойной буферизацией. Это означает, что изображение рисуется на скрытом окне (называемым буфером). Затем с помощью команды переключения буферов скрытый буфер копируется на экран. При этом получается плавная анимация без рывков, и зритель не замечает процесс рисования объектов.

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

Важно отметить, что этот код не будет скомпилирован на Си, он должен быть сохранен как .CPP файл.

15. Скомпилируйте и выполните проект. Итак, наша программа создает окно, размером 640x480, очищает его черным цветом и ожидает нажатия клавиши ESC (Alt+F4) для закрытия окна.
16. Попробуем нарисовать что-либо в этом окне. Добавьте в раздел ***DrawGLScene()*** следующий код:

```
        {
glClear(GL_COLOR_BUFFER_BIT);       // очистка экрана
    glPointSize(2);  //размер точки
    glBegin(GL_POINTS);
            glColor3d(1,0,0);
            glVertex3d(-0.45,-0.4,0);       // первая точка
            glColor3d(0,1,0);
            glVertex3d(0.4,0.4,0);          // вторая точка
            glColor3d(0,0,1);
            glVertex3d(-0.35,0.4,0);        // третья точка
    glEnd();
        }
```

17. Скомпилируйте и выполните проект. Сохраните результат вашей работы в своей папке. Этот шаблон будет использоваться в вашей дальнейшей работе.

Контрольные вопросы.

1. Что понимается под контекстом устройства?
2. Что представляет собой контекст воспроизведения?
3. Какие основные фрагменты должна содержать минимальная программа OpenGL.
4. Что входит в понятие *формат пиксела*?
5. Какие библиотечные файлы должны быть подключены к программе для работы с OpenGL?
6. Какая цветовая модель используется при определении цвета в данной программе?
7. Какие параметры следует указать, чтобы цвет фона был зеленым?
8. Для чего нужна функция масштабирования сцены?
9. В какой секции кода можно записывать команды рисования сцены?
10. Какое правило необходимо соблюдать, чтобы иметь возможность перехода в полноэкранный режим?

Лабораторная работа № 2. Примитивы OpenGL, основные приемы построения двумерных объектов.

Цель работы.

Знакомство с примитивами OpenGL, предназначенными для вывода точек, линий и многоугольников. Определение цвета объектов. Различные способы закрашивания объектов.

Необходимые теоретические сведения.

Для выполнения работы необходимо иметь представление об основных принципах формирования изображения на экране ([7] – лекция 17), о моделировании объектов на плоскости ([7] – лекции 7, 8, 10); знать особенности представления цвета в цветовой модели RGB ([7] – лекция 4); понимать смысл термина «антиэлайзинг» ([7] – лекция 6).

Область вывода в OpenGL задается командой

```
glViewPort(0, 0, ClientWidth, ClientHeight),
```

где первые два параметра задают положение левого верхнего угла окна вывода, параметры ***ClientWidth***, ***ClientHeight*** определяют размер окна в экранных координатах. Центр полученной области вывода имеет координаты (0, 0). Координаты изменяются в диапазоне [-1; 1] по каждой оси.

Если при установке формата пиксела был установлен режим двойной буферизации (флаг ***PFD_DOUBLEBUFFER***), то изображение готовится во внеэкранном буфере, и необходимо обеспечить перезапись содержимого внеэкранного буфера в основной. Сделать это можно командой

```
BOOL SwapBuffers(HDC hdc);
```

Все изображения строятся из отдельных примитивов, которые описываются с помощью набора вершин (***Vertex***). Примитивами OpenGl являются точки (одиночные вершины), линии (пары вершин), треугольники (три вершины), четырехугольники (четыре вершины) и полигоны (3 и более вершин).

Командные скобки. Использование функций glBegin и glEnd.

Команды рисования заключаются между командными скобками ***glBegin*** и ***glEnd***. Командные скобки библиотеки OpenGl представляют собой специальные функции (не имеющие никакого отношения к операторным скобкам языков программирования). Ошибка при использовании командных скобок не распознается компилятором, но может привести к непредсказуемым результатам работы программы.

Внутри командных скобок могут находиться любые операторы языка и многие функции OpenGL. Главное назначение командных скобок – задание режима (примитива) для команд ***glVertex*** (вершина), определяющих координаты вершин для рисования примитивов OpenGL.

Команды, устанавливающие размер точки, толщину и тип линии, включение и отключение режима сглаживания (антиэлайзинг) должны стоять вне командных скобок.

Цвет отдельных вершин или примитивов может устанавливаться как вне командных скобок, так и внутри них. Для установки цвета используется команда

```
glColor3f(0.3f, 0.5f, 0.1f)
```

где цвет формируется как сумма компонент красного, зеленого и синего цветов в указанной пропорции. Значения компонент задаются в виде вещественных чисел в интервале [0; 1].

Компоненты цвета могут быть заданы и в целочисленной форме, предельным значением в этом случае будет являться максимальное 8-битное целое без знака, например, белый цвет будет записан следующим образом:

```
glColor3i(214748647, 214748647,  214748647).
```

Однако предпочтительно использовать команду в вещественной форме, т.к. OpenGL хранит данные именно в вещественном формате.

Цифра 3 в названии команды означает число аргументов команды.

Аргументы функции glBegin.

Аргументами функции ***glBegin*** могут быть стандартные константы OpenGl, определяющие примитивы библиотеки: ***GL_POINTS***, ***GL_LINES***, ***GL_LINE_STRIP***, ***GL_LINE_LOOP***, ***GL_TRIANGLES***, ***GL_TRIANGLE_STRIP***, ***GL_TRIANGLE_FAN***, ***GL_QUADS***, ***GL_QUAD_STRIP***, ***GL_POLYGON***. В программе имена констант должны быть записаны именно так как это сделано здесь (все в верхнем регистре).

Включение и отключение режима сглаживания (антиэлайзинг).

Для установки режима сглаживания перед командными скобками должна стоять команда ***glEnable()*** с соответствующей константой в качестве аргумента, а после командных скобок – команда ***glDisable()*** также с соответствующей константой. Выбор константы определяется примитивом, сглаживание которого должно быть включено (или отключено): ***GL_POINT_SMOOTH*** (сглаживание для точек), ***GL_LINE_SMOOTH*** (сглаживание линий), ***GL_POLYGON_SMOOTH*** (сглаживание для полигонов).

Вывод точек в OpenGL.

Рассмотрим рисование точек. Точки представляют собой одиночные вершины. Для рисования вершины используется команда ***glVertex***. Если точка должна быть изображена на плоскости, то для определения ее положения необходимы две координаты. В этом случае используется функция с двумя аргументами: ***glVertex2f(0, 0)***. Буква ***f*** в названии функции определяет тип аргументов – вещественные числа (***float***). Точка в пространстве определяется тремя координатами, следовательно должна использоваться команда с тремя аргументами: ***glVertex3f(0.5, 0.3,-0.7)***.

Команды ***glVertex*** должны размещаться между командными скобками. При этом количество точек может быть любым.

Аргументом функции ***glBegin*** для рисования точек является константа ***GL_POINTS***.

Аргументом команд ***glEnable()*** и ***glDisable()*** для включения/отключения режима сглаживания является константа ***GL_POINT_SMOOTH***.

Для задания размера точки используется команда ***glPointSize()***. Аргументом является натуральное число, определяющее размер точки в пикселах. В режиме сглаживания существует ограничение на размер точек. Определите экспериментально максимальный размер точки, выводимой в режиме сглаживания.

Пример:

```
glColor3f(1.0, 0.0, 0.0);     // Установили красный цвет
glEnable(GL_POINT_SMOOTH);    // Включение режима сглаживания для
                              // точек
glPointSize(3);               // Установили размер точки 3 пиксела
glBegin(GL_POINTS);           // Режим рисования - точки
      glVertex2f(-0.5, -0.7);
      glVertex2f(0.0, 0.0);
      glVertex2f(0.1, 0.9);
      glVertex2f(0.3, -0.5);
glEnd();                      // Конец рисования
glDisable(GL_POINT_SMOOTH);   // Отключение режима сглаживания для
                              // точек
```

Линии: одиночные, ломаные, замкнутые ломаные.

Для рисования линий существует три режима: одиночные линии, ломаная, замкнутая ломаная.

Одиночная линия определяется двумя вершинами. Если требуется нарисовать несколько одиночных линий, то между командными скобками должны быть описаны координаты пар вершин, т.е. количество команд ***glVertex*** между командными скобками должно быть четным. В случае, если количество вершин нечетно – последняя вершина игнорируется.

Аргументом функции ***glBegin*** для рисования одиночных линий является константа ***GL_LINES***.

Аргументом команд ***glEnable()*** и ***glDisable()*** для включения/отключения режима сглаживания является константа ***GL_LINE_SMOOTH***.

Для задания толщины линии используется команда ***glLineWidth()***. Аргументом является натуральное число, определяющее толщину в пикселах. Так же как и при установке размера точки команду установки толщины линии записывают за пределами командных скобок.

Для изменения типа линии используется команда ***glLineStipple()***, имеющая два аргумента. Первый аргумент – масштабный множитель, а второй представляет собой шестнадцатиричную константу[4], определяющую шаблон штриховки (побитовым способом). Эта команда должна стоять вне операторных скобок.

ПРИМЕР:

```
glColor3f(1.0, 0.0, 0.0);           // Установили красный цвет
glEnable(GL_LINE_SMOOTH);     // Включение режима сглаживания для
                              // линий
glLineWidth(3);         // Установили толщину линии 3 пиксела
glLineStipple(1, 0xF0F0);           // Тип линии - пунктирная
glEnable(GL_LINE_STIPPLE);    // Разрешить изменение типа линии
glBegin(GL_LINES);            // Режим рисования - одиночные линии
      glVertex2f(-0.5, -0.7); // Начало первой линии
      glVertex2f(0.0, 0.0);         // Конец первой линии
      glVertex2f(0.1, 0.9);         // Начало второй линии
      glVertex2f(0.3, -0.5);        // Конец второй линии
glEnd();                      // Конец рисования
glDisable(GL_LINE_SMOOTH);    // Отключение режима сглаживания для
                              // линий
```

Если требуется нарисовать ломаную линию, то в командных скобках используют константу ***GL_LINE_STRIP***. Вершины, перечисленные между командными скобками, интерпретируются следующим образом: конечная точка первой линии является начальной точкой следующего звена ломаной и т.д. Количество вершин может быть как четным, так и нечетным. Ширина и тип ломаной линии задаются так же как и для одиночных линий.

Для рисования замкнутой ломаной аргументом функции ***glBegin*** должна быть установлена константа ***GL_LINE_LOOP***. Последний отрезок замкнутой ломаной в качестве начала имеет последнюю вершину списка, а в качестве конца – первую вершину.

Вывод треугольников: одиночные треугольники, ленты треугольников, веера треугольников.

Для рисования отдельных треугольников константа командных скобок: ***GL_TRIANGLES***. Количество вершин, перечисленных между командными скобками должно быть кратно трем. Каждые три вершины определяют треугольник.

Лента треугольников используется, если изображение может быть построено с помощью нескольких треугольников, имеющих смежные стороны (Рисунок 1. Лента треугольников).

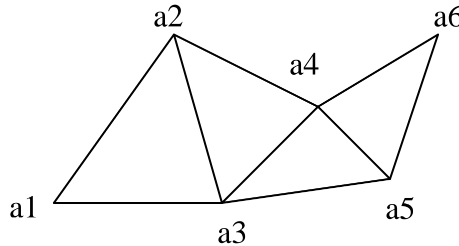


Рисунок 1. Лента треугольников.

[4] В Delphi шаблон запишется в виде, например `$F0F0`.

Здесь сторона ***a2a3*** является общей стороной для первого и второго треугольников, сторона ***a3a4*** – общей стороной второго и третьего треугольников и т.д. Если такую фигуру описывать с помощью одиночных треугольников, то необходимо задавать координаты всех вершин всех треугольников: ***a1***, ***a2***, ***a3***, ***a2***, ***a3***, ***a4***, ***a3***, ***a4***, ***a5***, ***a4***, ***a5***, ***a6*** – всего 12 вершин. Использование ленты треугольников позволяет не дублировать вершины при описании их координат. Изображенная на рисунке фигура может быть представлена лентой треугольников, координаты вершин перечисляются в следующем порядке: ***a1***, ***a2***, ***a3***, ***a4***, ***a5***, ***a6*** – достаточно 6 вершин.

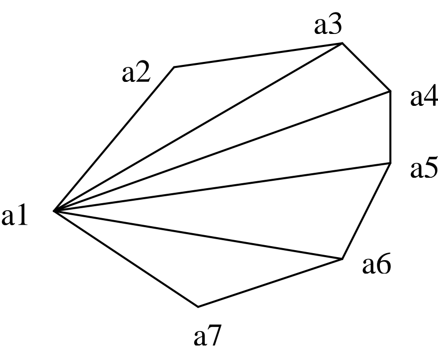


Рисунок 2. Веер треугольников.

Константа командных скобок для примитива «лента треугольников»: ***GL_TRIANGLE_STRIP***.

Другая возможность рисования с помощью треугольников – использование «веера треугольников» в тех случаях, когда несколько треугольников имеют общую вершину (Рисунок 2. Веер треугольников). При описании вершин первой в списке должна стоять общая вершина. Т.о. в списке из нескольких вершин первые три вершины определяют первый треугольник; первая, третья и четвертая – второй; первая, четвертая и пятая – третий; и т.д. Если в списке имеется ***N*** вершин, то будет изображено ***N-2*** треугольника.

Константа командных скобок для веера треугольников: ***GL_TRIANGLE_FAN***.

Вывод четырехугольников.

Константа командных скобок для рисования отдельных четырехугольников: ***GL_QUADS***.

Четырехугольник определяется группой из 4-х вершин, следовательно, количество вершин, записанных между командными скобками, должно быть кратно 4. Лишние вершины игнорируются. Каждая четверка вершин определяет отдельный четырехугольник.

Если изображение создается из связанных четырехугольников (каждая пара четырехугольников имеет общую сторону), то используется примитив «лента четырехугольников». Константа командных скобок: ***GL_QUAD_STRIP***.

Рисование полигонов, передние и задние грани полигонов.

Для рисования многоугольников в командных скобках используется константа ***GL_POLIGON***. При этом вершины, указанные между командными скобками определяют выпуклый многоугольник. Многоугольник строится из связанных треугольников с общей вершиной, в качестве общей вершины берется первая вершина списка (рисунок 2. Веер треугольников).

Список вершин для данного многоугольника: ***a1***, ***a2***, ***a3***, ***a4***, ***a5***, ***a6***, ***a7***.

При рисовании многоугольников следует иметь в виду наличие лицевых (передних) и обратных (задних) граней. Может возникнуть вопрос, зачем при построении плоского изображения различать лицевые и обратные грани? Однако это различие следует иметь в виду при выводе закрашенных полигонов и тем более при трехмерных построениях.

Грань считается лицевой, если вершины перечисляются в направлении против часовой стрелки (в положительном направлении для левой системы координат), и обратной, если обход вершин производится по часовой стрелке (в отрицательном направлении).

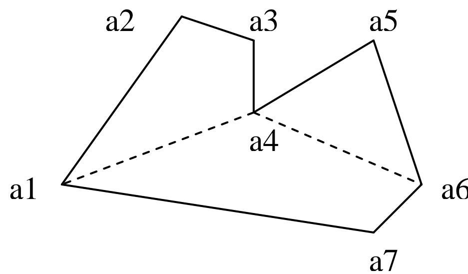


Рисунок 3. Представление невыпуклого полигона с помощью набора выпуклых полигонов.

Построение невыпуклых полигонов.

Если необходимо изобразить невыпуклый многоугольник, то он должен быть представлен как набор выпуклых многоугольников, каждый из которых описывается в своих командных скобках (Рисунок 3. Представление невыпуклого полигона с помощью набора выпуклых полигонов).

