Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 7

по курсу «Алгоритмы компьютерной графики»

Студент группы ИУ9-41Б Утебаева М. Б.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

Оптимизация приложения OpenGL, созданного в рамках предыдущей лабораторной работы, на основе выбора наиболее эффективных методик.

Нужно:

- изучить эффективные приемы организации приложений и оптимизации вызовов OpenGL.
- использовать дисплейные списки и массивы вершин и еще 2 любые различные оптимизации(в сумме минимум 4 оптимизации).

Оценка применимости выбранного метода оптимизации приложения OpenGL должна осуществляться на основании измерения производительности. Результаты замеров оформить в табличном виде.

2 Основная теория

- **Дисплейные списки:** дисплейный список – это один из популярных и довольно простых в реализации методов оптимизации, он позволяет скомпилировать последовательности команд OpenGL заранее, а это, в свою очередь, снижает нагрузку на процессор и ускоряет генерацию сложных объектов.
- **Буферы вершин:** использование буфера позволяет хранить объекты в видеопамяти, что уменьшает затраты на передачу данных межд ЦП и ГП.
- **Хранение текстур:** текстуры довольно затратны, поэтому изменение параметров их загрузки и хранения может увеличить скорость обработки фигур.
- **Источники света:** при оптимальной настройке источников света можно увеличить производительность, если установить источник света с помощью координат, то это повлияет на скорость расчёта и уменьшит вычислительную нагрузку.

```
#задание нового дисплейного списка
display_list = glGenLists(1)
glNewList(display_list, GL_COMPILE)
#удаление дисплейного списка
glDeleteLists(display_list, 1)
#вызов дисплейного листа
glCallList(display_list)
#создание буффера вершин
vbo = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBufferData(GL_ARRAY_BUFFER, np.array(vert, dtype=np.float32), GL_STATIC_DRAW)
```

