**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 6**

Тема: Создание шейдерных анимационных эффектов в OpenGL

Студент: Пищик Евгений Сергеевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

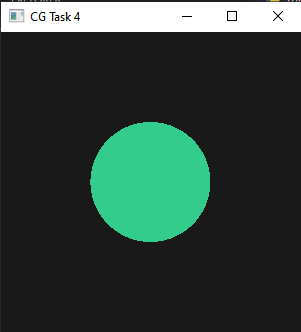
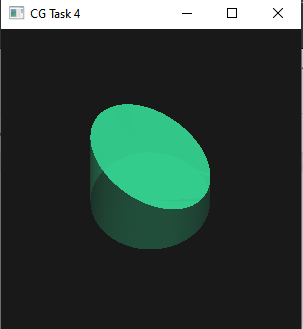
Москва, 2021

1. Постановка задачи

Для поверхности, созданной в л.р. №4-5, обеспечить выполнение следующего шейдерного эффекта: прозрачность вершины пропорциональна косинусу угла между нормалью и направлением на заданную точку.

1. Описание программы

Программа состоит из кода на ЯП python с использованием OpenGL. Используя буфер вершин строится заданное тело, при помощи средств OpenGL для каждого пиксельного шейдера задается прозрачность, которая пропорциональна косинусу угла между нормалью и направлением на заданную точку.

1. Набор тестов
2. Исходное положение.
3. Поворот.
4. Результаты выполнения тестов
5. 
6. 
7. Листинг программы

from OpenGL.GL import \*

from OpenGL.GLUT import \*

import sys

import numpy as np

def xflip(angle, point):

mx = np.matrix([[1.0, 0.0, 0.0], [0.0, np.cos(np.pi \* angle / 180.0), -np.sin(np.pi \* angle / 180.0)], [0.0, np.sin(np.pi \* angle / 180.0), np.cos(np.pi \* angle / 180.0)]])

inp = np.matrix([point[0], point[1], point[2]]).T

resp = mx @ inp

return [float(resp[0]), float(resp[1]), float(resp[2])]

def yflip(angle, point):

my = np.matrix([[np.cos(np.pi \* angle / 180.0), 0.0, np.sin(np.pi \* angle / 180.0)], [0.0, 1.0, 0.0], [-np.sin(np.pi \* angle / 180.0), 0.0, np.cos(np.pi \* angle / 180.0)]])

inp = np.matrix([point[0], point[1], point[2]]).T

resp = my @ inp

return [float(resp[0]), float(resp[1]), float(resp[2])]

def zflip(angle, point):

mz = np.matrix([[np.cos(np.pi \* angle / 180.0), -np.sin(np.pi \* angle / 180.0), 0.0], [np.sin(np.pi \* angle / 180.0), np.cos(np.pi \* angle / 180.0), 0.0], [0.0, 0.0, 1.0]])

inp = np.matrix([point[0], point[1], point[2]]).T

resp = mz @ inp

return [float(resp[0]), float(resp[1]), float(resp[2])]

def get\_normal(a, b, c):

v1 = [0, 0, 0]

v2 = [0, 0, 0]

v1[0] = a[0] - b[0]

v1[1] = a[1] - b[1]

v1[2] = a[2] - b[2]

v2[0] = b[0] - c[0]

v2[1] = b[1] - c[1]

v2[2] = b[2] - c[2]

wrki = ((v1[1] \* v2[2] - v1[2] \* v2[1])\*\*2 + (v1[2] \* v2[0] - v1[0] \* v2[2])\*\*2 + (v1[0] \* v2[1] - v1[1] \* v2[0])\*\*2)\*\*0.5

nx = (v1[1] \* v2[2] - v1[2] \* v2[1]) / wrki

ny = (v1[2] \* v2[0] - v1[0] \* v2[2]) / wrki

nz = (v1[0] \* v2[1] - v1[1] \* v2[0]) / wrki

return [nx, ny, nz]

def get\_center\_xy(ax, ay, bx, by, cx, cy):

d = 2 \* (ax \* (by - cy) + bx \* (cy - ay) + cx \* (ay - by))

ux = ((ax \* ax + ay \* ay) \* (by - cy) + (bx \* bx + by \* by) \* (cy - ay) + (cx \* cx + cy \* cy) \* (ay - by)) / d

uy = ((ax \* ax + ay \* ay) \* (cx - bx) + (bx \* bx + by \* by) \* (ax - cx) + (cx \* cx + cy \* cy) \* (bx - ax)) / d

return ux, uy

def get\_points():

global pointdata

global approx

pointdata = []

points3d = [[], []]

r = 0.4

height = 0.5

ang = 30

angles = np.linspace(0.0, 2 \* np.pi, approx)

for angle in angles:

points3d[0].append([r \* np.cos(angle), r \* np.sin(angle), np.tan(ang \* np.pi / 180) \* np.sin(r \* np.cos(angle))])

points3d[1].append([r \* np.cos(angle), r \* np.sin(angle), height])

px = 0.0

py = 0.0

pz = 0.0

for p1, p2 in zip(points3d[0], points3d[1]):

px += p1[0] + p2[0]

py += p1[1] + p2[1]

pz += p1[2] + p2[2]

px /= (len(points3d[0]) + len(points3d[1]))

