ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)**

Кафедра вычислительных машин, комплексов, систем и сетей
Л.А. Надейкина, Н.И. Черкасова

ИНЖЕНЕРНАЯ И КОМПЬЮТЕРНАЯ ГРАФИКА

ПОСОБИЕ
по выполнению
лабораторных работ № 1, 2

для студентов II курса
направления 09.03.01 (230100)
очной формы обучения

Москва - 2015

ББК 607
Н17

Рецензент канд. техн. наук, доц. Л.А. Вайнейкис

Надейкина Л.А., Черкасова Н.И.

Н17 Инженерная и компьютерная графика: пособие по выполнению лабораторных работ № 1, 2. – М.: МГТУ ГА, 2015. – 32 с.

Данное пособие издается в соответствии с рабочей программой учебной дисциплины «Инженерная и компьютерная графика» по Учебному плану для студентов II курса направления 09.03.01 (230100) очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 09.04.15 г. и методического совета 09.04.15 г.

Подписано в печать 15.05.2015 г.

Печать офсетная	Формат 60x84/16	1,6 уч.-изд. л.
1,86 усл.печ.л.	Заказ № 2017/	Тираж 80 экз.

Московский государственный технический университет ГА
125993 Москва, Кронштадтский бульвар, д.20
Редакционно-издательский отдел
125493 Москва, ул. Пулковская, д.6а

1 ВВЕДЕНИЕ

Многие годы в программистском сообществе идет дискуссия о преимуществах использования того или иного интерфейса для создания графики. Главные соперники в этой области - библиотеки OpenGL и Direct3D. Индустрия до сих пор так и не сделала однозначный выбор в пользу той или иной библиотеки.

При получении изображения в реальном времени на современном аппаратном ускорителе, качество и скорость рисования не зависят от применяемой библиотеки: Direct3D и OpenGL обладают схожими возможностями.

Таким образом, по поддержке аппаратных функций OpenGL и Direct3D, в общем, эквиваленты. В OpenGL новые функции доступны через механизм расширений, а в Direct3D они появляются только в новых версиях.

Объем кода, необходимого для написания простой программы на Direct3D, весьма велик (варьируется от 200 до 800 строк). В OpenGL все существенно проще - для решения такой же задачи необходимо менее 50 строк кода.

Серьезным достоинством OpenGL является прежде всего то, что это "открытый стандарт". Изменения в OpenGL предлагаются, обсуждаются и утверждаются представителями различных компаний.

Что касается Direct3D, то здесь ситуация прямо противоположная. Только Microsoft может вносить какие-либо изменения в библиотеку. Иначе говоря, именно Microsoft в конечном итоге определяет все пути развития библиотеки.

OpenGL

OpenGL (*Open Graphics Library*, открытая графическая библиотека), без сомнения, является превалирующим индустриальным программным интерфейсом (API) для разработки 2D и 3D-приложений.

OpenGL широко используется в системах автоматизированного проектирования, виртуальной реальности, научном и информационном моделировании, в движках компьютерных игр и многих других областях.

Библиотека проста в использовании, интуитивна, переносима.

Приложения OpenGL могут достаточно легко переноситься на практически любую платформу, присутствующую на сегодняшнем рынке.

OpenGL по своему дизайну является библиотекой, независимой от платформы и операционной системы.

OpenGL, прежде всего, это оптимизированная, высокопроизводительная графическая библиотека для отрисовки графики, и есть множество графических карт с акселераторами и специализированных 3D карт, которые выполняют примитивы OpenGL на аппаратном уровне.

Главной особенностью можно считать то, что она поддерживается практически всеми операционными системами, в отличие от Direct3D, которая написана только под Windows.

Для достижения аппаратной независимости OpenGL, команды для управления окнами, равно как и команды для получения входных данных от пользователя, были исключены.

Это может показаться серьезным недостатком при использовании OpenGL, но как мы увидим позже, есть возможность скомбинировать OpenGL с другими гибкими библиотеками, которые будут заботиться о задачах управления окнами и о получении входных данных от пользователя.

Более того, OpenGL не предоставляет никаких команд для описания сложных моделей (молекул, самолетов, строений, птиц и т.д.). В OpenGL вы найдете только наиболее примитивные геометрические объекты (точки, линии и многоугольники). Разработчику предоставляется самому сконструировать свои собственные модели, основываясь на этих нескольких простых примитивах. Есть также относящиеся к OpenGL библиотеки, которые предоставляют более сложные модели, и каждый может использовать эти библиотеки для построения собственных.

Ниже приведены некоторые возможности, реализованные в OpenGL:

- ***Геометрические примитивы*** позволяют конструировать математическое описание объекта. В настоящее время примитивами являются: точки, линии, многоугольники, изображения и битмапы (побитовые карты).
- ***Цветное кодирование*** в формате *RGBA (Red-Green-Blue-Alpha)* или в режиме индексированных цветов.
- ***Просмотр и моделирование*** позволяют размещать объекты на трехмерной сцене, передвигать камеры вокруг этого объекта и выбирать требуемую точку просмотра при отрисовке сцены.
- ***Текстуры*** помогают привнести реализм в модели при помощи отрисовки изображений реально выглядящих поверхностей поверх многоугольников модели.
- ***Освещение материалов*** - неотъемлемая часть всей трехмерной графики. OpenGL предоставляет в распоряжение команды для подсчета цвета любой точки при заданных свойствах материала и источников света в помещении.
- ***Двойная буферизация*** позволяет устранить мелькание при анимации. Каждое последующее изображение в анимационном ряду строится в отдельном буфере памяти и отображается только по завершении рендеринга. Рендеринг является процессом, благодаря которому получается изображение по модели с помощь компьютерной программы.

- ***Anti-aliasing*** сглаживает неровные края линий, отображаемых на компьютерном дисплее. Ломанные линии часто появляются при низком графическом разрешении. Anti-aliasing является стандартной техникой в компьютерной графике и заключается в изменении цвета и интенсивности точек вблизи линии для уменьшения искусственных зигзагов.
- ***Gouraud тени*** - это техника применения сглаженных теней трехмерного объекта для достижения тонких различий цветов на некоторой поверхности.
- ***Z-буферизация*** отслеживает Z-координату трехмерного объекта. Z-буфер используется для определения приближения к объекту. Он так же является определяющим при удалении заслоненных поверхностей.
- ***Атмосферные эффекты***, такие как туман, дым, дымка делают изображения, созданные компьютером, более реалистичными. Без атмосферных эффектов изображения зачастую выглядят нереально резкими и очерченными. *Fog* - это термин, который на самом деле описывает алгоритм моделирования тумана, дыма, загрязнений или просто эффект присутствия воздуха, придавая глубину изображению.
- ***Alpha смешивание*** использует значение *Alpha* (значение диффузии материала) как составляющую цветового кода *RGBA*, позволяя комбинировать цвет обрабатываемого фрагмента с цветом точек, которые уже хранятся в буфере. Представьте, например, отрисовку прозрачного голубого стекла перед красной коробкой. *Alpha* смешивание позволяет моделировать прозрачность стекла таким образом, что часть коробки, которая видна сквозь стекло будет иметь сиреневый оттенок.
- ***Трафаретные планы*** ограничивают отрисовку определенной областью экрана.
- ***Списки отображения*** позволяют сохранять команды отрисовки в некотором списке для дальнейшего рендеринга. При правильном применении, списки отображения могут заметно повысить производительность рендеринга.
- ***Полиномные вычисления*** дают возможность использовать неравномерные B-сплайны. Они помогают рисовать плавные кривые через несколько контрольных точек, исключая необходимость сохранять все промежуточные точки.
- ***Обратная связь, выделение и выбор*** дают возможность создавать приложения, которые позволяют пользователю выбирать область экрана или отдельный объект, изображенный на экране.
- ***Растровые примитивы*** (битмапы и прямоугольники из точек)
- ***Точечные операции***

