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 ***ИУК-4 «Программная инженерия»***

Практическая работа №5

ОСНОВЫ НАЛОЖЕНИЯ ТЕКСТУР В OPENGL

ДИСЦИПЛИНА: «Компьютерная графика»

Выполнил: студент гр. ИУК5-42Б _____ Ли Р. В.
(Подпись) (Ф.И.О.)

Проверил: _____ _____
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга , 2023

Цели: формирование практических навыков по работе с текстурами средствами OpenGL, их наложению на освещенные объекты подверженные проекционному сокращению.

Задачи: понимать принципы наложения растровых изображений на геометрические объекты, уметь реализовывать наложение текстур с использованием возможностей OpenGL, научиться использовать наложение множественных текстур, уметь создавать фотореалистичные сцены (корректное освещение), на которых присутствуют текстурированные геометрические объекты.

Вариант 4

Задание 1

На основе листинга 1. Внести следующие изменения:

1) С помощью меню менять не менее 4 параметров в glTexParameteri
2) Добавить еще 3 текстуры и вывести их одновременно в виде:

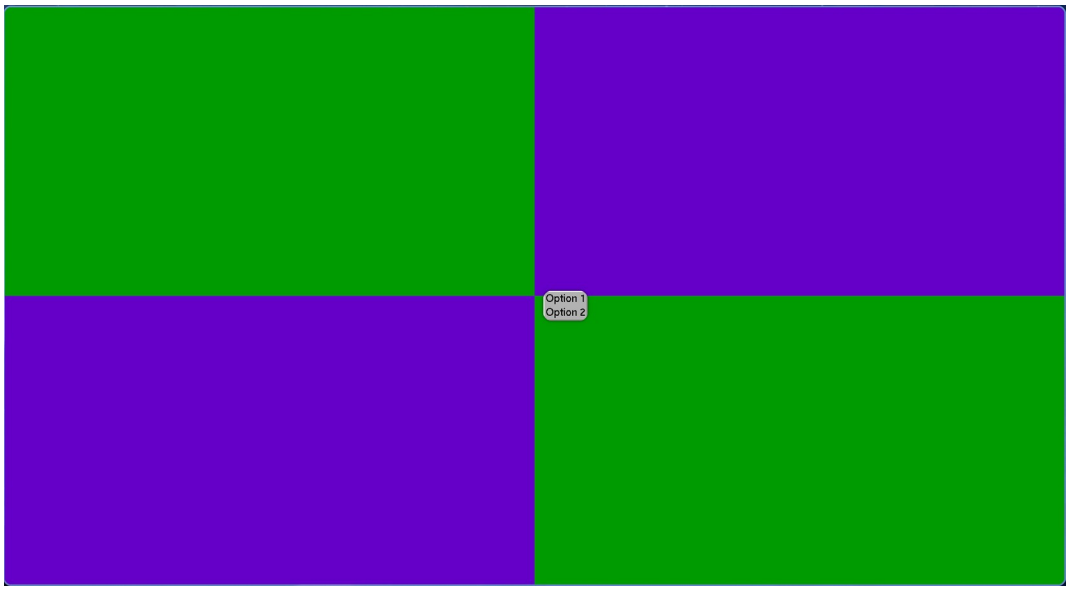
Листинг 1.1

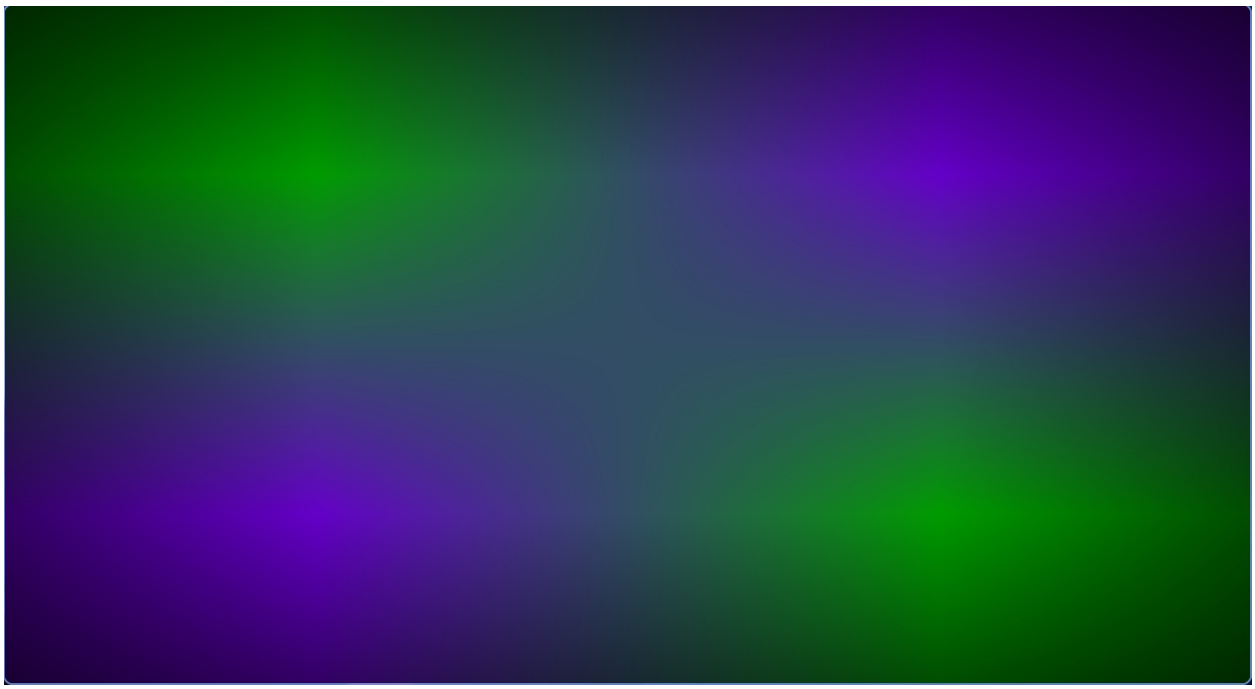
```
#include "GL/freeglut.h"
#include "GL/gl.h"
#include <GL/freeglut_std.h>
#include <cstring>
#include <iostream>

unsigned int texture;
// зададим массив координат квадрата
float vertex[] = {-1, -1, 0, 1, -1, 0, 1, 1, 0, -1, 1, 0};
