

Практическое занятие №1

ВВЕДЕНИЕ В OPENGL

Целью выполнения лабораторной работы является формирование практических навыков по работе с проекционной матрицей средствами OpenGL.

Основными задачами выполнения лабораторной работы являются: сформировать представление о методах и секторе решаемых OpenGL задач, изучить основные принципы работы OpenGL, представлять и понимать основные реализации OpenGL, уметь создавать типовой проект в различных средах разработки (Visual Studio), иметь представление о двойной буферизации.

Содержание отчета:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Предоставить отчет, по каждому заданию содержащий: формулировку задания, исходный код программы, скриншоты работающей программы (все названия окон – это фамилия и имя студента).
4. Ответить на вопросы преподавателя.

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

На базовом уровне, OpenGL — это просто *спецификация*, то есть документ, описывающий набор функций и их точное поведение. Производители оборудования на основе этой спецификации создают *реализации* — библиотеки функций, соответствующих набору функций спецификации. Реализация призвана эффективно использовать возможности оборудования. OpenGL освобождает программиста от написания программ для конкретного оборудования. Если устройство поддерживает какую-то функцию, то эта функция выполняется аппаратно, если нет, то библиотека выполняет её программно.

С точки зрения программиста OpenGL - это программный интерфейс для графических устройств, таких как графические ускорители. Он включает в себя около 150 различных команд, с помощью которых программист может определять различные объекты и производить рендеринг. Говоря более простым языком, вы определяете объекты, задаёте их местоположение в трёхмерном пространстве, определяете другие параметры (поворот, масштаб, ...), задаёте свойства объектов (цвет, текстура, материал, ...), положение наблюдателя, а библиотека OpenGL позаботится о том, чтобы

отобразить всё это на экране. Поэтому можно сказать, что библиотека OpenGL является только воспроизводящей (Rendering), и занимается только отображением 3D объектов, она не работает с устройствами ввода (клавиатуры, мыши). Также она не поддерживает менеджер окон.

OpenGL включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформе Windows конкурирует с Direct3D.

Игры, разработанные на OpenGL

- Ballenger
- Cube 2
- Doom (2016 video game)
- Minecraft, a famous sandbox video game
- Osu!

Фотографии и видео

- Adobe After Effects
- Adobe Photoshop
- Adobe Premiere Pro
- ArtRage
- Kodi

Моделирование и САПР

- 3D Studio Max
- Autodesk AutoCAD, 2D/3D CAD
- Autodesk Maya
- Blender, 3D CAD
- Cadence Allegro, Computer-aided design, Electronics
- LARSA4D
- Modo
- Rhinoceros
- SAP2000
- Scilab
- SketchUp
- VirtualMec

Визуализация и другие узкоспециализированные программы

- Algodoo
- Avogadro
- BALLView
- Celestia
- Enhanced Machine Controller (EMC2)
- Google Earth

- InVesalius
 - Mari (software), 3D texturing and painting software
 - PyMOL
 - QuteMol
 - Really Slick Screensavers, 3D Screensavers
 - SpaceEngine
-

Основные возможности OpenGL.

Что предоставляет библиотека в распоряжение программиста? Основные возможности:

- **Геометрические и растровые примитивы.** На основе геометрических и растровых примитивов строятся все объекты. Из геометрических примитивов библиотека предоставляет: точки, линии, полигоны. Из растровых: битовый массив(bitmap) и образ(image)
- **Использование В-сплайнов.** В-сплайны используются для рисования кривых по опорным точкам.
- **Видовые и модельные преобразования.** С помощью этих преобразований можно располагать объекты в пространстве, вращать их, изменять форму, а также изменять положение камеры из которой ведётся наблюдение.
- **Работа с цветом.** OpenGL предоставляет программисту возможность работы с цветом в режиме RGBA (красный-зелёный-синий-альфа) или используя индексный режим, где цвет выбирается из палитры.
- **Удаление невидимых линий и поверхностей.**
- **Двойная буферизация.** OpenGL предоставляет как одинарную так и двойную буферизацию. Двойная буферизация используется для того, чтобы устранить мерцание при мультипликации, т.е. изображение каждого кадра сначала рисуется во втором(невидимом) буфере, а потом, когда кадр полностью нарисован, весь буфер отображается на экране.
- **Наложение текстуры.** Позволяет придавать объектам реалистичность. На объект, например шар, накладывается текстура(просто какое-то изображение), в результате чего наш объект теперь выглядит не просто как шар, а как разноцветный мячик.
- **Сглаживание.** Сглаживание позволяет скрыть ступенчатость, свойственную растровым дисплеям. Сглаживание изменяет интенсивность и цвет пикселей около линии, при этом линия смотрится на экране без всяких зигзагов.
- **Освещение.** Позволяет задавать источники света, их расположение, интенсивность, и т.д.

- **Атмосферные эффекты.** Например туман, дым. Всё это также позволяет придать объектам или сцене реалистичность, а также "почувствовать" глубину сцены.
- **Прозрачность объектов.**
- **Использование списков изображений.**

Дополнительные библиотеки OpenGL

Несмотря на то, что библиотека OpenGL (сокращённо GL) предоставляет практически все возможности для моделирования и воспроизведения трёхмерных сцен, некоторые из функций, которые требуются при работе с графикой, отсутствуют в стандартной библиотеке OpenGL. Например, чтобы задать положение и направление камеры, с которой будет наблюдаться сцена, нужно самому рассчитывать модельную матрицу, а это далеко не все умеют. Поэтому для OpenGL существуют так называемые вспомогательные библиотеки.

Первая из этих библиотек называется GLU. Эта библиотека уже стала стандартом и поставляется вместе с главной библиотекой OpenGL. В состав этой библиотеки вошли более сложные функции, например для того чтобы определить цилиндр или диск потребуется всего одна команда. Также в библиотеку вошли функции для работы со сплайнами, реализованы дополнительные операции над матрицами и дополнительные виды проекций.

Следующая библиотека, также широко используемая - это **GLUT**. Это также независимая от платформы библиотека. Она реализует не только дополнительные функции OpenGL, но и предоставляет функции для работы с окнами, клавиатурой и мышкой. Для того чтобы работать с OpenGL в конкретной операционной системе, надо провести некоторую предварительную настройку и эта предварительная настройка зависит от конкретной операционной системы. С библиотекой GLUT всё намного упрощается, буквально несколькими командами можно определить окно, в котором будет работать OpenGL, определить прерывание от клавиатуры или мышки и всё это не будет зависеть от операционной системы. Библиотека предоставляет также некоторые функции, с помощью которых можно определять некоторые сложные фигуры, такие как конусы, тетраэдры, и даже можно с помощью одной команды определить чайник!

Freeglut — открытая альтернатива OpenGL Utility Toolkit (GLUT). Freeglut предназначена для полной замены GLUT, и имеет очень немного отличий от неё.

Существуют и другие дополнительные библиотеки для OpenGL. Все они добавляют что-то своё или ориентированы на какую-то платформу.

