КПІ ім. Ігоря Сікорського

Інститут прикладного системного аналізу

Кафедра Системного проектування

Лабораторна робота №3

з дисципліни «Комп’ютерна графіка»

«Параметри освітлення»

Виконав:

Студент
групи ДА-81

ННК «ІПСА»

Дзюбчик
Олександр

Варіант № 8

Київ – 2020

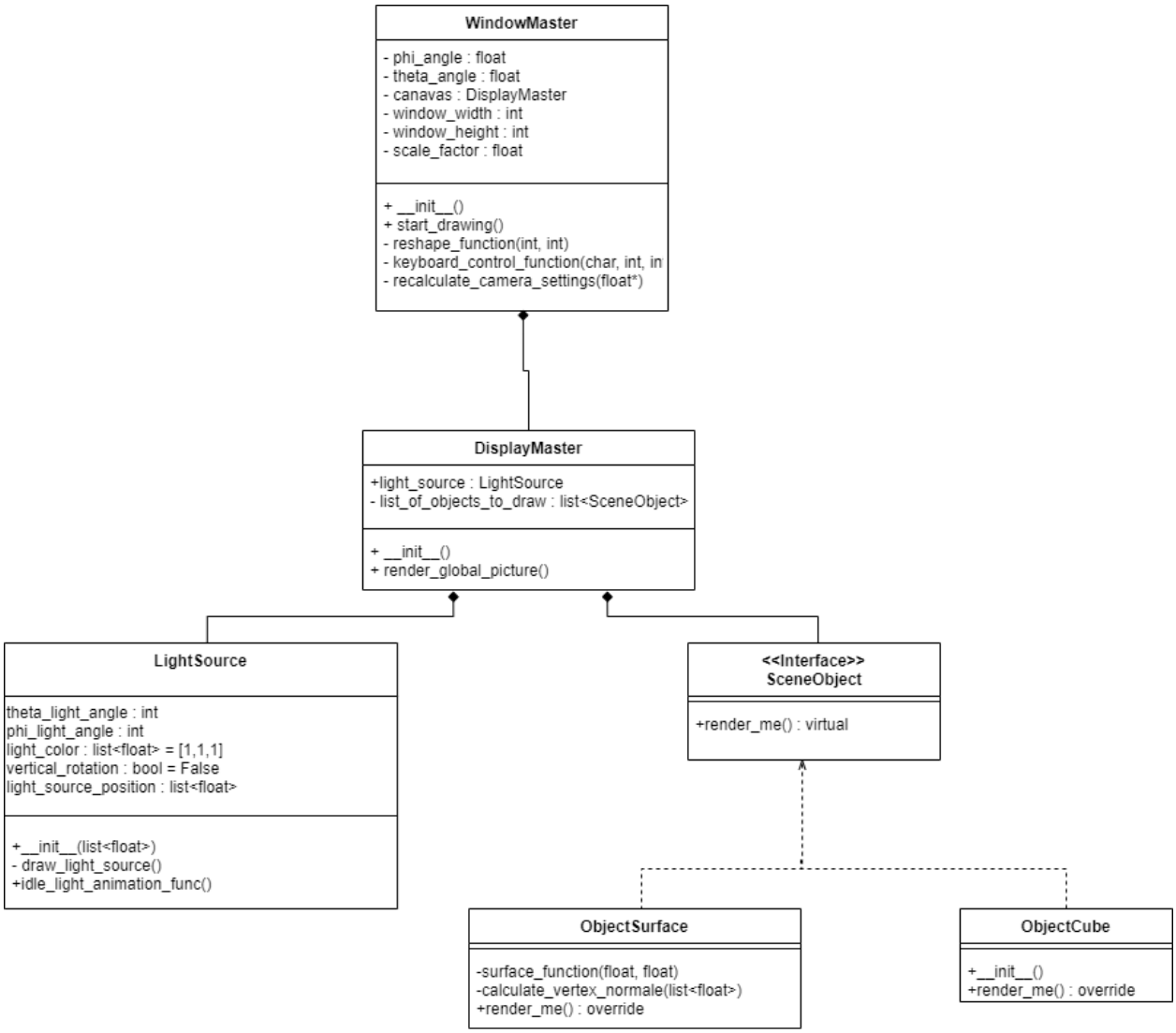
Мета роботи: здобути базові навички регулювання параметрів відображення, освітлення та перетворення об’єктів.

