| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |
| --- | --- |

**ФАКУЛЬТЕТ** ***ИУК "Информатика и управление"***

**КАФЕДРА** ***ИУК4 "Программная инженерия"***

**ЛАБОРАТОРНАЯ РАБОТА №1**

**«ВВЕДЕНИЕ В OPENGL»**

**ДИСЦИПЛИНА: «Компьютерная графика»**

| Выполнил: студент гр.ИУК5-41Б | | \_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_Шиндин А.О.\_\_\_\_)  (Подпись) (Ф.И.О.) |
| --- | --- | --- |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_Широкова Е.В.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2024

**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.

**Ход работы:**

Задание №1: Настроить перерисовку главного окна синим цветом.

Листинг программы:

#include "GL/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)

{

/\*

\* Функция `glClearColor` используется в библиотеке OpenGL для установки цвета,

\* который будет использоваться для очистки буфера цвета при вызове функции `glClear`.

\* Этот цвет задается в виде RGBA (красный, зеленый, синий, альфа) значений, где каждая

\* компонента цвета указывается в диапазоне от 0 до 1.

\* Прототип функции выглядит так:

? ```c

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

? ```

! - `red`, `green`, `blue` - значения компонент красного, зеленого и синего цвета соответственно.

! - `alpha` - значение компоненты альфа, которое указывает на прозрачность цвета

! (обычно используется для смешивания цветов и управления прозрачностью).

? ```c

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

\* Устанавливаем цвет очистки в синий (R=0, G=0, B=1) и полную непрозрачность (A=1)

? ```

\*/

glClearColor(0.0f, 0.0f, 1.0f, 1.0f); //Этот код устанавливает цвет очистки в синий цвет без прозрачности.

}

// Точка входа основной программы

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv); // Инициализация GLUT

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); // устанавливает режим отображения окна с использованием одиночного буфера рисования и цветовой модели RGB.

glutCreateWindow("Simple"); // создает окно с заголовком "Simple". Это окно будет использоваться для отображения графики

glutDisplayFunc(RenderScene); /\* устанавливает функцию RenderScene в качестве функции обратного вызова

для перерисовки содержимого окна. Таким образом, когда окно требуется перерисовать, будет вызываться

функция RenderScene, которая, как правило, содержит команды для рисования объектов сцены \*/

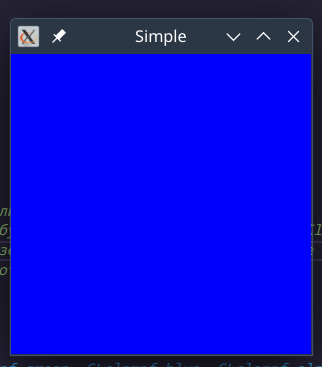
SetupRC(); // устанавливает цвет очистки экрана в синий цвет. Это происходит с помощью вызова функции glClearColor.

glutMainLoop(); // является последней функцией в вашей программе, так как она запускает бесконечный цикл обработки событий

return 0;

}

Результат:



Задание №1: Преобразуем программу. В центре окна с размерами 400 × 400 изобразим красный прямоугольник.

Листинг программы:

#include "GL/glut.h"

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

// Функция для отрисовки сцены

void RenderScene(void) {

// Очищаем буфер цвета текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Устанавливаем красный цвет для прямоугольника

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

// Рисуем прямоугольник в центре окна

glRectf(-0.5f, -0.5f, 0.5f, 0.5f);

// Принудительно выводим все команды в буфер OpenGL

glFlush();

}

// Устанавливаем состояние визуализации

void SetupRC(void) {

// Устанавливаем цвет очистки в синий

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

}

// Точка входа основной программы

int main(int argc, char\*\* argv) {

// Инициализация GLUT и создание окна

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutCreateWindow("Simple2");

// Устанавливаем функцию обратного вызова для отрисовки сцены

glutDisplayFunc(RenderScene);

// Устанавливаем начальное состояние визуализации

SetupRC();