Спецификация OpenGL существует уже давно и с тех пор сильно изменилась. Процесс модификации спецификации основан на механизме расширений. Каждое новое

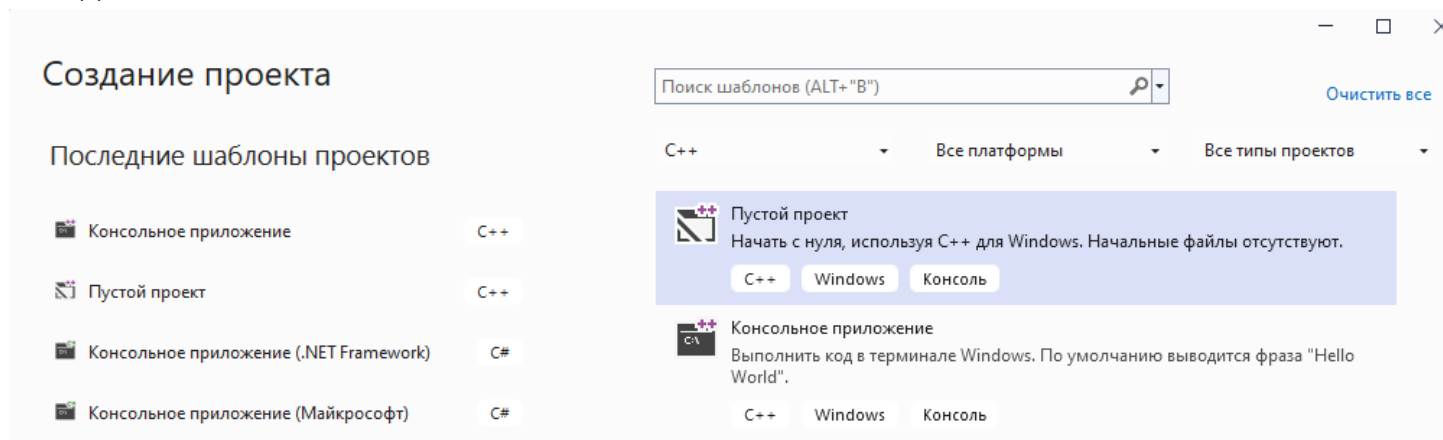
расширение предполагает введение нового набора функций, дополняющий функциональность OpenGL. Со временем, когда набирается достаточно расширений, комитет ARB, ведающий разработкой OpenGL, утверждает новую версию спецификации. Версии отличаются друг от друга только включенными в них расширениями. Текущая на данный момент версия спецификации — 4.6 (<https://registry.khronos.org/OpenGL>).

К несчастью, механизм расширений обладает серьезным недостатком. Обычно конечному программисту предоставляется базовый набор функций OpenGL, который находится на уровне версии 1.1 или 1.4. Для получения всех функций версий выше программисту предоставляются дополнительные функции по работе с расширениями, использование которых, мягко говоря, весьма трудоемко. Так как эту часть работы приходится проделывать в любом новом проекте, была выпущена специальная кросс-платформенная библиотека **GLEW** (The OpenGL Extension Wrangler Library <https://glew.sourceforge.net/>), задача которой и заключается в инициализации механизма расширений OpenGL и предоставлении конечному программисту полной ее функциональности. Эта функция инициализирует саму библиотеку. При этом она собирает сведения обо всех расширениях, поддерживаемых на конкретной машине с конкретным оборудованием и инициализирует адреса всех доступных программисту функций. При успешной инициализации функция возвращает значение `GLEW_NO_ERROR`, после чего весь функционал становится доступным программисту.

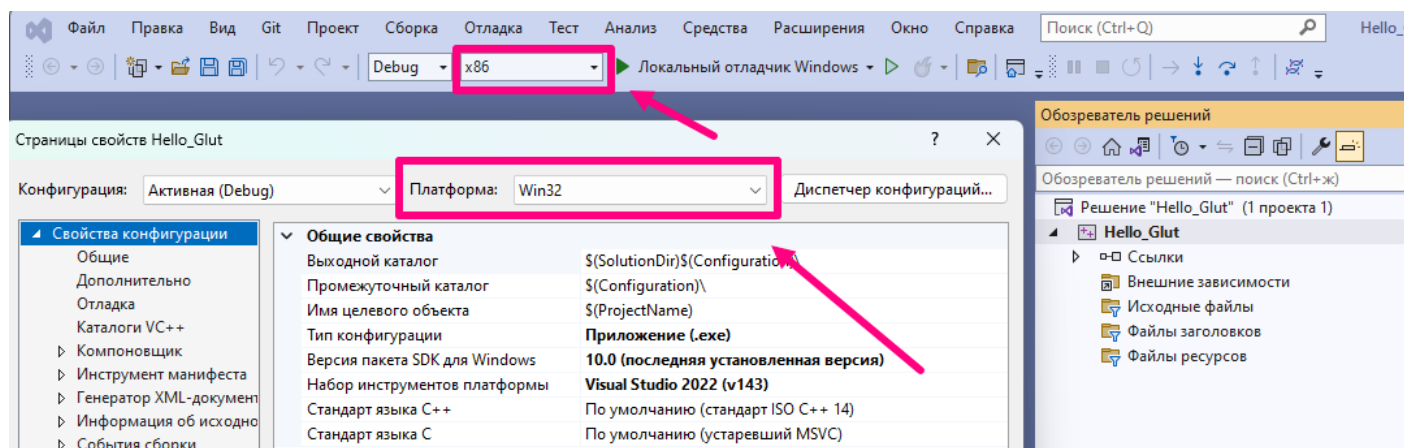
Основная библиотека, необходимая для работы на практических занятиях – это библиотека GLUT.

Первая программа. Подключаем библиотеки.

Для создания и настройки типичного проекта выполните следующее. Запустите Microsoft Visual Studio. Создайте пустой проект с использованием языка C++ для Windows.



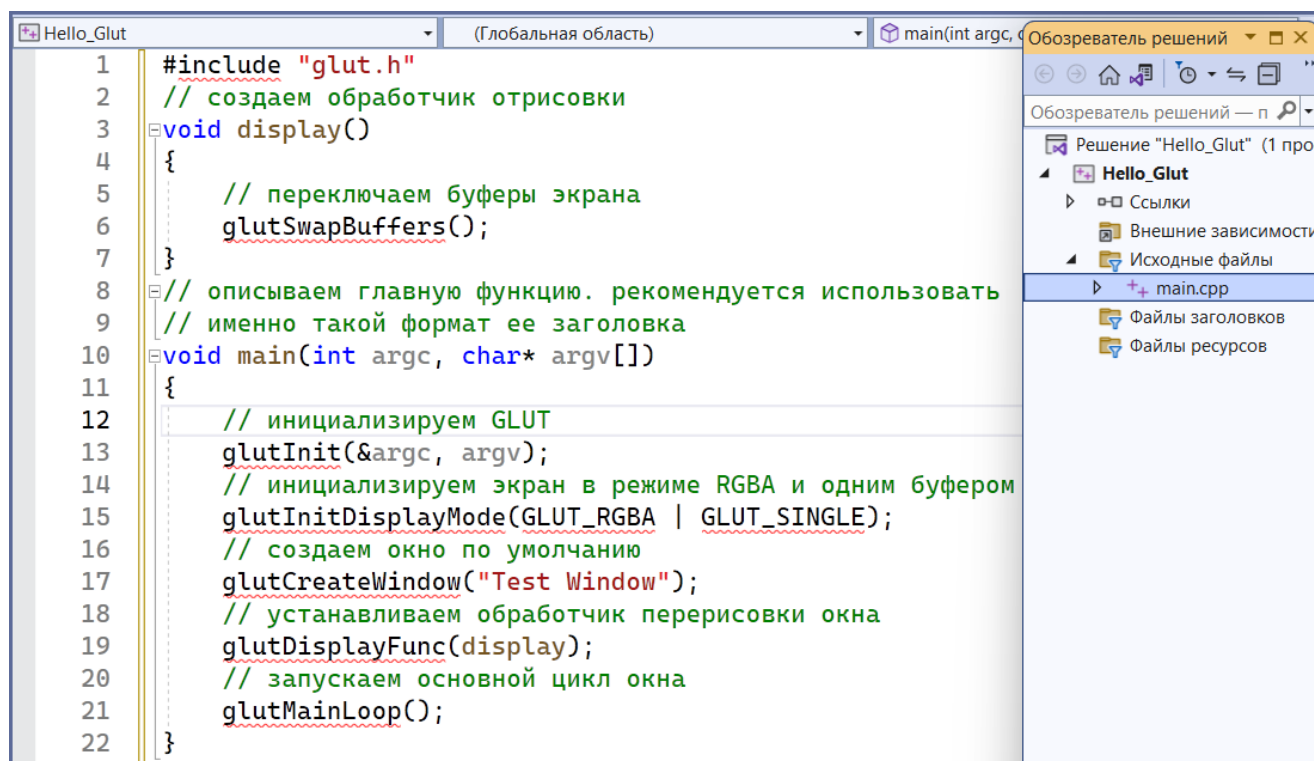
Далее в пустом проекте измените некоторые настройки по умолчанию: тип архитектуры с x64 на x86 и платформу Win32.