Данную фигуру можно разбить на 3 выпуклых многоугольника: ***a1a2a3a4***, ***a1a4a6a7*** и ***a4a5a6***. Следовательно фигура может быть описана как 3 примитива типа ***GL_POLIGON***. Однако заметим, что первая и вторая фигуры – смежные четырехугольники, а третья фигура – треугольник. Для воспроизведения треугольников и четырехугольников желательно использовать «родные» примитивы, причем, если это возможно – связанные. Т.е. для построения изображенной фигуры первые два четырехугольника мы опишем в одних командных скобках как связанные четырехугольники с общей стороной ***a1a4*** (последовательность перечисления вершин: ***a2***, ***a3***, ***a4***, ***a1***, ***a7***, ***a6***) – используем константу командных скобок ***GL_QUAD_STRIP***. Треугольник ***a4a5a6*** опишем в своих командных скобках с константой ***GL_TRIANGLES***.

Можно решить эту задачу и другими способами. Вообще оптимальным будет разбиение невыпуклой фигуры на треугольники, поскольку построение треугольников, как правило, реализовано на аппаратном уровне.

Для сглаживания (антиэлайзинга) многоугольников используется константа ***GL_POLYGON_SMOOTH***. Команда используется так же, как для точек и линий. Поскольку треугольники и четырехугольники являются многоугольниками, то сглаживание для них осуществляется как для многоугольников.

Особенности режимов закрашивания для многоугольников.

До сих пор вы изображали многоугольники с использованием заливки, причем, если для разных вершин примитивов были заданы различные цвета, то фигуры окрашивались градиентно, с плавным переходом цветов от вершины к вершине. Это происходит из-за того, что по умолчанию способ тонирования задан плавным. Чтобы изменить способ тонирования, необходимо перед командными скобками вызвать функцию ***glShadeModel(GL_FLAT)***. В этом случае связанные фигуры окрашиваются по правилу старшинства цвета второго примитива.

Для задания режима вывода многоугольников: в контурном виде (без заливки), с заливкой, - используется команда ***glPolygonMode***. Второй параметр команды указывает способ изображения грани фигуры с помощью следующих констант: ***GL_FILL*** – вывод граней с заливкой; ***GL_LINE*** – каркасный вывод (только контуры); ***GL_POINT*** - выводятся только вершины. Первый параметр определяет для каких граней установлен

режим: лицевые – ***GL_FRONT***; обратные – ***GL_BACK***; лицевые и обратные – ***GL_FRONT_AND_BACK***. Описанная команда помещается перед командными скобками.

Задания к лабораторной работе.

Используя программу-шаблон, созданную при выполнении предыдущей лабораторной работы выполните следующие задания.

Задание 1. Построить точки расположенные в вершинах правильного n-угольника. Установить режим сглаживания для точек. Экспериментально определить максимальный размер точки, при котором возможно сглаживание.

Задание 2. Используя примитив для вывода линий нарисовать правильный n-угольник. Изменить тип и ширину линий.

Задание 3. Используя примитив для вывода ломаной линии нарисовать фигуру, изображенную на рис.1.

Задание 4. Используя примитив для вывода замкнутой ломаной нарисовать фигуру, изображенную на рис.2

Задание 5. Построить фигуру, изображенную на рис.2, разбив ее на треугольники (каждый треугольник окрашен случайным цветом). Выполните три варианта построений с использованием примитивов:

А) треугольник;

Б) лента треугольников;

В) веер треугольников.

Чем отличаются результаты при изменении способа тонирования?

Задание 6. Используя примитив для вывода многоугольников построить правильный n-угольник.

Задание 7. Построить невыпуклый многоугольник, изображенный на рис.3, представив его в виде совокупности отдельных многоугольников, назначив каждому многоугольнику свой цвет. Посмотреть результат работы программы для различных способов тонирования.

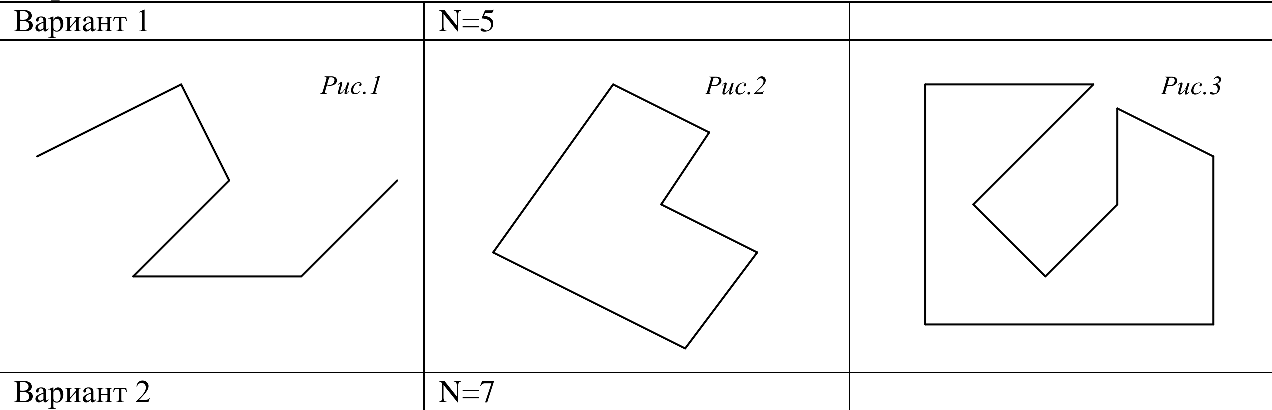
Задание 8. Изменить программу предыдущей задачи таким образом, чтобы

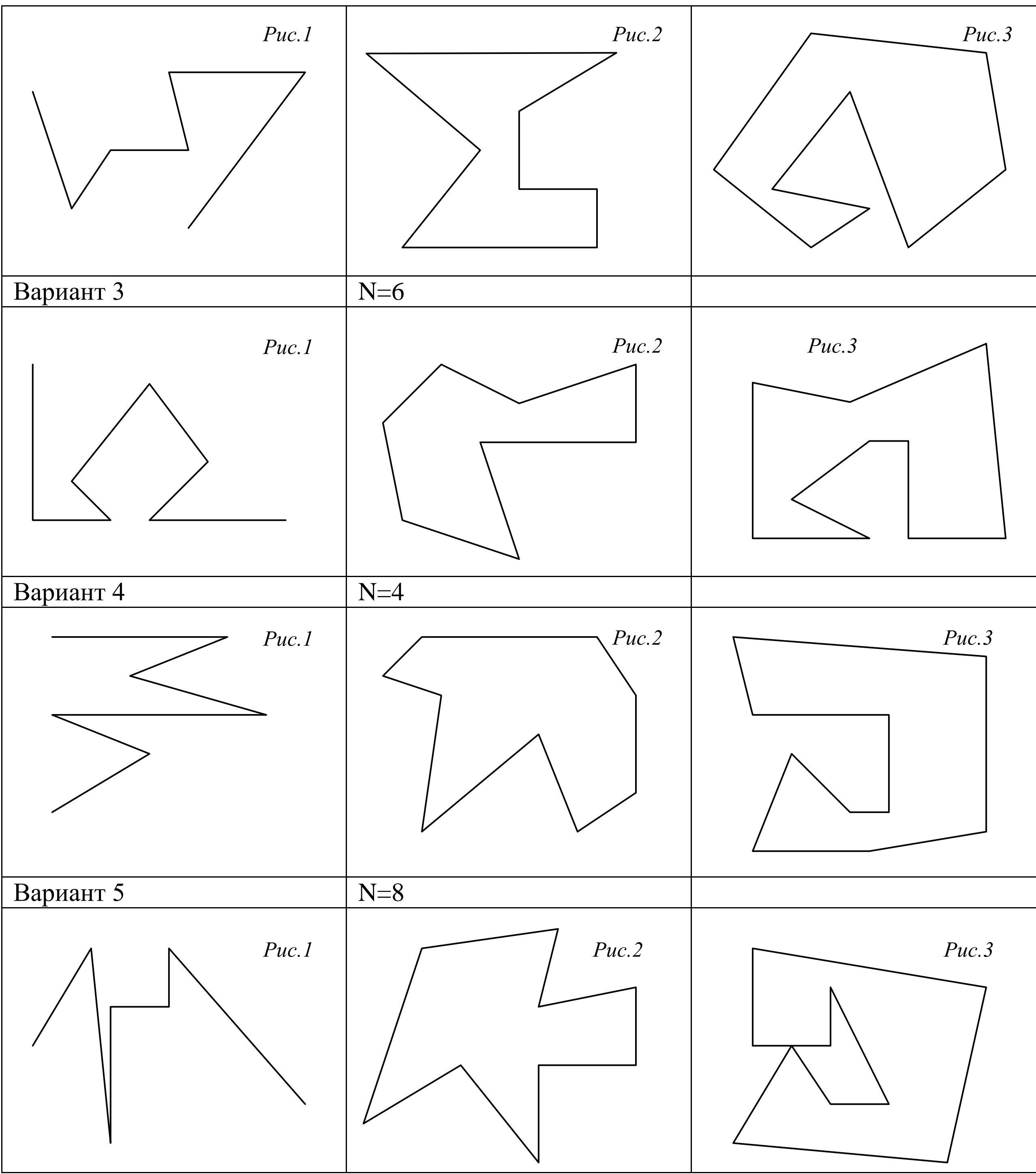
А) лицевые грани изображались только вершинами;

Б) лицевые грани изображались закрашенными, а обратные – линиями;

В) лицевые и обратные грани изображались линиями (каркасное изображение).

Варианты к заданию.

Вариант 1	N=5	
		
Вариант 2	N=7	



Дополнительные задания.

1. В установленном графическом окне построить N точек, окрашенных случайным образом и распределенных случайным образом по всей площади окна.
2. Фонтан. В некоторой окрестности заданной точки появляются и пропадают данное количество точек различного размера и цвета.
3. При нажатии клавиш управления курсором (при движении мыши) появляются и гаснут некоторое данное количество точек случайного размера и цвета.
4. Для области размером N*N точек (размер точки задан) реализовать алгоритм дизеринга с использованием трех заданных цветов.
5. Бенгальский огонь. В некоторой окрестности данной точки рисуются линии случайного размера, цвета и направления (длина линий ограничена).
6. Бенгальский огонь в движении. Картинка, построенная в результате предыдущей задачи, изменяется при нажатии клавиш управления курсором.

Контрольные вопросы.

1. Что такое командные скобки, каково их назначение?
2. Какие константы библиотеки OpenGL могут быть параметрами функции glBegin?
3. Что такое антиэлайзинг, для чего он служит?
4. Какие режимы существуют для рисования линий?
5. Какие режимы существуют для изображения треугольников, чем они различаются?
6. Что такое выпуклые и невыпуклые многоугольники?
7. Каким образом можно построить невыпуклый многоугольник?
8. Чем отличаются лицевые и обратные грани?
9. Какая команда изменяет способ тонирования?
10. Какие режимы вывода многоугольников вам известны? В каком месте программы должна быть записана команда, изменяющая режим вывода многоугольников?

Лабораторная работа № 3. Использование массивов вершин. Преобразования координат.

Цель работы:

продемонстрировать возможности использования специальных массивов OpenGL для моделирования графических объектов, дать понятие о векторной форме команд; исследовать возможности библиотеки для перемещения, масштабирования и поворотов объектов.

Порядок выполнения лабораторной работы.

Перед выполнением лабораторной работы следует ознакомиться с теоретическим материалом по теме «Аффинные преобразования» ([7] – лекция 7); изучить теоретические сведения, относящиеся к данной лабораторной работе.

Задания лабораторной работы следует выполнять в указанном порядке, используя данные указанного преподавателем варианта.

Необходимые теоретические сведения.

Массивы вершин.

Поскольку описывать сложные объекты вершинами достаточно трудоемкое занятие в OpenGL имеется возможность использования массива вершин.

Включение/выключение режима.

Работа с использованием массивов OpenGL это особый режим работы, для включения и отключения которого существуют команды, аналогичные ***glEnable*** и ***glDisable***:

glEnableClientState(<константа>); — включает режим
glDisableClientState(<константа>); — выключает режим

Параметром обеих команд является символьная константа, определяющая тип массива: ***GL_VERTEX_ARRAY***, ***GL_COLOR_ARRAY*** и др.

Объявление массива вершин:

GLfloat aVertex [<количество элементов-вершин>] [<число параметров>];

Первый индекс массива ***aVertex*** определяет необходимое число вершин, координаты которых будут храниться в массиве. Индексация вершин производится от 0 до значения, на единицу меньшего, чем указанное количество; второй индекс определяет число параметров процедуры ***glVertex*** – 2, 3 или 4. Соответственно вызов процедуры ***glVertex*** будет записан в векторной форме, например:

```
glVertex2fv(aVertex[5]);
```

отобразит вершину в двумерной системе координат, соответствующую 5-му элементу массива вершин.

GLfloat aColors [<количество элементов с данными цветами>] [<число параметров>];

Первый индекс определяет количество элементов изображения, для которых задаются цвета, второй индекс может принимать значения 3 (RGB) или 4 (RGBA).

Заполнение массива вершин

может производиться как в теле процедуры рисования, так и в отдельной процедуре.

Для формирования указателей на массивы используются процедуры

```
glVertexPointer(<число координат>,  GL_FLOAT, 0, aVertex);
glColorPointer(<число компонент цвета>, GL_FLOAT, 0, aColors);
```

Второй параметр указывает тип элементов массива (другие варианты: ***GL_SHORT***, ***GL_INT***, ***GL_DOUBLE***); третий – 0 – означает, что все элементы расположены последовательно; последний параметр обеих процедур представляет собой указатель на соответствующий массив.

Команды рисования.

После того как указатели на массивы сформированы, можно использовать команды рисования на основе массивов. Простейшей такой командой является команда

```
glArrayElement(<индекс элемента>);
```

Эта команда заменяет команду описания вершин и стоит внутри командных скобок какого-то примитива. Например

```
glBegin(GL_POLYGON);
    glArrayElement(0);
    glArrayElement(1);
    glArrayElement(2);
    glArrayElement(3);
glEnd;
```

Следующая команда позволяет нарисовать сразу несколько однотипных объектов:

```
glDrawArrays(<примитив>, <начальный индекс>, <количество элементов>);
```

Команда имеет три параметра: первый параметр определяет примитив и задается константой примитива (см. предыдущую л.р.); второй параметр – начальный индекс массива; третий параметр – количество элементов массива. Например:

```
glDrawArrays(GL_POLYGON, 0, 64);
```

рисует полигон из 64 вершин, заданных в массиве. Обратите внимание, что здесь не требуется командных скобок.

Геометрические преобразования.

В процессе построения изображения координаты вершин подвергаются различным преобразованиям, таким как сдвиг (перенос), поворот, масштабирование, преобразования проецирования. Данная лабораторная работа посвящена простейшим преобразованиям координат. Проекции будут рассмотрены позднее.

По умолчанию камера находится в начале координат и направлена вдоль отрицательного направления оси Oz.

Матрица моделирования.

Для преобразований координат в OpenGL применяется матрица моделирования (*modelview matrix*). Она служит для задания положения объекта и его ориентацию. Константа для матрицы моделирования: GL_MODELVIEW.

Все координаты внутри OpenGL хранятся при помощи однородных координат, т.е. в виде четырехмерных векторов (x, y, z, w). Если координата z не задана, то она полагается равной 0. Если координата w не задана, то она полагается равной 1.

При выполнении преобразования координат текущая матрица моделирования умножается на матрицу выбранного геометрического преобразования.

Текущая матрица моделирования задается при помощи процедуры

```
Void  glMatrixMode (GL_MODELVIEW);
```

Процедура

```
Void  glLoadIdenity();
```

устанавливает текущую матрицу равной единичной.

