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 №2

«ОСНОВНЫЕ ПРИМИТИВЫ 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

Цели: формирование практических навыков по работе с графическими примитивами OpenGL.

Задачи: научиться устанавливать размеры наблюдаемого объема, изучить параметры функции glVertex, изучить основные параметры функции glBegin, сформировать понимание особенности использования функции glEnable с конкретными геометрическими примитивами, выяснить основы построения сплошных объектов.

Ход работы:

Задание №1:Используя все графические примитивы, а также маску, создать изображение.

Листинг программы:

```
#include "GL/glut.h"
#include "math.h"
#include <GL/gl.h>

void RenderScene(void) {
   glClear(GL_COLOR_BUFFER_BIT);

   GLubyte fire[128] = {   0x00, 0x00, 0x00, 0x00,
                           0x00, 0x00, 0x00, 0x00,
                           0x00, 0x00, 0x00, 0x00,
                           0x00, 0x00, 0x00, 0x00,
                           0x0F, 0xFF, 0xFF, 0xF0,
                           0x0F, 0xFF, 0xFF, 0xF0,
                           0x0F, 0xFF, 0xFF, 0xF0,
                           0x0F, 0xFF, 0xFF, 0xF0,
                           0x0F, 0xFF, 0xFF, 0xF0,
                           0x00, 0xFF, 0xFF, 0x00,
                           0x00, 0xFF, 0xFF, 0x00,
                           0x00, 0xFF, 0xFF, 0x00,
                           0x00, 0xFF, 0xFF, 0x00,
```

```
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0xFF, 0xFF, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00 };

glBegin(GL_QUADS);
glColor3f(1.0, 0.6, 0.0);
glVertex2f(-200.f, -25.f);
glVertex2f(200.0f, -25.0f);
glColor3f(1.f, 0.f, 1.f);
glVertex2f(200.f, 100.f);
glVertex2f(-200.f, 100.0f);
glEnd();
```

```
glBegin(GL_QUADS);
glColor3f(0.707f, 0.f, 0.707f);
glVertex2f(0.f, 0.f);
glVertex2f(0.f, 0.f);
glColor3f(0.05f, 0.704f, 0.606f);
glVertex2f(100.f, 100.f);
glVertex2f(-100.f, 100.0f);
glEnd();

glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex3f(-10, 5, 0.0);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(-65, 5, 1.0);
glColor3f(0.0, 0, 1.0);
glVertex3f(-30, 20, 1.0);
glEnd();


glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex3f(-15, -5, 0.0);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(-75, -5, 1.0);
glColor3f(0.0, 0, 1.0);
glVertex3f(-45, 30, 1.0);
glEnd();

glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
```

```
glVertex3f(-20, -15, 0.0);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(-110, -15, 1.0);
glColor3f(0.0, 0, 1.0);
glVertex3f(-65, 40, 1.0);
glEnd();

glColor3f(1.0f, 0.0f, 0.0f);
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(fire);
glBegin(GL_TRIANGLES);
// glColor3f(1.0, 0.0, 0.0);
glVertex3f(-30, -25, 0.0);
// glColor3f(0.0, 1.0, 0.0);
glVertex3f(-140, -25, 1.0);
// glColor3f(0.0, 0, 1.0);
glVertex3f(-85, 50, 1.0);
glEnd();