- ***Преобразования:*** повороты, масштабирование, трансляция, трехмерные перспективы и так далее.

Как отмечалось ранее, для того, чтобы сделать OpenGL полностью переносимой и платформенно-независимой, было необходимо пожертвовать всеми командами, которые имеют отношение к системе управления окнами, например - открыть окно, закрыть окно, изменить размеры окна, перерисовать окно, прочитать позицию курсора; пришлось пожертвовать и командами, связанными с входными устройствами, например - чтения с клавиатуры. Все эти операции очень сильно зависят от операционной системы.

Для нивелирования данных проблем OpenGL обычно используется в связке с другими библиотеками.

Рассмотрим использование OpenGL в среде проектирования Visual Studio в связке с SDL2.

SDL

SDL – Simple DirectMedia Layer, это свободная всеобъемлющая кроссплатформенная мультимедийная библиотека, реализующая единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода для широкого спектра платформ. Данная библиотека активно используется при написании кроссплатформенных мультимедийных программ (в основном игр).

Дословно – «Простой директмедиа слой» - является кроссплатформенной библиотекой разработанной, чтобы обеспечить низкоуровневый доступ к аудио, клавиатуре, мыши, джойстику и аппаратной графике через OpenGL или DirectX.

Библиотека состоит из нескольких подсистем, таких как Video, Audio, CD-ROM, Joystick и Timer. В дополнение к этой базовой низкоуровневой функциональности, существует ряд стандартных библиотек, предоставляющих дополнительную функциональность:

SDL image — поддержка различных растровых форматов

SDL mixer — функции для организации сложного аудио, в основном, сведение звука из нескольких источников

SDL net — поддержка сетевых функций

SDL ttf — поддержка шрифтов TrueType

SDL rtf — отрисовка текста в формате RTF.

Самая последняя версия библиотеки доступна на официальном сайте библиотеки: https://www.libsdl.org/
Ссылки загрузки доступны на сайте:
https://www.libsdl.org/download-2.0.php

Лабораторные работы выполняются в среде Microsoft Visual Studio (2010 или 2013) как консольные приложения.

Использование OpenGL в связке с SDL2 в среде разработки Microsoft Visual Studio

1. Откройте Visual Studio и перейдите к созданию проекта. В качестве языка укажите "Visual C++ ", а в качестве типа проекта "пустой проект".
2. В разделе файлов исходного кода создайте единственный файл с расширением cpp и напишите в нем функцию main. Убедитесь, что код компилируется.
3. Убедившись, что проект создан правильно, скачайте с официального сайта библиотеки SDL архив "SDL2-devel-2.0.3-VC.zip" (http://libsdl.org/release/SDL2-devel-2.0.3-VC.zip) и распакуйте его в папку с проектом и вернитесь к Visual Studio.
4. Подключение SDL-библиотеку к проекту:
    1) Выберите ваш проект (НЕ РЕШЕНИЕ), откройте с помощью контекстного меню пункт "Свойства".
    2) Перейдите в "Свойства конфигурации" -> "C/C++" -> "Общее" и добавьте в "Дополнительные каталоги включаемых файлов" папку "SDL2-2.0.X\include", которую вы распаковали из архива.
    3) Перейдите в "Свойства конфигурации" -> "Компоновщик" -> "Общее" и добавьте в "Дополнительные каталоги библиотек" папку "SDL2-2.0.X\lib\xXX", которую вы распаковали из архива. Битность xXX определяется параметрами проекта системы и самой Visual Studio, может быть равной x86 или x64, но вероятно, это x86.
    4) Перейдите в "Свойства конфигурации" -> "Компоновщик" -> "Ввод", и добавьте в "Дополнительные зависимости" строки:
    SDL2.lib
    SDL2main.lib
    opengl32.lib
    glu32.lib
    5) Перейдите в "Свойства конфигурации" -> "Компоновщик" -> "Система", и выберите в "Подсистема" параметр "Консоль (/SUBSYSTEM:CONSOLE) "
    6) Скопируйте файл SDL2.dll из папки "SDL2-2.0.X\lib\xXX", которую вы распаковали из архива, в папку с проектом в целевую папку сборки (папка, в которой создаётся

исполняемый EXE-файл, который является результатом сборки вашей программы).

Для использования возможностей библиотеки SDL подключите соответствующий заголовочный файл и убедитесь, что программа компилируется:

```
#include  <SDL2/SDL.h>
#undef main
int main (int argc, char* argv[]) {
      return 0;
}
```

SDL представляет простой способ инициализации OpenGL окна. После инициализации самой библиотеки SDL, необходимо определить в программе версию используемой библиотеки OpenGL, вызвать функцию создания окна и создать OpenGL контекст, который представляет некоторую структуру для хранения всех данных, относящихся к визуализации. Когда приложение закрывается OpenGL контекст разрушается.

Таким образом, использование SDL обусловлено тем, что создание окна и контекста OpenGL не является частью спецификации OpenGL. Такие действия обеспечивает, например, библиотека SDL.

Рассмотрим ряд установочных функций.

Для инициализации SDL используется функция SDL_Init:

```
int SDL_Init (Unit32 flags),
```

где флаги должны указывать на то, какие части этой библиотеки для нас нужны среди следующих:

SDL_INIT_VIDEO	видео
SDL_INIT_AUDIO	аудио
SDL_INIT_CDROM	cd привод
SDL_INIT_TIMER	таймер
SDL_INIT_JOYSTICK	джойстик
SDL_INIT_EVERYTHING	все части

Функция возвращает положительный результат при успехе и значение меньше 0 при ошибке. В качестве аргумента стоит передать список подсистем, которые собираемся использовать. Например, SDL_INIT_EVERYTHING для инициализации всех подсистем, или SDL_INIT_VIDIO, для инициализации видеосистемы.