Для задания матрицы моделирования ее сначала устанавливают равной единичной, а затем последовательно применяют к текущей матрице различные геометрические преобразования.

Преобразование переноса:

Void glTranslatef (TYPE x, TYPE y, TYPE z);

реализует сдвиг объекта на величину (*x, y, z*).

Преобразование поворота:

Void glRotatef (TYPE angle, TYPE x, TYPE y, TYPE z);

обеспечивает поворот объекта на угол *angle* в направлении против часовой стрелки вокруг прямой с направляющим вектором (*x, y, z*).

Чтобы осуществить поворот на плоскости относительно начала координат, следует использовать вектор (*x, y, z*)=(0, 0, 1).

Преобразование масштабирования:

Void glScalef (TYPE kx, TYPE ky, TYPE kz);

осуществляет частичное масштабирование объекта вдоль каждой из координатных осей на значения, определяемые соответствующими параметрами.

Если последовательно указано несколько преобразований, то в результате текущая матрица будет последовательно умножена на матрицы соответствующих преобразований.

Команды преобразований помещаются перед командными скобками и/или после них.

Все объекты в OpenGL рисуются в точке отсчета системы координат (опорная точка), а команды геометрических преобразований изменяют положение системы координат, а не объекта относительно неподвижной системы.

Особенность команд преобразования координат заключается в том, что каждое повторное обращение к обработчику перерисовки экрана будет повторно применять преобразования к изображаемым объектам. Кроме того, бывает необходимо применить преобразование не ко всем объектам сцены, а лишь к некоторым из них. Поэтому важно возвращать систему в исходное положение. Например, если при изображении некоторого объекта система координат была сначала перенесена, а затем повернута:

```
glTranslatef(-0.3, 0.5,0.0);
glRotatef(60, 0, 0, 1);
```

то после рисования необходимо выполнить команды обратных преобразований в обратном порядке:

```
glRotatef(-60, 0, 0, 1);
glTranslatef(0.3, -0.5,0.0);
```

Описанный выше способ восстановления положения системы координат достаточно громоздок и неэффективен в вычислительном плане, поскольку обратные преобразования выполняются через перемножение матриц. Другой способ позволяет запоминать состояния матрицы моделирования и восстанавливать его в нужный момент.

Команды glPushMatrix и glPopMatrix позволяют запомнить и восстановить текущую матрицу. Эти команды оперируют со стеком, т.е. можно запоминать несколько величин, а при каждом восстановлении содержимое стека поднимается вверх на единицу данных (восстановление осуществляется в обратном порядке!).

Команды glPushMatrix и glPopMatrix устанавливаются соответственно до и после командных скобок.

Задания к лабораторной работе.

Задание 1. Создать массив вершин, содержащий координаты правильного ***n***-угольника. Используя сформированный массив нарисовать:

a. Полигон;
b. Набор четырехугольников;
c. Набор треугольников;
d. Набор линий.

Все объекты должны быть изображены каркасно.

Задание 2. Нарисуйте треугольник, прямоугольник и линию. Получите новое изображение на котором:

- треугольник следует смасштабировать с коэффициентами ***(kx, ky)*** и перенести на вектор ***p***;
- линию повернуть на угол α относительно начала координат;
- прямоугольник повернуть относительно точки с координатами ***(x, y)*** на угол β.

Задание 3. Используя примитив для рисования линий и операции геометрических преобразований изобразить фигуру, показанную на рисунке.

Варианты к заданиям.

Вариант	Параметры	Фигура	Вариант	Параметры	Фигура
Вариант 1	N=5 (kx,ky)=(2, 1.5) p=(12, 5) α=30 (x, y)=(3, 3) β=-15		Вариант 4	N=4 (kx,ky)=(2.5, 0.5) p=(2, 5) α=-30 (x, y)=(13, 7) β=35	
Вариант 2	N=7 (kx,ky)=(3, 0.5) p=(-5, 10) α=-70 (x, y)=(-5, -5) β=45		Вариант 5	N=8 (kx,ky)=(1.2, 1.5) p=(3, 15) α=40 (x, y)=(-3, 8) β=-25	
Вариант 3	N=6 (kx,ky)=(-0.5, 1.5) p=(10, -5) α=45 (x, y)=(-10, 3) β=-30				

Дополнительные задания.

1. Изобразите выпуклый многоугольник, заданный массивом вершин и точку белого цвета вне этого многоугольника. Точка может перемещаться по экрану при помощи клавиш управления курсором. При нажатии клавиши <ENTER> цвет точки становится: красным – точка внутренняя; зеленым – граничная; белым – внешняя точка.
2. Изобразите на экране отрезок прямой линии и некоторую точку вне этой линии. Пользуясь аффинными преобразованиями, постройте точку, симметричную данной, относительно линии.
3. Изобразите растущий прямоугольник, пользуясь только аффинными преобразованиями.
4. Изобразите мишень, состоящую из 10 концентрических окружностей. Раскрасьте мишень таким образом, чтобы каждое кольцо было окрашено в свой цвет.
5. Напишите программу для изображения гистограммы, значения категорий для которой хранятся в числовом массиве. Гистограмма должна
   а) полностью изображаться на экране;
   б) располагаться в заданной области окна.

Контрольные вопросы.

1. Для чего применяется массив вершин?

2. Каким образом производится объявление и заполнение массива вершин?
3. Какие команды служат для работы с массивом вершин?
4. Что необходимо сделать, прежде чем начать работать с массивом вершин?
5. Что такое матрица переноса и для чего она служит?
6. Какие существуют виды преобразований и какие команды openGL им соответствуют?

Лабораторная работа № 4. Трехмерные построения. Буфер глубины. Видовые параметры. Параллельная и перспективная проекции.

Цель работы.

Выяснить особенности трехмерного моделирования, построения объемных изображений как проекций на экранной плоскости; дать понятие о параметрах вида.

Порядок выполнения работы.

Перед выполнением лабораторной работы следует ознакомиться с необходимыми теоретическими сведениями, относящимися к возможностям трехмерного моделирования с использованием библиотеки OpenGL. Рекомендуется внимательно изучить теоретический материал, относящийся к различным способам проецирования ([7] – лекция 15).

Задания лабораторной работы рекомендуется выполнять в порядке их следования, в соответствии с вариантом, указанным преподавателем.

Необходимые теоретические сведения.

Трехмерные координаты.

В предыдущих лабораторных работах для рисования фигур использовалась версия команды glVertex с двумя параметрами. Координата по оси Z предполагалась равной 0.

Для изображения в пространстве используется команда glVertex с тремя параметрами:

glVertex3f(<координата x>, <координата y>, <координата z>).

Значение координаты Z лежит в пределах от –1 до 1. По умолчанию считается, что наблюдатель расположен в точке (0, 0, 0), что следует иметь в виду. Чтобы вершины с положительными координатами Z были изображены, следует сдвинуть систему координат вглубь экрана.

OpenGL воспроизводит только те части примитивов, координаты которых не превосходят по модулю 1. Примитивы с одинаковыми координатами рисуются по принципу: каждый последующий рисуется поверх предыдущего. Такое изображение не всегда дает правильные результаты, в частности, если не включена поддержка *буфера глубины.*

Буфер глубины.

Буфер глубины используется для передачи пространства. При воспроизведении каждого пиксела в этот буфер записывается информация о значении координаты Z пиксела, так называемая оконная Z. Если на пиксел приходится несколько точек, на экран выводится точка с наименьшим значением этой координаты.

Для правильного построения изображений (в соответствии с глубиной) необходимо включить режим тестирования глубины с помощью команды:

glEnable(GL_DEPTH_TEST);

Код сцены следует начинать с очистки двух буферов: буфера кадра и буфера глубины:

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

Точно так же, как перед очередным построением необходимо очистить поверхность рисования, для корректного воспроизведения требуется очистить буфер пространства.

Проекции.

Изображение трехмерных объектов на экране строится с помощью проекций: параллельной (ортографической) и перспективной.

При изображении трехмерных объектов следует иметь в виду, что изображение строится в пределах единичного куба (координаты по каждой из осей изменяются в пределах от –1 до 1), причем начало координат (центр куба) находится в центре окна, а ось Z направлена перпендикулярно плоскости экрана в сторону наблюдателя. Способ проецирования определяется выбором команды для установки параметров вида:

glFrustum – выбирается перспективный способ проецирования

glOrtho – выбирается параллельный способ проецирования

При параллельном проецировании сохраняется параллельность прямых. Например, при изображении куба в параллельной проекции задняя и передняя грани будут равны.

При использовании перспективной проекции параллельные линии объекта изображаются сходящимися в некоторой удаленной («вглубь» экрана) точке, что и создает перспективу.

Параметры проецирования относятся к *параметрам вида*.

Видовые параметры.

Кроме типа проекции изображения на экран, к видовым параметрам относят параметры, определяющие область воспроизведения в пространстве. Все, что выходит за пределы этой области, будет отсекаться при воспроизведении. Именно эти параметры и являются аргументами команд glFrustum и glOrtho:

1) координата плоскости отсечения слева;
2) координата плоскости отсечения справа;
3) координата плоскости отсечения снизу;
4) координата плоскости отсечения сверху;
5) расстояние от наблюдателя до ближней плоскости отсечения;
6) расстояние от наблюдателя до дальней плоскости отсечения.

Следует иметь в виду, что для команды glFrustum 5-й и 6-й параметры всегда положительны, а для команды glOrtho значения этих параметров могут быть и отрицательными.

Устанавливать видовые параметры не обязательно при каждой перерисовке экрана, достаточно делать это лишь при изменении размеров окна (или при создании нового окна).

Старайтесь переднюю и заднюю плоскости отсечения располагать таким образом, чтобы расстояние между ними было минимально возможным: чем меньший объем ограничен этими плоскостями, тем меньше вычислений приходится производить OpenGL.

Как правило, при установке видовых параметров производят и перенос системы координат таким образом, чтобы все точки, попадающие в область воспроизведения. Обратите внимание на то, что в этом случае (проецирование + перенос) область видимости расширяется.

Чтобы изображенная фигура выглядела пространственной, систему координат разворачивают вокруг оси X и вокруг оси Y.

Место команд в программе.

Для того чтобы при каждой последующей перерисовке экрана не происходило изменение размеров сцены, связанное с переносом и проецированием, следует использовать видовые команды по определенным правилам.

Первый способ – использование команды glLoadIdentity:

```
glLoadIdentity;                      //Сброс всех матриц в 1
glFrustum(-1, 1, -1, 1, 3, 10);      //видовые параметры
glTranslatef(0.0, 0.0, -5.0);        //начальный сдвиг системы координат
glRotatef(30.0, 1.0, 0.0, 0.0);      //поворот относительно оси X
glRotatef(70.0, 0.0, 1.0, 0.0);      //поворот относительно оси Y
```

```
        glBegin(…);
        …..                          // команды рисования
        glEnd;
```

Второй способ – использование команд glPushMatrix и glPopMatrix:

```
    glPushMatrix;                    //запоминаем текущую матрицу
    glFrustum(-1, 1, -1, 1, 3, 10);  //видовые параметры
    glTranslatef(0.0, 0.0, -5.0);    //начальный сдвиг системы координат
    glRotatef(30.0, 1.0, 0.0, 0.0);  //поворот относительно оси X
    glRotatef(70.0, 0.0, 1.0, 0.0);  //поворот относительно оси Y
        glBegin(…);
        …..                          // команды рисования
        glEnd;
    glPopMatrix;                     //восстанавливаем текущую матрицу
```

Матрица проецирования.

Для корректного выполнения операций проецирования используется *матрица проецирования.* Текущую матрицу проецирования можно установить с помощью команды:

```
glMatrixMode(GL_PROJECTION);
```

После выполнения этой команды следует установить матрицу проецирования в единичную с помощью команды glLoadIdentity.

Как правило, параметры вида помещаются в обработчике изменения размеров окна, стартовые сдвиги и повороты обычно располагаются здесь же, а код воспроизведения сцены заключается между командами glPushMatrix и glPopMatrix. Иногда поступают иначе: код воспроизведения начинается с glLoadIdentity, а далее идут стартовые трансформации и собственно код сцены.

Задания к лабораторной работе.

Задание 1. Постройте изображения трех плоских фигур в пространстве (в соответствии с вариантом). Оцените полученный результат: правильно ли изображены фигуры, если нет - объясните почему.

Задание 2. Установите режим проверки буфера глубины в программе к заданию 1. Сравните полученный результат с предыдущим.

Задание 3. Установите видовые параметры в соответствии с вариантом и параллельную проекцию; постройте изображение куба с координатами вершин, равными по модулю 1 (куб изображается с помощью 6 квадратов). Оцените результат.

Задание 4. Установите перспективную проекцию и постройте каркасное изображение куба

Задание 5. Поверните оси координат в соответствии с вариантом и постройте изображение куба. Оцените результат. Почему части куба отсечены? Измените видовые параметры таким образом, чтобы куб изображался без отсечений. Измените код программы таким образом, чтобы каждая грань куба изображалась своим цветом.

Задание 6. Напишите программу, в которой изображение куба изменяется при нажатии клавиш (в соответствии с вариантом).

Варианты заданий.

Вариант	Фигуры (заданы вершинами)	Параметры вида	Поворот осей	К заданию 6

	Треугольник: (0, 0.3, 0.2) (0.3, 0.3, 0) (0, 0, 0.7) Квадрат: (-0.5, 0.5, 0.3) (-0.5, -0.5, 0.3) (0.5, -0.5, 0.3) (0.5, 0.5, 0.3) Треугольник: (0, 0, 0.3) (0.4, 0, 0.5) (-0.5, 05, -1)	-0.7, 0.7 -0.85, 0.85 3, 10	Вокруг OX: 30 Вокруг OY: 70	Изменяется способ изображения: каркасное или заливка цветом
Вариант 2	Фигуры (заданы вершинами)	Параметры вида	Поворот осей	К заданию 6
	Четырехугольник: (-0.5, -0.5, -0.5) (-0.5, 0.5, -0.5) (0.8, 0.5, -0.5) (0.5, -0.8, -0.5) Треугольник: (-1, -1, 1) (-1, 1, 1) (1, 0, -1); Треугольник: (-0.7, -0.7, -0.7) (0, 0, 0) (0, 0, 1)	-0.9, 0.8 -0.8, 0.9 5, 10	Вокруг OX: 45 Вокруг OY: 45	Изменяется масштаб: увеличение или уменьшение объекта
Вариант 3	Фигуры (заданы вершинами)	Параметры вида	Поворот осей	К заданию 6
	Треугольник: (1, 1, 0) (1, 0, 1) (0, 1, 1) Четырехугольник: (0.8, 0.7, 1) (-0.8, 0.7, 1) (-0.8, -0.7, -0.8) (0.8, -0.7, -0.8) Треугольник: (0, 0.5, -0.5) (-0.5, 0, -0.5) (0.5, 0, 0.5)	-0.85, 0.85 -0.7, 0.7 6, 12	Вокруг OX: -25 Вокруг OY: 60	Куб поворачивается при нажатии на некоторую клавишу на заданный угол
Вариант 4	Фигуры (заданы вершинами)	Параметры вида	Поворот осей	К заданию 6
	Треугольник: (1, 1, 1) (-1, 0, -1) (1, -1, 1) Треугольник: (-0.7, -0.7, 0) (-0.7, 0.7, 0) (0.7, 0, 0) Треугольник: (0, 0, -0.9) (-1, -0.8, 0.5) (1, 1, 1)	-0.8, 0.8 -0.9, 0.9 3, 8	Вокруг OX: -15 Вокруг OY: 65	Куб перемещается: удаление или приближение

Вариант	Фигуры (заданы вершинами)	Параметры вида	Поворот осей	К заданию 6
Вариант 5	Квадрат: (-0.6, -0.6 , -0.2) (0.6, -0.6, -0.2) (0.6, 0.6, -0.2) (-0.6, 0.6, -0.2) Квадрат: (-0.5, 0.8, -0.8) (-0.5, 0.8, 0.8) (-0.5, -0.8, 0.8) (-0.5, -0.8, -0.8) Треугольник: (-1, -1, 0) (0, 1, 1) (1, 0, 0)	-1, 0.9 0.9, 1 2, 9	Вокруг ОХ: 35 Вокруг OY: 50	Изменяется способ проецирования: прямая или перспективная проекции

Дополнительные задания.