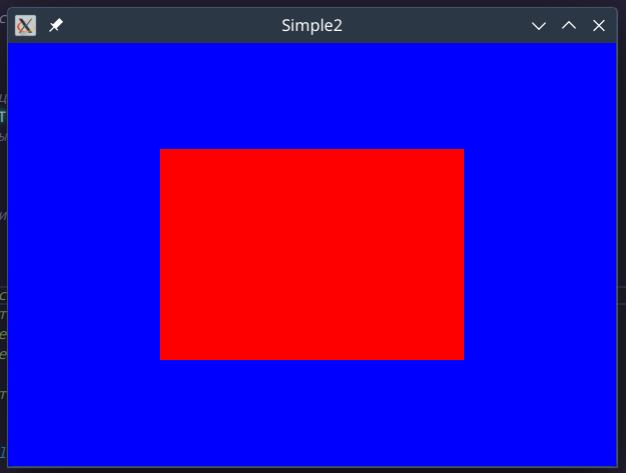
// Запускаем главный цикл GLUT

glutMainLoop();

return 0;

}

Результат:



Задание №3: Преобразовать программу так, чтобы прямоугольник всегда находился по центру окна (при любых растяжениях и сжатиях окна).

Листинг программы:

#include "GL/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();

}

void SetupRC(void) {

// Устанавливает в качестве цвета очистки розовый

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

}

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 ортографической системы координат

glOrtho(-100.0, 100.0, -100, 100.0, 1.0, -1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\*\* argv) {

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutCreateWindow("GLRect");

glutDisplayFunc(RenderScene);

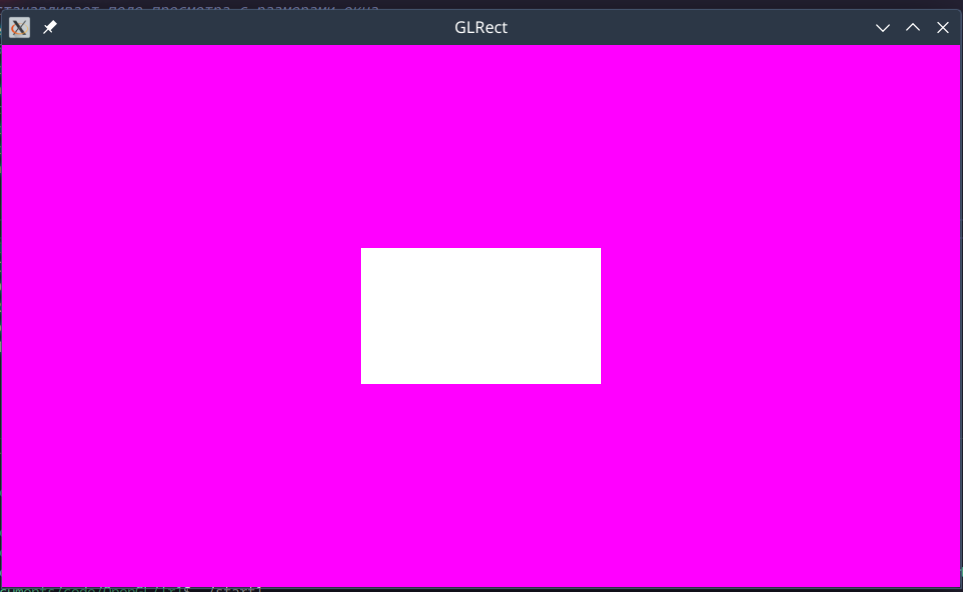
glutReshapeFunc(ChangeSize);

SetupRC();

glutMainLoop();

}

Результат:



Задание №4: Преобразовать программу так, чтобы получить центрированный прямоугольник.

Листинг программы:

#include "GL/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();

}

void SetupRC(void) {

// Устанавливает в качестве цвета очистки розовый

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

}

void ChangeSize(GLsizei w, GLsizei h)

{

GLfloat aspectRatio;

// Предотвращает деление на нуль

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 ортографической 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();

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutCreateWindow("GLRect");

glutDisplayFunc(RenderScene);

glutReshapeFunc(ChangeSize);

SetupRC();

glutMainLoop();

}

Результат:



Задание №5: Создать окно в позиции (100,100) и размерами 400х300 с зеленым цветом перерисовки.

Листинг программы:

#include "GL/glut.h"

// Функция отрисовки сцены

void RenderScene(void) {

// Очищаем окно зеленым цветом

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

glClear(GL\_COLOR\_BUFFER\_BIT);

// Принудительно выводим все команды в буфер OpenGL

glFlush();

}

void SetupRC(void) {

// Устанавливаем цвет очистки окна в зеленый

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

}

void ChangeSize(GLsizei w, GLsizei h) {

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливаем ортографическую проекцию с учетом новых размеров окна

glOrtho(0.0f, w, 0.0f, h, -1.0f, 1.0f);

