МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)

Расчетно-графическое задание по дисциплине «Компьютерная графика »
«Создание трехмерного приложения использующего OpenGL»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Осипов О.В.

Белгород 2019

Расчетно-графическое задание

«Создание трехмерного приложения использующего OpenGL»

Цель работы:получение навыков работы с высокоуровневым API OpenGL.

Постановка задачи: в качестве задания для работы была выбрана визуализация для алгоритма генерации ландшафта.

На вход для визуализатора подается двухмерный или трехмерный массив объектов класса *Block*. В классе *Block* имеется поле содержащее объект *Material*, в котором описаны правила необходимые для работы алгоритма генерации ландшафта, а также свойства и данных необходимые для визуализации:

- название текстурных файлов.
- цвет блока.
- прозрачность.

Каждый блок представлен в виде куба, с наложенной на него текстурой или окрашенный в цвет материала. При отрисовке проходя по массиву, генерируются вершины куба, и устанавливаются текстурные координаты.

Для возможности просмотра невидимых из-за непрозрачности блоков в трехмерном пространстве были реализованы срезы, используя слайдеры можно регулировать границы отрисовки массивов.

Текстуры хранятся в формате .bmp, и загружаются по именам из словаря материалов, для фильтрации текстур используется метод *gl.NEAREST*.

Для освещения используемой сцены используется 2 источника света размещенные по разные концы модели по оси Z, цвет свечения каждого источника света можно изменить используя диалоговое окно. GL_LIGHT1 направлен в центр модели.

