УДК 622:528.9:004
DOI: 10.25635/2313-1586.2018.04.051

Суханов Владимир Иванович
доктор технических наук, доцент, профессор
Центра ускоренного обучения
Института радиоэлектроники
и информационных технологий,
Уральский федеральный университет,
620002, Екатеринбург, ул. Мира, 19
e-mail: cyx-fat@mail.ru,

Sukhanov Vladimir I.
Doctor of Engineering,
Professor of the Center for accelerated education
of the Institute of radioelectronics
and information technologies,
Ural Federal University,
620002, Ekaterinburg, Mira str., 19
e-mail: cyx-fat@mail.ru

ПРОГРАММНЫЕ ПЛАТФОРМЫ ДЛЯ РАЗРАБОТКИ ГОРНОЙ ГИС

SOFTWARE PLATFORMS FOR THE DEVELOPMENT OF A MINING GIS

Аннотация:

Рассмотрены проблемы разработки отечественного свободного программного обеспечения горно-геологических информационных систем с открытым кодом для предприятий горной промышленности. Приведены основные сведения о программных платформах для разработки горной информационной системы на основе стандартов открытых систем и геометрического ядра с открытым кодом. Типовые задачи разработки иллюстрируются фрагментами кода на языке Python для графической библиотеки PyQt. Базовым программным обеспечением выбрано кроссплатформенное геометрическое ядро с открытым кодом Open CASCADE Technology, обеспечивающее создание и манипулирование геометрическими моделями элементов карьера в диалоге с пользователями. Инструментальные средства позволяют реализовать необходимые технологические операции по созданию модели месторождения и карьера, выбора направления развития горных работ, буровзрывных работ, организации транспортной инфраструктуры и других.

Ключевые слова: ГГИС, OGC, Open CASCADE, PythonOCC, PyQT, PostgreSQL*, программирование, карьер*

Abstract:

The problems of development of free software for mining and geological information systems with open source for mining enterprises are considered. The basic information about the program platforms for the development of mining information system on the basis of open system and geometric kernel standards is given. Typical development tasks are illustrated by Python code snippets for the PyQt graphics library. The basic software is a cross-platform geometric kernel with open-source Open CASCADE Technology, which provides creation and manipulation of geometric models of quarry elements in a dialogue with users. The tools allow to implement the necessary technological operations for the building of the field and pit model, for the choice of the mining development direction, for the drilling and blasting specifics, for the organization of transport infrastructure and others.

Keywords: MGIS, GIS, OGC, Open CASCADE, PythonOCC, PyQT, PostgreSQL, programming, pit, quarry

Введение

Удовлетворение потребностей в отечественном программном обеспечении горно-геологических информационных систем (ГГИС) ставит перед разработчиками следующие задачи:

1) Выбор архитектуры программно-аппаратного комплекса ГГИС.
2) Выбор базового программного обеспечения и средств разработки.
3) Разбиение информационных и технологических задач на модули, обеспечивающие потребности специалистов в компьютерных технологиях.
4) Выбор диалоговых и программных интерфейсов взаимодействия с пользователями и смежными информационными системами.
5) Программная реализация модулей и интерфейсов.

Основой аппаратных решений являются промышленные серверы, обладающие высокой надежностью, производительностью, объемами оперативной и дисковой памяти. Из-за требований высокой защищенности информационных систем в горном производстве

целесообразно отдавать предпочтения программному обеспечению с открытым кодом, что создает предпосылки для тщательной верификации программного обеспечения всех уровней от операционной системы до простейших утилит и сервисов.

Базовым программным обеспечением ГГИС следует считать операционные системы, системы управления базами данных и пакеты машинной графики. Остальное программное обеспечение (ПО) включает средства разработки и администрирования прикладных задач.

Разбиение информационных и технологических задач на модули задается правилами ведения и регламентами документирования горных работ, определяемыми разрешительными и надзорными инстанциями, с учетом специфики месторождений и технологий добычи полезных ископаемых. Обычно в их состав входят модели месторождения, карьерного пространства, отвалов, прилегающих территорий, коммуникаций и других инфраструктурных техногенных образований.

Выбор диалоговых и программных интерфейсов взаимодействия с пользователями и смежными информационными системами предполагает принятие решений об использовании локальных или сетевых взаимодействиях пользователей различных специальностей с информационными ресурсами и решении ими технологических задач. Основную сложность и объем имеют задачи геометрического моделирования, требующие точного позиционирования и манипулирования объектами в пространстве. Здесь имеют место существенно разные требования к задачам ввода/корректировки и просмотра геометрической информации. Существенное значение имеет учет полномочий пользователей и смежных подсистем на доступ к данным как на чтение, так и на запись.

Программная реализация модулей и интерфейсов — сложный, затратный эволюционный процесс с привлечением большого числа специалистов: аналитиков, кодировщиков, тестировщиков, менеджеров и программистов.

Анализ средств разработки

Open Geospatial Consortium (OGC) – международная добровольная организация по разработке стандартов и рекомендаций в области геоинформационных сервисов [1]. Ею подготовлены разнообразные стандарты и рекомендации по представлению и манипулированию геопространственными данными, на основе которых разработано большое количество общесистемного и прикладного программного обеспечения с открытым кодом, с использованием которого разработка доверенных горно-геологических систем для открытой разработки месторождений посильна для небольших коллективов разработчиков. Для реализации этих технологий предлагается следующее программное обеспечение: для работы с базами геоданных – СУБД PostgreSQL и ее расширение PostGIS [2], для администрирования БД и просмотра геоданных через веб-интерфейс – сервер GlassFish с установленным приложением GeoServer [3]. Редактирование геометрической информации – традиционно сложная отрасль компьютерных технологий, вследствие чего свободных проектов, удовлетворяющих требованиям ГГИС, практически нет. Имеющие чрезвычайно слабы по функциональности и интерфейсам с пользователями. Однако существует программный продукт с конкурентными возможностями для разработки редакторов геометрических данных и решения технологических задач – геометрическое ядро с открытым кодом Open CASCADE (OCC) [4, 5], имеющий программные интерфейсы с системой программирования Python и библиотеку PythonOCC.