Завдання: за варіантом розробити сцену з одним або більше джерелом світла. Обов’язково обрати тип джерела, налаштувати його інтенсивність та колір світла. Сцена має бути побудована з урахуванням перспективи.

2п варіант:

Побудувати геоцентричну систему представену: площиною (з ЛР2) та джерелом/джерелами світла, що рухаються по сферичній орбіті. Рух зірки за рахунок glutIdleFunc.

Структура проекту



Клас **WindowMaster** відповідає за:

• Створення вікна – відбувається у конструкторі.

• Запуск основного циклу – функція *start_drawing()*, викликається для об'єкту класу у головній функції програми.

• Обробку вікна – функції *reshape_function()* (зміна розмірів вікна) і *keyboard_control_function* (керуванням рисунком за допомогою клавіатури), *recalculate_camera_setting (*встановлення камери).

Атрибути класу:

theta_angle, phi_angle – кути у сферичних координатах між вектором з початку координат до позиції камери та осями Oy і Oz відповідно. Потрібні для повороту камери по сферичній траекторії.

window_width, window_height – розміри вікна

scale_factor – коефіцієнт масштабування

У конструкторі класу WindowMaster створюється об’єкт класу DisplayMaster**.**

Клас **DisplayMaster** відповідає безпосередньо за створення зображення у вікні. У конструкторі ініціалізується полотно.

Атрибути класу:

light_source – об’єкт класу *LightSource*, що виконує функції джерела світла

list_of_objects_to_draw – список об’єктів класу SceneObject, що потрібно відобразити

Методи класу:

render_global_picture() – виводить цілісну картину з усіма об’єктами та джерелом світла

Клас **SceneObject –** інтерфейс з чистовіртуальним методом *render_me()*, який відображає об’єкт на екран.

Класи **ObjectSurface** і **ObjectCube** наслідуються від SceneObject, перевизначають метод *render_me()* і створюють об’кти, що є поверхнею і кубом відповідно.

Клас *ObjectSurface* додатково має методи *surface_function()* – функція, якою задається поверхня та *calculate_vertex_normal()* – обчислення нормалі для кожної вершини, яка бере участь у побудові поверхні. Нормаль обчислюється через додаткові (сусідні) точки, що слугують для обчислення 2х векторів, векторний добуток яких і буде нормаллю. Нормалізувати вектор нормалі вручну не потрібно, оскільки увімкнено режим glEnable(GL_NORMALIZE).

Клас **LightSource -** потрібний для створення джерела світла та дій з ним.

Атрибути класу:

theta_light_angle, phi_light_angle – кути у сферичних координатах потрібні для обертання джерела світла по сферичній траекторії.

light_color – колір світла

vertical_rotation – змінна, яка визначає по якій траєкторії буде рухатись джерело (горизонтальна чи вертикальна).

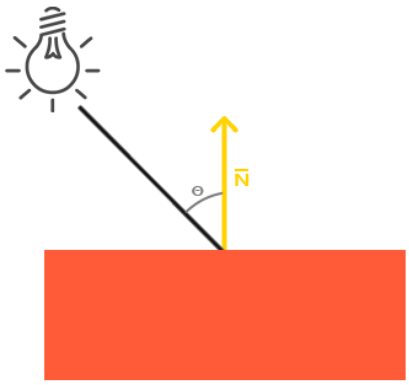
light_source_position – поточні координати джерела світла.

Методи класу:

draw_light_source() – встановлення джерела світла та відображення сфери, що «візуалізує» його.