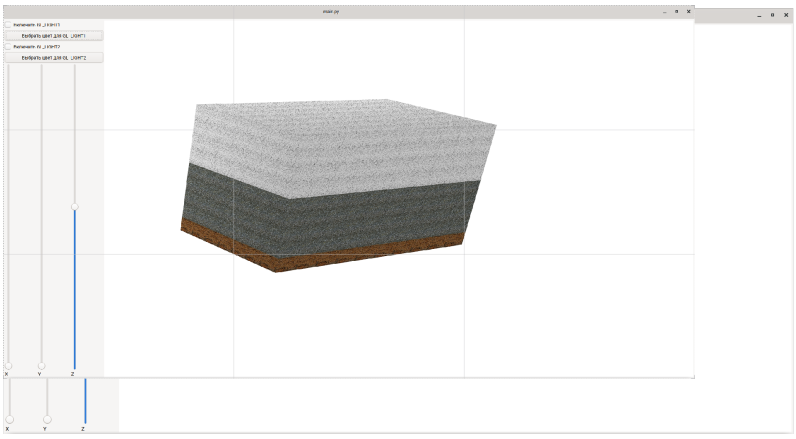

Рис. 1: Окно программы

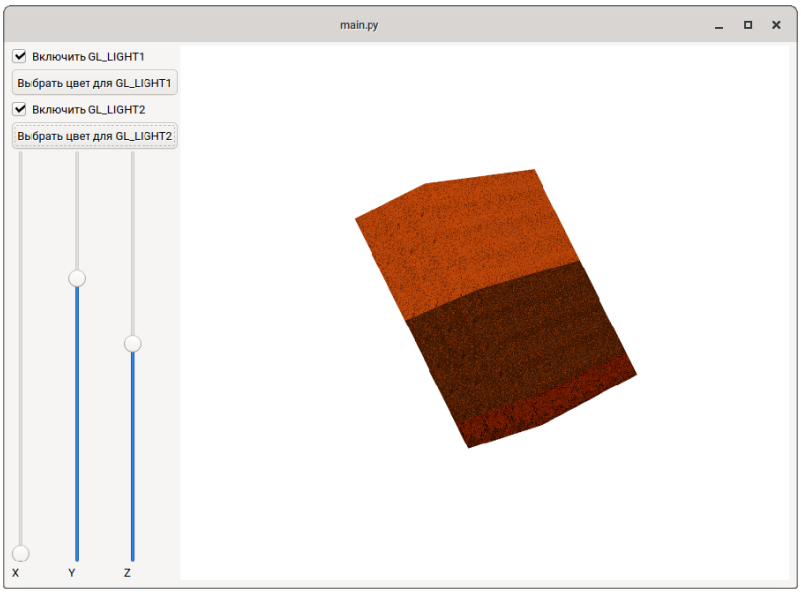

Рис. 2: Пример работы освещения

Приложение

Содержимое файла main.py

```
import sys
import math

from PyQt5.QtCore import Qt,QObject,pyqtSlot,QPoint
from PyQt5.QtGui import QColor,QMatrix4x4
from PyQt5.QtWidgets import QApplication,
↪  QMessageBox,QHBoxLayout,QVBoxLayout,QWidget,QSlider,QLabel,QSizePolicy,QGridLayout,QPushButton,QCol
from PyQt5.QtOpenGL import QGL, QGLFormat, QGLWidget
from PIL.Image import *

from blockmap import p_map
map_len_x = len(p_map[0][0])
map_len_y = len(p_map[0])
map_len_z = len(p_map)


try:
    from OpenGL import GL,GLU,GLE
except ImportError:
    app = QApplication(sys.argv)
    QMessageBox.critical(None, "OpenGL samplebuffers",
            "PyOpenGL must be installed to run this example.")
    sys.exit(1)


class GLWidget(QGLWidget):
    GL_MULTISAMPLE = 0x809D
    rot = 0.0
    def __init__(self, parent):
        super(GLWidget, self).__init__(QGLFormat(QGL.SampleBuffers), parent)
        self.setWindowTitle("OpenGL")
        self.globalScale = 0.1
        self.rotationMatrix = QMatrix4x4()
        self.centerMatrix = QMatrix4x4()
        self.pos = QPoint(0,0)
        self.z_offset = -10
        self.slice_x = 0
        self.slice_y = 0
        self.slice_z = 0

    @pyqtSlot(int)
    def slice_x_slot(self,int):
        self.slice_x = int
        self.repaint()

    @pyqtSlot(int)
    def slice_y_slot(self,int):
        self.slice_y = int
        self.repaint()

    @pyqtSlot(int)
    def slice_z_slot(self,int):
        self.slice_z = int
        self.repaint()

    def color_light1(self):
```

```
        color = QColorDialog.getColor()
        self.light1Diffuse = [color.red()/255, color.green()/255,  color.blue()/255]
        GL.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE, self.light1Diffuse)
        self.repaint()

    def color_light2(self):
        color = QColorDialog.getColor()
        self.light2Diffuse = [color.red()/255, color.green()/255,  color.blue()/255]
        GL.glLightfv(GL.GL_LIGHT2, GL.GL_DIFFUSE, self.light2Diffuse)
        self.repaint()

    def light1_check(self,checked):
        if(checked):
            GL.glEnable(GL.GL_LIGHT1)
        else:
            GL.glDisable(GL.GL_LIGHT1)
        self.repaint()

    def light2_check(self,checked):
        if(checked):
            GL.glEnable(GL.GL_LIGHT2)
        else:
            GL.glDisable(GL.GL_LIGHT2)
        self.repaint()

    def changeRotationMatrix(self,dx,dy):
        self.rotationMatrix.rotate(-dx, 0, 1, 0)
        self.rotationMatrix.rotate(-dy, 1, 0, 0)

    def centredScene(self,count_x,count_y,count_z):
        self.centerMatrix.setToIdentity()
        self.centerMatrix.translate(-count_x/2, 0, 0)
        self.centerMatrix.translate(0, -count_y/2, 0)
        self.centerMatrix.translate(0, 0, -count_z/2)

    def mouseMoveEvent(self,event):
        newPos = QPoint(event.pos())