Так, если нам потребуется использовать в своём приложении видео и звук, инициализация будет выглядеть так:

```
SDL_Init (SDL_INIT_VIDEO | SDL_INIT_AUDIO);
```

Все свои функции SDL будет загружать с динамической библиотеки (в Windows — это SDL.dll, в Linux – это SDL.so), которая должна быть расположена в директории нашего приложения, иначе программа её не найдёт.

Не забывайте проверять на возврат положительного результата. Ведь для инициализации мы используем функцию, которая возвращает определенное значение. Причем, если это значение меньше нуля, то тогда библиотека не загрузилась, и мы просто обязаны выйти из программы (при этом выдав сообщение по поводу ошибки), иначе продолжение работы приведёт к ошибке.

Можно использовать функцию SDL_GetError (), которая возвращает строку с информацией об ошибке.

Например,

```
#include <iostream>
#include <SDL.h>
int main ( int argc, char* argv[]){
        if ( SDL_Init (SDL_INIT_EVERYTHING)<0)
        std::cerr<< "Не инициализировалась SDL\n"<< SDL_GetError ()
       <<std::endl;
        system("pause");
        exit(1);
        SDL_Delay (1000);
        SDL_Quit ();
        return 0;
}
```

Когда мы завершили работу, то перед выходом из программы нужно выгрузить библиотеку SDL из памяти с помощью функции SDL_Quit ().

Для завершения работы не всего приложения, а какой-либо её части можно использовать: SDL_QuitSubSystem(<флаги>).

Для установки параметров OpenGL используется функция:

```
int SDL_GL_ SetAttribute (SDL_GLattr attr, int value);
```

Нас интересуют параметры SDL_GL_CONTEXT_MAJOR_VERSION и SDL_GL_CONTEXT_MINOR_VERSION, через которые мы сможем работать с имеющейся версией OpenGL*:*

Устанавливаем используемую версию OpenGL (2.1)*:*

```
SDL_GL_SetAttribute (SDL_GL_CONTEXT_MAJOR_VERSION, 2);
```

SDL_GL_SetAttribute (SDL_GL_CONTEXT_MINOR_VERSION, 1);

Для создания окна в библиотеке SDL имеется функция:

SDL_Window*initWindow (**const char***title, **int** w, **int** h, Unit32 flags);

в которую надо передать размеры и положение окна, а также учесть дополнительные параметры флаги. Нас интересуют такие флаги как SDL_WINDOW_OPENGL и SDL_WINDOW_FULLSCREEN, которые позволяют создать OpenGL окно в полноэкранном режиме.

Контекст устройства по окну создается путем вызова функции:

SDL_GLContext SDL_GL_CreateContext(SDL_ Window*window);

Далее можно приступать к рисованию

2 ЛАБОРАТОРНАЯ РАБОТА № 1

Геометрические преобразования на плоскости

2.1 Цель лабораторной работы

Целью работы является изучение математического аппарата, реализующего элементарные и сложные геометрические преобразования на плоскости, получение практических навыков разработки графических программа на языке C++ для формирования изображения на экране монитора динамически перемещаемых двумерных объектов.

2.2 Краткие теоретические сведения

Матрицы элементарных геометрических преобразований

1. Перенос

где a1 – константа переноса вдоль оси x;
a2 – константа переноса вдоль оси y.

2. Масштабирование

где sx – коэффициент масштабирования вдоль оси x;
sy – коэффициент масштабирования вдоль оси y;

3. Поворот на угол α

4. Отражение

- относительно оси X

- относительно оси Y

5. Результирующая матрица поворота на угол α вокруг произвольной точки с координатами (a, b)

$G = T^{-1}(a, b) * R(\alpha) * T(a, b) =$

$$\begin{vmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ -a*\cos(\alpha)+b*\sin(\alpha)+a & -a*\sin(\alpha)-b*\cos(\alpha)+b & 1 \end{vmatrix}$$

Преобразование угла из градусов в радианы:

α(радианов) = (π/180)*α(градусов) или
α(радианов) = α(градусов)*arctg(1) / 45.0

2.3 Задание на выполнение лабораторной работы

Разработать программу, реализующую операцию динамического вращения квадрата (равностороннего треугольника, трапеции, параметры фигуры вводятся с клавиатуры):
Вариант 1. Относительно одной из его вершин против часовой стрелки
Вариант 2. Относительно произвольной точки координат, лежащей вне квадрата, по часовой стрелке.
Вариант 3. Промасштабированного относительно центра в два раза; вращение осуществляется относительно центра квадрата по часовой стрелке.

Вариант 4. Промасштабированного относительно центра в ½ раза; вращение осуществляется относительно центра квадрата против часовой стрелки.

При этом предусмотреть в программе:

а) реализацию функции putPixel () на языке C++ и OpenGL;

б) реализацию требуемой в задании геометрической операции двумя способами:

- с помощью последовательности элементарных геометрических преобразований;
- с использованием результирующей матрицы сложного геометрического преобразования;

в) реализацию операции поворота (или масштабирования) относительно начала координат;

г) отображение всех графических результатов на одном экране в разных частях (рисунке 1).

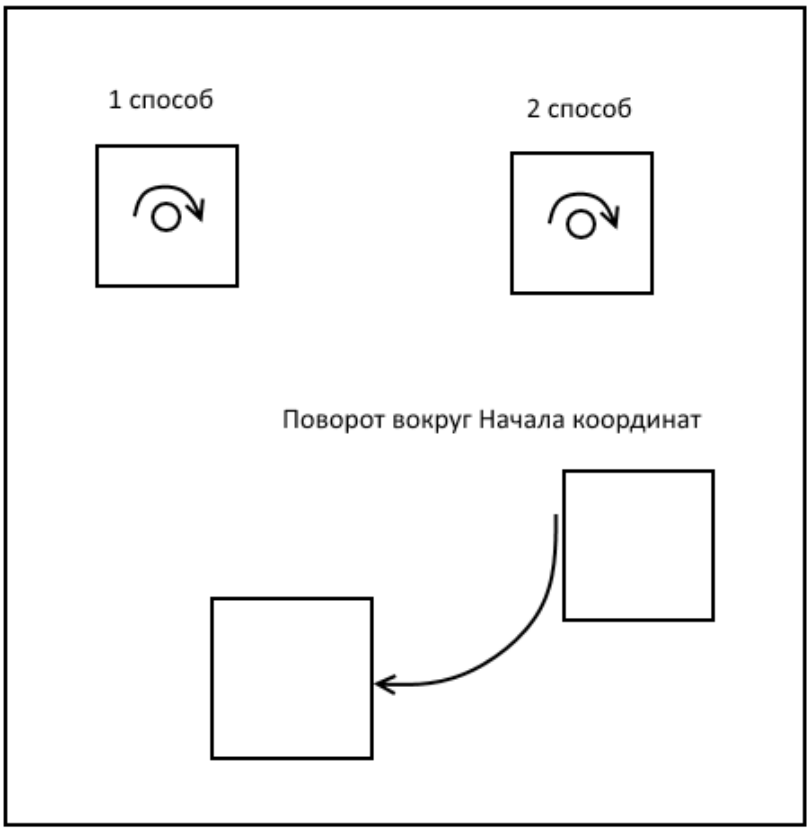

Рисунок 1 - Пример изображения графических результатов

2.4 Содержание отчёта по лабораторной работе

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.