// Устанавливаем текущую матрицу в модельно-видовую

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\*\* argv) {

// Инициализация GLUT и создание окна

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

// Устанавливаем позицию окна и его размеры

glutInitWindowPosition(100, 100);

glutInitWindowSize(400, 300);

glutCreateWindow("q5");

// Устанавливаем функцию отрисовки

glutDisplayFunc(RenderScene);

// Устанавливаем функцию изменения размеров окна

glutReshapeFunc(ChangeSize);

// Устанавливаем начальное состояние OpenGL

SetupRC();

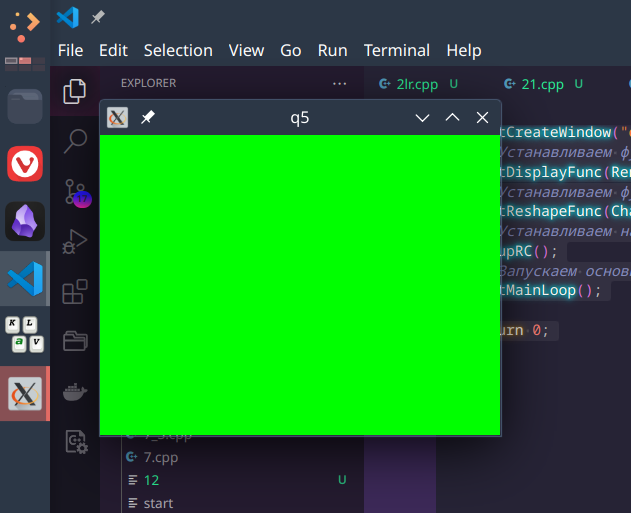
// Запускаем основной цикл GLUT

glutMainLoop();

return 0;

}

Результат:



Задание №6: Создать окно, в зависимости от размеров которого менялся бы его цвет (минимум 3 раза).

Листинг программы:

1. код реагирует на положение мыши:

#include "GL/glut.h"

// Переменные для хранения текущих координат курсора мыши

int mouseX = 0;

int mouseY = 0;

// Функция отрисовки сцены

void RenderScene(void) {

GLfloat r = (GLfloat)mouseX / glutGet(GLUT\_WINDOW\_WIDTH);

GLfloat g = (GLfloat)mouseY / glutGet(GLUT\_WINDOW\_HEIGHT);

GLfloat b = 1.0f - r;

// Очищаем окно новым цветом

glClearColor(r, g, b, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

// Принудительно выводим все команды в буфер OpenGL

glFlush();

}

// Функция для отслеживания перемещения курсора мыши

void OnMouseMove(int x, int y) {

mouseX = x;

mouseY = glutGet(GLUT\_WINDOW\_HEIGHT) - y; // Инвертируем координаты y

glutPostRedisplay(); // Вызываем перерисовку окна

}

// Устанавливаем начальное состояние OpenGL

void SetupRC(void) {

// Устанавливаем цвет очистки окна в черный

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

}

// Функция изменения размеров окна

void ChangeSize(GLsizei w, GLsizei h) {

// Обновляем матрицу проекции

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливаем ортографическую проекцию с учетом новых размеров окна

glOrtho(0.0f, w, 0.0f, h, -1.0f, 1.0f);

// Устанавливаем текущую матрицу в модельно-видовую

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\*\* argv) {

// Инициализация GLUT и создание окна

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600); // Задаем размеры окна

glutCreateWindow("Rainbow Window");

// Устанавливаем функцию отрисовки

glutDisplayFunc(RenderScene);

// Устанавливаем функцию изменения размеров окна

glutReshapeFunc(ChangeSize);

// Устанавливаем функцию отслеживания перемещения курсора мыши

glutPassiveMotionFunc(OnMouseMove);

// Устанавливаем начальное состояние OpenGL

SetupRC();

// Запускаем основной цикл GLUT

glutMainLoop();

return 0;

}

1. код реагирует на изменение размера окна:

#include "GL/glut.h"

// Переменные для хранения текущих координат курсора мыши

int mouseX = 0;

int mouseY = 0;

// Функция отрисовки сцены

void RenderScene(void) {

// Вычисляем компоненты цвета радуги в зависимости от положения курсора мыши

// GLfloat r = (GLfloat)mouseX / glutGet(GLUT\_WINDOW\_WIDTH);

// GLfloat g = (GLfloat)mouseY / glutGet(GLUT\_WINDOW\_HEIGHT);

// GLfloat b = 1.0f - r;