py /= (len(points3d[0]) + len(points3d[1]))

pz /= (len(points3d[0]) + len(points3d[1]))

points3d[0] = [(p[0] - px, p[1] - py, p[2] - pz) for p in points3d[0]]

points3d[1] = [(p[0] - px, p[1] - py, p[2] - pz) for p in points3d[1]]

xc, yc = get\_center\_xy(points3d[1][0][0], points3d[1][0][1], points3d[1][1][0], points3d[1][1][1], points3d[1][2][0], points3d[1][2][1])

centr = [xc, yc, height / 2]

zc = 0

for el in points3d[0]:

zc += el[2]

zc /= len(points3d[0])

for i in range(len(points3d[0]) - 1):

tl, tr, bl, br = points3d[1][i], points3d[1][i + 1], points3d[0][i], points3d[0][i + 1]

pointdata.append(bl)

pointdata.append(br)

pointdata.append(tl)

pointdata.append(tr)

pointdata.append(tl)

pointdata.append(br)

tl, tr, bl, br = points3d[1][-1], points3d[1][0], points3d[0][-1], points3d[0][0]

pointdata.append(bl)

pointdata.append(br)

pointdata.append(tl)

pointdata.append(tr)

pointdata.append(tl)

pointdata.append(br)

for i in range(len(points3d[1]) - 1):

bl, br = points3d[1][i], points3d[1][i + 1]

pointdata.append(centr)

pointdata.append(bl)

pointdata.append(br)

bl, br = points3d[1][-1], points3d[1][0]

pointdata.append(centr)

pointdata.append(bl)

pointdata.append(br)

for i in range(len(points3d[0]) - 1):

bl, br = points3d[0][i], points3d[0][i + 1]

pointdata.append([xc, yc, zc])

pointdata.append(bl)

pointdata.append(br)

bl, br = points3d[1][-1], points3d[1][0]

pointdata.append([xc, yc, zc])

pointdata.append(bl)

pointdata.append(br)

def specialkeys(key, x, y):

global xa

global ya

global pointdata

if key == GLUT\_KEY\_UP:

glRotatef(1, 1, 0, 0)

xa += 1

if key == GLUT\_KEY\_DOWN:

glRotatef(-1, 1, 0, 0)

xa -= 1

if key == GLUT\_KEY\_LEFT:

glRotatef(1, 0, 1, 0)

ya += 1

if key == GLUT\_KEY\_RIGHT:

glRotatef(-1, 0, 1, 0)

ya -= 1

glutPostRedisplay()

def init():

glClearColor(0.1, 0.1, 0.1, 1.0)

glEnable(GL\_DEPTH\_TEST)

glEnable(GL\_NORMALIZE)

glEnable(GL\_BLEND)

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)

def draw():

global pointdata

global color

global xa, ya

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

dir\_point = [0.0, 0.2, -1.0]

eps = 0.1

glBegin(GL\_TRIANGLES)

for i in range(0, len(pointdata), 3):

glVertex3f(pointdata[i][0], pointdata[i][1], pointdata[i][2])

glVertex3f(pointdata[i + 1][0], pointdata[i + 1][1], pointdata[i + 1][2])

glVertex3f(pointdata[i + 2][0], pointdata[i + 2][1], pointdata[i + 2][2])

norm = get\_normal(pointdata[i], pointdata[i + 1], pointdata[i + 2])

glNormal3f(norm[0], norm[1], norm[2])

alpha = (dir\_point[0] \* norm[0] + dir\_point[1] \* norm[1] + dir\_point[2] \* norm[2]) / ((dir\_point[0]\*\*2 + dir\_point[1]\*\*2 + dir\_point[2]\*\*2)\*\*0.5 \* (norm[0]\*\*2 + norm[1]\*\*2 + norm[2]\*\*2)\*\*0.5)

glColor4f(color[0], color[1], color[2], (alpha\*\*2)\*\*0.5 + eps)

glEnd()

glDisable(GL\_LIGHT0)

glutSwapBuffers()

def main():

W, H = 300, 300

XS, YS = 50, 50

glutInit(sys.argv)

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH)

glutInitWindowSize(W, H)

glutInitWindowPosition(XS, YS)

glutCreateWindow(f'CG Task 4')

init()

glutDisplayFunc(draw)

glutIdleFunc(draw)

glutSpecialFunc(specialkeys)

glutMainLoop()

if \_\_name\_\_ == '\_\_main\_\_':

global pointdata

global approx

global color

global xa

global ya

xa, ya = 0, 0

while True:

try:

approx = int(input('approximation number: int\n'))

if approx < 4:

raise ValueError('approximation number must be >= 4')

break

except ValueError as e:

print(e)

while True:

try:

color = list(map(float, input('color: 0-1f 0-1f 0-1f\n').strip().split()))

if len(color) != 3:

raise IndexError('must be 3 number for RGB values')

for el in color:

if el < 0.0 or el > 1.0:

raise ValueError('color value must be >= 0.0 and <= 1.0')

break

except IndexError as e0:

print(e0)

except ValueError as e1:

print(e1)

get\_points()

main()

ЛИТЕРАТУРА

1. Документация OpenGL[Электронный ресурс]. URL: <https://www.khronos.org/registry/OpenGL-Refpages/es3/> (дата обращения: 16.11.2021).