2. Цель лабораторной работы
3. Задание на выполнение лабораторной работы.
4. Аналитическое описание решаемой задачи с иллюстрациями.
5. Алгоритмы и листинги отлаживаемой программы.
6. Графические результаты выполнения лабораторной работы.
7. Выводы по лабораторной работе.

2.5 Контрольные вопросы

1. Способы реализации сложного геометрического преобразования на плоскости.
2. Построить результирующую матрицу заданного сложного геометрического преобразования на плоскости.
3. Относительно чего выполняется поворот (масштабирование, отражение) при использовании соответствующих матриц элементарных геометрических преобразований на плоскости.
4. Смысловое значение констант, используемых в результирующей матрице выполняемого в лабораторной работе геометрического преобразования.
5. Каким образом в программе реализована динамика вращения геометрической фигуры?
6. Математическая реализация геометрического преобразования поворота (масштабирования) относительно начала координат (произвольной точки) и операции переноса (объекта, системы координат).
7. Программная реализация операции записи/чтения кода цвета пикселя, расположенного в заданной позиции экрана.
8. Назначение аппаратных составляющих видеосистемы ПК.
9. Назначение регистров графического контроллера и способы программного доступа к ним.
10. Рассчитать адрес байта видеопамяти, содержащего код цвета пикселя с заданными координатами в 16-ти цветном режиме видеоадаптера VGA с разрешением 640x480.
11. Определить номер бита в байте видеопамяти, содержащего код цвета пикселя с заданными координатами в 16-ти цветном режиме видеоадаптера VGA с разрешением 640x480.
12. Коды цветов скольких пикселей хранятся в заданном количестве байт видеопамяти в различных графических режимах.