// Добавляем размеры окна в формулу цвета

GLfloat windowWidth = glutGet(GLUT\_WINDOW\_WIDTH);

GLfloat windowHeight = glutGet(GLUT\_WINDOW\_HEIGHT);

GLfloat r = windowWidth / 800.0f; // Нормализуем цвет по ширине окна

GLfloat g = windowHeight / 600.0f; // Нормализуем цвет по высоте окна

GLfloat b = (windowWidth + windowHeight) / ((800.f + 600.f)\*100.f);

// Очищаем окно новым цветом

glClearColor(r, g, b, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

// Принудительно выводим все команды в буфер OpenGL

glFlush();

}

// Функция для отслеживания перемещения курсора мыши

void OnMouseMove(int x, int y) {

mouseX = x;

mouseY = glutGet(GLUT\_WINDOW\_HEIGHT) - y; // Инвертируем координаты y

glutPostRedisplay(); // Вызываем перерисовку окна

}

// Устанавливаем начальное состояние OpenGL

void SetupRC(void) {

// Устанавливаем цвет очистки окна в черный

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

}

// Функция изменения размеров окна

void ChangeSize(GLsizei w, GLsizei h) {

// Обновляем матрицу проекции

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливаем ортографическую проекцию с учетом новых размеров окна

glOrtho(0.0f, w, 0.0f, h, -1.0f, 1.0f);

// Устанавливаем текущую матрицу в модельно-видовую

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\*\* argv) {

// Инициализация GLUT и создание окна

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600); // Задаем размеры окна

glutCreateWindow("Rainbow Window");

// Устанавливаем функцию отрисовки

glutDisplayFunc(RenderScene);

// Устанавливаем функцию изменения размеров окна

glutReshapeFunc(ChangeSize);

// Устанавливаем функцию отслеживания перемещения курсора мыши

glutPassiveMotionFunc(OnMouseMove);

// Устанавливаем начальное состояние OpenGL

SetupRC();

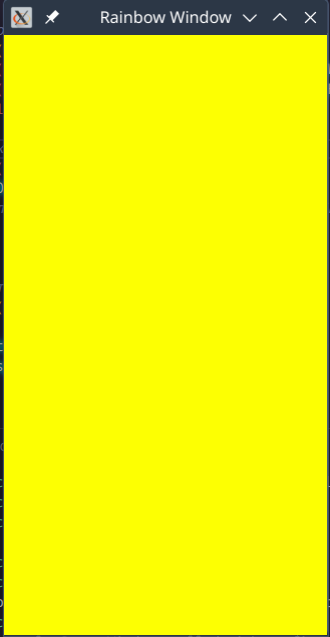
// Запускаем основной цикл GLUT

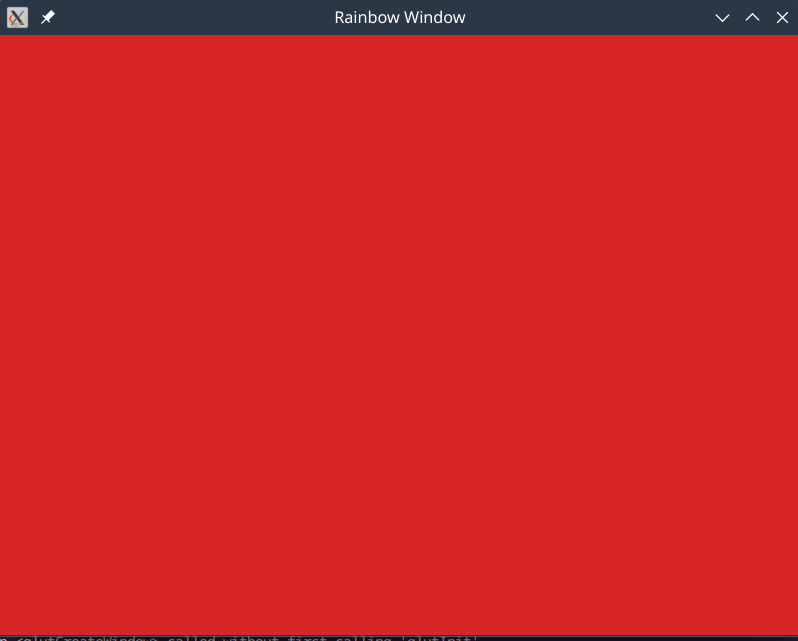
glutMainLoop();

return 0;

}

Результат:





Задание №7: 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)

Листинг программы:

#include "GL/glut.h"