В обозревателе решений надо вызвать контекстное меню, в котором выбрать **Добавить -> Создать элемент**. В созданный файл **main.cpp** поместите следующий программный код:

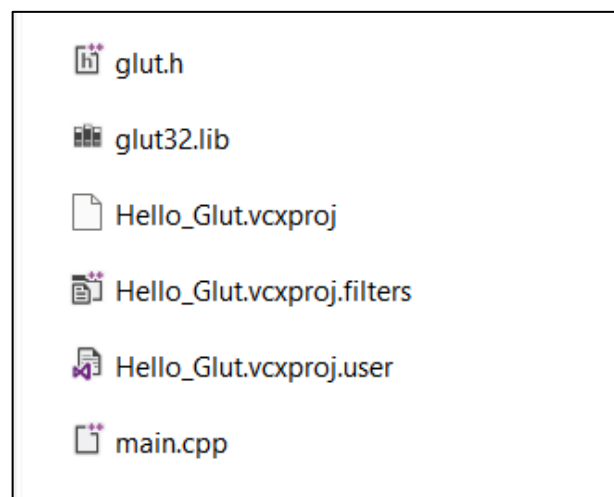
Листинг 1 – Простейшее окно GLUT

```
#include "glut.h"
// создаем обработчик отрисовки
void display()
{
    // переключаем буферы экрана
    glutSwapBuffers();
}
// описываем главную функцию. рекомендуется использовать
// именно такой формат ее заголовка
void main(int argc, char* argv[])
{
    // инициализируем GLUT
    glutInit(&argc, argv);
    // инициализируем экран в режиме RGBA и одним буфером
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    // создаем окно по умолчанию
    glutCreateWindow("Test Window");
    // устанавливаем обработчик перерисовки окна
    glutDisplayFunc(display);
    // запускаем основной цикл окна
    glutMainLoop();
}
```

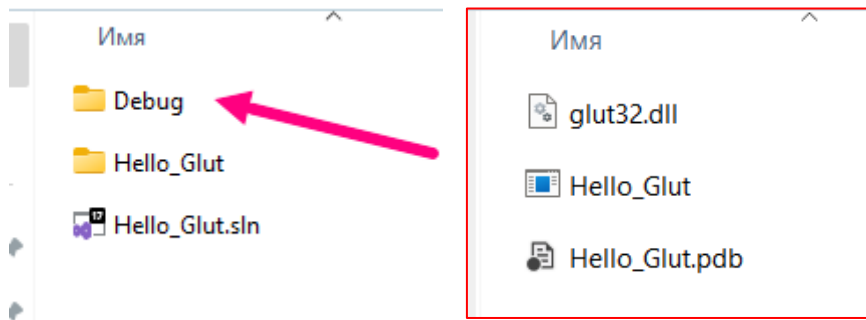



```
1  #include "glut.h"
2  // создаем обработчик отрисовки
3  void display()
4  {
5      // переключаем буферы экрана
6      glutSwapBuffers();
7  }
8  // описываем главную функцию. рекомендуется использовать
9  // именно такой формат ее заголовка
10 void main(int argc, char* argv[])
11 {
12     // инициализируем GLUT
13     glutInit(&argc, argv);
14     // инициализируем экран в режиме RGBA и одним буфером
15     glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
16     // создаем окно по умолчанию
17     glutCreateWindow("Test Window");
18     // устанавливаем обработчик перерисовки окна
19     glutDisplayFunc(display);
20     // запускаем основной цикл окна
21     glutMainLoop();
22 }
```

Теперь необходимо подключить к проекту библиотеки. Для этого скопируйте из папки с заданием заголовочный файл **glut.h**, а также **lib-файл** (именно lib, а не dll) в папку с проектом (там же должен располагаться файл с именем проекта и расширением **vcxproj**).



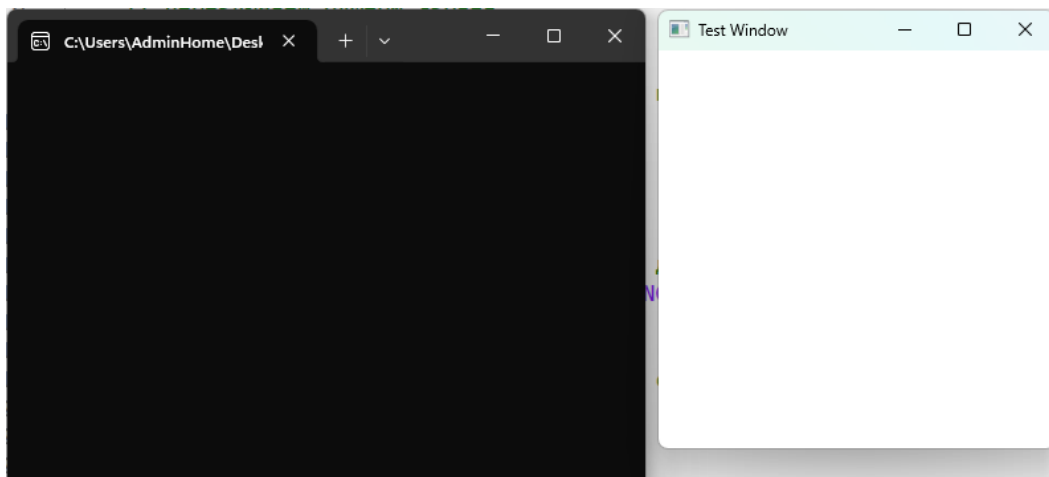
Далее сохраните все изменения **Файл-> Сохранить все**.
Итак, программа готова к запуску. Теперь надо собрать ее. Для этого перейдите в меню **Сборка -> Собрать решение**. Начнется процесс сборки программы, а снизу в закладке **Вывод** начнет выводиться информация по компиляции и компоновке. В конце должно последовать сообщение о том, что сборка завершена удачно.
После этого в папке проекта появится новая папка **Debug**, куда и разместим оставшийся файл библиотек DLL.



Любая динамически подключаемая библиотека (например, GLUT) состоит из трех частей — заголовочного файла .h, lib-файла и dll-файла.

Первые два нужны для компиляции и были скопированы нами в папку с самим проектом. Третий нужен уже при запуске программы.

После запуска программы появятся два окна.



Большое окно — это консольное приложение, а маленькое — окно GLUT.

Теперь разберемся с программным кодом. В начале подключаем заголовочный файл GLUT:

```
#include "glut.h"
```

Программы C и C++ консольного режима работы всегда начинают выполнение с функции main. После инициализации GLUT включаем режим отображения: один буфер.

```
glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
```

Приведенные метки указывают задействовать окно с простой буферизацией (с одним буфером) (GLUT_SINGLE) и режим цвета RGBA (GLUT_RGBA). Простая буферизация означает, что все команды рисования выполняются в отображенном окне. Альтернативой является двойная буферизация, когда команды рисования выполняются в буфере вне экрана, а затем быстро отображаются в окне. Этот метод часто применяется для создания эффектов анимации. Режим цвета RGBA означает, цвета задаются указанием различных интенсивностей красного, зеленого и синего

компонентов. Т.е. трехканальная цветовая модель RGB, дополненная четвертым альфа-каналом. Альфа указывает, насколько непрозрачен каждый пиксель, и позволяет комбинировать изображение поверх других с помощью альфа-компоновки с прозрачными областями и сглаживанием краев непрозрачных областей. Альтернативой является режим индексирования цвета, довольно распространенный и заключающийся в том, что вы задаете цвета с помощью указателей на палитру цветов.