1. Напишите программу для построения тетраэдра в выбранной проекции, пользуясь командой рисования ленты треугольников. Реализуйте изображение тетраэдра а) каркасно; б) с заливкой так, чтобы лицевые грани изображались красным цветом, а обратные – синим.
2. Изобразите октаэдр. Реализуйте вращение октаэдра относительно его центра симметрии.
3. Изобразите трехмерную лестницу, пользуясь операцией сдвига.
4. Изобразите винтовую лестницу.

Контрольные вопросы.

1. Для чего применяется буфер глубины?
2. Каким образом выводятся фигуры при отключенном тесте глубины?
3. Какие способы проецирования существуют?
4. Чем отличаются различные способы проецирования?
5. Какой способ проецирования целесообразно использовать для построения реалистичных изображений?
6. Какой способ проецирования удобен при построении чертежей?
7. Что такое видовые параметры и как они применяются?
8. С какой целью в командах, устанавливающих видовые параметры, используются плоскости отсечения?

Лабораторная работа № 5. Квадрик-объекты. Камера.

Цель работы.

Исследование возможностей моделирования графических объектов с использованием квадрик-объектов библиотеки OpenGL; установка и использование камеры; исследование и сравнительный анализ результатов, полученных при перемещении камеры и при перемещении объекта.

Порядок выполнения работы.

Перед выполнением работы следует ознакомиться с теоретическим материалом, относящимся к данной работе.

Задания следует выполнять последовательно в соответствии с вариантом, указанным преподавателем.

В задании 2 требуется изобразить сложный трехмерный объект с использованием квадрик-объектов библиотеки glu OpenGL. Рекомендуется использовать при моделировании изображения квадрик-объекты различных типов.

Необходимые теоретические сведения.

Для работы со стандартными трехмерными объектами в OpenGL используются команды библиотеки GLU, которая реализована в виде модуля glu32.dll. Эта библиотека

содержит несколько функций управления проекциями, функции работы с полигонами, кривыми и сплайновыми поверхностями и другие функции.

Квадрик-объекты.

Для изображения цилиндров и конусов используется примитив ***gluCylinder***.

Для изображения сферы – ***gluSphere***.

Для изображения диска – ***gluDisk***.

Для изображения части диска - ***gluPartialDisk***.

Перечисленные примитивы являются квадрик-объектами (*quadric objects*) и изображают соответствующие геометрические тела с помощью аппроксимации плоскими гранями. Поэтому, при задании таких объектов указывают число разбиений объекта на грани по различным направлениям, причем, чем больше число разбиений, тем более гладким получится изображение объекта.

Для работы с командами библиотеки GLU вводится переменная специального типа[5]:

```
GLUquadricObj *quadricObj
```

Функция ***gluCylinder*** имеет 6 параметров: ***quadricObj*** – указатель на квадрик-объект; ***R*** – первый радиус (центр этого основания цилиндра расположен в начале координат); ***r*** – второй радиус; ***H*** – длина; ***slices*** – число продольных граней (число вершин многоугольника, являющегося основанием цилиндра); ***stacks*** – число разбиений по длине:

```
gluCylinder(quadricObj, R, r, H, slices, stacks)
```

Функция ***gluSphere*** имеет 4 параметра: ***quadricOb***j – указатель на квадрик-объект; ***R*** – радиус сферы; ***slices*** – число параллелей; ***stacks*** – число меридианов:

```
gluSphere(quadricObj, R, slices, stacks)
```

Функция ***gluDisk*** рисует диск или кольцо и имеет 5 параметров: ***quadricObj*** – указатель на квадрик-объект; ***r*** – внутренний радиус; ***R*** – внешний радиус; ***slices*** – число секторов; ***stacks*** – число концентрических колец:

```
gluDisk(quadricObj, r, R, slices, stacks)
```

Функция ***gluPartialDisk*** рисует часть диска или кольца и имеет 7 параметров: первые пять параметров имеют тот же смысл, что и в предыдущей функции; ***A*** – начальный угол; ***SA*** – угол развертки (диск располагается в плоскости ***xOy***, его центр совпадает с началом координат, начальный угол отсчитывается по часовой стрелке от положительного направления направления оси ***Y***, углы измеряются в градусах):

```
gluPartialDisk(quadricObj, r, R, slices, stacks, A, SA)
```

Для того чтобы нарисовать квадрик-объект, сначала следует вызвать функцию ***gluNewQuadric*** (которая создает в динамической памяти указатель на объект), а после рисования объекта – освободить память вызовом функции ***gluDeleteQuadric***. Между указанными функциями заключается блок рисования квадрик-объекта:

```
GLUquadricObj *quadricObj
quadricObj=gluNewQuadric();
      …….          // рисуем объект quadricObj
gluDeleteQuadric (quadricObj);
```

Объекты данного типа располагаются в пространстве в центре координат с учетом матрицы ***GL_MODELVIEW***. Поэтому, чтобы нарисовать изображение объекта в нужном месте, необходимо соответствующим образом изменить эту матрицу, например, с помощью функций ***glTranslate*** и ***glRotate***. Указанные команды располагаются внутри блока рисования квадрик-объекта.

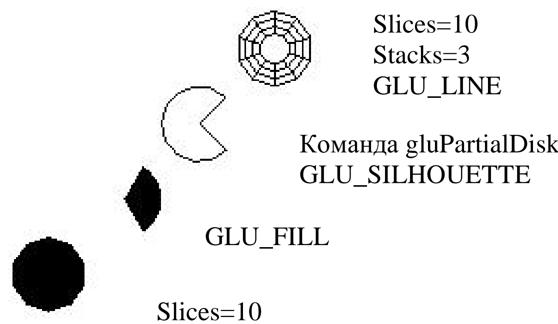
На базе одного квадрик-объекта можно строить сколько угодно фигур, не обязательно для каждой из них создавать собственный объект, если их параметры идентичны (например, при рисовании нескольких сфер, нескольких дисков, цилиндра и конуса). Однако следует иметь в виду, что рисуются объекты всегда в начале координат, поэтому необходимо корректно использовать команды ***glPushMatrix*** и ***glPopMatrix*** при преобразованиях системы координат.

---

[5] В Delphi переменную-указатель на квадрик-объект объявляют так

```
quadricObj: GLUquadricObj;
```

Рис. 1 Различные варианты использования команд gluDisk и gluPartialDisk.



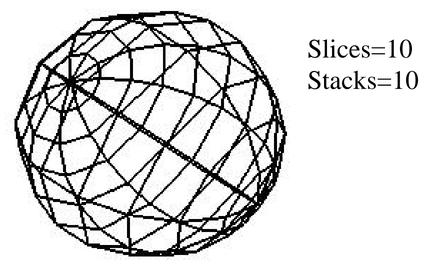


Рис. 2 Результат работы команды gluSphere в каркасном режиме.

По умолчанию каждый квадрик-объект рисуется со сплошным заполнением. Изменить стиль показа можно вызовом функции ***gluQuadricDrawStyle***, указав стиль показа: ***GLU_POINT*** – в виде точек, расположенных в вершинах многоугольников; ***GLU_LINE*** – каркасное изображение; ***GLU_FILL*** – сплошное заполнение; ***GLU_SILHOUETTE*** – силуэт, контур (разновидность каркасного).

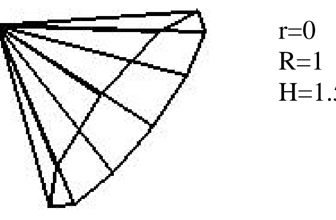


Рис. 3 Результат работы команды gluCilinder.

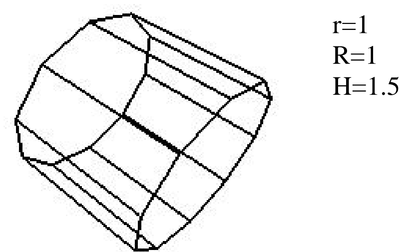


Рис. 4 Результат работы команды gluCilinder.

В дополнительной библиотеке GLUT (заголовочный файл glut.h) имеются команды рисования сферы (каркасное изображение сферы – ***glutWireSphere***, сплошное изображение – ***glutSolidSphere***), конуса (***glutWireCone***, ***glutSolidCone***), тора (***glutWireTorus***, ***glutSolidTorus***), чайника (***glutWireTeapot***, ***glutSolidTeapot***). Здесь же имеются команды для рисования правильных многогранников: тетраэдра (***glutWireTetrahedron***, ***glutSolidTetrahedron***), куба (***glutWireCube***, ***glutSolidCube***), октаэдра (***glutWireOctahedron***, ***glutSolidOctahedron***), додекаэдра (***glutWireDodecahedron***, ***glutSolidDodecahedron***), икосаэдра (***glutWireIcosahedron***, ***glutSolidIcosahedron***).

Библиотека GLU содержит удобные *команды для задания перспективы и моделирования камеры*.

Перспектива.

При выполнении предыдущей лабораторной работы использовались параллельная (***glOrtho***) и центральная (***glFrustum***) проекции для задания перспективы.

В библиотеке GLU для этой цели служит функция ***gluPerspective***, имеющая следующие 4 параметра: ***fovy*** – вертикальный угол конуса обзора; ***aspect*** – отношение размера области графического вывода по горизонтали к размеру по вертикали; ***near*** – расстояние до передней плоскости отсечения; ***far*** – расстояние до задней плоскости отсечения:

```
gluPerspective(fovy, aspect, near, far)
```

Для задания области отсечения графического вывода используется функция ***glViewPort***.

Моделирование камеры.

В OpenGL точка обзора по умолчанию располагается в центре координат. Направление зрения – вдоль оси ***Z*** противоположно ее направлению (т.е. все, что рисуется, мы видим из центра координат).

Для моделирования камеры можно использовать видовые преобразования (матрица ***MODELVIEW***):

```
glMatrixMode(GL_MODELVIEW);   //видовая матрица
glLoadIdentity();             //установим единичную матрицу
glTranslatef(0, 0, -10);      //сдвиг по оси Z вглубь экрана
glRotatef(27, 1, 0, 0);       //поворот вокруг оси X
glRotatef(-19, 0, 1, 0);      //поворот вокруг оси Y
```

Теперь наблюдатель (камера) как бы находится на расстоянии 10 единиц от центра координат (и значит от плоскости изображения), причем повернута относительно вертикали на 27 градусов, и развернута в горизонтальной плоскости на 19 градусов. При этом камера всегда смотрит в центр координат.

Для моделирования камеры, которая направлена в произвольную точку пространства и расположена в заданном месте, удобно использовать функцию библиотеки GLU – ***gluLookAt***, например:

```
glMatrixMode(GL_MODELVIEW);   //видовая матрица
glLoadIdentity();             //установим единичную матрицу
gluLookAt (x, y, z,           //местоположение камеры
           xp, yp, zp,        //камера смотрит в эту точку
           0, 1, 0);          //направление вектора «вверх»
```

Функция имеет 9 параметров: первые три параметра определяют координаты местоположения камеры; вторая тройка параметров задает координаты точки, в которую нацелена камера; последние три параметра – направление вектора «вверх» – используются для поворотов изображения в плоскости проецирования. Например:

```
gluLookAt (x, y, z, xp, yp, zp, 1, 0, 0);
```

определяет поворот изображения на 90 градусов (обычно Вы фотографируете, держа фотоаппарат горизонтально, но иногда бывает удобнее развернуть его вертикально, чтобы сфотографировать, например, высокое дерево – такой поворот и имеется в виду в примере).

Направление вектора «вверх» может быть любым, но не должно быть параллельно вектору нормали к плоскости наблюдения. Удобным является выбор вектора «вверх» параллельным оси ***Y***, т.е. (0, 1, 0).

Задания к лабораторной работе.

Задание 1. Установите перспективу в соответствии с вариантом.

Задание 2. Пользуясь командами библиотеки glu изобразите рисунок в соответствии с вариантом.

Задание 3. Установите камеру таким образом, чтобы она была направлена в центр Вашего изображения.

Задание 4. Реализуйте перемещение камеры относительно объекта:

- Объезд камеры вокруг объекта;
- Приближение/удаление камеры относительно объекта;

Задание 5. Реализуйте движение объекта относительно камеры:

- Движение объекта влево/вправо относительно камеры;
- Вращение объекта вокруг своей оси.

Варианты к заданиям лабораторной работы.

1 вариант.

Угол вертикального обзора 45 градусов.

Расстояние до передней плоскости отсечения: 3
Расстояние до задней плоскости отсечения: 10
Объект «Космический корабль»

2 вариант.

Угол вертикального обзора 40 градусов.
Расстояние до передней плоскости отсечения: 5
Расстояние до задней плоскости отсечения: 10
Объект «Солнечная система»

3 вариант.

Угол вертикального обзора 30 градусов.
Расстояние до передней плоскости отсечения: 6
Расстояние до задней плоскости отсечения: 12
Объект «Тележка»

4 вариант.

Угол вертикального обзора 25 градусов.
Расстояние до передней плоскости отсечения: 3
Расстояние до задней плоскости отсечения: 8
Объект «Настольная лампа»

5 вариант.

Угол вертикального обзора 35 градусов.
Расстояние до передней плоскости отсечения: 2
Расстояние до задней плоскости отсечения: 9
Объект «Ваза»

Дополнительные задания.

1. С помощью квадрик-объектов изобразите баскетбольную корзину и мячик, попадающий в эту корзину из некоторой точки, а затем падающий вниз.
2. С помощью квадрик-объектов изобразите выпуклую поверхность планеты и комету, летящую по параболе к горизонту планеты (комета должна иметь хвост).
3. С помощью квадрик-объектов изобразите мыльные пузыри, выдуваемые из кольца (пузыри растут и разлетаются в случайных направлениях, а затем лопаются).

Контрольные вопросы.

1. Что понимается под термином квадрик-объект?
2. Какие библиотеки OpenGL предоставляют возможности использования квадрик-объектов? Какие квадрик-объекты вам известны?
3. Какие действия необходимо описать в программе, чтобы квадрик-объект был изображен?
4. Как определяется положение квадрик-объекта в пространстве?
5. Сколько фигур может быть построено на базе данного квадрик-объекта?
6. Какие стили заполнения квадрик-объектов вам известны? Какая команда используется для изменения стиля заполнения квадрик-объекта?
7. Каким образом устанавливается перспектива с помощью библиотеки GLU?
8. Какая команда библиотеки GLU служит для моделирования камеры? Какие параметры имеет эта команда и каково их назначение?
9. Каково назначение вектора «верх»?
10. Каким образом моделируется камера в отсутствии библиотеки GLU?

Лабораторная работа № 6. Освещение.

Цель работы.

Знакомство с возможностями библиотеки OpenGL для задания параметров освещения сцены; анализ различных вариантов освещения.

Порядок выполнения работы.

Перед выполнением заданий данной лабораторной работы необходимо повторить сведения, относящиеся к цветовой модели RGB ([7] – лекция 4), изучить теоретический материал, связанный с различными уровнями визуализации ([7] – лекция 15), с учетом свойств материала объектов при их изображении; различия методов Гуро и Фонга ([7] – лекция 16).

Необходимые теоретические сведения.