#include <iostream>

const int rows = 7; // Количество строк

const int cols = 5; // Количество столбцов

const int cellSize = 100; // Размер клетки

// Перечисление для цветов

enum CellColor {

WHITE,

RED,

GREEN,

BLUE,

YELLOW

};

// Структура для представления клетки

struct Cell {

int x; // Координата x левого верхнего угла клетки

int y; // Координата y левого верхнего угла клетки

int width; // Ширина клетки

int height; // Высота клетки

CellColor color; // Цвет клетки

};

Cell cells[rows][cols]; // Двумерный массив клеток

// Функция отрисовки сцены

void RenderScene(void) {

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

// Отрисовываем клетки

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

Cell cell = cells[i][j];

switch (cell.color) {

case WHITE:

glColor3f(1.0f, 1.0f, 1.0f); // Белый цвет

break;

case RED:

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

break;

case GREEN:

glColor3f(0.0f, 1.0f, 0.0f); // Зеленый цвет

break;

case BLUE:

glColor3f(0.0f, 0.0f, 1.0f); // Синий цвет

break;

case YELLOW:

glColor3f(1.0f, 1.0f, 0.0f); // Желтый цвет

break;

}

glBegin(GL\_QUADS);

glVertex2i(cell.x, cell.y);

glVertex2i(cell.x + cell.width, cell.y);

glVertex2i(cell.x + cell.width, cell.y - cell.height);

glVertex2i(cell.x, cell.y - cell.height);

glEnd();

}

}

glFlush();

}

// Устанавливаем начальное состояние OpenGL

void SetupRC(void) {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Черный цвет фона

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, glutGet(GLUT\_WINDOW\_WIDTH), 0.0, glutGet(GLUT\_WINDOW\_HEIGHT));

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// Инициализируем координаты и размеры каждой клетки

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

cells[i][j].x = j \* cellSize;

cells[i][j].y = glutGet(GLUT\_WINDOW\_HEIGHT) - i \* cellSize;

cells[i][j].width = cellSize;

cells[i][j].height = cellSize;

cells[i][j].color = WHITE; // По умолчанию клетка белая

}

}

}

// Функция обработки событий мыши

void MouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

// Преобразуем координаты экрана в координаты окна

y = glutGet(GLUT\_WINDOW\_HEIGHT) - y;

// Проверяем, на какую клетку кликнули

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

if (x >= cells[i][j].x && x <= cells[i][j].x + cells[i][j].width &&

y >= cells[i][j].y - cells[i][j].height && y <= cells[i][j].y) {

// Меняем цвет клетки

switch (cells[i][j].color) {

case WHITE:

cells[i][j].color = RED;

break;

case RED:

cells[i][j].color = GREEN;

break;

case GREEN:

cells[i][j].color = BLUE;

break;

case BLUE:

cells[i][j].color = YELLOW;

break;

case YELLOW:

cells[i][j].color = WHITE;

break;

}

glutPostRedisplay(); // Перерисовываем сцену

return;

}

}

}

}

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(cols \* cellSize, rows \* cellSize);

glutInitWindowPosition(100, 100); // Позиция окна

glutCreateWindow("Clickable Grid");

glutDisplayFunc(RenderScene);

glutMouseFunc(MouseFunc);

SetupRC();

glutMainLoop();

return 0;

}

#include <GL/glut.h>

#include <algorithm>

#include <vector>

#include <random>

constexpr int initialRows = 5;

constexpr int initialCols = 7;

constexpr int cellSize = 50;

int rows = initialRows;

int cols = initialCols;

enum CellColor {

WHITE,

RED,

GREEN,

BLUE,

YELLOW,

NUM\_COLORS

};

struct Cell {

int x;

int y;

int size;

bool clicked;

CellColor color;

};

std::vector<std::vector<Cell>> cells;

void SetupRC() {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,

glutGet(GLUT\_WINDOW\_WIDTH),

glutGet(GLUT\_WINDOW\_HEIGHT), 0);

glMatrixMode(GL\_MODELVIEW);

}

void RenderScene() {

glClear(GL\_COLOR\_BUFFER\_BIT);

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

switch (cells[i][j].color) {

case WHITE:

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

break;

case RED:

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

break;

case GREEN:

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

break;

case BLUE:

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

break;

case YELLOW:

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

break;