Следующий вызов к библиотеке GLUT создает окно на экране. С помощью приведенного ниже кода создается окно `Test Window`.

```
glutCreateWindow("Test Window");
```

Единственным аргументом `glutCreateWindow` является надпись в строке заголовка окна. Следующая строка кода, относящаяся к GLUT, имеет такой вид:

```
glutDisplayFunc(display);
```

`glutDisplayFunc` - это функция, которая определяет, как будет отображаться каждая сцена в OpenGL. Она вызывается каждый раз, когда необходимо перерисовать окно. Эта функция принимает один аргумент, функцию обратного вызова, которая рисует сцену.

Последний вызов функции GLUT выполняется в конце программы.

```
glutMainLoop();
```

Это функция из библиотеки GLUT, которая запускает основной цикл обработки событий в программе. Этот цикл выполняется до тех пор, пока программа не будет закрыта, и отвечает за обработку событий, таких как ввод с клавиатуры, перемещение окна и перерисовка экрана. `glutMainLoop` продолжает вызывать функции отображения, а также сохраняет окно фактически открытым. Приложение должно вызывать ее только один раз.

ЗАДАНИЕ 1. Настроить перерисовку главного окна синим цветом.

Листинг 2 – Программа SIMPLE

```
// подключаем заголовочные файлы библиотек
#include "glew.h"
#include "glut.h"
void RenderScene(void)
{
    // Окно очищается текущим цветом очистки
    glClear(GL_COLOR_BUFFER_BIT);
    // В буфер вводятся команды рисования
    glFlush();
}
// Устанавливается состояние визуализации
void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
```

```

}
// Точка входа основной программы
void main(void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);
    SetupRC();
    glutMainLoop();
}

```

Следующая строка не относится ни к GLUT, ни к OpenGL:

```
SetupRC();
```

В этой функции выполняется вся инициализация OpenGL, которую нужно выполнить перед визуализацией. Функция SetupRC содержит единственный вызов функции OpenGL.

```
glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
```

Эта функция устанавливает цвет, используемый для очистки окна. Прототип функции выглядит так:

```
void glClearColor(GLclampf red, GLclampf green, GLclampf blue,
GLclampf alpha);
```

В большинстве реализаций OpenGL GLclampf определяется как величина типа float. В OpenGL единый цвет представляется как смесь красного, зеленого и синего компонентов. Каждый компонент может представляться величиной, принадлежащей диапазону 0,0-1,0. В OpenGL значением каждого компонента могут быть приемлемые значения с плавающей запятой от 0 до 1, следовательно, число возможных цветов практически бесконечно. На практике чаще всего используется величина 24 бит (16 миллионов цветов). Последний аргумент функции glClearColor — это компонент альфа, который используются при смешении и создании таких специальных эффектов, как просвечивание. Просвечиванием называется способность объекта пропускать свет. Предположим, требуется создать кусок красного стекла, за которым находится источник синего света. Синий свет влияет на вид красного стекла (синий + красный = фиолетовый). Для генерации красного цвета полупрозрачного объекта можно использовать компонент альфа, при этом объект будет выглядеть как кусок цветного стекла; кроме того, будут видны объекты, находящиеся за ним. В создании эффектов такого типа участвуют не только значения альфа.

Некоторые распространенные составные цвета

Составной цвет	Красный компонент	Зеленый компонент	Синий компонент
Черный	0.0	0.0	0.0
Красный	1.0	0.0	0.0
Зеленый	0.0	1.0	0.0
Желтый	1.0	1.0	0.0
Синий	0.0	0.0	1.0
Пурпурный	1.0	0.0	1.0
Голубой	0.0	1.0	1.0

Темно-серый	0.25	0.25	0.25
Светло-серый	0.75	0.75	0.75
Коричневый	0.60	0.40	0.12
Тыквенно-оранжевый	0.98	0.625	0.12
Пастельный розовый	0.98	0.04	0.7
Мягкий пурпурный	0.60	0.40	0.70
Белый	1.0	1.0	1.0

Очистка буфера цвета. Все, что было описано до этого момента, — это указали OpenGL использовать в качестве цвета очистки синий. В функции `RenderScene` требуется команда, выполняющая собственно очистку: `glClear(GL_COLOR_BUFFER_BIT);`

Функция `glClear` очищает определенный буфер или комбинацию буферов. Буфер — это область хранения информации об изображении. Красный, зеленый и синий компоненты рисунка обычно объединяются под общим названием буфера цветов или буфера пикселей. Буфер цветов — это место, в котором хранится отображаемое изображение, и что очистка буфера с помощью команды `glClear` удаляет из окна последний отображенный рисунок.

Последним идет завершающий вызов функции OpenGL.

`glFlush();`

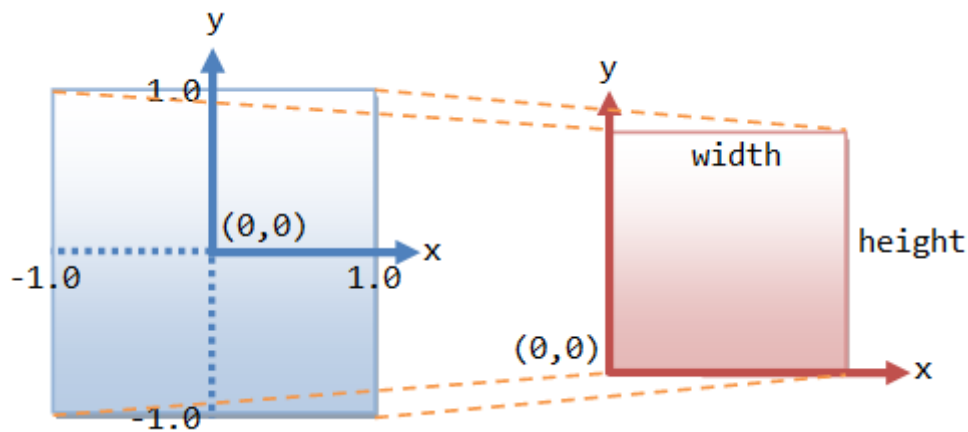
В этой строке указывается выполнить все невыполненные команды OpenGL; у нас есть только одна такая команда — `glClear`. Во внутреннем представлении OpenGL использует конвейер, последовательно обрабатывающий команды. Команды OpenGL часто выстраиваются в очередь, чтобы потом драйвер OpenGL обработал несколько "команд" одновременно. Такая схема увеличивает производительность. В короткой программе, приведенной в листинге 1, функция `glFlush` просто сообщает OpenGL, что он должен обработать команды рисования, переданные на этот момент, и ожидать следующих команд.

Viewport - это область окна, в которой будет отображаться результаты нашей работы. Для установки области видимости надо использовать функцию OpenGL

`glViewport(GLint x, GLint y, GLsizei width, GLsizei height).`

x и y - координаты левого нижнего угла области видимости, $width$ и $height$ - размеры. По умолчанию, OpenGL устанавливает область видимости равную размерам окна приложения в момент инициализации. Наиболее часто эту команду используют в обработчике изменения размеров окна.

При этом все, что будет изображаться в этой области окна задается декартовыми координатами в диапазоне $[-1; 1]$. Т.о. Viewport - это экранное преобразование, которое будет использоваться для преобразования из внутренних вещественных координат OpenGL в целочисленные экранные координаты на растре (в буфере кадра).



ЗАДАНИЕ 2. Преобразуем программу. В центре окна с размерами 400 × 400 изобразим красный прямоугольник.