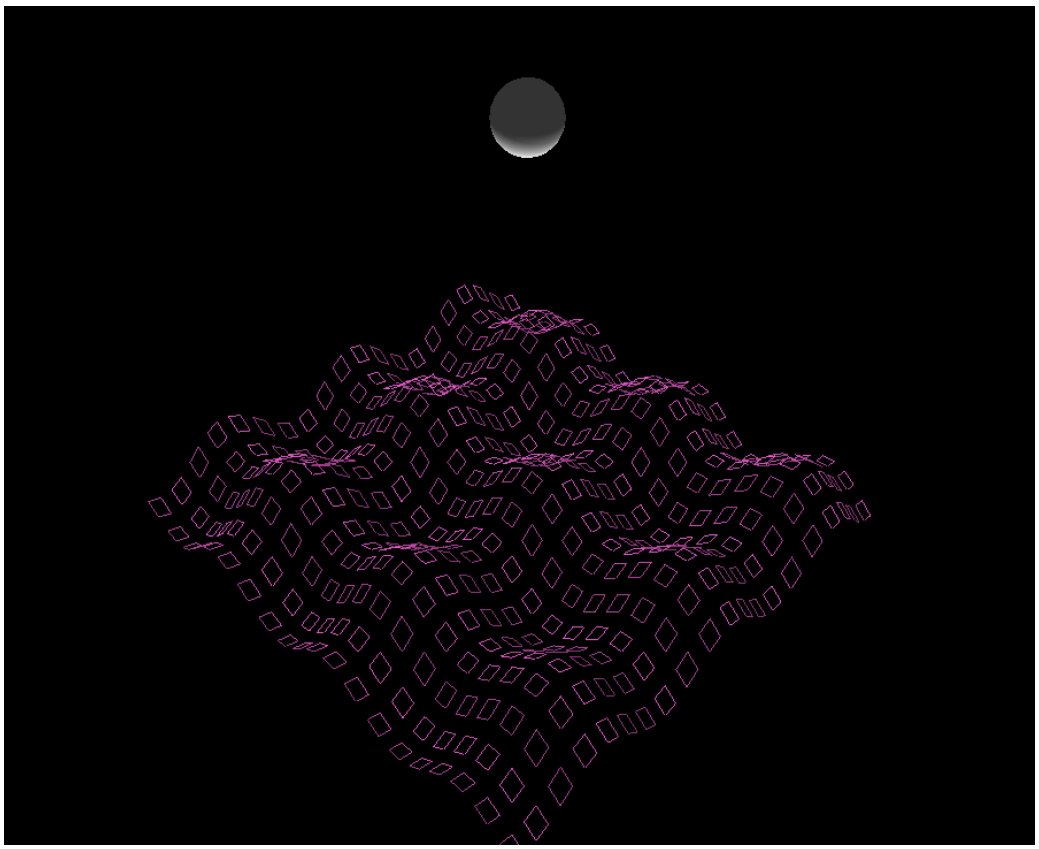
idle_light_animation_func() – зміна кутів в сферичних координатах, тобто рух джерела світла.

Вплив дифузного світла на об’єкт визначається кутом між напрямком променів світла і вектором нормалі до об’єкта: чим менший цей кут, тим більш освітлений об’єкт:

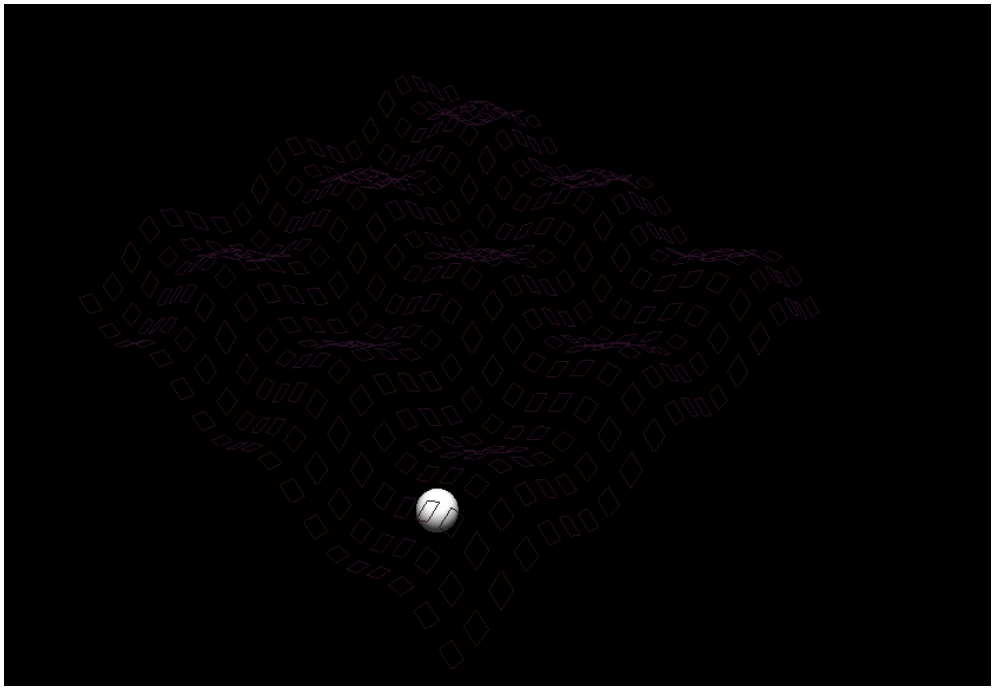


Для коректного освітлення поверхні для кожної точки, що бере участь у побудові об’єкту потрібно було обчислити і встановити (*glNormal3f()*) нормалі. Для кубу та сфери нормалі встановлені за замовчуваням, але для сфери («візуалізація джерела світла») потрібно було змінити напрямок нормалей на протилежний, оскільки за замовчуванням освітлювалась протилежна сторона.