// масив который хранит текстурные координаты для каждой вершины
```

```
float texCoord[] = {0, 0, 1, 0, 1, 1, 0, 1};
// процедура для создания квадрата и
void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, texture);
  // Активные цвета текстуры
  glColor3f(1, 1, 1);
  glPushMatrix();
  // Установка состояние для OpenGL, означающее, что мы будем использовать
  glEnableClientState(GL_VERTEX_ARRAY);
  // Установка состояние для OpenGL, означающее, что мы будем использовать
  glEnableClientState(GL_TEXTURE_COORD_ARRAY);
  // Установка массива вершинных координат квадрата
  glVertexPointer(3, GL_FLOAT, 0, vertex);
  // Установка массива текстурных координат
  glTexCoordPointer(2, GL_FLOAT, 0, texCoord);
  // Выводит примитивы по данным в массиве для квадрата
  glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
  glDisableClientState(GL_VERTEX_ARRAY);
  glDisableClientState(GL_TEXTURE_COORD_ARRAY);
  glPopMatrix();
  glutSwapBuffers();
}
void Text_Init() {
  // процедура для создания текстуры
  int width, height;
  width = 2;
  height = 2;
  // создаем двумерный массив 2 на 2 текселя
  struct {
    unsigned char r, g, b, a;
  } data[2][2];
  memset(data, 0, sizeof(data));
  // задаем цвет для каждого текселя структуры
  data[0][0].r = 100;
  data[0][0].b = 200;
  data[1][0].g = 155;
  data[0][1].g = 155;