glDisable(GL_POLYGON_STIPPLE);

glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex3f(10, 5, 0.0);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(65, 5, 1.0);
glColor3f(0.0, 0, 1.0);
glVertex3f(30, 20, 1.0);
glEnd();

glBegin(GL_TRIANGLES);
```

```
glColor3f(1.0, 0.0, 0.0);
glVertex3f(15, -5, 0.0);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(75, -5, 1.0);
glColor3f(0.0, 0, 1.0);
glVertex3f(45, 30, 1.0);
glEnd();

glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex3f(20, -15, 0.0);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(110, -15, 1.0);
glColor3f(0.0, 0, 1.0);
glVertex3f(65, 40, 1.0);
glEnd();

glColor3f(1.0f, 0.0f, 0.0f);
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(fire);
glBegin(GL_TRIANGLES);
// glColor3f(1.0, 0.0, 0.0);
glVertex3f(30, -25, 0.0);
// glColor3f(0.0, 1.0, 0.0);
glVertex3f(140, -25, 1.0);
// glColor3f(0.0, 0, 1.0);
glVertex3f(85, 50, 1.0);
glEnd();
glDisable(GL_POLYGON_STIPPLE);
```

```
glBegin(GL_QUADS);
glColor3f(0.707f, 0.f, 0.707f);
glVertex2f(0.f, 0.f);
glVertex2f(0.f, 0.f);
glColor3f(0.05f, 0.704f, 0.606f);
glVertex2f(-50.f, -25.f);
glVertex2f(50.f, -25.0f);
glEnd();

glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++) {
    float red = fabs(cos((i + 120) * 3.1415926 / 180));
    float green = fabs(cos((i + 240) * 3.1415926 / 180));
    float blue = fabs(cos(i * 3.1415926 / 180));

    glColor3f(red, green, blue);
    float angle = i * 3.1415926 / 180;
    float x = 25.0 * cos(angle) + 0.0;
    float y = 25.0 * sin(angle) + 20;
    glVertex2f(x, y);
}
glEnd();

glBegin(GL_QUADS);
glColor3f(1.0, 0.5, 0.3);
glVertex2f(-200.f, -25.f);
glVertex2f(200.0f, -25.0f);
glColor3f(0.05f, 0.704f, 0.606f);
glVertex2f(200.f, -100.f);
glVertex2f(-200.f, -100.0f);
```

```
glEnd();

glBegin(GL_TRIANGLE_STRIP);
glColor3f(0.707f, 0.1f, 0.707f);
glVertex3f(-140.f, -100.f, 0.f);
glVertex3f(140.f, -100.f, 0.f);
glColor3f(0.05f, 0.704f, 0.606f);
glVertex3f(-50, -25, 0);
glVertex3f(50, -25, 0);
glEnd();

glBegin(GL_TRIANGLE_STRIP);
glColor3f(0.707f, 0.3f, 0.707f);
glVertex3f(-90.f, -100.f, 0.f);
glVertex3f(90.f, -100.f, 0.f);
glColor3f(0.05f, 0.704f, 0.606f);
glVertex3f(-40, -25, 0);
glVertex3f(40, -25, 0);
glEnd();

glColor3f(1.f, 1.f, 1.f);
glBegin(GL_TRIANGLE_STRIP);
glVertex3f(-3.f, -100.f, 0.f);
glVertex3f(3.f, -100.f, 0.f);
glColor3f(0.05f, 0.704f, 0.606f);
glVertex3f(-0.50, -25, 0);
glVertex3f(0.50, -25, 0);
glEnd();

glFlush();
```

```
}

void SetupRC(void) {
    glClearColor(0.f, 0.f, 0.0f, 1.0f);
}

void ChangeSize(GLsizei w, GLsizei h) {
    GLfloat aspectRatio;
    if (h == 0)
        h = 1;

    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        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);
    glutInitWindowSize(1000, 1000);
```

```
    glutCreateWindow("images");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();

    return 0;
}
```