Листинг 3 – Программа SIMPLE -квадрат

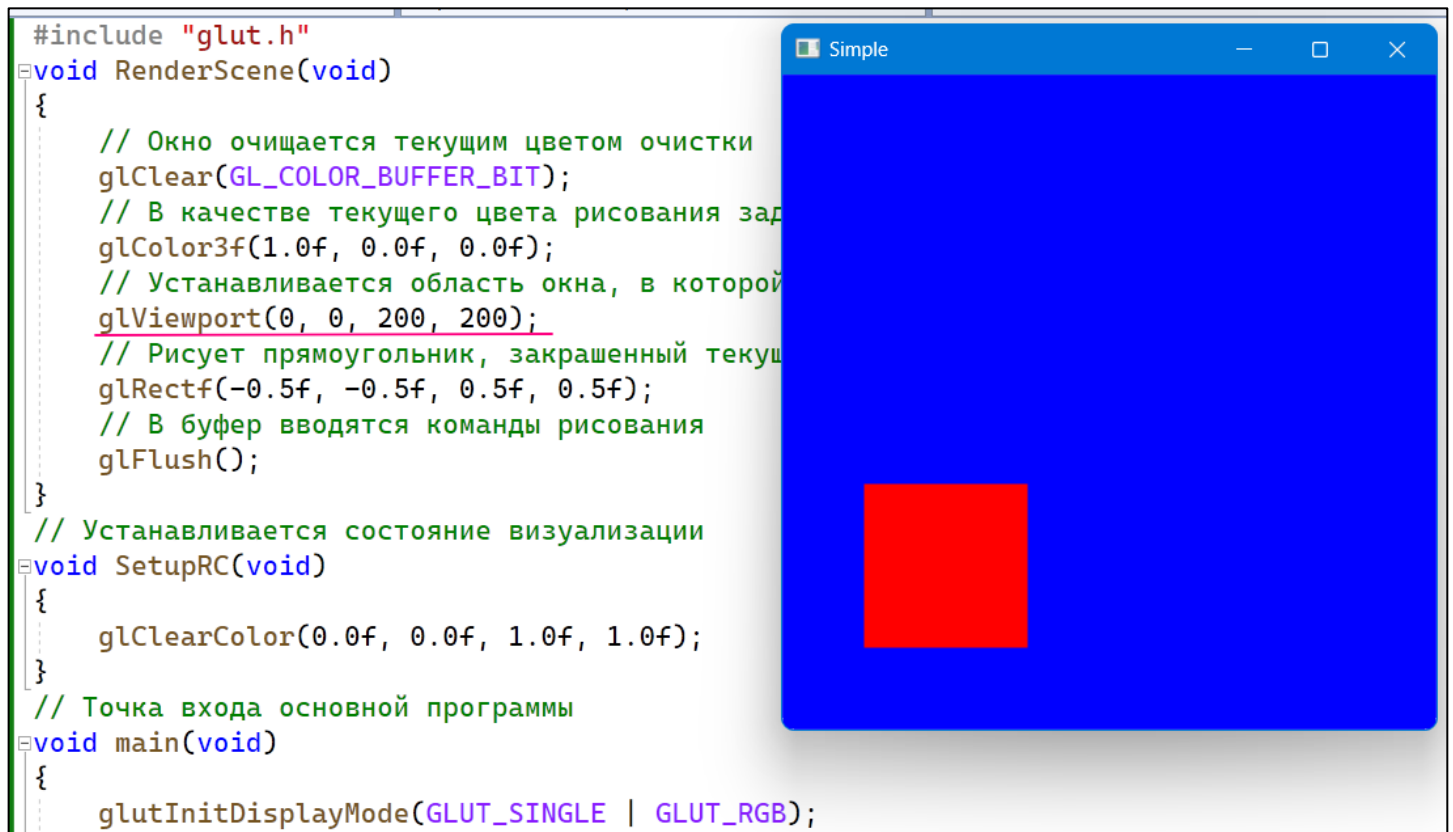
```
// подключаем заголовочные файлы библиотек
#include "glut.h"
void RenderScene(void)
{
    // Окно очищается текущим цветом очистки
    glClear(GL_COLOR_BUFFER_BIT);
    // В качестве текущего цвета рисования задает красный
    glColor3f(1.0f, 0.0f, 0.0f);
    // Устанавливается область окна, в которой будет отображаться прямоугольник
    glViewport(0, 0, 400, 400);
    // Рисует прямоугольник, закрашенный текущим цветом
    glRectf(-0.5f, -0.5f, 0.5f, 0.5f);
    // В буфер вводятся команды рисования
    glFlush();
}
// Устанавливается состояние визуализации
void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
// Точка входа основной программы
void main(void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    // Устанавливаем размеры окна
    glutInitWindowSize(400, 400);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);
```

```

    SetupRC();
    glutMainLoop();
}

```

Обратим внимание, что если поменять размеры GLsizei width, GLsizei height с 400 на 200, то прямоугольник будет рисоваться только в видимой области окна размера 200×200.



Установим двумерную ортографическую систему координат.

Листинг 4

```

void RenderScene(void)
{
    // Окно очищается текущим цветом очистки
    glClear(GL_COLOR_BUFFER_BIT);
    // В качестве текущего цвета рисования задает красный //RGB
    glColor3f(1.0f, 0.0f, 0.0f);
    glViewport(0, 0, 400, 400);
    // Обновляет систему координат
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // Установка двумерной ортографической системы координат
    glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // Рисует прямоугольник, закрашенный текущим цветом

```

```

    glRectf(-25.f, -25.f, 25.f, 25.f);
    // В буфер вводятся команды рисования
    glFlush();
}

```

Суть и особенности ортогографической проекции будут рассмотрены в рамках следующих практических работ, а сейчас необходимо понимать только то, что с помощью команды OpenGL `glOrtho` мы можем установить двумерную или трёхмерную систему координат.

```

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far);

```

В трёхмерном декартовом пространстве значения `left` и `right` задают минимальную и максимальную координату точек, отображаемых вдоль оси `x`; `bottom` и `top` делают то же для оси `y`. Параметры `near` и `far` предназначены для оси `z`, обычно удалению от наблюдателя соответствуют отрицательные значения. Для двухмерной системы можно задать `near = -1, far = 1`.

Обратите внимание на два вызова функций сразу после кода с `glOrtho`.

```

// Обновляет систему координат
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

```

ЗАДАНИЕ 3. Преобразовать программу так, чтобы прямоугольник всегда находился по центру окна (при любых растяжениях и сжатиях окна).

Листинг 5 – Устойчивый к растяжению окна квадрат

```

// подключаем заголовочные файлы библиотек
#include "glut.h"
void RenderScene(void) {
    // Очищаем окно, используя текущий цвет очистки
    glClear(GL_COLOR_BUFFER_BIT);
    // В качестве текущего цвета рисования задает белый
    glColor3f(1.0f, 1.0f, 1.0f);
    // Рисует прямоугольник, покрашенный текущим цветом
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    // Очищает очередь текущих команд
    glFlush();
}
////////// Задаёт состояние визуализации
void SetupRC(void)
{
    // Устанавливает в качестве цвета очистки розовый
    glClearColor(1.0f, 0.0f, 1.0f, 1.0f);
}
//////////Вызывается библиотекой GLUT при изменении размеров окна

```

```

void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    // Устанавливает поле просмотра с размерами окна
    glViewport(0, 0, w, h);
    // Обновляет систему координат
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // Установка двумерной ортогографической системы координат
    glOrtho(-100.0, 100.0, -100, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
//////////Точка входа основной программы
void main(void) {
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```



ЗАДАНИЕ 4. Преобразовать программу так, чтобы получить центрированный прямоугольник.

Листинг 6 – Центрированный прямоугольник.