  data[1][1].r = 100;
  data[1][1].b = 200;
  // Создадим имена текстур. 1-количество текстур.
  glGenTextures(1, &texture);
  // выбирает указанную текстуру как активную для наложения ее на объекты.
После
  glBindTexture(GL_TEXTURE_2D, texture);
  // Основные настройки текстуры
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
               GL_UNSIGNED_BYTE, data);
  // Отключаем активную текстуру
  glBindTexture(GL_TEXTURE_2D, 0);
}
void processMenu(int value) {
  switch (value) {
  case 0:
```

```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    // Handle menu option 0
    break;
  case 1:
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // Handle menu option 1
    break;
    // Add more cases for other menu items as needed
  }
  glutPostRedisplay();
}
void createMenu() {
  int menu = glutCreateMenu(processMenu);
  glutAddMenuEntry("Option 1", 0);
  glutAddMenuEntry("Option 2", 1);
  // Add more menu items as needed
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}
int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(600, 600);
  glutCreateWindow("Текстура-квадраты");
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glEnable(GL_DEPTH_TEST);
  Text_Init();
  createMenu();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

Результат:



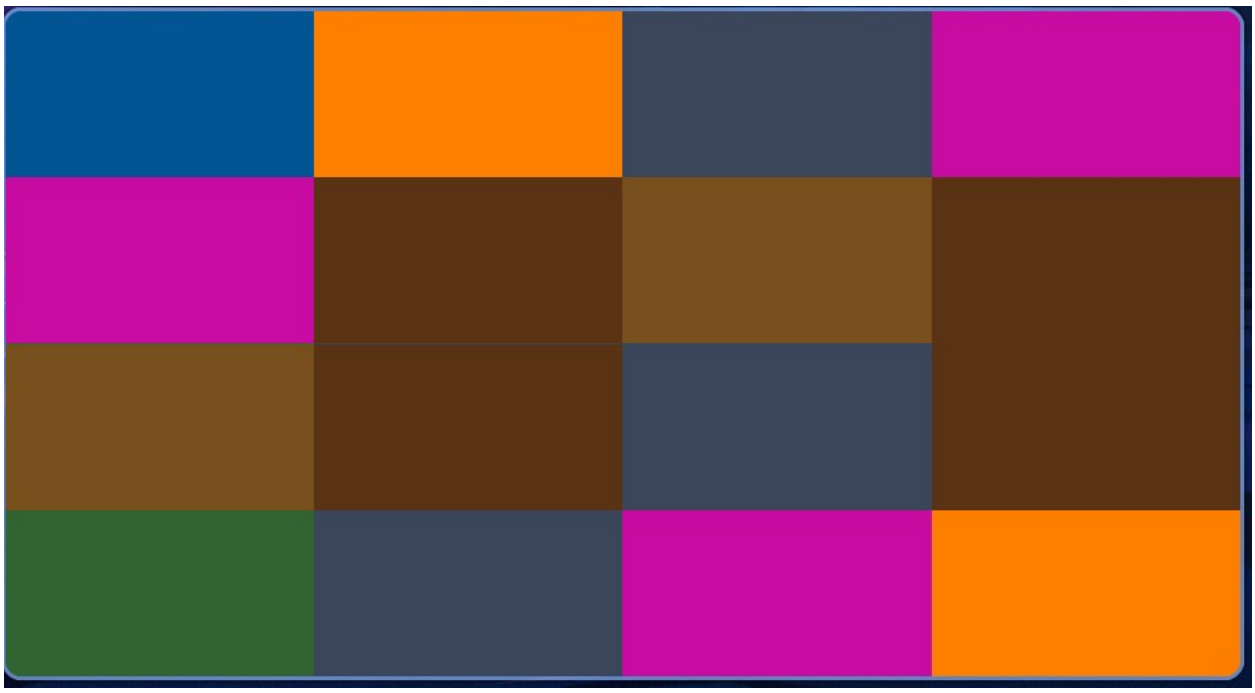


Листинг 1.2