Результат:

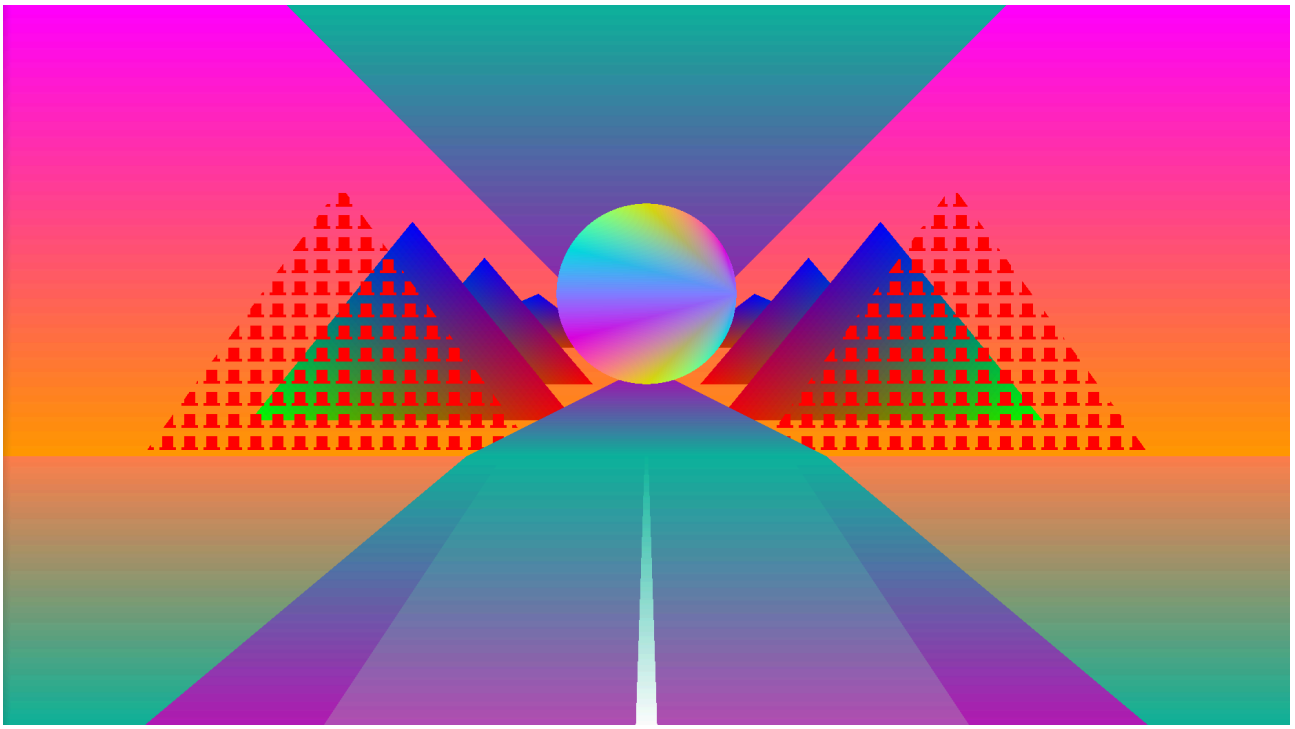

Рисунок 1 - результат выполнения первого задания

Задание №2: Создать анимацию с помощью изменения координат объекта, выполнить пульсирующее масштабирование квадрата относительно одной из его вершин.

Листинг программы:

```
#include "GL/glut.h"
#include "GL/gl.h"

GLfloat x1 = 0.0f;
```

```
GLfloat y1 = 0.0f;
GLfloat rsize = 40;
GLfloat scale = 1.0f;
GLfloat scaleCoef = 0.01f;

void RenderScene(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0f, 1.0f, 0.0f);
   glPushMatrix();
   glTranslatef(x1, y1, 0.0f); // Переносим начало координат
     glTranslatef(rsize, -rsize, 0.0f); // Перемещаем начало координат в верхнюю правую вершину квадрата
   glScalef(scale, scale, 1.0f); // Масштабируем квадрат
   glTranslatef(-rsize, rsize, 0.0f); // Возвращаем начало координат обратно
    glBegin(GL_QUADS); // Начинаем определение последовательности вершин для квадрата
   glVertex2f(-rsize, -rsize);
   glVertex2f(rsize, -rsize);
   glVertex2f(rsize, rsize);
   glVertex2f(-rsize, rsize);
   glEnd();
   glPopMatrix();
   glutSwapBuffers();
}

void TimerFunction(int value)
{
   scale += scaleCoef;
```

```
    if (scale > 1.5f || scale < 1.0f) { // Если масштаб больше 1.5 или меньше 1.0, меняем направление масштабирования
        scaleCoef = -scaleCoef;
    }
    glutPostRedisplay();
    glutTimerFunc(10, TimerFunction, 1); // Устанавливаем таймер с интервалом 10 миллисекунд
}

void ChangeSize(GLsizei w, GLsizei h)
{
    if (h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h) {
        gluOrtho2D(-100.0, 100.0, -100.0 * (GLfloat)h / (GLfloat)w, 100.0 * (GLfloat)h / (GLfloat)w);
    }
    else {
        gluOrtho2D(-100.0 * (GLfloat)w / (GLfloat)h, 100.0 * (GLfloat)w / (GLfloat)h, -100.0, 100.0);
    }
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