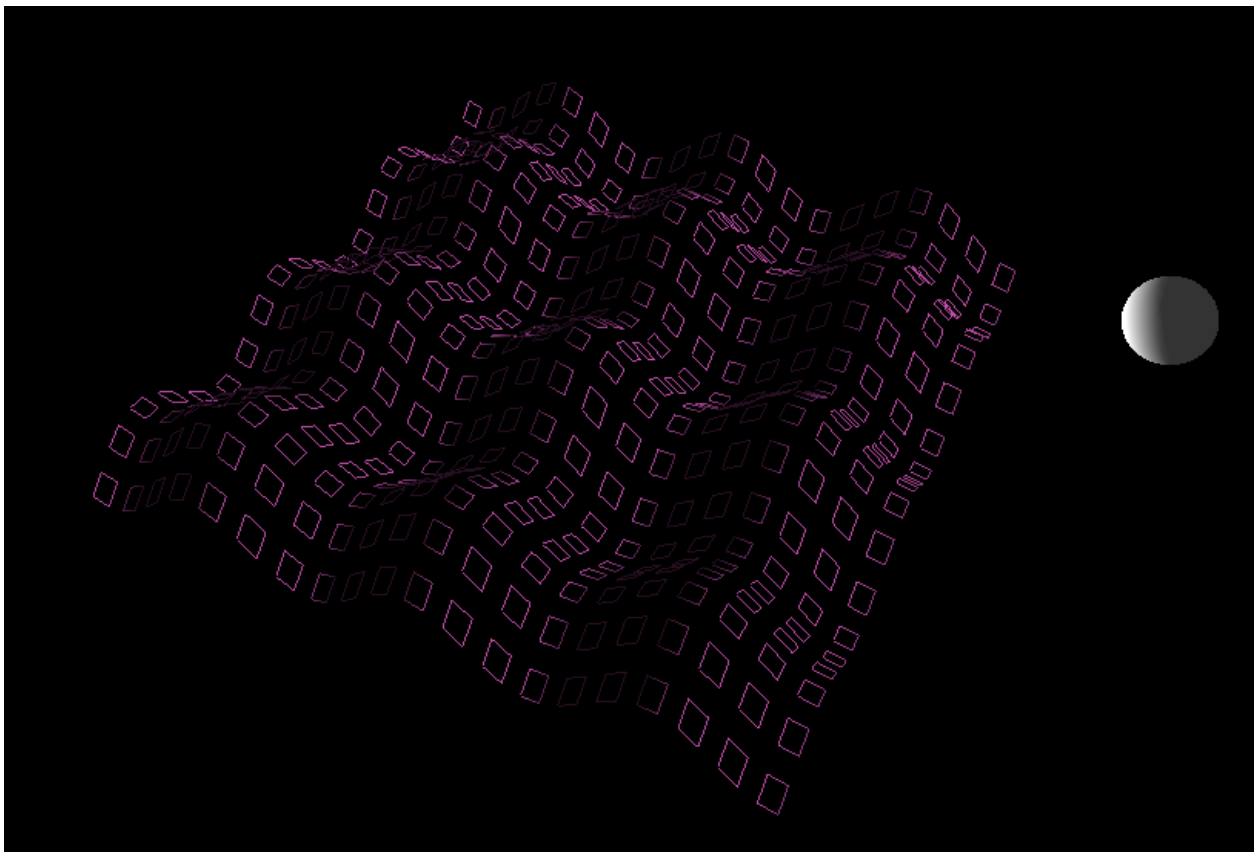
Результати роботи програми



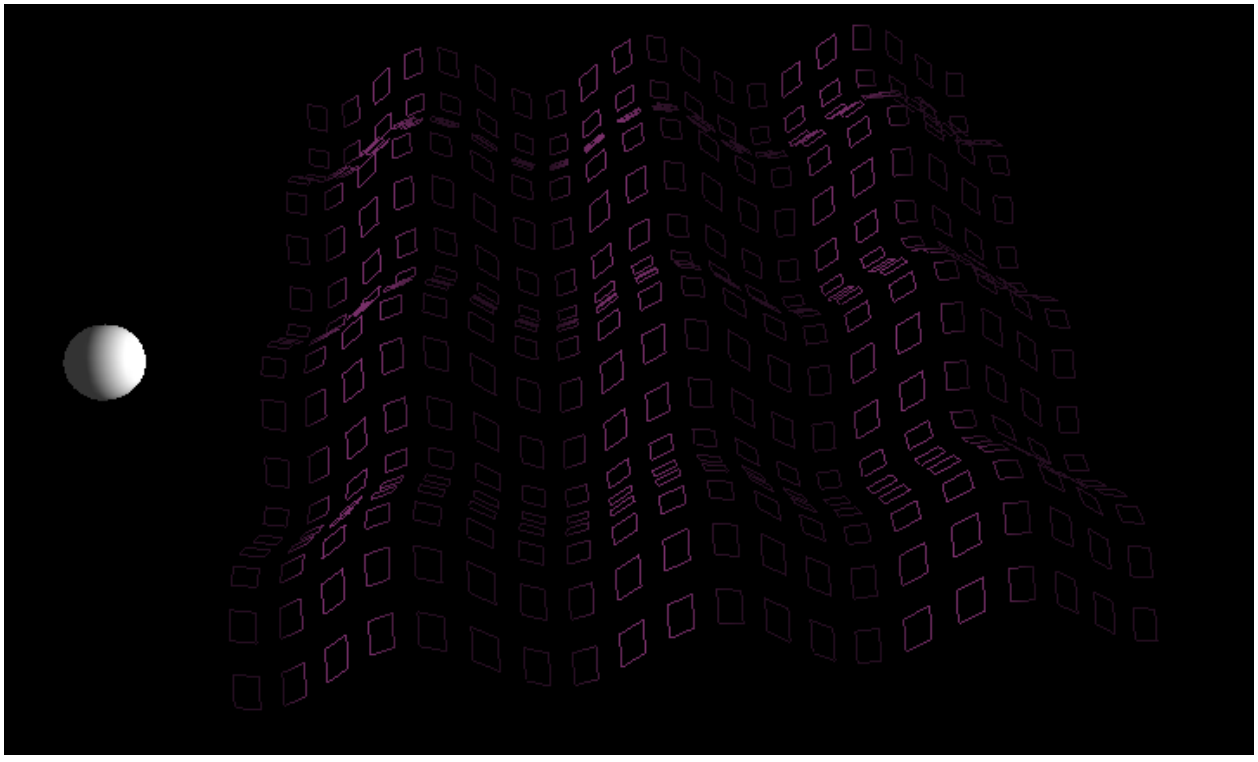
Джерело світла над поверхнею (максимальне освітлення)



Джерело світла під поверхнею (освітлення відсутнє)



Освітлення з одного боку



Освітлення з іншого боку, але джерело нижче, а, відповідно, і освітлення менше

w, a, s, d – для руху камери

m, n – маштабування

t – зміна траекторії руху джерела світла.

Якшо натиснути кнопку з розкладкою відмінною від англійської – програма вилітає.

Висновок: в результаті виконання лабораторної роботи було отримано практичні навички роботи з освітленням у OpenGL.