```
#include "GL/freeglut.h"
#include "GL/gl.h"
#include <GL/freeglut_std.h>
#include <cstring>
#include <iostream>

const int texturesNum = 4;
unsigned int textures[texturesNum];

// зададим массив коордтнат квадрата
float vertex[][12] = {{-2, -2, 0, -2, 0, 0, 0, 0, 0, 0, -2, 0},
                      {0, 0, 0, 0, 2, 0, 2, 2, 0, 2, 0, 0},
                      {0, 0, 0, 2, 0, 0, 2, -2, 0, 0, -2, 0},
                      {0, 0, 0, -2, 0, 0, -2, 2, 0, 0, 2, 0}};
// масив который хранит текстурные координаты для каждой вершины
float texCoord[] = {0, 0, 2, 0, 2, 2, 0, 2};
// процедура для создания квадрата и
void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_TEXTURE_2D);
  for (int i = 0; i < texturesNum; i++) {
    glBindTexture(GL_TEXTURE_2D, textures[i]);
    // Активные цвета текстуры
    glColor3f(1, 1, 1);
    glPushMatrix();
    // Установка состояние для OpenGL, означающее, что мы будем использовать
    glEnableClientState(GL_VERTEX_ARRAY);
    // Установка состояние для OpenGL, означающее, что мы будем использовать
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    // Установка массива вершинных координат квадрата
    glVertexPointer(3, GL_FLOAT, 0, vertex[i]);
    // Установка массива текстурных координат
```

```
    glTexCoordPointer(2, GL_FLOAT, 0, texCoord);
    // Выводит примитивы по данным в массиве для квадрата
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
    glPopMatrix();
  }
  glutSwapBuffers();
}
void Text_Init() {
  // процедура для создания текстуры
  int width, height;
  width = 2;
  height = 2;
  // создаем двумерный массив 2 на 2 текселя
  char data[][4] = {{50, 100, 50}, {120, 80, 30}, {60, 70, 90}, {90, 50,
20}};
  // задаем цвет для каждого текселя структуры
  // Color variation 1
  // Создадим имена текстур. 1-количество текстур.
  glGenTextures(4, textures);
  // выбирает указанную текстуру как активную для наложения ее на объекты.
После
  for (int i = 0; i < texturesNum; i++) {
    glBindTexture(GL_TEXTURE_2D, textures[i]);
    // Основные настройки текстуры
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
                 GL_UNSIGNED_BYTE, data[i]);
    // Отключаем активную текстуру
    glBindTexture(GL_TEXTURE_2D, 0);
  }
}
int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(600, 600);
  glutCreateWindow("Текстура-квадраты");
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glEnable(GL_DEPTH_TEST);
  Text_Init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

Результат:



Задание 2

Наложить произвольную текстуру на геометрический объект.

- 4) Цилиндр

Листинг 2

```
#include <GL/freeglut_std.h>
#include <GL/gl.h>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#include <cstring>
#include <vector>

unsigned int texture;
float angle = 0;
std::vector<GLfloat> texCoords;
void display() {

  float radius = 2;
  float height = 4;
  float segments = 50;
  const float PI = 3.141592;
  float position[] = {0, -2, 0};
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glRotatef(angle, 0.0f, 1.0f, 0.0f);
  glEnable(GL_TEXTURE_2D);
  glEnableClientState(GL_TEXTURE_COORD_ARRAY);
  glBindTexture(GL_TEXTURE_2D, texture);
  glColor3f(1, 1, 1);
  glPushMatrix();
```

```
  glTranslatef(position[0], position[1], position[2]);


  // Draw the top and bottom circles
  glBegin(GL_TRIANGLE_FAN);
  glVertex3f(0.0f, 0.0f, 0.0f); // Center of bottom circle
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
    float z = radius * sin(theta);
    float s = static_cast<float>(i) /
              static_cast<float>(
                  segments); // Calculate texture coordinate in the S
direction

    // Bottom circle
    glTexCoord2f(s, 0.0f);
    glVertex3f(x, 0.0f, z);
  }
  glEnd();

  glBegin(GL_TRIANGLE_FAN);
  glVertex3f(0.0f, height, 0.0f); // Center of top circle
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
    float z = radius * sin(theta);
    float s = static_cast<float>(i) /
              static_cast<float>(
                  segments); // Calculate texture coordinate in the S
direction

    // Top circle
    glTexCoord2f(s, 1.0f);
    glVertex3f(x, height, z);
  }
  glEnd();


  glBegin(GL_QUAD_STRIP);
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
    float z = radius * sin(theta);
    float s = static_cast<float>(i) /
              static_cast<float>(
                  segments); // Calculate texture coordinate in the S
direction

    // Bottom circle
    glTexCoord2f(s, 0.0f);
    glVertex3f(x, 0.0f, z);
    // Top circle
    glTexCoord2f(s, 1.0f);
    glVertex3f(x, height, z);
  }
```