```
    glutCreateWindow("Pulsating Square");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
     glutTimerFunc(50, TimerFunction, 1); // Устанавливаем таймер с интервалом 50
миллисекунд
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Устанавливаем цвет фона
    glutMainLoop();
    return 0;
}
```

Результат:

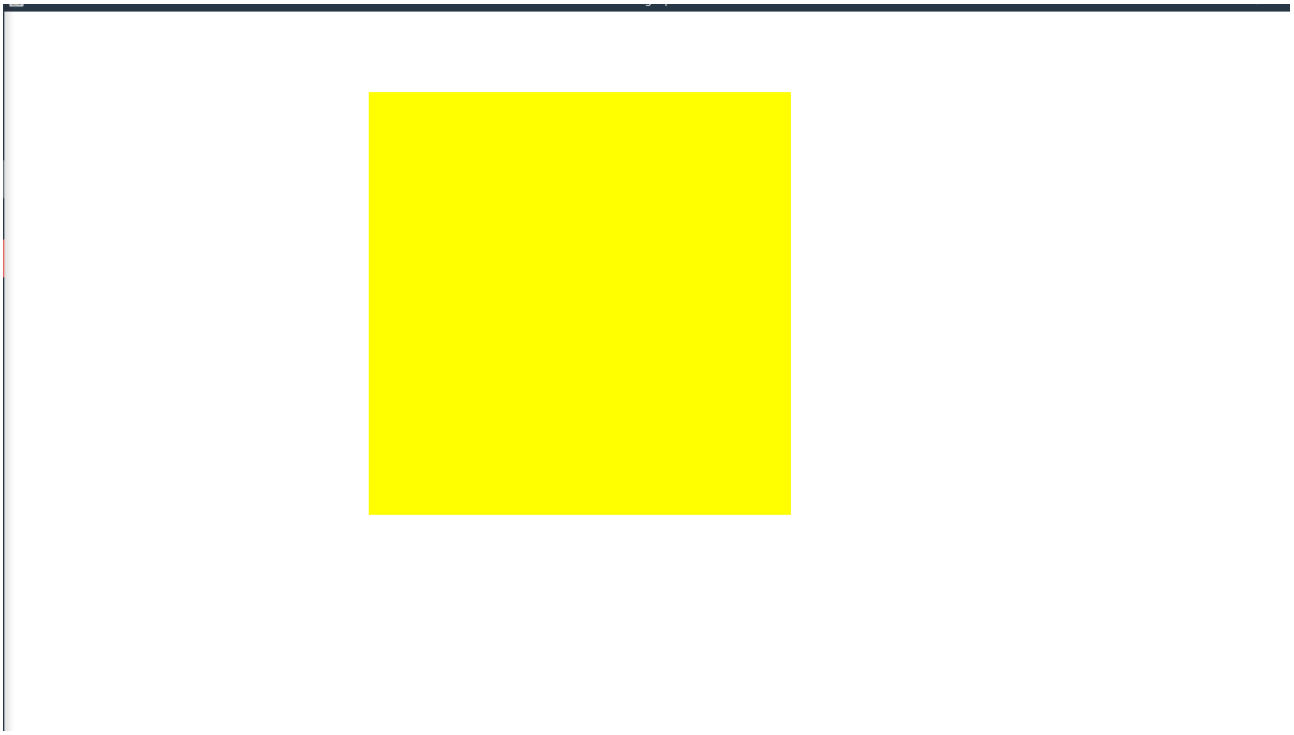

Рисунок 2 - пульсирующий квадрат

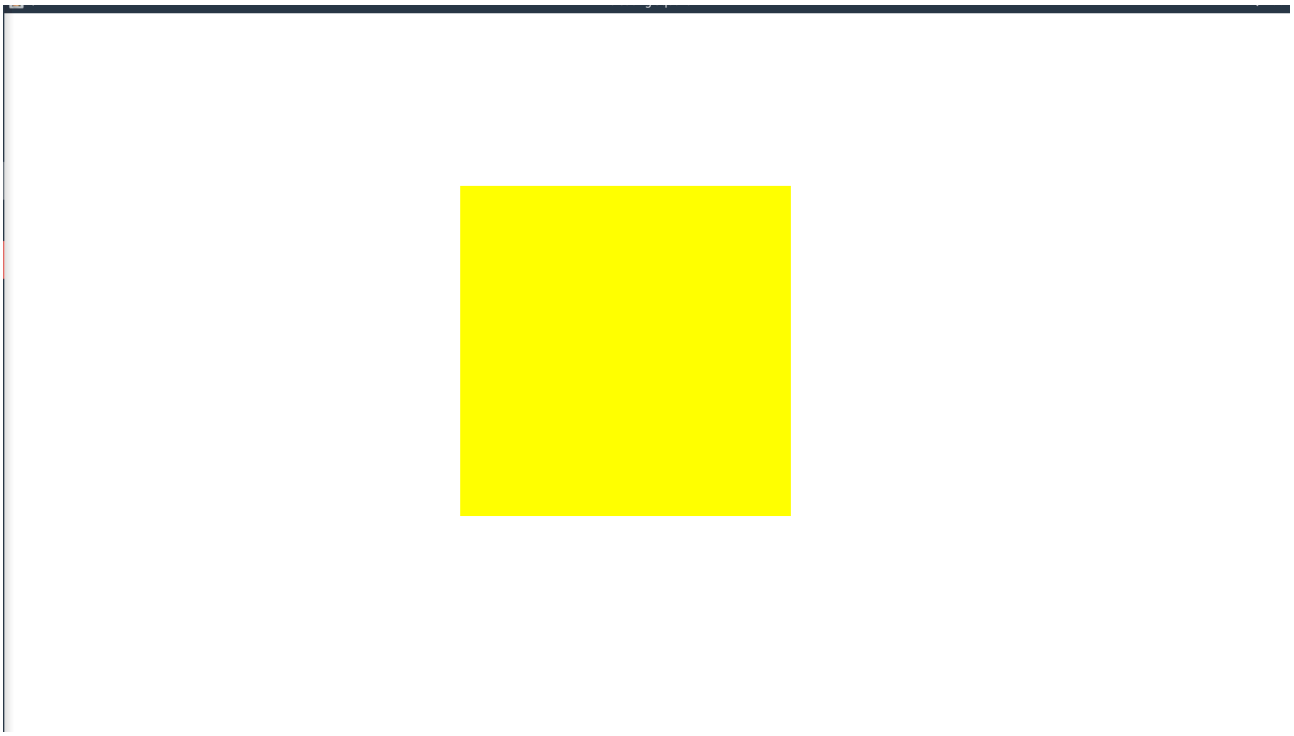

Рисунок 3 - пульсирующий квадрат

Вывод: в ходе выполнения лабораторной работы были сформированы практические навыки по работе с графическими примитивами OpenGL.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Известные виды геометрических примитивов в OpenGL:
- Точка (GL_POINTS)
- Линия (GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP)
- Треугольник (GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN)
- Квадрат (GL_QUADS, GL_QUAD_STRIP)

2. Тесселяция - это процесс разбиения сложной поверхности на простые геометрические фигуры, такие как треугольники или квадраты. Назначение тесселяции заключается в том, чтобы аппроксимировать сложные геометрические формы с помощью простых примитивов, что облегчает их отрисовку и обработку.

3. Для описания криволинейных структур средствами OpenGL можно использовать кривые Безье или сплайны. Например, для отображения кривых Безье можно использовать команды glBegin(GL_LINE_STRIP) и glVertex*() для задания контрольных точек.

4. Описание точки в OpenGL осуществляется с помощью команды glVertex*(), где * - это размерность точки (2, 3 или 4). Например, для задания точки в двумерном пространстве используется glVertex2f(x, y), где x и y - координаты точки.

5. Основные свойства лицевой грани многоугольника в OpenGL:
- Лицевая грань имеет ориентацию, определяемую порядком обхода вершин (по или против часовой стрелки).
- Лицевая грань может быть закрашена цветом или текстурой.
- Лицевая грань может иметь нормали, используемые для освещения.

6. Вееры (GL_TRIANGLE_FAN, GL_QUAD_STRIP) и ленты (GL_TRIANGLE_STRIP) используются для оптимизации рендеринга множества полигонов путем использования общих вершин. Это позволяет сократить количество вызовов API и улучшить производительность.