        dx = newPos.x() - self.pos.x()
        dy = newPos.y() - self.pos.y()
        self.changeRotationMatrix(dx / 2, dy / 2)
        self.pos = newPos
        self.resetModelView()
        self.repaint()

    def mousePressEvent(self,event):
        self.pos = event.pos()
        self.repaint()

    def resetProjection(self):
        GL.glMatrixMode(GL.GL_PROJECTION)
        GL.glLoadIdentity()
        if(self.width()==0):
            width = 1
        else:
            width = self.width()
        if(self.height()==0):
            height = 1
        else:
            height = self.width()
        GLU.gluPerspective(30.0, width/height, 0.1, 20)
```

```
def resetModelView(self):
    GL.glMatrixMode(GL.GL_MODELVIEW)
    GL.glLoadIdentity()
    GL.glTranslatef(0, 0, self.z_offset)
    GL.glMultMatrixf(self.rotationMatrix.transposed().data())
    GL.glScalef(self.globalScale, self.globalScale, self.globalScale)
    GL.glMultMatrixf(self.centerMatrix.data())

def wheelEvent(self,event):
    numPixels = QPoint(event.pixelDelta())
    numDegrees = QPoint(event.angleDelta() / 8)
    if (not numPixels.isNull()):
        self.globalScale = self.globalScale -
        ↪  (event.pixelDelta().x()+event.pixelDelta().y())/600
    elif (not numDegrees.isNull()):
        self.globalScale = self.globalScale - ((numDegrees.x()+numDegrees.y()) / 15)/600
    self.resetProjection()
    self.resetModelView()
    self.repaint()

def initializeGL(self):
    GL.glClearColor(1.0, 1.0, 1.0, 0.0)
    GL.glEnable(GL.GL_DEPTH_TEST)
    GL.glEnable(GL.GL_NORMALIZE)
    GL.glEnable(GL.GL_LIGHTING)
    #GL.glEnable(GL.GL_MULTISAMPLE)
    #обрезка внутренностей
    GL.glEnable(GL.GL_CULL_FACE)
    GL.glMatrixMode(GL.GL_PROJECTION)
    #альфа канал
    GL.glEnable(GL.GL_BLEND)
    GL.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA)
    GL.glEnable(GL.GL_TEXTURE_2D)
    self.textures = {}
    self.textures['stone'] = self.open_textures('stone')
    self.textures['metal'] = self.open_textures('metal')
    self.textures['entity'] = self.open_textures('entity')
    #освещение
    light_dir = [map_len_x/2,map_len_y/2,map_len_z/2]
    sp = [1.0, 1.0, 1.0, 1.0]
    self.light1Diffuse = [1.0, 0.0, 0.0]
    self.light2Diffuse = [0,0, 1.0, 0.0]
    light1Position = [map_len_x+1,map_len_y/2,map_len_z, 1.0]
    light2Position = [map_len_x+1,map_len_y/2,-map_len_z, 1.0]
    GL.glLightfv(GL.GL_LIGHT1, GL.GL_SPECULAR, sp)
    GL.glLightfv(GL.GL_LIGHT2, GL.GL_SPECULAR, sp)
    GL.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE, self.light1Diffuse)
    GL.glLightfv(GL.GL_LIGHT2, GL.GL_DIFFUSE, self.light2Diffuse)
    GL.glLightfv(GL.GL_LIGHT1, GL.GL_SPOT_DIRECTION, light_dir)
    GL.glLightfv(GL.GL_LIGHT1, GL.GL_POSITION, light1Position)
    GL.glLightfv(GL.GL_LIGHT2, GL.GL_POSITION, light2Position)
    GL.glTexEnvf(GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE, GL.GL_MODULATE)
    GL.glEnable(GL.GL_LIGHT1)
    GL.glEnable(GL.GL_LIGHT2)
    self.makeObject()