Лістинг програми

LightSource.py:

```
from OpenGL.GL import glColor3f, glTranslate, glLightfv, glLoadIdentity, GL_LIGHT
0, GL_POSITION, GL_DIFFUSE, GL_CONSTANT_ATTENUATION, GL_QUADRATIC_ATTENUATION
from OpenGL.GLU import gluNewQuadric, gluQuadricOrientation, gluSphere, GLU_INSID
E
from OpenGL.GLUT import glutPostRedisplay

from math import sin, cos, pi

class LightSource:
    """
        A class for creating light source
        ...
        Attributes
        ----------
        theta_light_angle: int
            Angle between light vector and Oy in spherical coordinates, 45 degree
s by default, increase/decrease by 1 degrees each second

        phi_light_angle: int
            Angle between light vector and Oz in spherical coordinates, 45 degree
s by default, increase/decrease by 1 degrees each second

        light_color: list
            light color stored as rbg unsigned byte

        vertical_rotation: bool
            define plane of rotation, true if vertical, false by default - horizo
ntal. May be changed by clicking "t"
        ...
        Methods
        ----------
        __init__(light_color_vector=[1.0, 1.0, 1.0]):
           Set light source default parameters and draw visual envelope

        draw_light_source():
            draw visual envelope

        idle_light_animation_func():
            Function executed in main loop need for light source moving animation
    """
    def __init__(self, light_color_vector=[1.0, 1.0, 1.0]):

        self.theta_light_angle = 45
        self.phi_light_angle = 45

        self.light_color = light_color_vector
```

```
        self.vertical_rotation = False

        self.draw_light_source()

    def draw_light_source(self):

        glColor3f(1.0, 1.0, 1.0)

        #15 is radius of the sphere that light source around
        self.light_source_position = [
            15 * sin(self.theta_light_angle * pi / 180) * sin(self.phi_light_angl
e * pi / 180),
            15 * cos(self.theta_light_angle * pi / 180),
            15 * sin(self.theta_light_angle * pi / 180) * cos(self.phi_light_angl
e * pi / 180)
        ]

        glTranslate(self.light_source_position[0], self.light_source_position[1],
 self.light_source_position[2])
        #set light source position
        glLightfv(GL_LIGHT0, GL_POSITION, [*self.light_source_position, 1])
        #set light color
        glLightfv(GL_LIGHT0, GL_DIFFUSE, self.light_color)
        #set light intensity
        glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.7)
        glLightfv(GL_LIGHT0,  GL_QUADRATIC_ATTENUATION, 0.0001)

        #turning the rays of light so that the back of the ball is shaded
        gl_q = gluNewQuadric()
        gluQuadricOrientation(gl_q, GLU_INSIDE)

        #draw visual envelope (ball)
        gluSphere(gl_q, 1, 20, 20)

        glLoadIdentity()

    def idle_light_animation_func(self):

        if self.vertical_rotation:
            self.theta_light_angle += 1
        else:
            self.phi_light_angle += 1

        self.draw_light_source()

        glLoadIdentity()
        glutPostRedisplay()
```