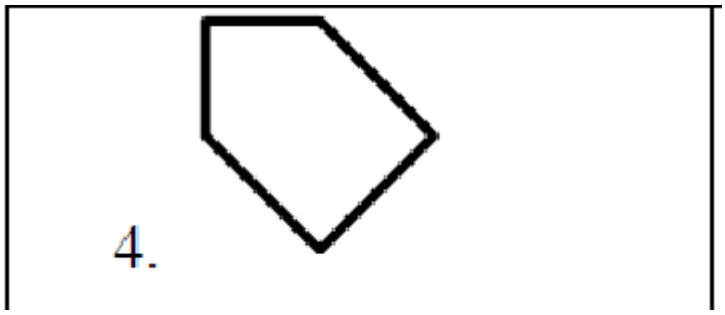
```
  glEnd();

  glPopMatrix();

  glDisableClientState(GL_TEXTURE_COORD_ARRAY);
  glutSwapBuffers();
}
void Game_Init() {
  int width, height, cnt;
  // загрузим данные картинки
  unsigned char *data = stbi_load("1.jpg", &width, &height, &cnt, 0);
  glGenTextures(1, &texture);
  glBindTexture(GL_TEXTURE_2D, texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
               cnt == 4 ? GL_RGBA : GL_RGB, GL_UNSIGNED_BYTE, data);
  glBindTexture(GL_TEXTURE_2D, 0);
  stbi_image_free(data);
}
void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  float aspectRatio = (float)w / (float)h;

  // Set up the perspective projection
  float distanceToObject = 10;
  float fov = 60.0f;                       // Field of view in degrees
  float nearPlane = 1.0f;                  // Near clipping plane distance
  float farPlane = distanceToObject + 100; // Far clipping plane distance

  gluPerspective(fov, aspectRatio, nearPlane, farPlane);
  gluLookAt(0.0f, 0.0f, distanceToObject, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
0.0f);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
void spinView(int key, int x, int y) {
  switch (key) {
  case GLUT_KEY_LEFT:
    angle += .5f;
    break;
  case GLUT_KEY_RIGHT:
    angle -= .5f;
    break;
  }
  glutPostRedisplay(); // Mark the current window as needing to be
redisplayed
}

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("Текстура-файл");
  glutInitWindowSize(500, 500);
  Game_Init();
```

```
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutSpecialFunc(spinView);
  glutMainLoop();
  return 0;
}
```

Результат:

Задание 3

Используя Листинг 3 создать коридор по форме указанной в варианте. Для каждой грани использовать свою собственную текстуру т. е. отдельный файл



Листинг 3

```
#include <GL/freeglut_std.h>
#include <GL/gl.h>
#define STB_IMAGE_IMPLEMENTATION
#include "GL/freeglut.h"
#include "stb_image.h"
#include <math.h>
// Rotation amounts
static GLfloat zPos = -60.0f;
static GLfloat xPos = 0;
unsigned int texture[3];
void ProcessMenu(int value) {
  GLint iLoop;
  for (iLoop = 0; iLoop < 3; iLoop++) {
    glBindTexture(GL_TEXTURE_2D, texture[iLoop]);
    switch (value) {
    case 0:
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      break;
    case 1:
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      break;
    case 2:
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                      GL_NEAREST_MIPMAP_NEAREST);
      break;
    case 3:
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                      GL_NEAREST_MIPMAP_LINEAR);
      break;
    case 4:
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                      GL_LINEAR_MIPMAP_NEAREST);
      break;
    case 5:
    default:
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                      GL_LINEAR_MIPMAP_LINEAR);
      break;
    }
  }
  glutPostRedisplay();
}
void SetupRC() {
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  // разрешаем наложение текстуры
  glEnable(GL_TEXTURE_2D);
  // функции преобразования цветов источника света, цвета образа текстуры,
  // цвета вершин примитивов и цвета конфигурации текстуры для получения
  // поверхности с наложенной на нее текстурой
  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
  int width, height, cnt;
  // генерируем 3 текстуры
  glGenTextures(3, texture);
  // flooring
  unsigned char *data1 = stbi_load("floor.jpg", &width, &height, &cnt, 0);
  glBindTexture(GL_TEXTURE_2D, texture[0]);
```