```

#include "glut.h"
void RenderScene(void) {
    // Очищаем окно, используя текущий цвет очистки
    glClear(GL_COLOR_BUFFER_BIT);
    // В качестве текущего цвета рисования задает белый
    glColor3f(1.0f, 1.0f, 1.0f);
    // Рисует прямоугольник, закрашенный текущим цветом
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    // Очищает очередь текущих команд
}

```

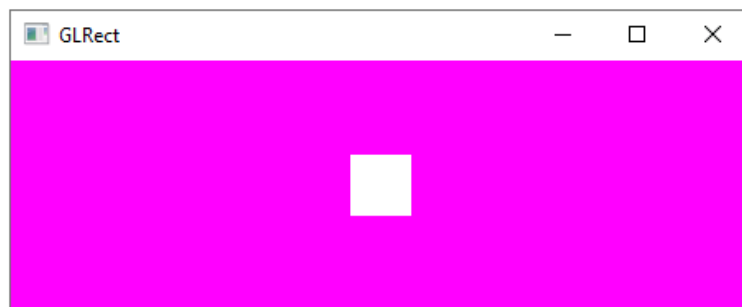


```

    glFlush();
}
////////// Задаёт состояние визуализации
void SetupRC(void)
{
    // Устанавливает в качестве цвета очистки розовый
    glClearColor(1.0f, 0.0f, 1.0f, 1.0f);
}
//////////Вызывается библиотекой GLUT при изменении размеров окна
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    // Предотвращает деление на ноль
    if (h == 0)
        h = 1;
    // Устанавливает поле просмотра с размерами окна
    glViewport(0, 0, w, h);
    // Обновляет систему координат
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h; if (w <= h)
        // Установка двумерной ортогонафической системы координат
        glOrtho(-100.0, 100.0, -100 / aspectRatio, 100.0 / aspectRatio,
1.0, -1.0);
    else
        glOrtho(-100.0 * aspectRatio, 100.0 * aspectRatio, -100.0, 100.0,
1.0, -1.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void main(void) {
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

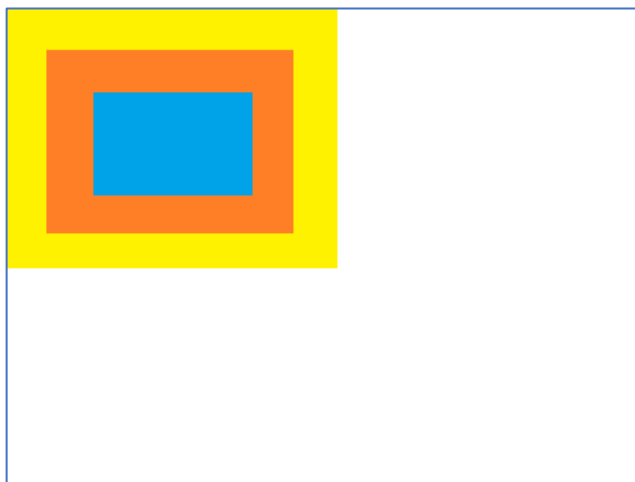


ЗАДАНИЕ 5. Создать окно в позиции (100,100) и размерами 400x300 с зеленым цветом перерисовки.

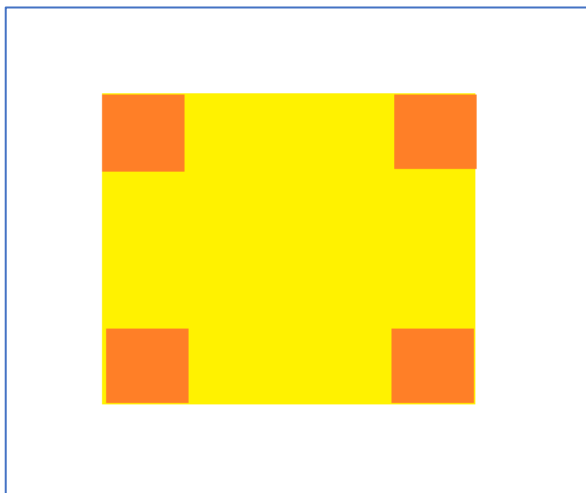
ЗАДАНИЕ 6. Создать окно, в зависимости от размеров которого менялся бы его цвет (ка минимум 3 раза).

ЗАДАНИЕ 7 (по вариантам). Отобразить в окне рисунок, составленный из прямоугольников согласно образцу своего варианта. Пропорции окна, расположения фигур и их цвета должны соответствовать рисунку. При изменении размеров окна рисунок должен быть устойчив к растяжению (т.е. пропорции рисунка не должны меняться).

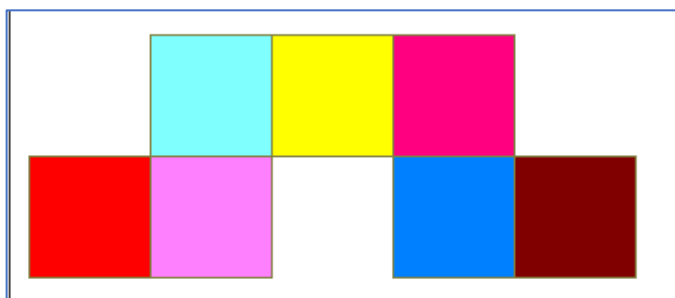
1.



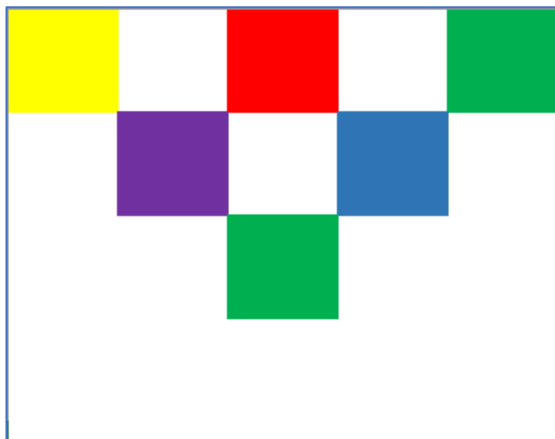
2.



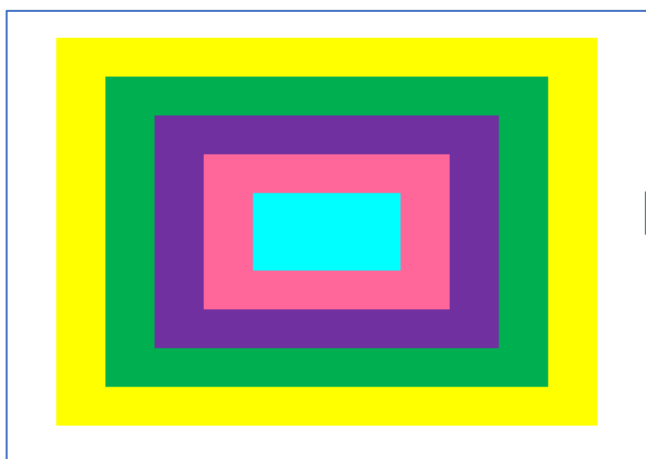
3.



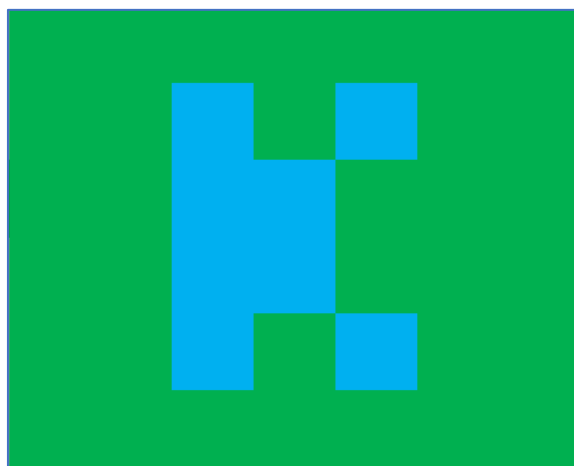
4.