SceneObject.py:

```
from OpenGL.GL import glColor3f, glBegin, glEnd, glNormal3f, glVertex3f,  glLoadI
dentity, glTranslate, GL_LINES
```

```
from OpenGL.GLUT import glutSolidCube

from abc import ABCMeta, abstractmethod

from math import sin, cos

class SceneObject:
    """
        A an abstract class of objects that must be drawed
    """
    __metaclass__ = ABCMeta

    @abstractmethod
    def render_me(self):
        return


class ObjectSurface(SceneObject):
    """
        A class for creating surface that must be drawed
        ...
        Methods
        ----------
        surface_function(x, y):
            Define function that presents surface

        calculate_vertex_normal(vertex):
            Calculate normal vector for passed vertex

        render_me():
            Render surface
    """
    def __init__(self):
        pass

    def surface_function(self, x, y):
        return sin(x) + cos(y)

    def calculate_vertex_normal(self, vertex):

        step = 0.5

        #get neighbor vertexes for passed vector to calculate normal
        addit_point_1 = [vertex[0] + step, self.surface_function(vertex[0] + step
, vertex[2]), vertex[2]]
        addit_point_2 = [vertex[0], self.surface_function(vertex[0], vertex[2] +
step), vertex[2] + step]

        v1 = [c2 - c1 for c1, c2 in zip(vertex, addit_point_2)]
        v2 = [c2 - c1 for c1, c2 in zip(vertex, addit_point_1)]
```

```
        #calculate normal
        normal = [v1[1] * v2[2] - v1[2] * v1[1],
                  v1[2] * v2[0] - v1[0] * v2[2],
                  v1[0] * v2[1] - v1[1] * v2[0]]

        return normal

    def render_me(self):

        glColor3f(0.75, 0.3, 0.66)
        #glColor3f(1, 1, 1)

        #surface bounds
        x_lower_bound = -10
        x_upper_bound = 10
        y_lower_bound = -10
        y_upper_bound = 10

        step = 0.5

        x_current = x_lower_bound + step
        y_current = y_lower_bound + step

        #draw part of surface
        glBegin(GL_LINES)

        while x_current <= x_upper_bound + 0.0001:

            while y_current < y_upper_bound + 0.0001:
                normal = self.calculate_vertex_normal([x_current,
                                                       self.surface_function(x_cu
rrent, y_current),
                                                       y_current])
                glNormal3f(normal[0], normal[1], normal[2])
                glVertex3f(x_current, self.surface_function(x_current, y_current)
, y_current)

                y_current += step

            y_current = y_lower_bound + step
            x_current += step

        x_current = x_lower_bound + step
        y_current = y_lower_bound + step

        #draw part of surface
        while y_current <= y_upper_bound + 0.01:

            while x_current <= x_upper_bound + 0.01:
                normal = self.calculate_vertex_normal([x_current,
```

```
                                                    self.surface_function(x_cu
rrent, y_current),
                                                    y_current])
                glNormal3f(normal[0], normal[1], normal[2])
                glVertex3f(x_current, self.surface_function(x_current, y_current)
, y_current)

                x_current += step

            x_current = x_lower_bound + step
            y_current += step

        glEnd()


class ObjectCube(SceneObject):
    """
        A class for creating cube that must be drawed
        ...
        Methods
        ----------
        __init__(position, side_size):
            Init cube position and size

        render_me():
            Render cube
    """
    def __init__(self, position, side_size):
        self.position = position
        self.side_size = side_size

    def render_me(self):
        glColor3f(1, 1, 1)

        glLoadIdentity()

        glTranslate(self.position[0], self.position[1], self.position[2])

        glutSolidCube(self.side_size)

        glLoadIdentity()
```

DisplayMaster.py:

```
from OpenGL.GL import *
from OpenGL.GLU import gluPerspective
from OpenGL.GLUT import glutSwapBuffers

from LightSource import LightSource
from SceneObject import *

class DisplayMaster:
```

```
    """
        A class for managing program window. It collects data, initializes window
, draws figure, manages keyboard actions.
        ...
        Attributes
        ----------
        light_source: LightSource
            class-
object LightSource (i.e. the light itself and the visual envelope (ball))

        list_of_objects_to_draw: list
            list of all figures that must be drawed
        ...
        Methods
        ----------
        __init__():
           Set canvas default parameters as matrix mode, grid etc. Create light s
ource and initializes list of all figures that must be drawed

        render_global_picture():
            Render list of all figures
    """
    def __init__(self):
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(60.0, 800.0 / 800.0, 0.01, 100.0)

        glMatrixMode(GL_MODELVIEW)

        glEnable(GL_DEPTH_TEST)

        glEnable(GL_LIGHTING)
        glEnable(GL_LIGHT0)
        glEnable(GL_COLOR_MATERIAL)
        glEnable(GL_NORMALIZE)

        glClearColor(0.0, 0.0, 0.0, 0.0)

        self.light_source = LightSource()

        self.list_of_objects_to_draw = []

        #self.list_of_objects_to_draw.append(ObjectCube([0, 0, 0], 7))
        self.list_of_objects_to_draw.append(ObjectSurface())


    def render_global_picture(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()

        self.light_source.draw_light_source()
```

```
        for object in self.list_of_objects_to_draw:
            object.render_me()

        glutSwapBuffers()
```