2.6 Листинг программы

```
#include <iostream>
#include <cmath>
#include <SDL2/SDL.h>
#include <SDL2/SDL_opengl.h>
#undef main
#include <GL/GLU.h>
#ifdef _WIN32
#include <conio.h>
#else
#include <ncurses.h>
#endif
//---------------Глобальные переменные-----------------------
const int WIDTH = 800;   // ширина окна
const int HEIGHT = 600; //высота
float Xr[5], Yr[5];     //массив координат
float vx;               //абсцисса вершины
float vy;               //ордината вершины
float l;                // длина стороны квадрата
struct Rgb     // структурный тип определяющий цвет
{  double r;   //интенсивность красного цвета  от 0.0 до 1.0
   double g;   //    -//-      зелёного  цвета от 0.0 до 1.0
   double b;   //    -//-      синего цвета  от 0.0 до 1.0
};
//Определение некоторых интенсивных цветов
#define RED    {1.0, 0.0, 0.0}
#define BLUE   {0.0, 0.0, 1.0}
#define GREEN  {0.0, 1.0, 0.0}
#define YELLOW {1.0, 1.0, 0.0}
#define WHITE  {1.0, 1.0, 1.0}
#define BLACK  {0.0, 0.0, 0.0}

SDL_GLContext OpenGL_Context; // Контекст OpenGL
// --------------------Пользовательские функции-------------
void err (const char *message);
void line (int x1, int y1, int x2, int y2, Rgb colour);
void putPixel (int x, int y, Rgb colour);
void getPixel (int x, int y, Rgb &colour);
SDL_Window *initWindow (const char* title, int w, int h);
void Render ();
void initCoord ();
void drawRect (Rgb color);
void move (int,int);
void rotate (float);
void resMatrix (float,float,float,float,float);
float rad(float);
void mash (float, float);
//шаблон функции, не использующей параметр
template <typename T>
void unused(T const &) {}
```

```
//----------------------Определения функций----------------
//Функция завершения программы с ошибкой, указанной в
//параметре message
void err(const char *message){
std::cerr<<message<< std::endl;
system("pause");
exit(1);
}
//-------------------------------------------------
// Функция рисование линии между двумя заданными точками
//координаты выражены в пикселях, центр системы координат –
//левый верхний угол окна
void line (int x1, int y1, int x2, int y2, Rgb colour)
{float w=WIDTH;
 float h=HEIGHT;
//Перевод к координатам OpenGL. Ширина и высота окна равны 2-м
//единицам, центр системы координат находится в центре окна.
// Координаты по оси X и Y изменяются от -1 до 1
float nx1 = x1 / (w / 2) - 1;
float ny1 = (h - y1 - 1) / (h / 2) - 1;
float nx2 = x2 / (w / 2) - 1;
float ny2 = (h - y2 - 1) / (h / 2) - 1;
//установка цвета в формате Rgb
glColor3f (colour.r, colour.g, colour.b);
//начало рисования примитива –линии
glBegin (GL_LINES);
//задаем координаты двух вершин линии
glVertex3f (nx1, ny1, 0.0);
glVertex3f (nx2, ny2, 0.0);
//Закончим рисованием линии
glEnd ();
}
//-----------------------------------------------------------
// Функция записи цвета пикселя по указанным координатам
void putPixel (int x, int y, Rgb colour, float w=WIDTH, float
h=HEIGHT)
{//Производим преобразование координат
float nx = x / (w / 2) - 1;
float ny = (h - y - 1) / (h / 2) - 1;
// Выбираем адрес, соответствующий подсчитанным координатам
glRasterPos2f (nx, ny);
//Отправляем по данному адресу указанный цвет
glDrawPixels (1, 1, GL_RGB, GL_FLOAT, colour);}
//-----------------------------------------------------------
// Функция получения цвета пикселя по указанным координатам
//экрана, результат помещается в colour
void getPixel(int x, int y, Rgb &colour, int h=HEIGHT)
{GlReadBuffer (GL_FRONT); //Выбираем буфер для чтения

float in[3] = {0,0,0};
glReadPixels(x, h-y, 1, 1, GL_RGB, GL_FLOAT, &in);
```

```
colour.r = in[0];colour.g = in[1];colour.b = in[2];
return;
}
//-------------------------------------------------------------
// Функция создания окна и инициализации OpenGL, принимает
// в качестве параметров название и размеры окна, возвращает
// созданное окно SDL
SDL_Window *initWindow (const char* title, int w, int h)
{
if (SDL_Init (SDL_INIT_VIDEO) < 0)
err ("Couldn't initialise SDL");

// Устанавливаем версию использованной OpenGL (2.1)
SDL_GL_SetAttribute (SDL_GL_CONTEXT_MAJOR_VERSION, 2);
SDL_GL_SetAttribute (SDL_GL_CONTEXT_MINOR_VERSION, 1);

SDL_Window *window = SDL_CreateWindow (title,
SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
w, h, SDL_WINDOW_SHOWN | SDL_WINDOW_OPENGL);

if (!window)
err ("Couldn't create SDL window");

// Создадим OpenGL контекст, с которым будем работать
OpenGL_Context = SDL_GL_CreateContext (window);

if(!OpenGL_Context)
err("Couldn't create OpenGL context");

//Попытаемся включить вертикальную синхронизацию
if(SDL_GL_SetSwapInterval(1) <0)
printf ("Warning: Unable to set VSync! SDL Error: %s\n",
SDL_GetError());

// Произведём инициализацию OpenGL
// Будем помещать ошибки в эту переменную при их появлении
GLenum error = GL_NO_ERROR;

// инициализируем матрицы Projection и Modelview
GLuint matrixes[2] = { GL_PROJECTION, GL_MODELVIEW };
for (int i = 0; i < 2; i++)
{
GlMatrixMode (matrixes[i]);
glLoadIdentity ();
error = glGetError ();
if (error != GL_NO_ERROR) // Возникла ошибка
err (reinterpret_cast<const char *> (gluErrorString (error)));
}
return window;
}
//-------------------------------------------------------------
```

```
//Функция масштабирования
void mash(float sx, float sy)
{
for(int i=0;i<5;i++)
{Xr[i]*=sx; Yr[i]*=sy;}
}
//---------------------------------------------------------------
// Функция преобразования градусов в радианы
float rad (float grad)
{return (atanf (1.0f)/45.0f)*grad;
}
//---------------------------------------------------------------
// Функция инициализации исходных состояний матрицы
void initCoord()
{
float X0[5] = {vx, vx+l, vx+l, vx, vx};
float Y0[5] = {vy, vy, vy+l, vy+l, vy};
for(int i=0;i<5;i++)
{ Xr[i] = X0[i]; Yr[i] = Y0[i];}
}
//---------------------------------------------------------------
// Функция рисования квадрата
void drawRect(Rgb color)
{
for(int i=0;i<4;i++)
{line (Xr[i], Yr[i], Xr[i+1], Yr[i+1], color); }
}
//---------------------------------------------------------------
// Функция изменения координат вершин квадрата при переносе
void move(int a, int b)
{
for(int i=0;i<5;i++)
{Xr[i]+=a; Yr[i]+=b;}
}
//---------------------------------------------------------------
// Функция изменения координат квадрата при вращении на угол
//Альфа относительно начала координат
void rotate (float alpha)
{float X;
for (int i=0;i<5;i++)
{X=Xr[i];
Xr[i] = X*cosf(alpha) -Yr[i]*sinf(alpha);
Yr[i] = X*sinf(alpha) + Yr[i]*cosf(alpha);
}}
//---------------------------------------------------------------
// Функция геометрического преобразования координат вершин
//квадрата с помощью результирующей матрицы преобразования
void resMatrix(float a, float b, float sx, float sy, float
alpha)
{
float X;
```

```
for(int i=0; i<5; i++)
{X=Xr[i];
Xr[i] = X*sx*cosf(alpha) - Yr[i]*sy*sinf(alpha) -
a*sx*cosf(alpha) + b*sy*sinf(alpha) + a;
Yr[i] = X*sx*sinf(alpha) + Yr[i]*sy*cosf(alpha) -
a*sx*sinf(alpha) - b*sy*cosf(alpha) + b;
}}
//------------------Функция отрисовки-----------------------
float grad=0,radx=0;

void Render()
{
glClearColor (0.0f, 0.0f, 0.0f, 1.0f); // Очистим фон чёрным
glClear (GL_COLOR_BUFFER_BIT); // Применим изменение цвета
if (grad==0) grad=360;
if (radx==0) radx=360;
initCoord();
drawRect(YELLOW);
move(-(vx+l/2),-(vy+l/2));
rotate(rad(-grad));
mash(1,1);
move(vx+l/2,vy+l/2);
drawRect (YELLOW);
//------------Рисуем матрицу---------------------------------
initCoord ();
move (l+100,0);
drawRect (RED);
resMatrix (vx+l+100+l, vy+l, 1, 1, rad(-grad));
drawRect (RED);

putPixel (vx+l, vy+l, RED);
//------------Вращение вокруг начала координат----------------
initCoord();
move (l+100, l+100);
drawRect (GREEN);
rotate (rad (radx));
drawRect (GREEN);
grad+=1;
radx+=1;
}
//---------------------------Главная функция-----------------
int main (int argc, char* argv[])
{
unused(argc);
unused(argv);
std::cout << "Vvedite vershinu kvadrata x:";
std::cin >> vx;
std::cout << "Vvedite vershinu kvadrata y:";
std::cin >> vy;
std::cout << "Vvedite dlinu storoni kvadrata l:";
std::cin >> l;
```

```
std::cout << "Zapusk graphiki...\n";

SDL_Window *window = initWindow ("lab_1", WIDTH, HEIGHT);
SDL_Event event;
bool working=true;
//Рабочий цикл программы
while (working)
{// Бесконечный цикл обработки событий
// (движение мыши, перемещение и закрытие окон)
while (SDL_PollEvent(&event)!= 0)
{
// Если пользователь закрыл окно, выходим из программы
if (event.type == SDL_QUIT)
{
working=false;
return 0;
}
// Если пользователь нажал клавишу, выполним какое-нибудь
//действие, например, если пользователь нажал клавишу ESC -
//выходим из программы
else if (event.type == SDL_KEYDOWN)
{
switch(event.key.keysym.sym)
{//Проверка клавиш
case SDLK_ESCAPE:
working=false;
break;
}
}
}
//Отрисовать текущий кадр
Render();
//Выполнить отрисовку кадра в буфере OpenGL
glFlush();
//Обновить содержимое окна
SDL_GL_SwapWindow(window);
//Подождать 10 миллисекунд
SDL_Delay(10);
}
std::cout << "Bye!\n";
SDL_DestroyWindow(window);
SDL_Quit();
return 0;
}
```

3 ЛАБОРАТОРНАЯ РАБОТА № 2

Реалистическое представление трехмерных объектов. Геометрические преобразования в пространстве

3.1 Цель лабораторной работы

Целью лабораторной работы является изучение математического аппарата центрального одноточечного проецирования, методов видового и перспективного преобразования трехмерных объектов, получение практических навыков элементарных и сложных геометрических преобразований в пространстве и изображения на экране монитора динамически перемещаемых трехмерных объектов.