OpenGL вычисляет цвет каждого пикселя в результирующей, отображаемой сцене, содержащейся в буфере кадра. Часть этого расчета зависит от того, какое освещение используется в сцене, и как объекты сцены отражают и поглощают свет. В качестве примера этому вспомните, что океан (или море, или река – вообще говоря, любой водоем) имеет различный цвет в солнечный или в облачный день. Присутствие света или облаков определяет, будет ли вода выглядеть ярко синей или грязно зеленой. По правде говоря, большинство объектов вообще не выглядят трехмерными, если они не освещены. Скажем, неосвещенная сфера ничем не отличается от двумерного круга. Освещение любого объекта зависит от двух факторов. Первый - это материал, из которого сделан объект. Второй - это свет, которым он освещен.

Модель освещения.

OpenGL рассчитывает свет и освещение так, как будто свет может быть разделен на красный, зеленый и синий компоненты. Таким образом, источник света характеризуется количеством красного, зеленого и синего света, которое он излучает, а материал поверхности характеризуется долями красного, зеленого и синего компонентов, которые он отражает в различных направлениях. Уравнения освещенности в OpenGL являются всего лишь аппроксимациями, но зато они работают достаточно хорошо и могут быть вычислены относительно быстро.

В модели освещения OpenGL свет исходит от нескольких источников, которые могут включаться и выключаться индивидуально. Часть света обычно исходит из какого-либо определенного направления или позиции, часть распределена по всей сцене. Например, если вы включите лампочку в комнате, большая часть света будет исходить от нее, но часть света падает на поверхности предметов в комнате после того, как он отразился от одной, двух, трех или более стен. Считается, что этот многократно отраженный свет (называемый фоновым светом) распределен настолько сильно, что не существует никакого способа определить его исходное направление, однако он исчезает при выключении определенного источника света.

Наконец, в сцене может также присутствовать общий фоновый свет, у которого нет никакого конкретного источника, как будто он был отражен столько раз и распределен так сильно, что его оригинальный источник установить невозможно.

В модели OpenGL эффект от источника света присутствует только если есть поверхности поглощающие или отражающие свет. Считается, что каждая поверхность состоит из материала с несколькими свойствами. Материал может излучать свой собственный свет (например, фара автомобиля), он может распределять некоторое количество входящего света во всех направлениях, также он может отражать часть света в определенном направлении (например, зеркало или другая блестящая поверхность).

В модели освещения OpenGL предполагается, что освещение может быть разделено на 4 компонента: фоновое (ambient), диффузное (diffuse), зеркальное (specular) и исходящее (эмиссионное – emissive). Все 4 компонента рассчитываются независимо и только затем суммируются.

Фоновый, диффузный, зеркальный и исходящий свет.

Фоновое излучение – это свет, который настолько распределен средой (предметами, стенами и так далее), что его направление определить невозможно – кажется, что он исходит отовсюду. Лампа дневного света имеет большой фоновый компонент, поскольку большая часть света, достигающего вашего глаза, сначала отражается от множества поверхностей. Уличный фонарь имеет маленький фоновый компонент: большая часть его света идет в одном направлении, кроме того, поскольку он находится на улице, очень небольшая часть света попадает вам в глаз после того, как отразится от других объектов. Когда фоновый свет падает на поверхность, он одинаково распределяется во всех направлениях.

Диффузный компонент – это свет, идущий из одного направления, таким образом, он выглядит ярче, если падает на поверхность под прямым углом, и выглядит тусклым, если касается ее всего лишь вскользь. Однако, когда он падает на поверхность, он распределяется одинаково во всех направлениях, то есть его яркость одинакова вне зависимости от того, с какой стороны вы смотрите на поверхность. Вероятно, любой свет, исходящий из определенного направления или позиции, имеет диффузный компонент.

Зеркальный свет исходит из определенного направления и отражается от поверхности в определенном направлении. При отражении хорошо сфокусированного лазерного луча от качественного зеркала происходит почти 100 процентное зеркальное отражение. Блестящий метал или пластик имеет высокий зеркальный компонент, а кусок ковра или плюшевая игрушка – нет. Вы можете думать о зеркальности как о том, насколько блестящим выглядит материал.

Помимо фонового, диффузного и зеркального цветов, материалы могут также иметь исходящий цвет, имитирующий свет, исходящий от самого объекта. В модели освещения OpenGLисходящий свет поверхности добавляет объекту интенсивности, но на него не влияют никакие источники света, и он не производит дополнительного света для сцены в целом.

Хотя источник света излучает единое распределение частот, фоновый, диффузный и зеркальный компоненты могут быть различны. Например, если в вашей комнате красные стены и белый свет, то этот свет, отражаясь от стен будет скорее красным, чем белым (несмотря на то, что падающий на стену свет – белый). OpenGL позволяет устанавливать значения красного, зеленого и синего независимо для каждого компонента света.

Цвет материала и света.

Модель освещения OpenGL делает допущение о том, что цвет материала зависит от долей падающего красного, зеленого и синего света, которые он отражает. Например, максимально красный шар отражает весь красный свет, который на него падает и поглощает весь зеленый и синий. Если вы посмотрите на такой мяч под белым светом (состоящим из одинакового количества красного, зеленого и синего), весь красный свет отразится, и вы увидите красный мяч. Если смотреть на мяч при красном свете, он также будет выглядеть красным. Если, однако, посмотреть на него под зеленым светом, он будет выглядеть черным (весь зеленый свет поглотится, а красного нет, то есть никакой свет отражен не будет).

Также как и свет, материалы имеют разные фоновый, диффузный и зеркальный цвета, которые задают реакцию материала на фоновый, диффузный и зеркальный компоненты света. Фоновый цвет материала комбинируется с фоновым компонентом всех источников света, диффузный цвет с диффузным компонентом, а зеркальный с зеркальным. Фоновый и диффузный цвета задают видимый цвет материала, они обычно близки, если не эквивалентны. Зеркальный цвет обычно белый или серый. Он задет цвет блика на объекте (то есть он может совпадать с зеркальным компонентом источника света).

Цветовые компоненты, задаваемые для источников света, означают совсем не то же самое, что для материалов. Для источника света число представляет собой процент от полной интенсивности каждого цвета. Если R, G и B – величины цвета источника света все равны 1.0, свет будет максимально белым. Если величины будут равны 0.5, свет все равно останется белым, но лишь с половиной интенсивности (он будет казаться серым). Если R=G=1 и B=0 (полный красный, полный зеленый, отсутствие синего), свет будет желтым.

Для материалов числа соответствуют отраженным пропорциям этих цветов. Так что, если для материала R=1, G=0.5 и B=0, этот материал отражает весь красный свет, половину зеленого и совсем не отражает синего. Другими словами, если обозначить компоненты источника света как (LR, LG, LB), а компоненты материала как (MR, MG, MB) и проигнорировать все остальные взаимодействия, то свет, который поступит в глаз можно определить как (LR–MR, LG–MG, LB–MB).

Похожим образом, если два источника света с характеристиками (R1, G1, B1) и (R2, G2, B2) направлены в глаз, OpenGL сложит компоненты: (R1+R2, G1+G2, B1+B2). Если какая-либо из сумм будет больше 1 (соответствуя цвету, который нельзя отобразить), компонент будет урезан до 1.

Включение фонового освещения.

По умолчанию освещение отключено. Включается оно командой glEnable(GL_LIGHTING). Без освещения работать практически невозможно. Сфера всегда будет показываться как круг, а конус - как круг или треугольник. Если монотонное тело у равномерно освещено, то увидеть его рельеф невозможно. Поэтому необходимо использовать источники света.

Когда освещение разрашаное, то можно устанавливать фоновую освещенность. По умолчанию, значение фоновой освещенности равно (0.2, 0.2, 0.2, 1). Фоновое освещение устанавливается с помощью функции glLightModel. Если повысить фоновое освещение до (1,1,1,1), т.е. до максимума, то включать источники света не понадобится. Их действие просто не будет заметно, т.к. объект уже максимально освещен

Добавление в функцию main вызов следующей функции позволит установить фоновое освещение.

```
float ambient[4] = {0.5, 0.5, 0.5, 1};
...
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambient);
```

Задание параметров материала.

Материал может рассеивать, отражать и излучать свет. Свойства материала устанавливаются при помощи функции

```
glMaterialfv(GLenum face, GLenum pname, GLtype* params)
```

Первый параметр определяет грань, для которой устанавливаются свойства. Он может принимать одно из следующих значений:

GL_BACK задняя грань
GL_FONT передняя грань
GL_FRONT_AND_BACK обе грани

Второй параметр функции glMaterialfv определяет свойство материала, которое будет установлено, и может принимать следующие значения.

GL_AMBIENT рассеянный свет
GL_DIFFUSE тоже рассеянный свет, пояснения смотри ниже
GL_SPECULAR отраженный свет
GL_EMISSION излучаемый свет
GL_SHININESS степень отраженного света
GL_AMBIENT_AND_DIFFUSE оба рассеянных света

Ambient и diffuse переводятся на русский как "рассеянный". Разница между ними не очень понятна. Я использую только GL_DIFFUSE. Третий параметр определяет цвет соответствующего света, кроме случая GL_SHININESS. Цвет задается в виде массива из четырех элементов - RGBA. В случае GL_SHININESS params указывает на число типа float, которое должно быть в диапазоне от 0 до 128.

Создание, позиционирование и включение одного или более источников света.

В разговорной речи источники света часто именуются просто лампами. Все параметры лампы задаются с помощью функции glLight, которая имеет следующий прототип:

```
void glLight[if][v](
            GLenum light,
            GLenum pname,
            GLfloat param)
```

Буква v в названии функции определяет, используется ли векторная версия команды.

Первый аргумент определяет номер лампы. Его можно задавать двумя способами. Первый - явно указать GL_LIHGTi, где GL_LIGHTi предопределено в файле gl.h следующим образом:

```
/* LightName */
#define GL_LIGHT0          0x4000
#define GL_LIGHT1          0x4001
#define GL_LIGHT2          0x4002
#define GL_LIGHT3          0x4003
#define GL_LIGHT4          0x4004
#define GL_LIGHT5          0x4005
#define GL_LIGHT6          0x4006
#define GL_LIGHT7          0x4007
```

Второй способ - GL_LIGHT0 + i, где i номер лампы. Такой способ используется, когда существует необходимость в цикле изменять параметры ламп. Второй аргумент определяет имя параметра, а третий его значение. С помощью следующих функций разрешаем освещение и включаем нулевую лампу.

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Массивы pos и dir содержат координаты местоположения лампы и направления, куда она светит. Массив dir содержит три координаты - x,y,z. Массив pos - четыре, назначение четвертого мне не очень ясно. Если его значение отличается от нуля, то изображение вполне логичное получается. Если же он ноль, то получается что-то непотребное. По умолчанию цвет всех источников света кроме GL_LIGHT0 – черный.

```
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
```

После настройки параметров источника света его нужно активизировать командой glEnable().

Параметры источника света.

В аргументе param задается значение или значения, в которые следует установить характеристику pname. Если используется векторная версия команды, param представляет собой вектор величин, а если невекторная, то param – одно единственное значение. Невекторная версия команды может использоваться только для указания параметров, чье значение выражается одним числом.

Значения параметров по умолчанию.

Имена параметров	Значения по умолчанию	Смысл
GL_AMBIENT	(0.0,0.0,0.0,1.0)	Интенсивность фонового света
GL_DIFFUSE	(1.0,1.0,1.0,1.0) или (0.0,0.0,0.0,1.0)	Интенсивность диффузного света (значение по умолчанию для 0-го источника - белый свет, для остальных - черный)
GL_SPECULAR	(1.0,1.0,1.0,1.0) или (0.0,0.0,0.0,1.0)	Интенсивность зеркального света (значение по умолчанию для 0-го источника - белый свет, для остальных - черный)
GL_POSITION	(0.0,0.0,1.0,0.0)	Положение источника света (x,y,z,w)
GL_SPOT_DIRECTION	(0.0,0.0,-1.0)	Направление света прожектора (x,y,z)
GL_SPOT_EXPONENT	0.0	Концентрация светового луча
GL_SPOT_CUTOFF	180.0	Угловая ширина светового луча
GL_CONSTANT_ATTENUATION	1.0	Постоянный фактор ослабления
GL_LINEAR_ATTENUATION	0.0	Линейный фактор ослабления
GL_QUADRATIC_ATTENUATION	0.0	Квадратичный фактор ослабления

Значения по умолчанию для GL_DIFFUSEи GL_SPECULAR в таблице различаются для GL_LIGHT0 и других источников света (GL_LIGHT1, GL_LIGHT2, ...). Для параметров GL_DIFFUSEи GL_SPECULAR источника света GL_LIGHT0 значение по умолчанию – (1.0, 1.0, 1.0, 1.0). Для других источников света значение тех же параметров по умолчанию – (0.0, 0.0, 0.0, 1.0).

Цвет.

OpenGL позволяет ассоциировать с каждым источником света три различных параметра, связанных с цветом: GL_AMBIENT, GL_DIFFUSE и GL_SPECULAR. Параметр GL_AMBIENT задает RGBA интенсивность фонового света, который каждый отдельный источник света добавляет к сцене.

```
Glfloat      light_ambient[]={0.0,0.0,1.0,1.0};
             glLightfv(GL_LIGHT0,GL_AMBIENT,light_ambient);
```

Параметр GL_DIFFUSE наверное наиболее точно совпадает с тем, что вы привыкли называть «цветом света». Он определяет RGBA цвет диффузного света, который отдельный источник света добавляет к сцене. По умолчанию для GL_LIGHT0 параметр GL_DIFFUSE равен (1.0, 1.0, 1.0, 1.0), что соответствует яркому белому свету. Значение по умолчанию для всех остальных источников света (GL_LIGHT1, GL_LIGHT2, ..., GL_LIGHT7) равно (0.0, 0.0, 0.0, 0.0).

Параметр GL_AMBIENT влияет на цвет зеркального блика на объекте. В реальном мире на объектах вроде стеклянной бутылки имеется зеркальный блик соответствующего освещению цвета (часто белого). Для создания эффекта реалистичности нужно установить

GL_SPECULAR в то же значение, что и GL_DIFFUSE. По умолчанию GL_SPECULAR равно (1.0, 1.0, 1.0, 1.0) для GL_LIGHT0 и (0.0, 0.0, 0.0, 0.0) для остальных источников.

Позиция и ослабление.

Можно выбрать, следует ли считать источник света расположенным бесконечно далеко от сцены или близко к ней. На источники света первого типа ссылаются как на направленные (directional): эффект от бесконечно далекого расположения источника заключается в том, что все лучи его света могут считаться параллельными к моменту достижения объекта. Примером реального направленного источника света может служить солнце. Источники второго типа называются позиционными (positional), поскольку их точное положение на сцене определяет их эффект и, в частности, направление из которого идут лучи. Примером позиционного источника света является настольная лампа.