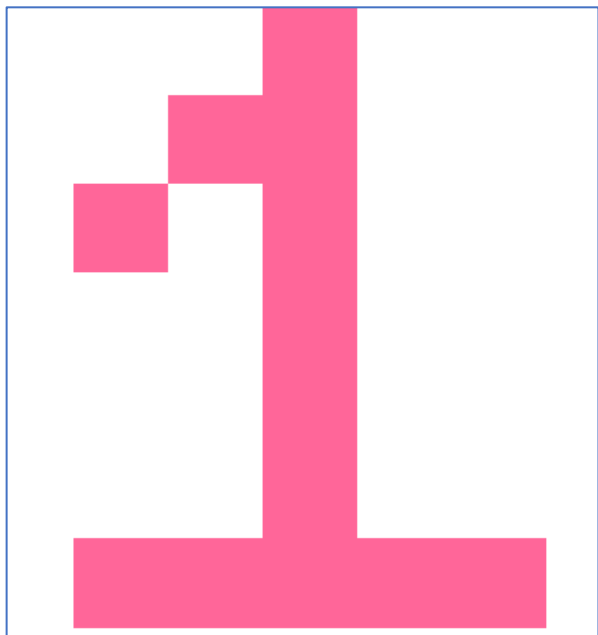
5.



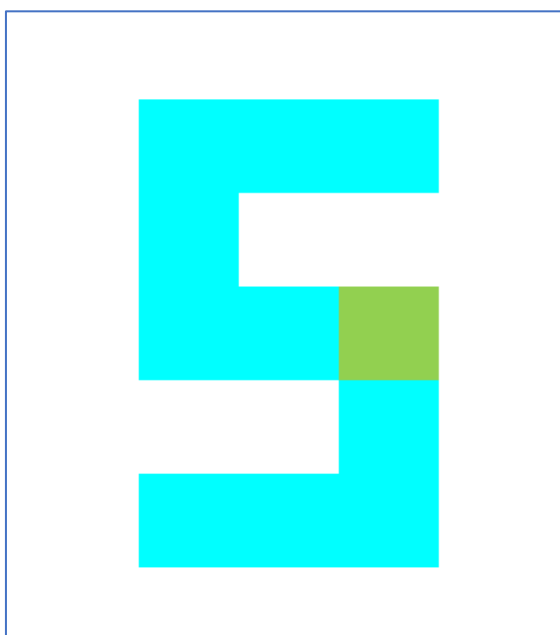
6.



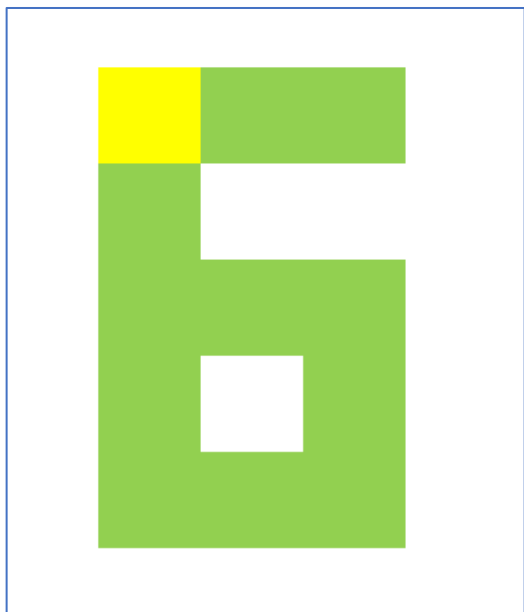
7.



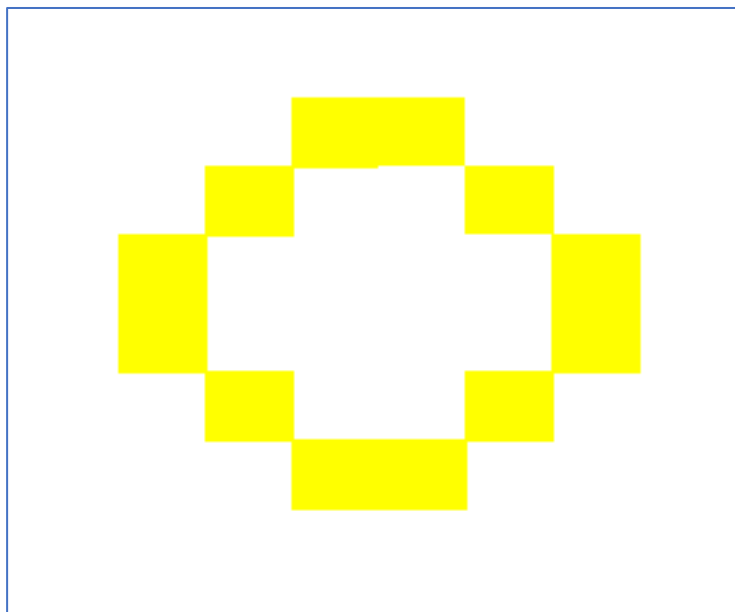
8.



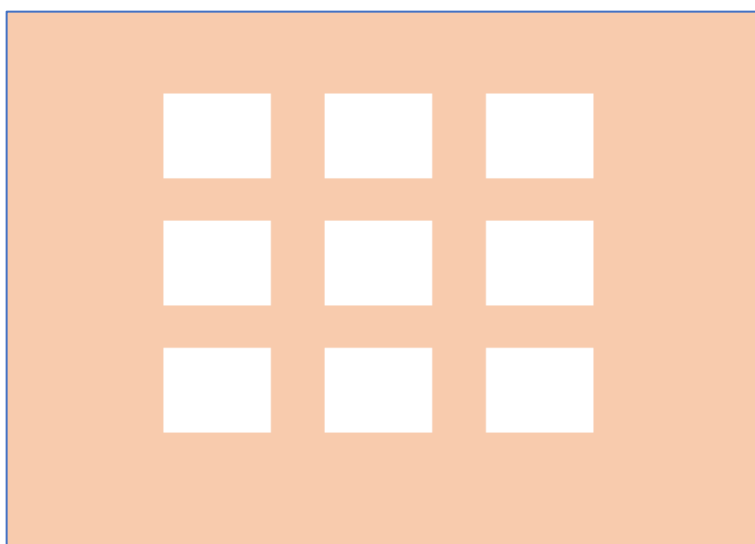
9.



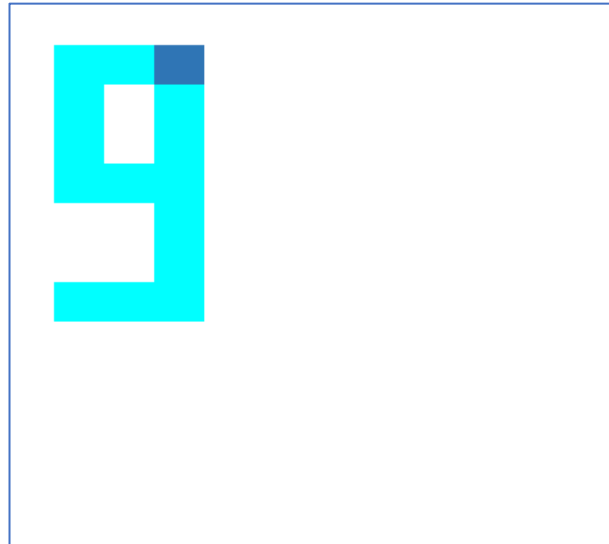
10.