3.2 Краткие теоретические сведения

Очевидно, что изображение трехмерного объекта, которое увидит наблюдатель, зависит от его положения по отношению к этому объекту. Поэтому для того, чтобы можно было изображать объект в разных ракурсах, необходимо выполнять видовое преобразование. Для выполнения видового преобразования должны быть заданы: описание объекта и положение точки наблюдения (глаза, камеры) в исходной трехмерной системе координат.

Пусть центр мировой системы координат находится в точке *O,* а положение точки наблюдения *E* в мировой системе координат может быть задано либо сферическими (ρ, θ, φ) либо декартовыми (x_E, y_E, z_E) координатами

Видовое преобразование, заключающееся в переходе от мировой системы координат к видовой системе координат, состоит из следующей последовательности элементарных геометрических операций:

а). Перенос системы координат, при котором точка наблюдения E становится новым началом координат.

б). Поворот системы координат вокруг оси Z на угол ($\pi/2-\theta$) в отрицательном направлении.

В результате выполнения операции «б» вектор наблюдения *EO* будет находиться в плоскости YZ, а горизонтальная проекция вектора *OE* (*OE* = –*EO*) совпадает по направлению с осью Y новой системы координат.

в). Поворот системы координат вокруг оси X на угол ($\pi-\varphi$) в положительном направлении.

Поскольку ось Z видовой системы координат должна совпадать по направлению с вектором наблюдения *EO*, повернем систему координат вокруг оси X на угол ($\pi-\varphi$) в положительном направлении.

г). Отражение относительно координатной плоскости YZ.

В результате выполненных предыдущих преобразований имеют правильную ориентацию оси Y и Z, а ось X должна быть направлена в противоположную сторону. Данная операция обеспечивает переход от *правосторонней* мировой системы координат к *левосторонней* видовой системе координат.

Видовые координаты $[x_E\ y_E\ z_E\ 1]$ опорных точек трехмерного объекта при заданных сферических координатах точки наблюдения могут быть рассчитаны по заданным мировым координатам этих точек $[x_w\ y_w\ z_w\ 1]$ с помощью выражения

$$[x_E\ y_E\ z_E\ 1] = [x_w\ y_w\ z_w\ 1] * V$$

где V - матрица видового преобразования:

$$V = \begin{Vmatrix} -\sin\theta & -\cos\varphi\cos\theta & -\sin\varphi\cos\theta & 0 \\ \cos\theta & -\cos\varphi\sin\theta & -\sin\varphi\sin\theta & 0 \\ 0 & \sin\varphi & -\cos\varphi & 0 \\ 0 & 0 & \rho & 1 \end{Vmatrix}$$

θ - угол отклонения проекции вектора наблюдения на плоскость XY от оси X;

φ - угол отклонения вектора наблюдения от оси Z;

ρ - расстояние от точки наблюдения до начала мировой системы координат (центра объекта).

Центральное одноточечное проецирование выполняется с помощью математических выражений:

$$X = (x_E/z_E) * d + c1;$$
$$Y = (y_E/z_E) * d + c2,$$

где (X, Y) - экранные координаты проецируемой точки;
d - расстояние от центра проецирования (точки наблюдения) до проекционной плоскости (экрана);
(c1, c2) - желаемые координаты проекции центра объекта на экране.

Поскольку ось Y экранной системы координат направлена вниз (то есть противоположно направлению координатной оси «Y» видовой системы координат), то на экране будет получено перевернутое изображение. Для того чтобы исключить искажения, при расчете экранной координаты Y требуется изменить знак на противоположный:

$$Y = -(y_E/z_E) * d + c2.$$

Матрицы элементарных геометрических преобразований в пространстве

1. Перенос

$$T(\lambda,\mu,\nu)=\begin{vmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\\lambda&\mu&\nu&1\end{vmatrix}$$

где λ, μ, ν = const – константы переноса вдоль осей X, Y, Z.

2. Масштабирование

$$S(sx,sy,sz)=\begin{vmatrix}sx&0&0&0\\0&sy&0&0\\0&0&sz&0\\0&0&0&1\end{vmatrix}$$

где sx, sy, sz – коэффициенты масштабирования вдоль осей X, Y, Z.

3. Поворот на угол alfa:

- вокруг координатной оси X

$$R_x(alfa)=\begin{vmatrix}1&0&0&0\\0&\cos(alfa)&\sin(alfa)&0\\0&-\sin(alfa)&\cos(alfa)&0\\0&0&0&1\end{vmatrix}$$

- вокруг координатной оси Y:

$$R_y(alfa)=\begin{vmatrix}\cos(alfa)&0&-\sin(alfa)&0\\0&1&0&0\\\sin(alfa)&0&\cos(alfa)&0\\0&0&0&1\end{vmatrix}$$

- вокруг координатной оси Z:

$$R_z(alfa)=\begin{vmatrix}\cos(alfa)&\sin(alfa)&0&0\\-\sin(alfa)&\cos(alfa)&0&0\\0&0&1&0\\0&0&0&1\end{vmatrix}$$

4. Отражение:

- относительно плоскости «XY»

$$O_{XY} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- относительно плоскости «YZ»

$$O_{YZ} = \begin{vmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- относительно плоскости «XZ»

$$O_{XZ} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

При построении геометрической модели куба можно воспользоваться схемой, представленной на рис. 2

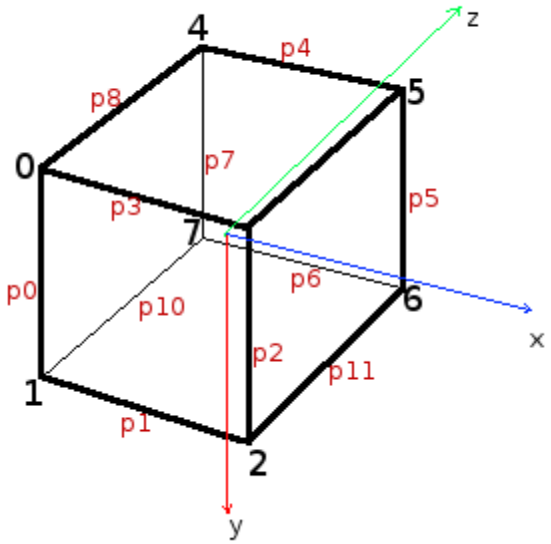

Рисунок 2 Схема построения геометрической модели куба

3.3 Задание на выполнение лабораторной работы

Разработать программу, реализующую операцию динамического вращения куба:

Вариант 1. Вокруг одного из его ребер, параллельного:

а) оси координат x;
б) оси координат y;
в) оси координат z.

Вариант 2. Вокруг произвольного вектора, проходящего вне куба и параллельного:

а) оси координат x;
б) оси координат y;
в) оси координат z.