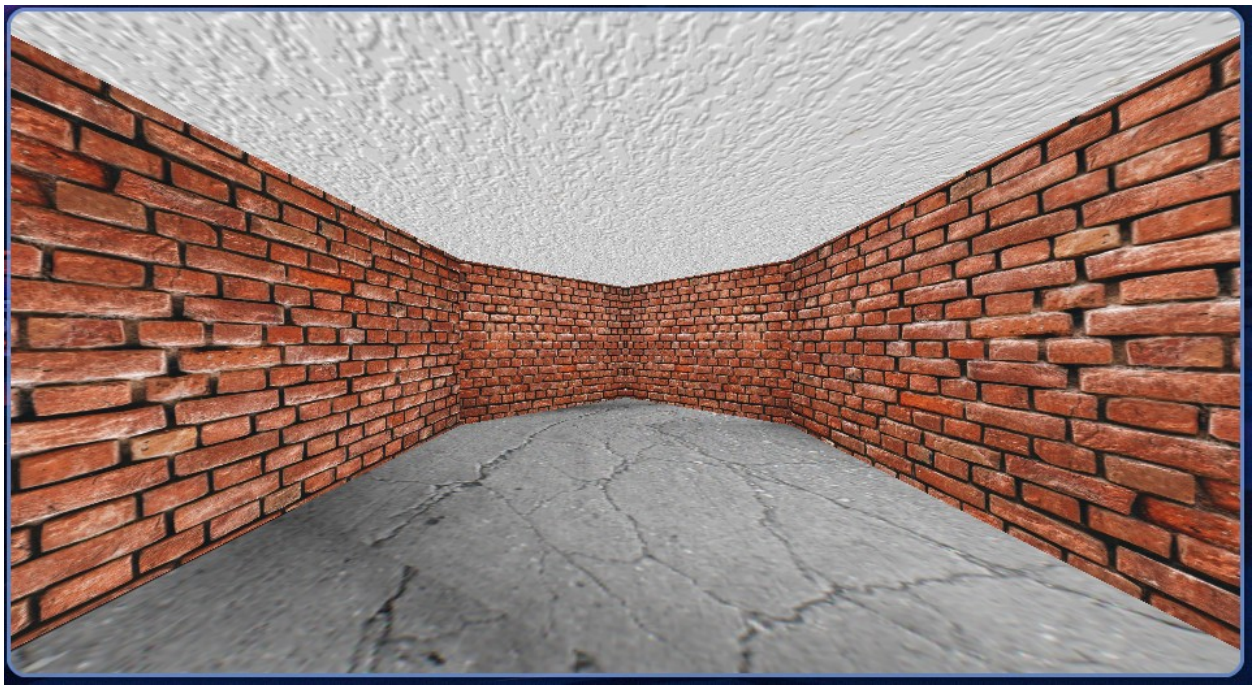
```
  // создает все изображения MIP-карты
  gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height, GL_RGB,
                    GL_UNSIGNED_BYTE, data1);
  // задаем параметры
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
  // теперь активных текстур 0
  glBindTexture(GL_TEXTURE_2D, 0);
  // освобождаем память
  stbi_image_free(data1);
  // walls
  unsigned char *data2 = stbi_load("wall.jpg", &width, &height, &cnt, 0);
  glBindTexture(GL_TEXTURE_2D, texture[1]);
  gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height, GL_RGB,
                    GL_UNSIGNED_BYTE, data2);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glBindTexture(GL_TEXTURE_2D, 0);
  stbi_image_free(data2);
  // ceiling
  unsigned char *data3 = stbi_load("ceiling.jpg", &width, &height, &cnt, 0);
  glBindTexture(GL_TEXTURE_2D, texture[2]);
  gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height, GL_RGB,
                    GL_UNSIGNED_BYTE, data3);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glBindTexture(GL_TEXTURE_2D, 0);
  stbi_image_free(data3);
}
void SpecialKeys(int key, int x, int y) {
  if (key == GLUT_KEY_UP)
    zPos += 1.0f;
  if (key == GLUT_KEY_DOWN)
    zPos -= 1.0f;
  if (key == GLUT_KEY_RIGHT)
    xPos -= 1.0f;
  if (key == GLUT_KEY_LEFT)
    xPos += 1.0f;
  // Refresh the Window
  glutPostRedisplay();
}
void ChangeSize(int w, int h) {
  GLfloat fAspect;
  // Prevent a divide by zero
  if (h == 0)
    h = 1;
  // Set Viewport to window dimensions
  glViewport(0, 0, w, h);
  fAspect = (GLfloat)w / (GLfloat)h;
  // Reset coordinate system
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  // Produce the perspective projection
```

```
  gluPerspective(90.0f, fAspect, 1, 120);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
void RenderScene(void) {
  GLfloat z;
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_TEXTURE_2D);
  glPushMatrix();
  glTranslatef(xPos, 0.0f, zPos);
  // пол
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  glBegin(GL_TRIANGLE_FAN);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(20.0f, -10.0f, -20);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(20.0f, -10.0f, 20);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(-20.0f, -10.0f, 20.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(-20.0f, -10.0f, -20.0f);

  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(0.0f, -10.0f, -40.0f);
  glEnd();

  // потолок
  glBindTexture(GL_TEXTURE_2D, texture[2]);
  glBegin(GL_TRIANGLE_FAN);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(20.0f, 10.0f, -20);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(20.0f, 10.0f, 20);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(-20.0f, 10.0f, 20.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(-20.0f, 10.0f, -20.0f);