Направленный источник:

```
Glfloat light_position[]={1.0,1.0,1.0,0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Как видно, для параметра GL_POSITION задается вектор из четырех величин (x, y, z, w). Если последняя величина w равна 0, соответствующий источник света считается направленным, и величины (x, y, z) определяют его направление. Это направление преобразуется видовой матрицей. По умолчанию параметру GL_POSITION соответствуют значения (0, 0, 1, 0), что задает свет, направленный вдоль отрицательного направления оси z. (Заметьте, что никто не запрещает вам создать свет, направленный в (0, 0, 0), однако такой свет не даст должного эффекта).

Если значение w не равно 0, свет является позиционным, и величины (x, y, z) задают местоположение источника света в однородных объектных координатах. Это положение преобразуется видовой матрицей и сохраняется в видовых координатах. Кроме того, по умолчанию позиционный свет излучается во всех направлениях, но вы можете ограничить распространение света, создав конус излучения, определяющий прожектор.

Задания лабораторной работе.

Задание 1. Построить 2 сферы с координатами и радиусом, заданными в таблице 1 для каждого варианта.

Задание 2. Задать фоновое освещение с параметрами, заданными в таблице 1 для каждого варианта.

Задание 3. Задать параметры материала для сфер. Одна из сфер должна иметь свойство GL_DIFFUSE , а вторая GL_SHININESS. Цвет и степень блеска заданы в таблице 1.

Задание 4. Создать источники света, количество которых различно для каждого варианта. Параметры источников света задаются произвольно, однако они должны быть различны для разных ламп.

Варианты к заданию лабораторной работы.

№	Координаты центра сферы	Радиус сферы	Параметры фонового освещения	Цвет, степень блеска	Кол-во ламп
1	1,0,4	3	0.2,0.2,0.3,1	красный, 45	2
2	2,7,9	5	0.1,0.4,0.1,1	синий, 79	3
3	9,4,10	2.9	0.5,0.3,0.5,1	желтый, 93	1
4	-15,9,2.2	6	0.3,0.6,0.2,1	зеленый, 39	2
5	8,3.8,-1.7	3.4	0.2,0.7,0.1,1	розовый, 122	1

Дополнительные задания.

1. В установленном графическом окне построить куб, задать для различных его граней различные параметры материала: GL_AMBIENT, GL_DIFFUSE,

GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_AMBIENT_AND_DIFFUSE.

2. Установить диффузный источник света, освещающий белую неблестящую сферу. По нажатию клавиши на клавиатуры лампа должна изменять свой цвет от красного до фиолетового в порядке следования цветов в спектре.
3. Установить диффузный источник света, освещающий белую неблестящую сферу. При нажатии клавиш курсора лампа должна передвигаться в указанном направлении.
4. Построить пирамиду, освещаемую двумя источниками света. Красным и зеленым. При нажатии клавиши “G” или “R” интенсивность соответственно зеленого или красного источника света увеличивается на 10%. При достижении интенсивности в 100%, она должна автоматически падать до 10%.
5. Построить куб, освещенный вращающимся вокруг него источником света. При нажатии клавиши «вправо» источник света начинает двигать с большей скоростью. При нажатии клавиши «влево», скорость движения лампы должна уменьшаться.

Контрольные вопросы.

1. Какая модель освещения используется в OpenGL?
2. Какие виды освещения можно установить в программе OpenGL?
3. Чем определяется цвет поверхности объекта?
4. Какие команды используются для установки источника освещения?
5. Сколько источников освещения можно определить в программе OpenGL?
6. Как влияют свойства материала на видимый цвет объекта?
7. В чем заключаются различия между направленными и позиционными источниками освещения?
8. В чем заключается эффект ослабления?

Лабораторная работа № 7. Текстуры: режимы фильтрации, режимы взаимодействия текстуры с накладываемым объектом, автоматическая генерация текстурных координат.

Цель работы.

Дать представление о возможностях наложения текстур на поверхности объектов средствами OpenGL.

Необходимые теоретические сведения.

Текстура – одномерное, двухмерное или трехмерное изображение, которое имеет множество ассоциированных с ним параметров, определяющих каким образом будет производиться наложение изображения на поверхность. Проще говоря, текстура – это изображение, накладываемое на поверхность. С физической точки зрения текстура - массив данных, например цветовых, световых или цветовых и альфа. Каждый элемент этого массива называется ***текселем. Текстурные координаты*** – координаты текселя, назначаемого вершине.

Общепринятые имена для текстурных координат – s, t, r, q.

Для использования текстур необходимо выполнить следующую последовательность действий:*

1. Подготовка изображения для использования в текстуре
    - Загрузка изображения из файла.
2. Создание текстуры
    1) Генерация уникального имени текстуры*

2) Выполнение операции первичного связывания*
3) Связывание изображения с текстурой*
4) Установка режимов фильтрации текстуры
5) Установка параметров взаимодействия объекта, с накладываемой текстурой.
6) Установка режима мозаичного наложения текстуры.
7) Автоматическая генерация текстурных координат, эффект отражения.

3. Использование текстуры
   1) Выбор текстуры для использования (вторичное связывание).
   2) Согласование геометрических и текстурных координат
   3) Создание эффекта отражения с помощью текстур.

* - Звездочкой обозначены обязательные пункты.

Подготовка изображения для использования в текстуре.

Первый этап выполняется при инициализации OpenGL.

OpenGL имеет свой формат хранения изображений. Подготовительный этап заключается в том, что бы преобразовать изображение в формат OpenGL. В библиотеке GLAUX имеется функция, которая считывает изображение из DIB или BMP файла и автоматически преобразовывает его к формату OpenGL.

AUX_RGBImageRec* auxDIBImageLoad(strFile);

Функция возвращает указатель на структуру типа AUX_RGBImageRec, которая хранит само изображение и его основные параметры – ширину и высоту.

```
struct AUX_RGBImageRec
{
    unsigned char   *data;    //  изображение
    GLint           sizeX;    //  ширина    изображения
    GLint           sizeY;    //  высота    изображения
};
```

Передаваемое значение – путь к файлу с изображением.

Пример использования функции auxDIBImageLoad – загрузка изображения из файла sky.bmp в объект структуры ImageSky.

```
AUX_RGBImageRec   *ImageSky;
ImageSky = auxDIBImageLoad(“sky.bmp”);
```

Создание текстуры

Этот этап, как и подготовка изображения, обычно выполняется во время инициализации OpenGL.

Для того, что бы разрешить текстуррирование используется функция

glEnable(GL_TEXTURE_2D);

Единственный параметр этой функции определяет тип текстур, используемых в программе. Обычно этот параметр принимает значение GL_TEXTURE_2D

1) Генерация уникального имени текстуры

Каждая текстура, которая используется в OpenGL должна иметь свое уникальное имя – идентификатор типа GLuint. Для автоматической генерации этого идентификатора служит функция

void glGenTextures(GLsizei n, GLLuint *textures);

Передаваемые параметры:

n	количество генерируемых идентификаторов
textures	массив, куда будут записаны сгенерированые идентификаторы

Пример генерации уникальных идентификаторов в первом случае для одной текстуры, во втором – для пяти текстур.

```
GLuint     SingleTex;
```

```
Gluint    ArrayTex[5];
glGenTextures(1, &SingleTex);// Генерация одного идентификатора
glGenTextures(5, &ArrayTex); // Генерация пяти идентификаторов
```

2) Первичное связывание

Перед тем как задать конкретное изображение текстуре необходимо произвести операцию первичного связывания. При этом создается новый объект текстуры, и переданное имя текстуры ассоциируется с ним. Реализуется это посредством вызова функции

void glBindTexture (GLenum target, GLuint texture)

target задает целевой объект с которым связывается текстура, может принимать значения GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D.

texture идентификатор текстуры

Пример первичного связывания создает текстурный объект и сопоставляет ему идентификатор SingleTex.

```
glBindTexture(GL_TEXTURE_2D, SingleTex);
```

3) Связывание изображения с текстурой

Конкретное изображением ассоциирует с текстурой с помощью функции **glTexImage2D.**

Для изображения, подготовленного функцией **auxDIBImageLoad** необходимо в качестве параметра **target** использовать значение GL_TEXTURE_2D. Изображение, которые вы используете должно иметь ширину и высоту кратную степень двойки, ширина границы текстуры тоже должна быть степенью двойки. Если вы не используете границу текстуры то в качестве параметра border следует указать 0. Формат цвета пикселя - GL_RGB.

Внутренний формат – число 3. Количество уровней детализации – можно установить в 0. Ширина, высота и само изображение определяются из объекта типа AUX_RGBImageRec, подготовленного с помощью функции auxDIBImageLoad.

```
void glTexImage2D(
  GLenum target,          // GL_TEXTURE_2D
  GLint level,            // Уровень детализации
  GLint internalformat,   // Внутренний формат
  GLsizei width,          // ширина изображения
  GLsizei height,         // высота изображения
  GLint border,           // ширина границы
  GLenum format,          // формат цвета пикселя
  GLenum type,            // тип данных в массиве pixels
  const GLvoid *pixels);  // указатель на изображение
```

Пример Создаем текстуру с идентификатором SingleTex и изображением ImageSky.

```
glTexImage2D(GL_TEXTURE_2D, 0, 3,
             ImageSky->sizeX, TypeGL_RGBImage->sizeY,
             0, GL_RGB, GL_UNSIGNED_BYTE, TypeGL_RGBImage->data);
```

4) Режимы фильтрации.

При различии размеров текстуры и размеров объекта, на который она накладывается, используются различные режимы фильтрации. Режим без сглаживания, линейная фильтрация, мип-мэп наложение.

а) Использование *режима без сглаживания* позволяет добиться максимальной производительности, однако изображение при этом будет выглядеть неестественно. Этот режим рекомендовано применять только на очень слабых машинах.

Для задания режимов фильтрации используется функция **glTexParameteri**

void glTexParameteri(GLenum target, GLenum pname, GLint param);

Первый параметр определяет тип текстуры (в нашем случае GL_TEXTURE_2D)

Второй параметр - определяет устанавливаемые параметры

При установке параметров уменьшающего фильтра (размер текстуры больше размера поверхности на которую она накладывается) этот параметр принимает значение GL_TEXTURE_MIN_FILTER, а при установке параметров увеличивающего фильтра - GL_TEXTURE_MAX_FILTER.
Последний параметр определяет значение параметра pname
При использовании фильтра без сглаживания этот параметр принимает значение GL_NEAREST
Пример задания уменьшающего и увеличивающего фильтра без сглаживания

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
```

б) Использование *линейной фильтрации* позволяет заметно улучшить качество изображения, однако производительность при этом снижается, так как этот режим требует определенных вычислений. В этом режиме увеличения/уменьшения используется взвешенная сумма массива текселей 2x2, которые находятся ближе всего к отображаемому пикселю.
Вызов функции **glTexParameteri** для установки режима фильтрации аналогичен режиму без сглаживания, за исключением последнего параметра – он принимает значение GL_LINEAR
Пример задания уменьшающего и увеличивающего фильтра линейной фильтрации

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
```

в) *Мип-мэп наложение*. При использовании этого метода OpenGL создает набор картинок разного размера (от исходного до размера в 1x1 тексель). Во время работы программы автоматически выбирается картинка наиболее подходящего размера. При использовании этого способа возможно несоблюдение условия того, что высота и ширина изображения должна быть степенью двойки. Так же существует различие в связывании изображения с текстурой (см. «**3 Связывание изображения с текстурой**»). Этот способ фильтрации дает наилучшее изображение, однако является наиболее затратным с точки зрения производительности.
Пример использования связывания изображения с текстурой с последующим включением режима фильтрации - мип-мап наложение.

```
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, TypeGL_RGBImage->sizeX,
                  TypeGL_RGBImage->sizeY, GL_RGB, GL_UNSIGNED_BYTE,
                  TypeGL_RGBImage->data);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
```

Параметры, передаваемые в функцию gluBuild2Dmipmaps аналогичны параметрам передаваемым в функцию glTexImage2D.
5) Установка параметров взаимодействия объекта, с накладываемой текстурой.
В OpenGL существует возможность смешивать цвет накладываемой текстуры с исходным цветом объекта. Для этого необходимо вызвать функцию **glTexEnv**.
void glTexEnv(Glenum target, Glenum pname, GLType param);
taget первый параметр - определяет конфигурацию текстуры и должен быть равен GL_TEXTURE_ENV.
pname - определяет символическое имя конфигурации текстуры. Должен быть равен GL_TEXTURE_ENV_MODE.
param – символическая константа, для которой допустимы значения GL_MODULATE, GL_DECALL, GL_BLEND.

При использовании GL_MODULATE цвет текстуры как бы сливается с цветом объекта, при режиме DECALL, цвет объекта, соответствует цвету накладываемой текстуры.
Пример. В первом случае результирующий цвет объекта складывается, из цвета объекта и цвета текстуры, во втором результирующий цвет – цвет текстуры.

```
// Слияние цветов
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODUALTE);
// Цвета текстуры
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
```

6) Установка режима мозаичного(тайлового) наложения текстуры.
Иногда возникает необходимость в повторении текстуры, например при отрисовке кирпичной стены. OpenGL предоставляет такую возможность, достаточно только выполнить два условия:
а) Разрешить режим повторения текстуры. Режим повторения задается отдельно для вертикальной и для горизонтальной оси:

```
GlTexParametri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
GlTexParametri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Последний параметр определяет режим наложения текстуры при значении GL_REPEAT текстура будет повторятся, при наложении на объект. Если вместо последнего параметра – GL_REPEAT использовать GL_CLAMP, текстура будет растягиваться при наложении на накладываемую поверхность.При
б) Задать координаты текстуры > 1.0, например 4.0 в этом случае текстура будет повторена на накладываемой поверхности 4 раза.
7) Автоматическая генерация текстурных координат, эффект отражения.
С помощью текстурирования в OpenGL возможно реализовать эффект отражения окружающих предметов от блестящих поверхностей. Ниже приведен пример задания режима зеркального отражения. В двух строчках определяются параметры генерации текстурных координат – в данном случае текстурные координаты вычисляются в сфере вокруг позиции образа. При использовании автоматической генерации текстурных координат необходимо в качестве фильтра использовать мип-мапы.

```
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, TypeGL_RGBImage->sizeX,
                  TypeGL_RGBImage->sizeY, GL_RGB, GL_UNSIGNED_BYTE,
                  TypeGL_RGBImage->data);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
```

Использование текстуры.

Этот этап выполняется непосредственно в функции прорисовки сцены.
1) Выбор текстуры для использования (вторичное связывание)
Для использования текстуры в программе ее надо сделать активной этот процесс и называется вторичным связыванием. Вторичное связывание, как и первичное, выполняется с помощью функции

void glBindTexture (GLenum target, GLuint textureName);

target — определяет тип текстуры (GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D). Этот параметр должен совпадать с тем, который был при операции первичного связывания.

textureName идентификатор текстуры**.**

Пример вторичного связывания – делает активной текстуру с идентификаторм SingleTex.

```
glBindTexture(GL_TEXTURE_2D, SingleTex);
```

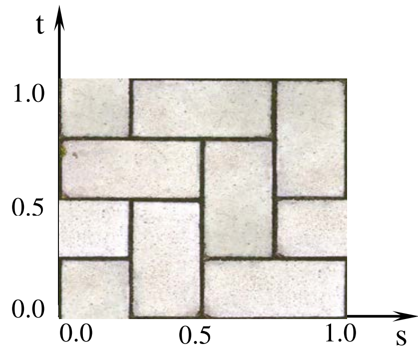
2) Согласование геометрических и текстурных координат
При прорисовке объекта, на который накладывается текстура, необходимо согласовать текстурные координаты с геометрическими. Например при наложении текстуры на

прямоугольный объект вы должны отобразить правый верхний угол текстуры в правый верхний угол объекта, правый нижний угол текстуры в правый нижний угол объекта и т.д. Текстурные координаты являются частью данных, связанных с вершинами многоугольника. Для установки текстурных координат используется функция **glTexCoord2f(GLfloat s, GLLfloat t);**

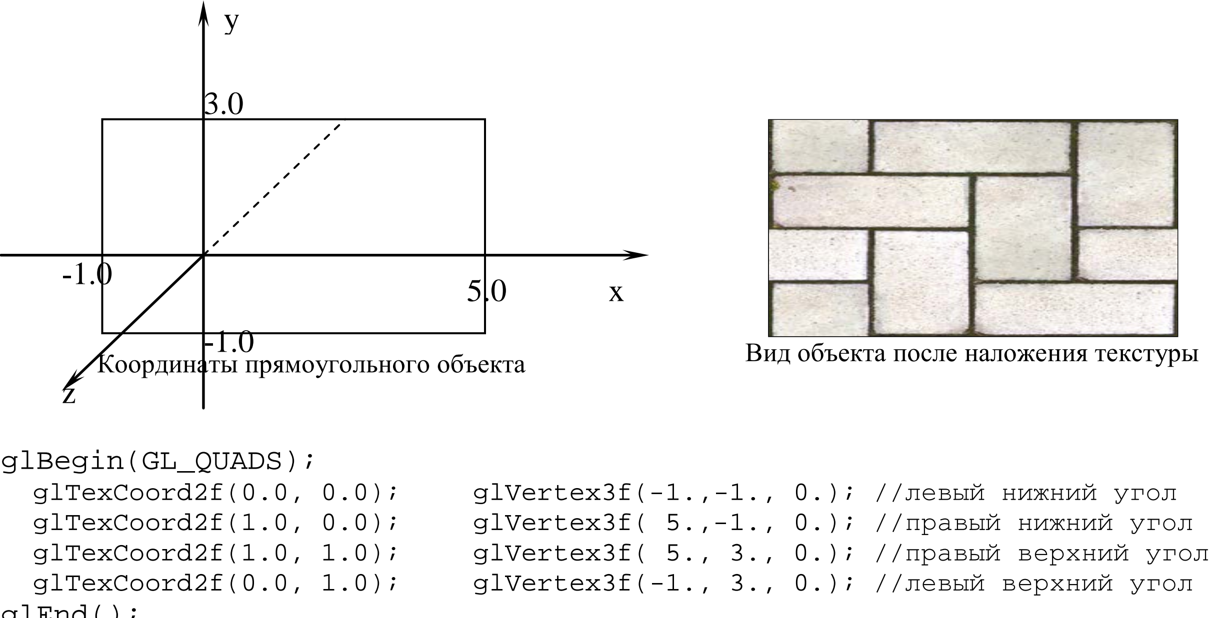
s - определяет горизонтальную текстурную координату

t - определяет вертикальнуютекстурную координату

Система текстурных координат имеет следующий вид:



Пример наложения текстуры с растяжением на прямоугольный объект.