def resizeGL(self, w, h):
    GL.glViewport(0, 0, w, h)
    self.resetProjection()
```

```
def paintGL(self):
    GL.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT)
    GL.glMatrixMode(GL.GL_MODELVIEW)
    GL.glEnable(GLWidget.GL_MULTISAMPLE)
    self.makeObject()

def makeObject(self):
    for z in range(0,map_len_z-self.slice_z):
        for y in range(0,map_len_y-self.slice_y):
            for x in range(0,map_len_x-self.slice_x):
                self.geometry(p_map[z][y][x],x,y,z)
    self.centredScene(map_len_x,map_len_y,map_len_z)

def geometry(self,Block,x,y,z):
    block_color = QColor(
        Block.material.color_red,
        Block.material.color_green,
        Block.material.color_blue,
        Block.material.color_alpha
    )

    GL.glBindTexture(GL.GL_TEXTURE_2D, self.textures[Block.material.textureName])
    #self.qglColor(block_color)
    GL.glBegin(GL.GL_QUADS)
    #нижний полигон
    GL.glTexCoord2f(0.0, 1.0)
    GL.glVertex3d(x, y, z)

    GL.glTexCoord2f(1.0, 0.0)
    GL.glVertex3d(1+x, y, z)

    GL.glTexCoord2f(1.0, 1.0)
    GL.glVertex3d(1+x, y, 1+z)

    GL.glTexCoord2f(0.0, 0.0)
    GL.glVertex3d(x, y, 1+z)
    #GL.glEnd()

    #GL.glBegin(GL.GL_QUADS)
    #фронтальный полигон
    GL.glTexCoord2f(0.0, 0.0)
    GL.glVertex3d(x, y, z)

    GL.glTexCoord2f(1.0, 0.0)
    GL.glVertex3d(x ,y+1, z)

    GL.glTexCoord2f(1.0, 1.0)
    GL.glVertex3d(x+1, y+1, z)

    GL.glTexCoord2f(0.0, 1.0)
    GL.glVertex3d(x+1, y, z)
    #GL.glEnd()

    #GL.glBegin(GL.GL_QUADS)
    #верхний полигон
    GL.glTexCoord2f(0.0, 0.0)
    GL.glVertex3d(x, y+1, z)

    GL.glTexCoord2f(1.0, 0.0)
```

```
        GL.glVertex3d(x, y+1, 1+z)

        GL.glTexCoord2f(1.0, 1.0)
        GL.glVertex3d(1+x, y+1, 1+z)

        GL.glTexCoord2f(0.0, 1.0)
        GL.glVertex3d(1+x, y+1, z)
        #GL.glEnd()

        #GL.glBegin(GL.GL_QUADS)
        #задний полигон
        GL.glTexCoord2f(0.0, 1.0)
        GL.glVertex3d(x, y, z+1)

        GL.glTexCoord2f(1.0, 0.0)
        GL.glVertex3d(x+1, y, z+1)

        GL.glTexCoord2f(1.0, 1.0)
        GL.glVertex3d(x+1, y+1, z+1)

        GL.glTexCoord2f(0.0, 0.0)
        GL.glVertex3d(x ,y+1, z+1)
        #GL.glEnd()

        #GL.glBegin(GL.GL_QUADS)
        #боковой левый полигон
        GL.glTexCoord2f(0.0, 1.0)
        GL.glVertex3d(x, y, z)

        GL.glTexCoord2f(1.0, 0.0)
        GL.glVertex3d(x, y, z+1)

        GL.glTexCoord2f(1.0, 1.0)
        GL.glVertex3d(x, y+1, z+1)

        GL.glTexCoord2f(0.0, 0.0)
        GL.glVertex3d(x, y+1, z)
        #GL.glEnd()

        #GL.glBegin(GL.GL_QUADS)
        #боковой правый полигон
        GL.glTexCoord2f(0.0, 0.0)
        GL.glVertex3d(x+1, y, z)

        GL.glTexCoord2f(1.0, 0.0)
        GL.glVertex3d(x+1, y+1, z)

        GL.glTexCoord2f(1.0, 1.0)
        GL.glVertex3d(x+1, y+1, z+1)

        GL.glTexCoord2f(0.0, 1.0)
        GL.glVertex3d(x+1, y, z+1)
        GL.glEnd()

    def open_textures(self,textureName):
        texture = GL.glGenTextures(1)