3 Практическая реализация

```
import glfw
from OpenGL.GL import *
from math import cos, sin, sqrt, asin, pi
from PIL import Image
import random
import time
import numpy as np
```

```
alpha, beta = 0.0, 0.0
vector_speed, speed = 0.009, 0.0
fill = False
is_move, is_light, is_tek = 0, 0, 0
R, G, B = 155, 456, 89
last_time, cnt_frame = 0.0, 0.0
display_list, vbo = None, None
vert = []
fps = []

sides = [
    (1/2, 1/2, 8**(1/4)/4),#A
    (0, sqrt(2)/2, -8**(1/4)/4),#E
    (-1/2, 1/2, 8**(1/4)/4),#B
    (-sqrt(2)/2, 0, -8**(1/4)/4),#H
    (-1/2, -1/2, 8**(1/4)/4),#D
    (0, -sqrt(2)/2, -8**(1/4)/4),#F
    (1/2, -1/2, 8**(1/4)/4),#C
    (sqrt(2)/2, 0, -8**(1/4)/4)#G
]
root1 = [
    (1/2, 1/2, 8**(1/4)/4),#A
    (-1/2, 1/2, 8**(1/4)/4),#B
    (-1/2, -1/2, 8**(1/4)/4),#D
    (1/2, -1/2, 8**(1/4)/4)#C
]
root2 = [
    (sqrt(2)/2, 0, -8**(1/4)/4),#G
    (0, sqrt(2)/2, -8**(1/4)/4),#E
    (-sqrt(2)/2, 0, -8**(1/4)/4),#H
    (0, -sqrt(2)/2, -8**(1/4)/4),#F
]
tek_list = [
    (0, 0),
    (0, 1),
    (1, 0),
    (1, 1)
]


def compute_norm(a, b, c):
    x0, y0, z0 = a
    x1, y1, z1 = b
    x2, y2, z2 = c
    ux, uy, uz = [x1 - x0, y1 - y0, z1 - z0]
```

```
    vx, vy, vz = [x2 - x0, y2 - y0, z2 - z0]
    normal = [uy * vz - uz * vy, uz * vx - ux * vz, ux * vy - uy * vx]
    l = sqrt(normal[0] * normal[0] + normal[1] * normal[1] + normal[2] * normal[2])
    normal[0] /= l
    normal[1] /= l
    normal[2] /= l
    return normal


def move():
    global vector_speed, speed
    speed -= vector_speed
    if speed < -1 or speed > 1:
        vector_speed = -vector_speed


def load_tex():
    glEnable(GL_TEXTURE_2D)
    glBindTexture(GL_TEXTURE_2D, glGenTextures(1))

    image = Image.open('/Users/molivka/vs/iu9-graphics/laba7/ruby.jpeg')
    image = image.transpose(Image.FLIP_TOP_BOTTOM)
    img_data = image.convert("RGBA").tobytes()

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    #UNSIGNED_BYTE - оптимизация
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, image.width, image.height, 0, GL_RGBA, GL_UNSIGNED_BYTE, im


def light():
    glShadeModel(GL_SMOOTH)
    glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE)
    glEnable(GL_NORMALIZE)

    #позиция источника света - оптимизация
    light_pos = [10.0, 10.0, 10.0, 0.0]
    glLightfv(GL_LIGHT0, GL_POSITION, light_pos)

    glLightfv(GL_LIGHT0, GL_AMBIENT, [0, 0, 0, 1])
    glLightfv(GL_LIGHT0, GL_DIFFUSE, [1, 1, 1, 1])
    glLightfv(GL_LIGHT0, GL_SPECULAR, [1, 1, 1, 1])

    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 180.0)
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.5)
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.05)
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.05)
```

```
    glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, 0)

    ambient = [0.0, 0.0, 0.0, 0.0]
    diffuse = [0.4, 0.7, 0.2]
    specular = [1.0, 1.0, 1.0, 1.0]
    shininess = [1]
    emission = [0.0, 0.0, 0.0, 0.0]

    glMaterialfv(GL_FRONT, GL_AMBIENT, ambient) #цвет фонового отражения материала
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse) #цвет рассеянного отражения материала
    glMaterialfv(GL_FRONT, GL_SPECULAR, specular) #цвет зеркального отражения материала
    glMaterialfv(GL_FRONT, GL_SHININESS, shininess) #степень в формуле зеркального отражения
    glMaterialfv(GL_FRONT, GL_EMISSION, emission) #цвет, испускаемый объектом


def create_display_list():
    global R, G, B, display_list
    #дисплейный список - оптимизация
    display_list = glGenLists(1)
    glNewList(display_list, GL_COMPILE)

    glBegin(GL_POLYGON)
    #верхняя
    if (1 - is_tek):
        glColor3f(R / 255, G / 255, B / 255)
    for i in range(len(root1)):
        if is_tek:
            glTexCoord2f(tek_list[i][0], tek_list[i][1])
        glVertex3f(root1[i][0], root1[i][1], root1[i][2])
    normal = compute_norm(root1[0], root1[1], root1[2])
    glNormal3f(normal[0], normal[1], normal[2])
    glEnd()
    #стороны
    glBegin(GL_TRIANGLES)
    for i in range(8):
        if is_tek:
            glTexCoord2f(tek_list[0][0], tek_list[0][1])
        else:
            glColor3f(R / 255, G / 255, B / 255)
        a, b, c = sides[i]
        glVertex3f(a, b, c)
        if is_tek:
            glTexCoord2f(tek_list[1][0], tek_list[1][1])
        a, b, c = sides[(i + 1) % 8]
        glVertex3f(a, b, c)
        if is_tek:
            glTexCoord2f(tek_list[2][0], tek_list[2][1])
```

```
            a, b, c = sides[(i + 2) % 8]
            glVertex3f(a, b, c)
            if is_tek:
                glTexCoord2f(tek_list[3][0], tek_list[3][1])
            normal = compute_norm(sides[i], sides[(i + 1) % 8], sides[(i + 2) % 8])
            glNormal3f(normal[0], normal[1], normal[2])
        glEnd()
        #нижняя
        glBegin(GL_POLYGON)
        if (1 - is_tek):
            glColor3f(R / 255, G / 255, B / 255)