Вариант 3. Вокруг вектора, проходящего внутри куба (не центр) и параллельного:

а) оси координат x;
б) оси координат y;
в) оси координат z.

Вариант 4. Вокруг вектора, проходящего через середину грани куба и параллельного:

а) оси координат x;
б) оси координат y;
в) оси координат z.

Вариант 5. Вокруг диагонали куба.

Геометрическое преобразование трехмерного объекта должно выполняться с помощью результирующей матрицы.

Ось вращения должна быть показана на экране.

Графические результаты отображать на одном экране.

3.4 Содержание отчета по лабораторной работе

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.
2. Цель лабораторной работы.
3. Задание на выполнение лабораторной работы.
4. Аналитической описание решаемой задачи с иллюстрациями.
5. Алгоритм и листинг отлаженной программы.

6. Графические результаты выполнения лабораторной работы.
7. Выводы по лабораторной работе.

3.5 Контрольные вопросы

1. Виды и подвиды проецирования, их характеристики.
2. Назначение и математическая реализация видового преобразования.
3. Назначение и математическая реализация перспективного преобразования.
4. Математическая реализация сложного геометрического преобразования в пространстве.
5. Объяснить значение констант, используемых в результирующей матрице реализованного в лабораторной работе геометрического преобразования.
6. Объяснить значение констант, используемых в функции перспективного преобразования разработанной программы.
7. Объяснить значение констант, используемых в функции видового преобразования разработанной программы.
8. Построить результирующую матрицу заданного сложного геометрического преобразования в пространстве.
9. Привести матрицу для математической реализации заданного элементарного геометрического преобразования в пространстве (например, геометрического преобразования поворота точки M (0, 0, 2) в точку M^* (0, 2, 0)).
10. Рассчитать экранные координаты заданной точки в пространстве.
11. Рассчитать видовые координаты заданной точки в пространстве.
12. Влияние значений видовых параметров на изображение трехмерного объекта на экране.