  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(0.0f, 10.0f, -40.0f);
  glEnd();

  glEnable(GL_DEPTH_TEST);
  glBindTexture(GL_TEXTURE_2D, texture[1]);
  // front wall left
  glBegin(GL_QUADS);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(-20, 10.0f, -20);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(0.0f, 10.0f, -40.0f);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(0, -10.0f, -40.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(-20, -10.0f, -20);
  glEnd();
  // front wall right
  glBegin(GL_QUADS);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(20, 10.0f, -20);
```

```
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(0.0f, 10.0f, -40.0f);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(0, -10.0f, -40.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(20, -10.0f, -20);
  glEnd();
  // правая стена
  glBegin(GL_QUADS);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(20.0f, 10.0f, 20);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(20.0f, 10.0f, -20);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(20.0f, -10.0f, -20);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(20.0f, -10.0f, 20);
  glEnd();

  // левая стена
  glBegin(GL_QUADS);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(-20.0f, -10.0f, 20);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(-20.0f, -10.0f, -20.0f);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(-20.0f, 10.0f, -20.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(-20.0f, 10.0f, 20);
  glEnd();
  // back wall
  glBegin(GL_QUADS);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(20, 10.0f, 20);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(-20.0f, 10.0f, 20.0f);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(-20, -10.0f, 20.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(20, -10.0f, 20);
  glEnd();


  glPopMatrix();
  glutSwapBuffers();
}
int main(int argc, char *argv[]) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Tunnel");
  glutReshapeFunc(ChangeSize);
  glutSpecialFunc(SpecialKeys);
  glutDisplayFunc(RenderScene);
  // настройки меню
  glutCreateMenu(ProcessMenu);
  glutAddMenuEntry("GL_NEAREST", 0);
  glutAddMenuEntry("GL_LINEAR", 1);
  glutAddMenuEntry("GL_NEAREST_MIPMAP_NEAREST", 2);
```

```
  glutAddMenuEntry("GL_NEAREST_MIPMAP_LINEAR", 3);
  glutAddMenuEntry("GL_LINEAR_MIPMAP_NEAREST", 4);
  glutAddMenuEntry("GL_LINEAR_MIPMAP_LINEAR", 5);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  SetupRC();
  glutMainLoop();
  return 0;
}
```

Результат:



Вывод: были сформированы практические навыки по работе с текстурами средствами OpenGL, их галожению на освещенные объекты подверженные проекционному сокращению.