}

glBegin(GL\_QUADS);

glVertex2i(cells[i][j].x, cells[i][j].y);

glVertex2i(cells[i][j].x + cells[i][j].size, cells[i][j].y);

glVertex2i(cells[i][j].x + cells[i][j].size, cells[i][j].y + cells[i][j].size);

glVertex2i(cells[i][j].x, cells[i][j].y + cells[i][j].size);

glEnd();

}

}

glFlush();

}

void Resize(int width, int height) {

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, width, height, 0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

rows = height / cellSize;

cols = width / cellSize;

cells.resize(rows);

for (int i = 0; i < rows; ++i) {

cells[i].resize(cols);

for (int j = 0; j < cols; ++j) {

cells[i][j].x = j \* cellSize;

cells[i][j].y = i \* cellSize;

cells[i][j].size = cellSize;

cells[i][j].clicked = false;

// Генерируем случайный цвет для каждой клетки

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<int> dis(0, CellColor::NUM\_COLORS - 1);

cells[i][j].color = static\_cast<CellColor>(dis(gen));

}

}

}

void MouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

int row = y / cellSize;

int col = x / cellSize;

cells[row][col].clicked = !cells[row][col].clicked;

glutPostRedisplay();

}

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 300);

glutInitWindowPosition(100, 100);

glutCreateWindow("Resizable Grid");

glutDisplayFunc(RenderScene);

glutReshapeFunc(Resize);

glutMouseFunc(MouseFunc);

SetupRC();

Resize(400, 300); // Вызываем Resize в начале для создания сетки с размерами окна по умолчанию

glutMainLoop();

return 0;

}

#include <GL/glut.h>

#include <vector>

constexpr int initialRows = 5;

constexpr int initialCols = 7;

constexpr int cellSize = 75;

int rows = initialRows;

int cols = initialCols;

enum CellColor {

WHITE,

RED,

GREEN,

BLUE,

YELLOW,

VIOLET,

NUM\_COLORS

};

struct Cell {

int x;

int y;

int size;

CellColor color;

};

std::vector<std::vector<Cell>> cells;

void SetupRC() {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT), 0);

glMatrixMode(GL\_MODELVIEW);

}

void RenderScene() {

glClear(GL\_COLOR\_BUFFER\_BIT);

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

switch (cells[i][j].color) {

case WHITE:

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

break;

case RED:

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

break;

case GREEN:

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

break;

case BLUE:

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

break;

case YELLOW:

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

break;

case VIOLET:

glColor3f(0.5f, 0.0f, 0.5f);

break;

}

glBegin(GL\_QUADS);

glVertex2i(cells[i][j].x, cells[i][j].y);

glVertex2i(cells[i][j].x + cells[i][j].size, cells[i][j].y);

glVertex2i(cells[i][j].x + cells[i][j].size, cells[i][j].y + cells[i][j].size);

glVertex2i(cells[i][j].x, cells[i][j].y + cells[i][j].size);

glEnd();

}

}

glFlush();

}

void Resize(int width, int height) {

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, width, height, 0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

rows = height / cellSize;

cols = width / cellSize;

cells.resize(rows);

for (int i = 0; i < rows; ++i) {

cells[i].resize(cols);

for (int j = 0; j < cols; ++j) {

cells[i][j].x = j \* cellSize;

cells[i][j].y = i \* cellSize;

cells[i][j].size = cellSize;

cells[i][j].color = WHITE;

}

}

}

void MouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

int row = y / cellSize;

int col = x / cellSize;

cells[row][col].color = static\_cast<CellColor>((cells[row][col].color + 1) % NUM\_COLORS);

glutPostRedisplay();

}

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(750\*1.4, 750);

glutInitWindowPosition(100, 100);

glutCreateWindow("Resizable Grid");

glutDisplayFunc(RenderScene);

glutReshapeFunc(Resize);

glutMouseFunc(MouseFunc);

SetupRC();

Resize(400, 300);

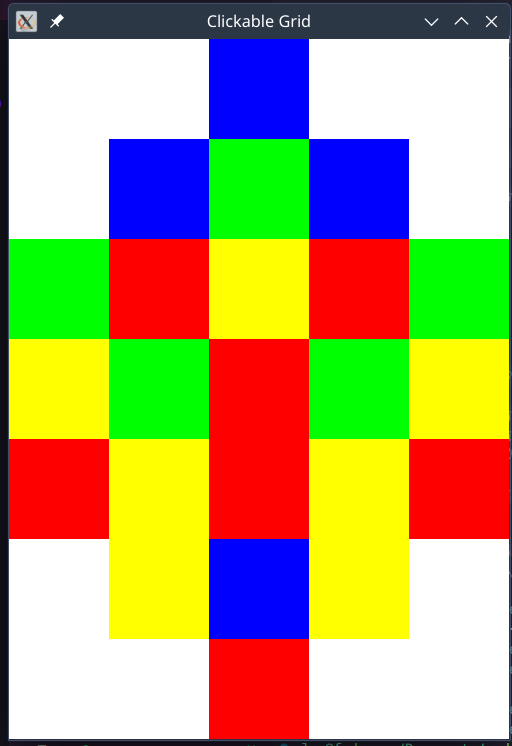
glutMainLoop();

return 0;