7. Многоугольники классифицируются по количеству вершин и форме. Например:
- Треугольники (3 вершины)
- Четырехугольники (4 вершины)
- Пятиугольники (5 вершин)

Допустимые многоугольники должны быть выпуклыми и не пересекаться сами с собой.

8. Декартов объем - это трехмерное пространство, ограниченное вдоль трех осей: X, Y и Z. Он часто используется для задания координат объектов в

трехмерной графике. Декартов объем можно задать, указав его ширину, высоту и глубину.

9. Установка размера точки и ширины линии осуществляется с помощью функций glPointSize() и glLineWidth() соответственно.

10. Переход от определения отдельных точек к многоугольникам можно осуществить, используя команду glBegin() для начала определения последовательности вершин многоугольника и последующие вызовы glVertex*() для задания координат каждой вершины.

11. Шаблонирование линии - это процесс повторения текстуры вдоль линии. Это позволяет создавать эффекты, такие как полосы и узоры на поверхности линии.

12. Отсечения в OpenGL могут влиять на производительность, поскольку они могут приводить к тому, что OpenGL будет рендерить только те части сцены, которые видимы для камеры. Отсечение выполняется для уменьшения количества объектов, которые не будут отображаться на экране, что может улучшить производительность рендеринга.

13. Цвет многоугольника в OpenGL задается с помощью функции glColor*(), где * - это спецификация цвета (например, glColor3f() для указания RGB-компонент цвета).

14. Узор-заполнитель (pattern-fill) - это текстура, которая используется для заполнения многоугольника или другой фигуры. Он задается с помощью текстурных координат и может быть повторен или масштабирован в соответствии с размерами фигуры.

15. Механизм проверки глубины в OpenGL используется для определения порядка отображения объектов на экране в трехмерной сцене. Это позволяет определить, какие пиксели должны быть отображены на переднем плане, а какие - на заднем, что важно для правильного отображения пересекающихся объектов.