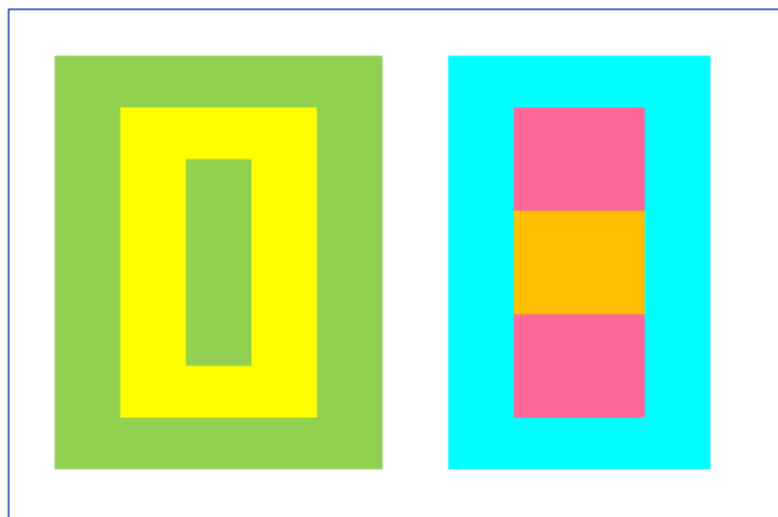
11.



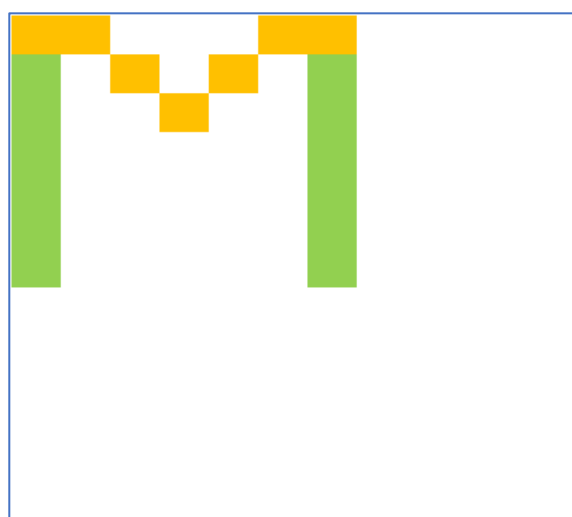
12.



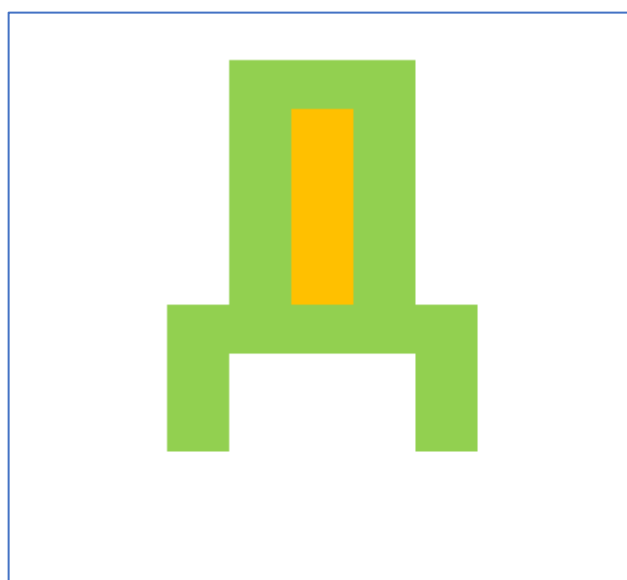
13.



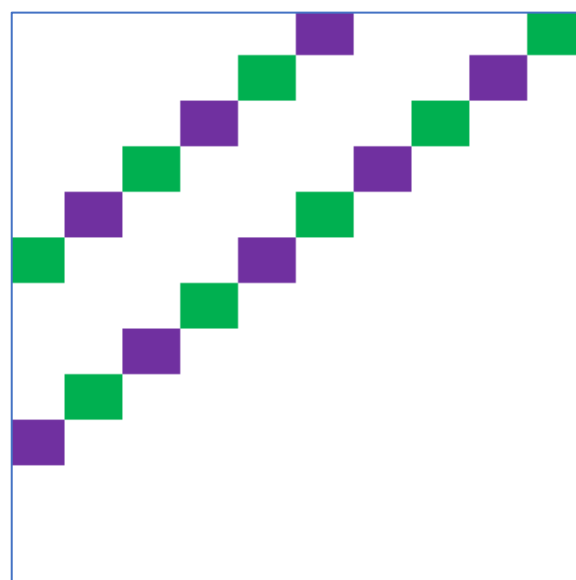
14.



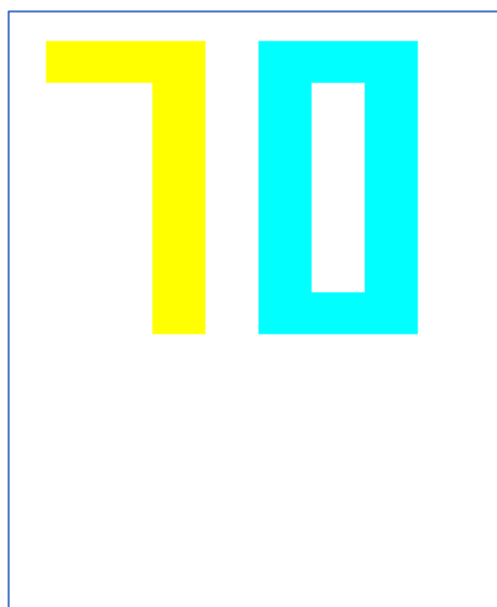
15.



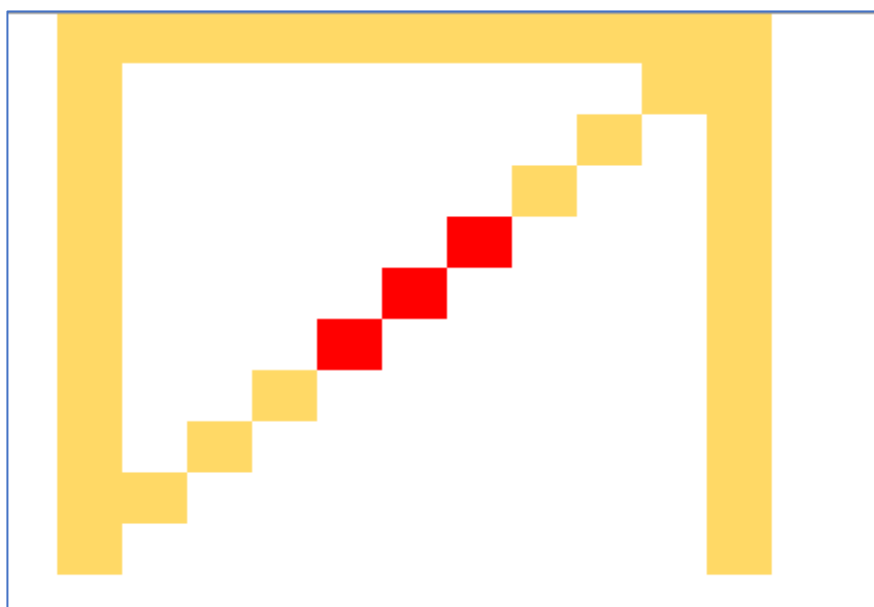
16.



17.



18.



КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение OpenGL и GLUT.
2. Приведите классификацию видов проекций и опишите особенности их реализации в OpenGL.
4. Объясните сущность процесса подключения библиотек OpenGL к проекту.
5. Охарактеризуйте, как описывается декартово пространство в приложении к окну.
6. Что означают параметры в команде `glViewport`?
7. Для чего предназначена функция `glutMainLoop()`?
8. Раскройте значение термина поле просмотра, и назовите функцию его определяющую.
9. Объясните принцип работы функции очистки буфера.
10. Опишите сущность и назначение двойной буферизации.

ЛИТЕРАТУРА:

1. "OpenGL Red Book" http://pm.samgtu.ru/sites/pm.samgtu.ru/files/materials/comp_graph/RedBook_OpenGL.pdf
2. <https://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf>
3. WebGL: Программирование трехмерной графики. Коичи Мацуда, Роджер Ли. / Пер. с англ. Киселев А. Н. – М.: ДМК Пресс, 2015. – 494 с.