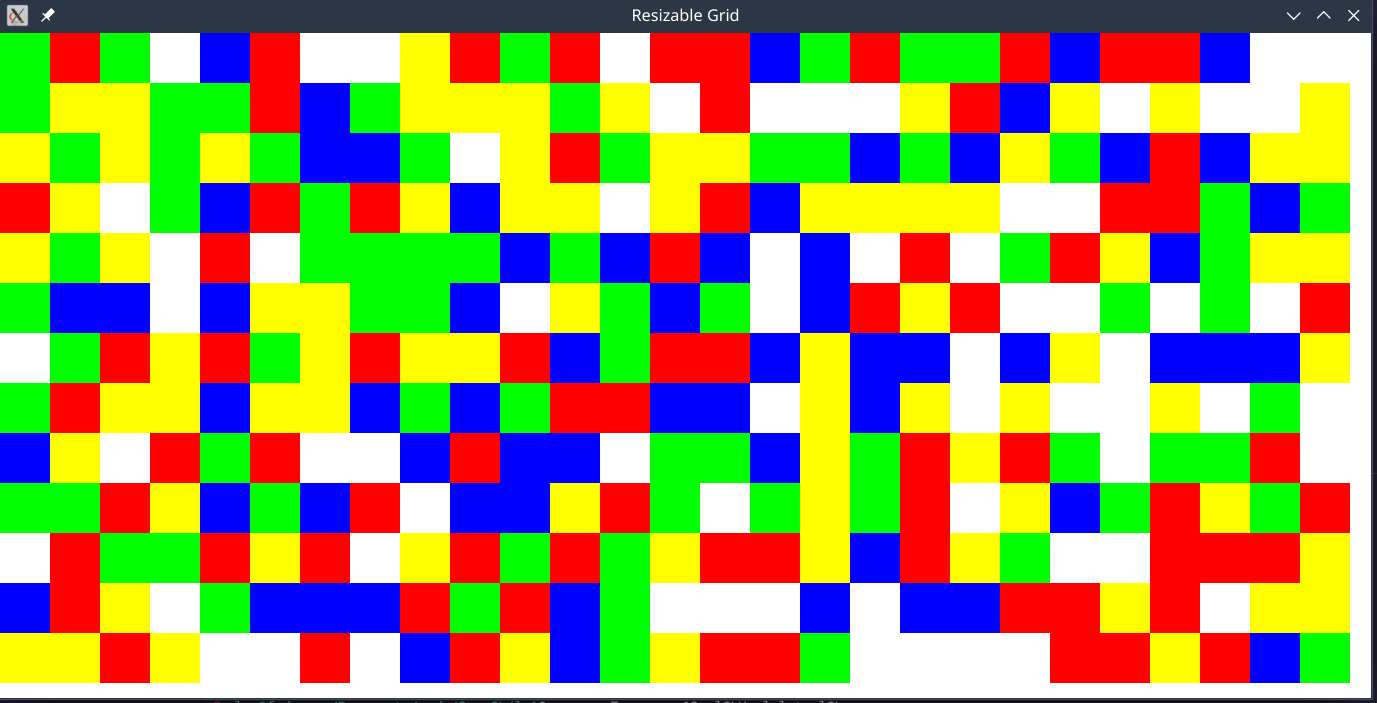
}

Результат:

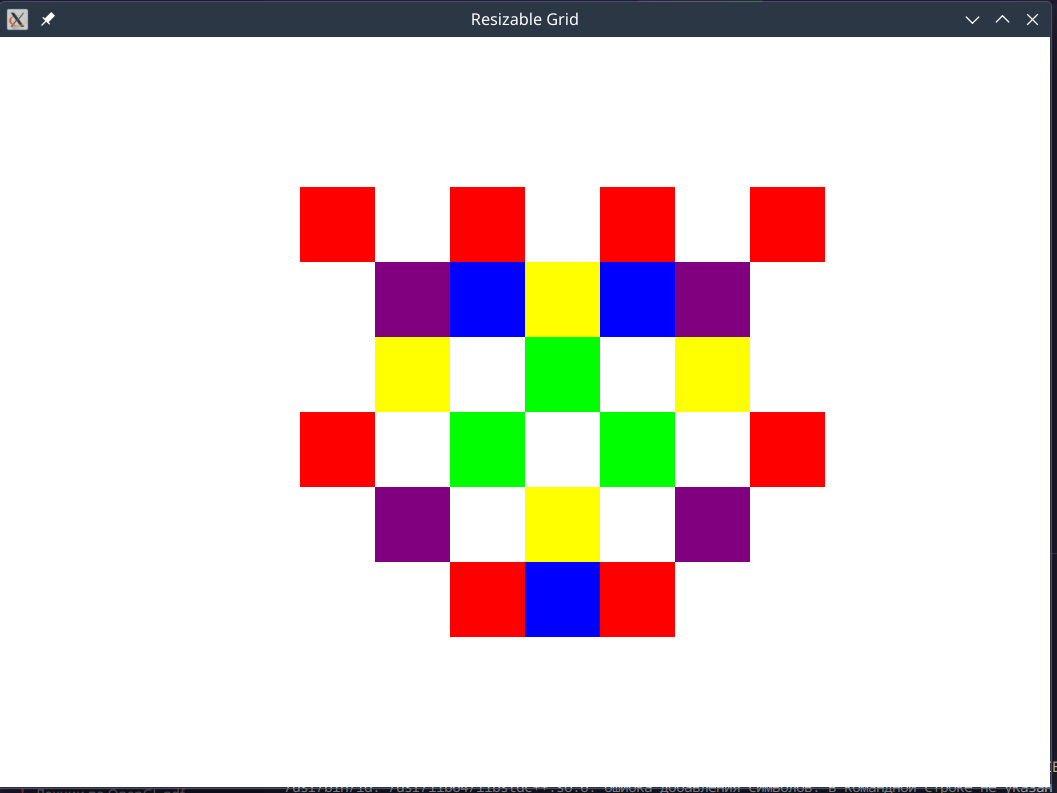
1. программа реагирует на действия пользователя, на клик пользователя меняет цвет:



2) программа не реагирует на действия пользователя, генерация происходит случайно:



3) программа реагирует на действия пользователя, на клик пользователя меняет цвет, предусмотрено расширение окна:



**Вывод:** в ходе выполнения лабораторной работы были сформированы практические навыки работы с проекционной матрицей средствами OpenGL. была изучена теория, а также выполнены задания согласно варианту.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. OpenGL (Open Graphics Library) - это кроссплатформенная библиотека для разработки графических приложений. GLUT (OpenGL Utility Toolkit) - это набор инструментов, облегчающих создание оконных приложений с использованием OpenGL.

2. Виды проекций классифицируются на ортографическую и перспективную. В ортографической проекции параллельные линии остаются параллельными, а в перспективной проекции они сходятся к одной точке - центру проекции. В OpenGL ортографическая проекция реализуется функцией glOrtho(), а перспективная - функцией gluPerspective().

3. Для подключения библиотек OpenGL к проекту нужно добавить заголовочные файлы и библиотеки в проект, а также указать компилятору путь к этим файлам и библиотекам.

4. Декартово пространство в приложении к окну описывается с помощью координат x и y, где (0,0) находится в левом нижнем углу окна, положительное направление оси x направлена вправо, а положительное направление оси y - вверх.

5. Параметры в команде glViewport указывают на размеры окна, в котором будет отображаться изображение OpenGL.

6. Функция glutMainLoop() предназначена для запуска основного цикла обработки событий GLUT, в котором происходит обработка событий ввода, рисование и другие операции.

7. Поле просмотра (viewing frustum) - это объем в пространстве, который виден из точки зрения наблюдателя. Функция glFrustum() определяет поле просмотра.

8. Принцип работы функции очистки буфера заключается в том, что она очищает содержимое указанного буфера (например, цвета, глубины, шаблоны) перед началом нового кадра.

9. Двойная буферизация - это техника рендеринга, при которой используется два буфера: передний и задний. Во время рендеринга изображение отображается в заднем буфере, а когда рендеринг завершен, содержимое заднего буфера переключается в передний буфер, что предотвращает появление эффекта разрыва изображения (мерцание).

**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.