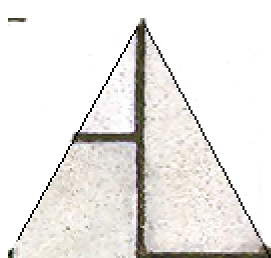
Координаты прямоугольного объекта

Вид объекта после наложения текстуры

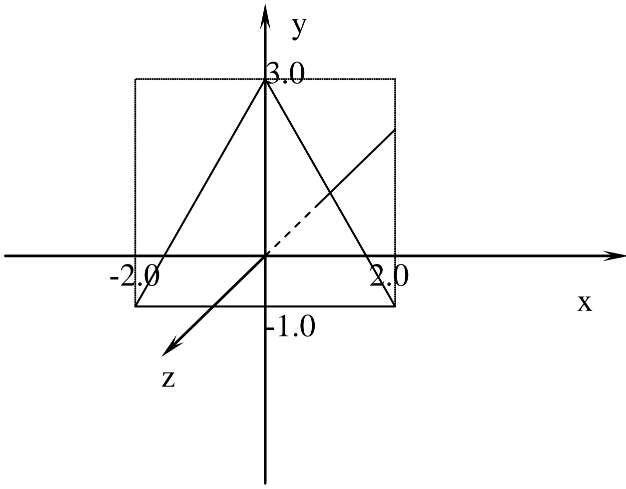
```
glBegin(GL_QUADS);
  glTexCoord2f(0.0, 0.0);     glVertex3f(-1.,-1., 0.); //левый нижний угол
  glTexCoord2f(1.0, 0.0);     glVertex3f( 5.,-1., 0.); //правый нижний угол
  glTexCoord2f(1.0, 1.0);     glVertex3f( 5., 3., 0.); //правый верхний угол
  glTexCoord2f(0.0, 1.0);     glVertex3f(-1., 3., 0.); //левый верхний угол
glEnd();
```

Пример наложения текстуры на треугольник.

В данном примере используется только левая нижняя часть текстуры



Вид объекта после наложения текстуры

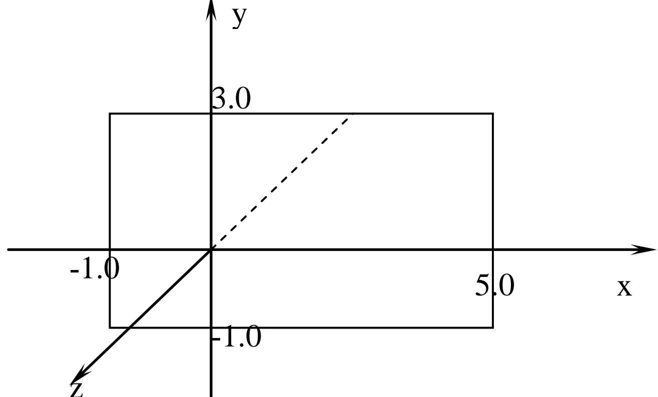



```
glBegin(GL_TRIANGLES);
  glTexCoord2f(0.0,  0.0);   glVertex3f(-2.,-1., 0.); //левая нижняя вершина
  glTexCoord2f(0.5,  0.0);   glVertex3f( 2.,-1., 0.); //правая нижняя вершина
  glTexCoord2f(0.25, 0.5);   glVertex3f( 0., 3., 0.); //верхняя вершина
glEnd();
```

Пример мозаичного(тайлового) покрытия текстурой

Координаты прямоугольника совпадают с координатами прямоугольника в первом примере. Текстура будет повторена по горизонтальной оси 3 раза, по вертикальной – 2 раза. Для возможности повторения текстуры этот режим должен быть активирован с помощью команд

```
GlTexParametri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
GlTexParametri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```



Вид объекта после наложения текстуры

```
glBegin(GL_QUADS);
  glTexCoord2f(0.0, 0.0);    glVertex3f(-1.,-1., 0.); //левый нижний угол
  glTexCoord2f(3.0, 0.0);    glVertex3f( 5.,-1., 0.); //правый нижний угол
  glTexCoord2f(3.0, 2.0);    glVertex3f( 5., 3., 0.); //правый верхний угол
  glTexCoord2f(0.0, 2.0);    glVertex3f(-1., 3., 0.); //левый верхний угол
glEnd();
```

Пример создания с помощь текстур эффекта отражения.

Необходимо разрешить OpenGL использовать автоматически сгенерированные координаты. Параметры, с помощью которых эти координаты будут генерироваться, хранятся в текстурном объекте, созданном на втором этапе (см. пункт 2.7).

```
// 1 - разрешаем автогенерацию текстурных координат по оси S
// 2 - разрешаем автогенерацию текстурных координат по оси T
```

```
glEnable(GL_TEXTURE_GEN_S); // 1
glEnable(GL_TEXTURE_GEN_T); // 2
glBegin(GL_QUADS);
      glVertex3f(-1.,-1., 0.); //левый нижний угол
      glVertex3f( 5.,-1., 0.); //правый нижний угол
      glVertex3f( 5., 3., 0.); //правый верхний угол
      glVertex3f(-1., 3., 0.); //левый верхний угол
glEnd();
// 3 - запрещаем автогенерацию текстурных координат по оси S
// 4 - запрещаем автогенерацию текстурных координат по оси T
glEnable(GL_TEXTURE_GEN_S); // 3
glEnable(GL_TEXTURE_GEN_T); // 4
```

Задания к лабораторной работе и порядок выполнения.

1. Подготовьте изображение к использованию в качестве текстуры. В качестве изображения выбирается любое изображение, имеющееся на компьютере. При отсутствии таковых изображение создается в графическом редакторе.

2. Создайте текстурный объект.

3. Настройте параметры текстурных объектов, в соответствии с вариантом.

Номер варианта	Увеличивающий фильтр	Уменьшающий фильтр	Взаимодействие с поверхностью
1	линейная фильтрация	без сглаживания	смешивание
2	без сглаживания	линейная фильтрация	без смешивания
3	мип-мап наложение	без сглаживания	без смешивания
4	линейная фильтрация	линейная фильтрация	смешивание
5	мип-мап наложение	линейная фильтрация	без смешивания

4. Изобразите объект в соответствии с вариантом

Номер варианта	Вид объекта	Номер варианта	Вид объекта
1		4	
2		5	
3			

5. Наложите текстуры на созданный геометрический объект.

6. Реализуйте изменение параметров текстуры при нажатии клавиш в соответствии с вариантом.

Номер варианта	**Вносимые изменение**
1	Установка уменьшающего фильтра в режим линейной фильтрации. Вращение объекта вокруг оси X.
2	Разрешение тайлового покрытия текстурой. Изменения количества повторения текстуры. Вращение объекта вокруг оси Z.
3	Изменение режима взаимодействия с поверхностью в режим смешивания. Вращение текстуры вокруг оси Y.
4	Установка увеличающего и уменьшающего фильтра. Вращение текстуры вокруг оси Z.
5	Создание эффекта зеркального отражения. Вращение текстуры вокруг осей X, Y.

Контрольные вопросы.

1. Что понимается под термином «текстура»?
2. Каков смысл использования текстур при визуализации? Какому уровню визуализации соответствует текстурирование?
3. Что такое «тексель»?
4. Что представляют собой текстурные координаты?
5. Какие действия необходимо выполнить для наложения текстуры на объект?
6. Какие режимы фильтрации используются при наложении текстур?
7. В чем заключается первичное связывание?
8. Каким образом осуществляется связывание изображения с текстурой?
9. Каким образом при помощи текстур может быть достигнут эффект отражения?
10. Почему необходимо согласование геометрических и текстурных координат? Как это достигается?

ПРИЛОЖЕНИЯ.

Приложение 1. Минимальный код программы OpenGL на C++.

```
#include <windows.h>          // Заголовочный файл для Windows
#include <gl\gl.h>            // Заголовочный файл для OpenGL32 библиотеки
#include <gl\glu.h>           // Заголовочный файл для GLu32 библиотеки
#include <gl\glaux.h>         // Заголовочный файл для GLaux библиотеки
    static HGLRC hRC;         // Постоянный контекст рендеринга
static HDC hDC;               // Приватный контекст устройства GDI
   BOOL    keys[256];         // Массив для процедуры обработки клавиатуры
       GLvoid InitGL(GLsizei Width, GLsizei Height)  // Вызвать после
создания
                                                     // окна GL
{
   glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // Очистка экрана в черный цвет
}
    GLvoid ReSizeGLScene(GLsizei Width, GLsizei Height)
{
      if (Height==0)          // Предотвращение деления на ноль,
                              //если окно слишком мало
            Height=1;

      glViewport(0, 0, Width, Height);   // Сброс текущей области вывода
}
   GLvoid DrawGLScene(GLvoid)
{
         glClear(GL_COLOR_BUFFER_BIT);    // Очистка экрана
// Здесь создается рисунок
         glPointSize(2);
       glBegin(GL_POINTS);
       glColor3f(1,0,0);
       glVertex2f(-0.45,-0.4); // первая точка

         glColor3f(0,1,0);
       glVertex2f(0.4,0.4);   // вторая точка

         glColor3f(0,0,1);    //третья точка
       glVertex2f(-0.35,0.4);
      glEnd();

         glLineWidth(3);

      glBegin(GL_LINE_STRIP);  // ломаная линия
         glColor3f(0.7,0.3,0);
         glVertex2f(-0.10,0);
         glVertex2f(1,0.13);
         glColor3f(0,1,0);
         glVertex2f(-0.15,0.33);
         glColor3f(0,0,1);
         glVertex2f(-0.12,0.35);
      glEnd();
// здесь закончилось создание рисунка
}

 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
SuitablePixelFormat.","Error",MB_OK|MB_ICONERROR);
                    PostQuitMessage(0);
                    // Это сообщение говорит, что программа должна завершиться
                    break;      // Предтовращение повтора кода
              }
     if(!SetPixelFormat(hDC,PixelFormat,&pfd))
              {
                    MessageBox(0,"Can't Set
ThePixelFormat.","Error",MB_OK|MB_ICONERROR);
                    PostQuitMessage(0);
                    break;
              }
    hRC = wglCreateContext(hDC);
        if(!hRC)
              {
        MessageBox(0,
                   "Can't Create A GLRenderingContext.",
                   "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);
}
 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 дескриптора 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 WindowClass.",
                "Error",MB_OK|MB_ICONERROR);
      return FALSE;
      }
 hWnd = CreateWindow("OpenGL WinClass",
                    "First OpenGL program",    // 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;
      }
 if(!hWnd)
      {
      MessageBox(0,"Window Creation Error.","Error",MB_OK|MB_ICONERROR);
            return FALSE;
```

```
	}

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
	}
}
```

Приложение 2. Минимальный код программы для использования OpenGL в программе на Delphi.

В приложении приводятся три варианта программ, позволяющих использовать библиотеку OpenGL для графического вывода. Примеры взяты с диска к книге Краснова М.В. «OpenGL в проектах Delphi».

1) Оконное приложение.

(часть 1, пример 20)

```
{**********************************************************************}
program GLmin;
uses
  Forms,
  Unit1 in 'Unit1.pas' {frmGL};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TfrmGL, frmGL);
  Application.Run;
end.
{**********************************************************************}
unit Unit1;
interface
uses
  Windows, Messages, Forms, Classes, Controls, ExtCtrls, ComCtrls,
  StdCtrls, Dialogs, SysUtils,
  OpenGL;
type
  TfrmGL = class(TForm)
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    hrc: HGLRC;  // ссылка на контекст воспроизведения
  end;
var
  frmGL: TfrmGL;

implementation
{$R *.DFM}
{============================================Рисование картинки}
procedure TfrmGL.FormPaint(Sender: TObject);
begin
 wglMakeCurrent(Canvas.Handle, hrc);
 glClearColor (0.5, 0.5, 0.75, 1.0); // цвет фона
 glClear (GL_COLOR_BUFFER_BIT);      // очистка буфера цвета
 //glBegin(GL_LINES);
 //  glVertex(-0.5,-0.5);
 //  glVertex(0.5,0.5);
 //glEnd;
 wglMakeCurrent (0, 0);
end;
{============================================Формат пикселя}
procedure SetDCPixelFormat (hdc : HDC);
var
 pfd : TPixelFormatDescriptor;
 nPixelFormat : Integer;
begin
 FillChar (pfd, SizeOf (pfd), 0);
 nPixelFormat := ChoosePixelFormat (hdc, @pfd);
 SetPixelFormat (hdc, nPixelFormat, @pfd);
end;
{============================================Создание формы}
procedure TfrmGL.FormCreate(Sender: TObject);
begin
 SetDCPixelFormat(Canvas.Handle);
 hrc := wglCreateContext(Canvas.Handle);
end;
{====================================Конец работы приложения}
procedure TfrmGL.FormDestroy(Sender: TObject);
begin
 wglDeleteContext(hrc);
end;
end.
{*******************************************************************}
```

2) Консольное приложение.

(часть 1, пример 21).