WindowMaster.py:

```
from OpenGL.GL import glViewport, glMatrixMode, glLoadIdentity, glScalef, GL_PROJ
ECTION, GL_MODELVIEW
from OpenGL.GLU import gluPerspective, gluLookAt
from OpenGL.GLUT import *

from math import sin, cos, pi

from DisplayMaster import DisplayMaster

class WindowMaster:
    """
        A class for managing program window. It initializes window, draws figure,
 manages keyboard actions and camera moving.
        ...
        Attributes
        ----------
        window_width, window_height: int, int
            window's width and height

        theta_angle: int
            Angle between camera vector and Oy in spherical coordinates, 45 degre
es by default, increase/decrease by 8 degrees each time changed

        phi_angle: int
            Angle between camera vector and Oz in spherical coordinates, 45 degre
es by default, increase/decrease by 8 degrees each time changed

        canvas: DisplayMaster
            object of class DisplayMaster used to draw main figure

        scaling_coef: float
            equal 1 by defult, increase/decrease by 10% each time changed
        ...
        Methods
        ----------
        __init__():
            Create window, set attributes by default

        start_drawing():
            Start main loop.

        reshape_function(width, heigth):
            Manage changing window's size and camera moving
```

```
        keyboard_control_function():
            Manage keyboard actions

        recalculate_camera_settings():
            Manage changing camera pozition
    """
    def __init__(self):

        glutInit(sys.argv)
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)
        glutInitWindowSize(800, 800)
        glutInitWindowPosition(320, 100)
        glutCreateWindow("Dzyubchik_CG_L3_V8")

        self.canvas = DisplayMaster()

        self.theta_angle = 45
        self.phi_angle = 45

        self.window_width = 800
        self.window_height = 800

        self.scale_factor = 1

    def start_drawing(self):
        glutDisplayFunc(self.canvas.render_global_picture)
        glutReshapeFunc(self.reshape_function)
        glutKeyboardFunc(self.keyboard_control_function)
        glutIdleFunc(self.canvas.light_source.idle_light_animation_func)
        glutMainLoop()

    def reshape_function(self, w, h):

        if h == 0:
            h = 1

        self.window_width = w
        self.window_height = h

        proportional_rate = float(w) / h

        glViewport(0, 0, w, h)

        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()

        gluPerspective(60.0, proportional_rate, 0.1, 100)

        camera_settings_matrix = [0 for i in range(9)]

        #calculate camera's point of view
```

```
        camera_settings_matrix = self.recalculate_camera_settings(camera_settings
_matrix)

        #set camera's point of view
        gluLookAt(camera_settings_matrix[0], camera_settings_matrix[1], camera_se
ttings_matrix[2],
                  camera_settings_matrix[3], camera_settings_matrix[4], camera_se
ttings_matrix[5],
                  camera_settings_matrix[6], camera_settings_matrix[7], camera_se
ttings_matrix[8])

        glScalef(self.scale_factor, self.scale_factor, self.scale_factor)

        glMatrixMode(GL_MODELVIEW)

    def keyboard_control_function(self, key, x, y):

        key = key.decode("utf-8").lower()

        if key == 'a':
            self.phi_angle = (self.phi_angle + 8) % 360

        elif key == 'd':
            self.phi_angle = (self.phi_angle - 8) % 360

        elif key == 'w':
            self.theta_angle = (self.theta_angle + 8) % 360

        elif key == 's':
            self.theta_angle = (self.theta_angle - 8 + 360) % 360

        elif key == 'm':
            self.scale_factor *= 1.1

        elif key == 'n':
            self.scale_factor *= 0.9

        elif key == 't':
            self.canvas.light_source.vertical_rotation = not self.canvas.light_so
urce.vertical_rotation

        self.reshape_function(self.window_width, self.window_height)

    def recalculate_camera_settings(self, camera_settings_matrix):

        #25 is radius of the sphere that camera moves around
        z_camera_coordinate = 25 * sin(self.theta_angle * pi / 180) * cos(self.ph
i_angle * pi / 180)
        x_camera_coordinate = 25 * sin(self.theta_angle * pi / 180) * sin(self.ph
i_angle * pi / 180)
        y_camera_coordinate = 25 * cos(self.theta_angle * pi / 180)
```

```
        camera_settings_matrix[0] = x_camera_coordinate
        camera_settings_matrix[1] = y_camera_coordinate
        camera_settings_matrix[2] = z_camera_coordinate

        #this if_statements prevents the camera from scrolling when it passes a p
oint perpendicular to the xOz plane
        if self.theta_angle > 180:
            camera_settings_matrix[7] = -1
        else:
            camera_settings_matrix[7] = 1

        return camera_settings_matrix
```

main.py:

```
from WindowMaster import WindowMaster

if __name__ == "__main__":
    window = WindowMaster()
    window.start_drawing()
```