ЛИТЕРАТУРА:

1. Боресков А.В. Основы работы с технологией CUDA / А.В. Боресков, А.А. Харламов - Издательство "ДМК Пресс", 2010. - 232 с. - ISBN 978-5-94074-578-5; ЭБС «Лань». - URL: https://e.lanbook.com/book/1260#book_name (23.12.2017).

2. Васильев С.А. OpenGL. Компьютерная графика : учебное пособие / С.А. Васильев. — Электрон. текстовые данные. — Тамбов: Тамбовский государственный технический университет, ЭБС АСВ, 2012. — 81 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/63931.html — ЭБС «IPRbooks», по паролю

3. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов/ Вольф Д. - Издательство "ДМК Пресс", 2015. - 368 с. - 978-5-97060-255-3; ЭБС «Лань». - URL: https://e.lanbook.com/book/ 73071#book_name (23.12.2017).

4. Гинсбург Д. OpenGL ES 3.0. Руководство разработчика/Д. Гинсбург, Б. Пурномо. - Издательство "ДМК Пресс", 2015. - 448 с. - ISBN 978-5-97060-256-0; ЭБС «Лань». - URL: https://e.lanbook.com/book/ 82816#book_name (29.12.2017).

5. Лихачев В.Н. Создание графических моделей с помощью Open Graphics Library / В.Н. Лихачев. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 201 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/39567.html

6. Забелин Л.Ю. Основы компьютерной графики и технологии трехмерного моделирования : учебное пособие/ Забелин Л.Ю., Конюкова О.Л., Диль О.В.— Новосибирск: Сибирский государственный университет телекоммуникаций и информатики, 2015.— 259 с.— Режим доступа: http://www.iprbookshop.ru/54792.— ЭБС «IPRbooks», по паролю

7. Папуловская Н.В. Математические основы программирования трехмерной графики : учебно-методическое пособие / Н.В. Папуловская. — Электрон. текстовые данные. — Екатеринбург: Уральский федеральный университет, 2016. — 112 c. — 978-5-7996-1942-8. — Режим доступа: http://www.iprbookshop.ru/68345.html

8. Перемитина, Т.О. Компьютерная графика : учебное пособие / Т.О. Перемитина ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск : Эль Контент, 2012. - 144 с. : ил.,табл., схем. - ISBN 978-5-4332-0077-7 ; - URL: http//biblioclub.ru/index.php?page=book&id=208688 (30.11.2017).