        for i in range(len(root2)):
            if is_tek:
                glTexCoord2f(tek_list[i][0], tek_list[i][1])
            glVertex3f(root2[i][0], root2[i][1], root2[i][2])
        normal = compute_norm(root2[0], root2[1], root2[2])
        glNormal3f(normal[0], normal[1], normal[2])
        glEnd()
        #конец дисплейного списка
        glEndList()


def update_display_list():
    global display_list
    if display_list is not None:
        glDeleteLists(display_list, 1)
    create_display_list()


def create_vbo():
    global vbo, vert

    for i in range(len(root1)):
        vert.extend([root1[i][0], root1[i][1], root1[i][2]])
    #стороны
    for i in range(8):
        a, b, c = sides[i]
        vert.extend([a, b, c])
        a, b, c = sides[(i + 1) % 8]
        vert.extend([a, b, c])
        a, b, c = sides[(i + 2) % 8]
        vert.extend([a, b, c])
    #нижняя
    for i in range(len(root2)):
        vert.extend([root2[i][0], root2[i][1], root2[i][2]])
    #буфер вершин - оптимизация
    vbo = glGenBuffers(1)
```

```
    glBindBuffer(GL_ARRAY_BUFFER, vbo)
    glBufferData(GL_ARRAY_BUFFER, np.array(vert, dtype=np.float32), GL_STATIC_DRAW)


def prizma():
    global display_list, vert, vbo

    if display_list is None:
        create_display_list()
    glCallList(display_list)
    #vbo
    if vbo is None:
        create_vbo()
    glBindBuffer(GL_ARRAY_BUFFER, vbo)
    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 0, None)
    glDrawArrays(GL_QUAD_STRIP, 0, int(len(vert) / 3))
    glDisableClientState(GL_VERTEX_ARRAY)


def display(window):
    global is_tek, display_list, last_time, cnt_frame, fps

    glEnable(GL_DEPTH_TEST)
    glDepthFunc(GL_LESS)
    glClearColor(0.0, 0.0, 0.0, 0.0)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)

    if is_move == 1:
        move()
    glTranslatef(speed, 0, 0) #движение по оси
    glTranslatef(0, speed, 0)
    glRotatef(alpha, 1, 0, 0)
    glRotatef(beta, 0, 1, 0)

    prizma()
    #рассчёт fps
    current_time = time.time()
    delta_time = current_time - last_time
    cnt_frame += 1
    if delta_time >= 1.0:
        cur_fps = cnt_frame / delta_time
        fps.append(cur_fps)
        cnt_frame = 0
        last_time = current_time
```

```
    glfw.swap_buffers(window)
    glfw.poll_events()


def key_callback(window, key, scancode, action, mods):
    global alpha, beta, is_move, is_light, is_tek
    if action == glfw.PRESS or action == glfw.REPEAT:
        if key == glfw.KEY_RIGHT:
            beta += 1
        elif key == glfw.KEY_LEFT:
            beta -= 1
        elif key == glfw.KEY_UP:
            alpha += 1
        elif key == glfw.KEY_DOWN:
            alpha -= 1
        elif key == glfw.KEY_F:
            global fill
            if fill :
                glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)
            else:
                glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
            fill = not fill
        elif key == glfw.KEY_L:
            update_display_list()
            if is_light:
                glEnable(GL_LIGHTING)
                glEnable(GL_LIGHT0)
            else:
                glDisable(GL_LIGHTING)
            is_light = 1 - is_light
        elif key == glfw.KEY_M:
            is_move = 1 - is_move
        elif key == glfw.KEY_T:
            update_display_list()
            is_tek = 1 - is_tek


def main():
    global R, G, B
    if not glfw.init():
        return
    window = glfw.create_window(640, 640, "laba7", None, None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)
```

```
    R, G, B = (random.randint(1, 255), random.randint(1, 255), random.randint(1, 255))
    load_tex()
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    light()
    for _ in range(500):
        display(window)
    # while not glfw.window_should_close(window):
    #     display(window)
    glfw.destroy_window(window)
    glfw.terminate()

if __name__ == '__main__':
    start = time.monotonic()
    main()
    stop = time.monotonic()
    print('time:', stop - start)
    print('FPS:', sum(fps)/len(fps))
```

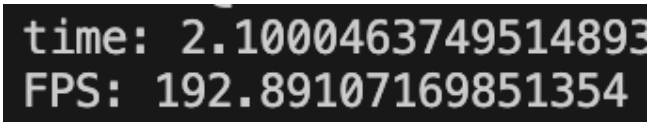

Рис. 1 — Результаты без оптимизаций

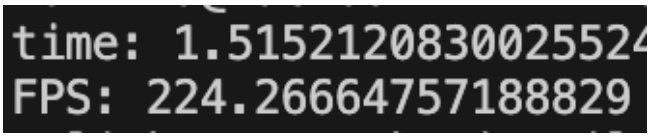

Рис. 2 — Результаты с оптимизацией

Параметр	time	FPS
До оптимизации	2.1	192
После оптимизации	1.5	225

Таблица 1: Сравнение средних значений производительности

4 Заключение

Существуют различные способы оптимизации приложений написанных с OpenGL, одни из главных: дисплейный список, буфер вершин, эффективные методы хранения текстур, правильная настройка источников света. Эти методы позволяют значительно улучшить производительность, например, уменьшить время отрисовки объекта и увеличить частоту кадров, поэтому применение этих оптимизаций – одна из главных частей разработки графического приложения.