        image = open('./texture/'+textureName+'/512x512.bmp')
        ix = image.size[0]
        iy = image.size[1]
        image = image.tobytes("raw", "RGBX", 0, -1)
```

```
        GL.glBindTexture(GL.GL_TEXTURE_2D, texture)    # 2d texture (x and y size)
        GL.glPixelStorei(GL.GL_UNPACK_ALIGNMENT,1)
        GL.glTexImage2D(GL.GL_TEXTURE_2D, 0, 3, ix, iy, 0, GL.GL_RGBA, GL.GL_UNSIGNED_BYTE,
        ↪  image)
        GL.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_CLAMP)
        GL.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_CLAMP)
        GL.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT)
        GL.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT)
        GL.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_LINEAR)
        GL.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.GL_LINEAR)
        GL.glTexEnvf(GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE, GL.GL_DECAL)
        print('ok')
        return texture

if __name__ == '__main__':

    app = QApplication(sys.argv)

    my_format = QGLFormat.defaultFormat()
    my_format.setSampleBuffers(True)
    QGLFormat.setDefaultFormat(my_format)

    widget = QWidget()
    drawWidget = GLWidget(None)

    layout = QGridLayout()
    toolsLayout = QVBoxLayout()

    xLayout = QVBoxLayout()
    yLayout = QVBoxLayout()
    zLayout = QVBoxLayout()

    light1_box = QCheckBox('Включить GL_LIGHT1')
    color1_button = QPushButton('Выбрать цвет для GL_LIGHT1')
    toolsLayout.addWidget(light1_box)
    toolsLayout.addWidget(color1_button)
    light2_box = QCheckBox('Включить GL_LIGHT2')
    color2_button = QPushButton('Выбрать цвет для GL_LIGHT2')
    toolsLayout.addWidget(light2_box)
    toolsLayout.addWidget(color2_button)

    hLayout = QHBoxLayout()
    slider_x = QSlider(Qt.Vertical)
    slider_x.setRange(0,map_len_x)
    label_slider_x = QLabel('X')
    xLayout.addWidget(slider_x)
    xLayout.addWidget(label_slider_x)
    hLayout.addLayout(xLayout)

    slider_y = QSlider(Qt.Vertical)
    slider_y.setRange(0,map_len_y)
    label_slider_y = QLabel('Y')
    yLayout.addWidget(slider_y)
    yLayout.addWidget(label_slider_y)
    hLayout.addLayout(yLayout)

    slider_z = QSlider(Qt.Vertical)
    slider_z.setRange(0,map_len_z)
    label_slider_z = QLabel('Z')
    zLayout.addWidget(slider_z)
```

```
zLayout.addWidget(label_slider_z)
hLayout.addLayout(zLayout)

toolsLayout.addLayout(hLayout)

layout.addLayout(toolsLayout,0,0,1,1)
layout.addWidget(drawWidget,0,1,1,6)

slider_x.sliderMoved.connect(drawWidget.slice_x_slot)
slider_y.sliderMoved.connect(drawWidget.slice_y_slot)
slider_z.sliderMoved.connect(drawWidget.slice_z_slot)
color1_button.pressed.connect(drawWidget.color_light1)
color2_button.pressed.connect(drawWidget.color_light2)
light1_box.toggled.connect(drawWidget.light1_check)
light2_box.toggled.connect(drawWidget.light2_check)
widget.resize(300,300)
widget.setLayout(layout)
widget.show()

sys.exit(app.exec_())
```