3.6 Листинг программы

```
#include <iostream>
#include <cmath>
#include <SDL2/SDL.h>
#include <SDL2/SDL_opengl.h>
#undef main
#include <GL/GLU.h>

const int WIDTH = 800;
const int HEIGHT = 600;

//внешние переменные
int i, a1, b1;
float AL=0, temp, c1, c2, ro, phi, teta, a, d;

//массивы мировых координат вершин куба
float xw[8], yw[8], zw[8];
```

```
// массивы видовых координат вершин
float xe[8], ye[8], ze[8];

// массивы экранных координат вершин
int x[8], y[8];

//массивы мировых, видовых, экранных координат линий вращения
float xwl[2], ywl[2], zwl[2];
float xel[2], yel[2], zel[2];
float xl[2], yl[2];

//массив ребер по номерам вершин
int MR[12][2] = { { 0, 1 }, { 1, 2 }, { 2, 3 }, { 3, 0 },
{ 4, 5 }, { 5, 6 },{ 6, 7 }, { 7, 4 }, { 1, 7 },
 { 2, 6 }, { 0, 4 }, { 3, 5 } };

struct Rgb     // структурный тип определяющий цвет
{    double r;   //интенсивность красного цвета  от 0.0 до 1.0
     double g;   //    -//-      зелёного  цвета от 0.0 до 1.0
     double b;   //    -//-      синего цвета  от 0.0 до 1.0
};

// Функция завершения программы с ошибкой, указанной в
// параметре message
void err(const char *message) {
     std::cerr << message << std::endl;
     system("pause");
     exit(1);}

//шаблон функции, не использующей параметр
template<typename T>
void unused(T const &) {}
//функция инициализации мировых координат вершин куба
void init()
{
int x0[8] = {-a/2, -a/2, a/2, a/2, -a/2, a/2, a/2, -a/2},
    y0[8] = {-a/2, a/2, a/2, -a/2, -a/2, -a/2, a/2, a/2},
    z0[8] = {-a/2, -a/2, -a/2, -a/2, a/2, a/2, a/2, a/2};

for (i = 0; i<8; i++)
     {    xw[i] = x0[i];
          yw[i] = y0[i];
          zw[i] = z0[i];
     }
}
//видовое преобразование координат i-ой вершины куба
void vidkub(int i)
{
xe[i] = -xw[i]*sin(teta) + yw[i]*cos(teta);
ye[i] = -xw[i]*cos(phi)*cos(teta) - yw[i]*cos(phi)*sin(teta) +
zw[i]*sin(phi);
```

```
ze[i] = -xw[i]*sin(phi)*cos(teta) - yw[i]*sin(phi)*sin(teta) -
zw[i]*cos(phi) + ro;
}
//видовое преобразование координат линии
void vidline(int i)
{
xel[i] = -xwl[i]*sin(teta) + ywl[i]*cos(teta);
yel[i] = -xwl[i]*cos(phi)*cos(teta) - ywl[i]*
cos(phi)*sin(teta) + zwl[i]*sin(phi);
zel[i] = -xwl[i] * sin(phi)*cos(teta) - ywl[i]*
sin(phi)*sin(teta) - zwl[i]*cos(phi) + ro;
}
// преобразование координат при перспективном проецировании
void perspkub(int i)
{
     x[i] = xe[i] / ze[i] * d + c1;
     y[i] = ye[i] / ze[i] * d + c2;
}

void perspline(int i)
{
     xl[i] = xel[i] / zel[i] * d + c1;
     yl[i] = yel[i] / zel[i] * d + c2;
}
 //-----------------------------------------------------
// Функция рисование линии между двумя заданными точками
//координаты выражены в пикселях, центр системы координат –
//левый верхний угол окна
void line(int x1, int y1, int x2, int y2, Rgb colour)
{
     float w = WIDTH;
     float h = HEIGHT;

 //Перевод к координатам OpenGL.
 // Ширина и высота окна равны 2-м единицам,
 // центр системы координат находится в центре окна.
 // Координаты по оси X и Y изменяются от -1 до 1
     float nx1 = x1 / (w / 2) - 1;
     float ny1 = (h - y1 - 1) / (h / 2) - 1;
     float nx2 = x2 / (w / 2) - 1;
     float ny2 = (h - y2 - 1) / (h / 2) - 1;

 //установка цвета в формате Rgb
     glColor3f (colour.r, colour.g, colour.b);
     //начало рисования примитива –линии
     glBegin(GL_LINES);
     //задаем координаты двух вершин линии
     glVertex3f (nx1, ny1, 0.0);
     glVertex3f (nx2, ny2, 0.0);
     //Закончим рисованием линии
     glEnd ();
```

```
}
//рисование всех ребер куба на экране
void drawkub (Rgb colour)
{
     for (i = 0; i<8; i++)
     {    vidkub(i);
          perspkub (i);
     }
     for (i = 0; i<12; i++)
     {
          a1 = MR[i][0];
          b1 = MR[i][1];
          line(x[a1], y[a1], x[b1], y[b1], colour);
     }
}
//-------------------------------------------------------------
void drawline(Rgb colour)
{
     for (i = 0; i<2; i++)
     {    vidline(i);
          perspline(i);
     }
     line(xl[0], yl[0], xl[1], yl[1], colour);
}
//-------------------------------------------------------------
void  POVOROT_X()
{
     for (i = 0; i<8; i++)
     {
     temp = yw[i];
     yw[i]= yw[i]*cos(AL)-zw[i]*sin(AL)-ywl[0]*cos(AL)+
     zwl[0]*sin(AL)+ywl[0];

     zw[i]= temp*sin(AL)+zw[i]*cos(AL)-ywl[0]*sin(AL) –
     cos(AL)*zwl[0] + zwl[0];
     }
}
//-------------------------------------------------------
void  POVOROT_Y()
{
     for (i = 0; i<8; i++)
     {
     temp = xw[i];
     xw[i]= -xw[i]*sin(AL)+zw[i]*cos(AL)+xwl[0]*sin(AL) –
     zwl[0]*cos(AL)+xwl[0];

     zw[i] = temp*cos(AL)+zw[i]* sin(AL)-xwl[0]*cos(AL) –
     sin(AL)*zwl[0] + zwl[0];
     }}
//-------------------------------------------------------
void  POVOROT_Z()
```

```
{
     for (i = 0; i<8; i++)
     {
      temp = xw[i];
      xw[i]= xw[i]*cos(AL)-yw[i]*sin(AL)-xwl[0]*cos(AL) +
      ywl[0]*sin(AL)+xwl[0];

      yw[i] = temp*sin(AL)+yw[i]*cos(AL)-xwl[0]*sin(AL) –
      cos(AL)*ywl[0]+ywl[0];
     }
}
//-------------------------------------------------------------
// Функция создания окна и инициализации OpenGL, принимает
// в качестве параметров название и размеры окна, возвращает
// созданное окно SDL
SDL_Window *initWindow (const char* title, int w, int h) {
     if (SDL_Init (SDL_INIT_VIDEO) < 0)
          err("Couldn't initialise SDL");

// Устанавливаем версию использованной OpenGL (2.1)
     SDL_GL_SetAttribute (SDL_GL_CONTEXT_MAJOR_VERSION, 2);
     SDL_GL_SetAttribute (SDL_GL_CONTEXT_MINOR_VERSION, 1);

     SDL_Window *window = SDL_CreateWindow (title,
     SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
      w, h, SDL_WINDOW_SHOWN | SDL_WINDOW_OPENGL);

     if (!window)
          err("Couldn't create SDL window");

// Создадим OpenGL контекст, с которым будем работать
     SDL_GLContext context = SDL_GL_CreateContext (window);
     if (!context)
          err("Couldn't create OpenGL context");
// Проверим доступность вертикальной синхронизации
     if (SDL_GL_SetSwapInterval(1) <0) {
          printf ("Warning: Unable to set VSync! SDL Error:"
         "%s\n", SDL_GetError ());
     }
// Произведём инициализацию OpenGL
// Будем помещать ошибки в эту переменную при их появлении
     GLenum error = GL_NO_ERROR;
// Инициализируем матрицы Projection (проецирования) и
//Modelview (вида)

     GLuint matrixes [2] = { GL_PROJECTION, GL_MODELVIEW };

     for (int i = 0; i < 2; i++) {
          glMatrixMode (matrixes[i]);
          glLoadIdentity ();
          error = glGetError ();
```

```
          if (error != GL_NO_ERROR) // Что-то пошло не так
          err (reinterpret_cast<const char*>
         (gluErrorString (error)));
     }
     return window;
}

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

unused (argc);
unused (argv);
std::cout<<"Enter values: (Leave empty to use default "
"values)\n\n";
phi=35
std::cout<<"phi ("<<phi<<"): ";
std::cin >> phi;
teta = 53;
std::cout<<"teta ("<<teta<<"): ";
std::cin >> teta;
ro = 200;
std::cout<<"ro ("<<ro<<"): ";
std::cin >> ro;
d = 150;
std::cout<<"d (side Length, "<<d<<"): ";
std::cin >> d;
a = 80;
std::cout<<"a ("<<a<<"): ";
std::cin >> a;

phi=phi*atan(1)/45;
teta=teta*atan(1)/45;
std::cout << "Zapusk graphiki...\n";

SDL_Window *window = initWindow("lab_2", WIDTH, HEIGHT);
SDL_Event event;

while (true) {
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);   // Очистим фон чёрным
glClear(GL_COLOR_BUFFER_BIT);      // Применим изменение цвета
glFlush();

// Цикл, в котором обрабатываются системные и пользовательских
//событий (движение мыши, перемещение и закрытие окон)
     while (SDL_PollEvent(&event) != 0) {
// Если пользователь закрыл окно, выходим из программы
               if (event.type == SDL_QUIT) {
                    SDL_Quit();
                    return 0;
               }
             }
```

```
AL += 0.01;

c1 = 150; c2 = 150;
xwl[0] = a/2;  xwl[1] = -a/2;
ywl[0] = -a/2; ywl[1] = -a/2;
zwl[0] = -a/2; zwl[1] = -a/2;
init();

POVOROT_X();
drawkub({ 1.0, 1.0, 1.0 });
drawline({ 1.0, 0.0, 0.0 });

c1 = 600; c2 = 200;
init();
xwl[0] = a/2;  xwl[1] = a/2;
ywl[0] = -a/2; ywl[1] = a/2;
zwl[0] = -a/2; zwl[1] = -a/2;
POVOROT_Y();
drawkub({ 1.0, 1.0, 1.0 });
drawline({ 1.0, 0.0, 0.0 });

c1 = 400; c2 = 500;
init();
xwl[0] = a/2;  xwl[1] = a/2;
ywl[0] = a/2;  ywl[1] = a/2;
zwl[0] = -a/2; zwl[1] = a/2;
POVOROT_Z();
drawkub ({ 1.0, 1.0, 1.0 });
drawline ({ 1.0, 0.0, 0.0 });

// Отправим все ожидающие отрисовки объекты на экран
 glFlush ();
// Обновим окно
 SDL_GL_SwapWindow(window);
 SDL_Delay (10);
}
return 0;
}
```

4. СПИСОК ЛИТЕРАТУРЫ

1) Федотова Т.Н. Компьютерная графика. Часть 1. Учебное пособие. – М: МГТУ ГА, 2008.

2) Баяковский Ю.М. Игнатенко А.В. Начальный курс OpenGL - М: Планета знаний, 2007.

3) Богуславский А.А., Соколов С.М. Основы программирования на языке Си++. Часть II. Основы программирования трехмерной графики — Коломна: КГПИ, 2002.

СОДЕРЖАНИЕ