```
{*******************************************************************}
program OpenGL_min;
uses
```

```
  Messages, Windows, OpenGL;
const
  AppName = 'OpenGL_Min';
Var
  Window : HWnd;
  Message : TMsg;
  WindowClass : TWndClass;
  dc : HDC;
  hrc : HGLRC;          // контекст воспроизведения OpenGL
  MyPaint : TPaintStruct;
// Процедура заполнения полей структуры PIXELFORMATDESCRIPTOR
procedure SetDCPixelFormat (hdc : HDC);
var
 pfd : TPixelFormatDescriptor; // данные формата пикселей
 nPixelFormat : Integer;
Begin
 With pfd do begin
  nSize := sizeof (TPixelFormatDescriptor); // размер структуры
  nVersion := 1;                       // номер версии
  dwFlags := PFD_DRAW_TO_WINDOW or PFD_SUPPORT_OPENGL; // множество
битовых флагов, определяющих устройство и интерфейс
  iPixelType := PFD_TYPE_RGBA; // режим для изображения цветов
  cColorBits := 16;              // число битовых плоскостей в каждом буфере цвета
  cRedBits := 0;                 // число битовых плоскостей красного в каждом буфере RGBA
  cRedShift := 0;                // смещение от начала числа битовых плоскостей красного
                                 //в каждом буфере RGBA
  cGreenBits := 0;           // число битовых плоскостей зелёного в каждом буфере RGBA
  cGreenShift := 0;              // смещение от начала числа битовых плоскостей зелёного
                                 //в каждом буфере RGBA
  cBlueBits := 0;            // число битовых плоскостей синего в каждом буфере RGBA
  cBlueShift := 0;               // смещение от начала числа битовых плоскостей синего
                                 //в каждом буфере RGBA
  cAlphaBits := 0;           // число битовых плоскостей альфа в каждом буфере RGBA
  cAlphaShift := 0;              // смещение от начала числа битовых плоскостей альфа
                                 //в каждом буфере RGBA
  cAccumBits := 0;           // общее число битовых плоскостей в буфере аккумулятора
  cAccumRedBits := 0;        // число битовых плоскостей красного в буфере аккумулятора
  cAccumGreenBits := 0;      // число битовых плоскостей зелёного в буфере аккумулятора
  cAccumBlueBits := 0;       // число битовых плоскостей синего в буфере аккумулятора
  cAccumAlphaBits := 0;      // число битовых плоскостей альфа в буфере аккумулятора
  cDepthBits := 32;          // размер буфера глубины (ось z)
  cStencilBits := 0;         // размер буфера трафарета
  cAuxBuffers := 0;          // число вспомогательных буферов
  iLayerType := PFD_MAIN_PLANE;// тип плоскости
  bReserved := 0;            // число плоскостей переднего и заднего плана
  dwLayerMask := 0;          // игнорируется
  dwVisibleMask := 0;        // индекс или цвет прозрачности нижней плоскости
  dwDamageMask := 0;         // игнорируется
 end;
nPixelFormat := ChoosePixelFormat (hdc, @pfd);  // запрос системе - поддерживается ли
                                                //выбранный формат пикселей
 SetPixelFormat (hdc, nPixelFormat, @pfd);      // устанавливаем формат пикселей в
```

```
                                                    //контексте устройства
End;

function WindowProc (Window : HWnd; Message, WParam : Word;
        LParam : LongInt) : LongInt; stdcall;
Begin
  WindowProc := 0;
  case Message of
  wm_Create:
      begin
      dc := GetDC (Window);
      SetDCPixelFormat (dc);          // установить формат пикселей
      hrc := wglCreateContext (dc);   // создаёт контекст воспроизведения OpenGL
      ReleaseDC (Window, dc);
      end;
  wm_Paint:
      begin
      dc := BeginPaint (Window, MyPaint);
      wglMakeCurrent (dc, hrc);     // устанавливает текущий контекст воспроизведения
{****** ЗДЕСЬ РАСПОЛАГАЮТСЯ КОМАНДЫ РИСОВАНИЯ OpenGL*********}
      glClearColor (0.85, 0.75, 0.5, 1.0); // определение цвета фона
      glClear (GL_COLOR_BUFFER_BIT);       // установление цвета фона
{*****************************************************************}
      wglMakeCurrent (dc, 0);         // перед завершением работы необходимо,
                                      // чтобы контекст никем не использовался
      EndPaint (Window, MyPaint);
      ReleaseDC (Window, dc);
      end;
  wm_Destroy :
      begin
      wglDeleteContext (hrc);         // удаление контекста воспроизведения
      DeleteDC (dc);
      PostQuitMessage (0);
      Exit;
      end;
  end; // case
  WindowProc := DefWindowProc (Window, Message, WParam, LParam);
End;
Begin
  With WindowClass do
      begin
      Style := cs_HRedraw or cs_VRedraw;
      lpfnWndProc := @WindowProc;
      cbClsExtra := 0;
      cbWndExtra := 0;
      hInstance := 0;
      hCursor := LoadCursor (0, idc_Arrow);
      lpszClassName := AppName;
      end;
  RegisterClass (WindowClass);
  Window := CreateWindow (AppName, AppName,
      ws_OverLappedWindow or ws_ClipChildren
```

```
     or ws_ClipSiBlings,              // обязательно для OpenGL
    cw_UseDefault, cw_UseDefault,
    cw_UseDefault, cw_UseDefault,
    HWND_DESKTOP, 0, HInstance, nil);
    ShowWindow (Window, CmdShow);
    UpdateWindow (Window);
  While GetMessage (Message, 0, 0, 0) do begin
    TranslateMessage (Message);
    DispatchMessage (Message);
  end;
  Halt (Message.wParam);
end.
{*********************************************************************}
```

3) Пример рисования на поверхности визуального компонента.

(пример 10, часть 2).

```
{*********************************************************************}
program Dots;
uses
  Forms,
  Unit1 in 'Unit1.pas' {frmGL};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TfrmGL, frmGL);
  Application.Run;
end.

{*********************************************************************}
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  OpenGL;
type
  TfrmGL = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    hrc: HGLRC;
  end;
var
  frmGL: TfrmGL;

implementation
{$R *.DFM}
{=======================================Перерисовка окна}
procedure TfrmGL.FormPaint(Sender: TObject);
begin
 wglMakeCurrent(Canvas.Handle, hrc);
 glViewPort (0, 0, ClientWidth, ClientHeight);       // область вывода
```

```
 glClearColor (0.5, 0.5, 0.75, 1.0);               // цвет фона
 glClear (GL_COLOR_BUFFER_BIT);                    // очистка буфера цвета
 glPointSize (20);                                 // размер точек
 glColor3f (1.0, 0.0, 0.5);                        // текущий цвет примитивов
 glBegin (GL_POINTS);                              // открываем командную скобку
  glVertex2f (-1, -1);
  glVertex2f (-1, 1);
  glVertex2f (0, 0);
  glVertex2f (1, -1);
  glVertex2f (1, 1);
 glEnd;                                            // закрываем командную скобку
 SwapBuffers(Canvas.Handle);                       // содержимое буфера - на экран
 wglMakeCurrent(0, 0);
 Canvas.Brush.Color := clGreen;
 Canvas.Ellipse (10, 10, 50, 50);
end;
{============================================Формат пикселя}
procedure SetDCPixelFormat (hdc : HDC);
var
 pfd : TPixelFormatDescriptor;
 nPixelFormat : Integer;
begin
 FillChar (pfd, SizeOf (pfd), 0);
 pfd.dwFlags  := PFD_DRAW_TO_WINDOW or PFD_SUPPORT_OPENGL or
PFD_DOUBLEBUFFER;
 nPixelFormat := ChoosePixelFormat (hdc, @pfd);
 SetPixelFormat (hdc, nPixelFormat, @pfd);
end;
{=============================================Создание формы}
procedure TfrmGL.FormCreate(Sender: TObject);
begin
 SetDCPixelFormat(Canvas.Handle);
 hrc := wglCreateContext(Canvas.Handle);
end;
{================================Завершение работы приложения}
procedure TfrmGL.FormDestroy(Sender: TObject);
begin
 wglDeleteContext(hrc);
end;
end.
{*******************************************************************}
```

Приложение 3. Ссылки на Интернет-ресурсы по использованию библиотек OpenGL в программировании графики.

http://www.opengl.org
http://gl.satel.ru
http://www.sgi.com/software/opengl

www.opengl.org.ru/ - Подборка материалов книг, документации, учебников. Примеры программ написанных с использованием OpenGL.

opengl.gamedev.ru/ - Программирование компьютерной графики средствами OpenGL. Документация, статьи, советы.

www.codenet.ru/progr/opengl/ - Минимальная программа OpenGL. Введение в OpenGL.

nehe.gamedev.net/ - главная страница компании NeHe, учебники, ресурсы.
www.rsdn.ru/article/opengl/ogltut2.xml - Учебное пособие по OpenGL.
www.citforum.ru/programming/opengl/index.shtml - книга Игоря Тарасова «OpenGL».
talk.mail.ru/forum/fido7.ru.opengl - Форумы@Mail.Ru:
Форум Стандарт визуализации OpenGL.
pmg.org.ru/nehe/ - OpenGL - уроки от NeHe: переводы по OpenGL, трехмерная графика, игры, мультимедия Translations Russian OpenGL.
www.firststeps.ru/mfc/opengl/opengl1.html - первые шаги в создании программы с OpenGL
www.bib.com.ua/cat_art28.html - руководство по написанию программ.

Библиографический список.

11. Боресков А. Графика трехмерной компьютерной игры на основе OpenGL. – М.: Диалог-МИФИ, 2005.
12. Гашников М. и др. Методы компьютерной обработки изображений. - М.: Физматлит, 2001.
13. Павлидис Т. Алгоритмы машинной графики и обработка изображений. - М.: Радио и связь. 1986.
14. Петров М. Компьютерная графика. Учебник для вузов. - СПб.: Питер, 2002.
15. Рейбоу В. Компьютерная графика. Энциклопедия. - СПб.: Питер, 2002.
16. Роджерс Д. Алгоритмические основы машинной графики. - М.: Мир, 1989.
17. Снижко Е. Компьютерная геометрия и графика: Конспект лекций. – СПб.: Изд. БГТУ, 2005.
18. Шикин Е., Боресков А. Компьютерная графика. Динамика, реалистические изображения. - М.: Диалог - Мифи, 1985.
19. Шикин Е., Боресков А. Компьютерная графика. Полигональные модели. – М.: Диалог-МИФИ, 2005.
20. Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGl. - М.: Изд. Дом «Вильямс», 2001.
21. Херн Д., Бейкер М. Компьютерная графика и стандарт OpenGL. – М.: Изд. Дом «Вильямс», 2005.

Оглавление.

ПРЕДИСЛОВИЕ......3

ЛАБОРАТОРНАЯ РАБОТА № 1. ПОДКЛЮЧЕНИЕ БИБЛИОТЕК; КОНТЕКСТ УСТРОЙСТВА, КОНТЕКСТ ВОСПРОИЗВЕДЕНИЯ; ОБЩИЙ ВИД ПРОГРАММЫ. 4

Цель работы......4
Необходимые теоретические сведения......4
Основные понятия, используемые в данной лабораторной работе: контекст устройства, контекст воспроизведения, формат пиксела......4
Порядок выполнения работы......4
Контрольные вопросы......10

ЛАБОРАТОРНАЯ РАБОТА № 2. ПРИМИТИВЫ OPENGL, ОСНОВНЫЕ ПРИЕМЫ ПОСТРОЕНИЯ ДВУМЕРНЫХ ОБЪЕКТОВ......11

Цель работы......11
Необходимые теоретические сведения......11
Командные скобки. Использование функций glBegin и glEnd......11
Аргументы функции glBegin......12
Включение и отключение режима сглаживания (антиэлайзинг)......12
Вывод точек в OpenGL......12
Линии: одиночные, ломаные, замкнутые ломаные......12
Вывод треугольников: одиночные треугольники, ленты треугольников, веера треугольников......13
Вывод четырехугольников......14
Рисование полигонов, передние и задние грани полигонов......14
Построение невыпуклых полигонов......15
Особенности режимов закрашивания для многоугольников......15
Задания к лабораторной работе......16
Варианты к заданию......16
Дополнительные задания......17
Контрольные вопросы......18

ЛАБОРАТОРНАЯ РАБОТА № 3. ИСПОЛЬЗОВАНИЕ МАССИВОВ ВЕРШИН. ПРЕОБРАЗОВАНИЯ КООРДИНАТ......18

Цель работы:......18
Порядок выполнения лабораторной работы......18
Необходимые теоретические сведения......18
Массивы вершин......18
Включение/выключение режима......18
Заполнение массива вершин......19
Команды рисования......19
Геометрические преобразования......19
Матрица моделирования......19
Преобразование переноса:......20
Преобразование поворота:......20
Преобразование масштабирования:......20
Задания к лабораторной работе......20
Варианты к заданиям......21
Дополнительные задания......21
Контрольные вопросы......21

ЛАБОРАТОРНАЯ РАБОТА № 4. ТРЕХМЕРНЫЕ ПОСТРОЕНИЯ. БУФЕР ГЛУБИНЫ. ВИДОВЫЕ ПАРАМЕТРЫ. ПАРАЛЛЕЛЬНАЯ И ПЕРСПЕКТИВНАЯ ПРОЕКЦИИ........22

Цель работы. 22
Порядок выполнения работы. 22
Необходимые теоретические сведения. 22
Трехмерные координаты. 22
Буфер глубины. 22
Проекции. 23
Видовые параметры 23
Место команд в программе. 23
Матрица проецирования. 24
Задания к лабораторной работе. 24
Варианты заданий. 24
Дополнительные задания. 26
Контрольные вопросы. 26

ЛАБОРАТОРНАЯ РАБОТА № 5. КВАДРИК-ОБЪЕКТЫ. КАМЕРА. 26

Цель работы. 26
Порядок выполнения работы. 26
Необходимые теоретические сведения. 26
Квадрик-объекты. 27
Перспектива. 28
Моделирование камеры. 29
Задания к лабораторной работе. 29
Варианты к заданиям лабораторной работы. 29
Дополнительные задания. 30
Контрольные вопросы. 30

ЛАБОРАТОРНАЯ РАБОТА № 6. ОСВЕЩЕНИЕ. 30

Цель работы. 31
Порядок выполнения работы. 31
Необходимые теоретические сведения. 31
Модель освещения. 31
Фоновый, диффузный, зеркальный и исходящий свет. 32
Цвет материала и света. 32
Включение фонового освещения. 33
Задание параметров материала. 33
Создание, позиционирование и включение одного или более источников света. 34
Параметры источника света. 34
Цвет. 35
Позиция и ослабление. 36
Задания лабораторной работе. 36
Варианты к заданию лабораторной работы. 36
Дополнительные задания. 36
Контрольные вопросы. 37

ЛАБОРАТОРНАЯ РАБОТА № 7. ТЕКСТУРЫ: РЕЖИМЫ ФИЛЬТРАЦИИ, РЕЖИМЫ ВЗАИМОДЕЙСТВИЯ ТЕКСТУРЫ С НАКЛАДЫВАЕМЫМ ОБЪЕКТОМ, АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ТЕКСТУРНЫХ КООРДИНАТ. 37

Цель работы. 37
Необходимые теоретические сведения. 37

Подготовка изображения для использования в текстуре. 38
Создание текстуры 38
Использование текстуры. 41
Пример наложения текстуры с растяжением на прямоугольный объект. 42
Пример наложения текстуры на треугольник. 42
Пример мозаичного(тайлового) покрытия текстурой 43
Пример создания с помощь текстур эффекта отражения. 43
Задания к лабораторной работе и порядок выполнения. 44

ПРИЛОЖЕНИЯ....46

Приложение 1. Минимальный код программы OpenGL на С++. 46
Приложение 2. Минимальный код программы для использования OpenGL в программе на Delphi. 49
1) Оконное приложение (пример 20, часть 1). 49
2) Консольное приложение (пример 21, часть 1). 50
3) Пример рисования на поверхности визуального компонента (пример 10, часть 2). 53
Приложение 3. Ссылки на Интернет-ресурсы по использованию библиотек OpenGL в программировании графики. 54

БИБЛИОГРАФИЧЕСКИЙ СПИСОК....56

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «КОМПЬЮТЕРНАЯ ГРАФИКА»

для обучающихся по направлению 54.03.01 «Дизайн»,
профиль «Промышленный дизайн» всех форм обучения

Составители:
Кузовкин Алексей Викторович
Суворов Александр Петрович
Золототрубова Юлия Сергеевна

Подписано в печать 04.06.2021
Формат 60x84 1/8 Бумага для множительных аппаратов
Уч.-изд. л. 3,3 Усл. печ. л. 3,0.

ФГБОУ ВО «Воронежский государственный технический университет»
396026 Воронеж, Московский просп., 14

Участок оперативной полиграфии издательства ВГТУ
396026 Воронеж, Московский просп., 14