Базовый пакет OCC создан для использования в программах на языке C++. В последнее время программисты все больше отдают предпочтение скриптовым безтиповым языкам, в частности, языку Python, позволяющим существенно сократить объем текста и повысить скорость разработки программ, пренебрегая незначительной потерей скорости работы приложений. Группа единомышленников разработала для OCC интерфейс про-

граммирования приложений PythonOCC на языке Python [6, 7], снабдив его высокотехнологичными средствами установки на ПК разработчика, что сделало эту платформу безусловно привлекательной.

Для отображения геометрических объектов на экране дисплея PythonOCC имеет интерфейсы с двумя распространенными графическими библиотеками *wxPython* и *PyQt* [8]. Обе библиотеки функционально эквивалентны, позволяют отображать геометрические объекты и манипулировать ими в диалоге с пользователем [9, 10]. При сравнении их технологических свойств предпочтение можно отдать библиотеке PyQt. Пакет Qt – кроссплатформенная свободная библиотека, входящая в стандартную комплектацию большинства операционных систем. Расширение *PyQt* включено во все репозитории бинарных пакетов операционных систем семейства Linux. Внешний вид приложения на платформе *PyQt5* показан на рис. 1

Установка пакета PythonOCC для программирования на языке Python версии 3.6 выполняется следующей последовательностью команд `pip3 install conda`

```
conda create --name OCE
conda info --env
conda activate OCE
conda install -c conda-forge -c dlr-sc -c pythonocc -c oce pythonocc-
        core==0.18.1
```

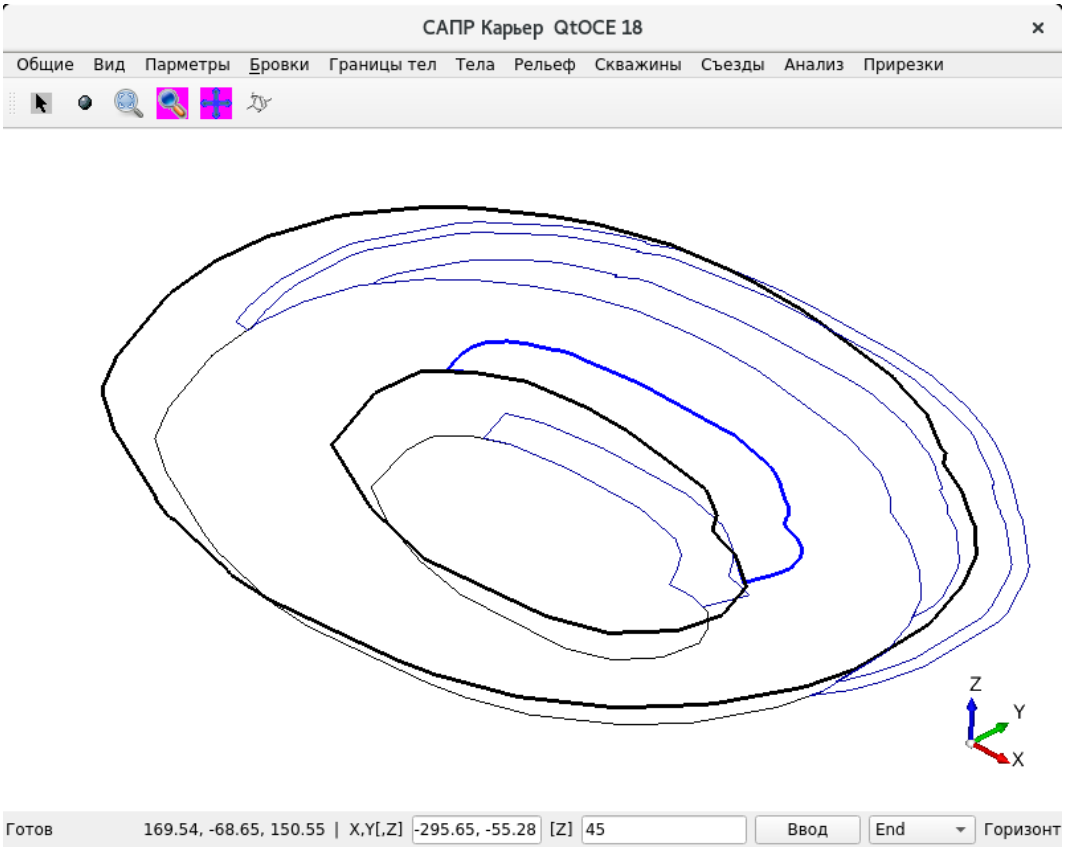

Рис. 1 – Интерфейсы приложения на платформе PyQt5

Для проверки работоспособности можно создать папку ~/.conda/envs/OCE/Public/OCEexamples
Скопировать из http://www.pythonocc.org/download/ файл pythonocc-core-0.18.1.zip, извлечь из его папки pythonocc-core-0.18.1/examples примеры для демонстрации и запустить
python3.6 core_classic_occ_bottle.py

Работа с базами данных

Доступ к БД PostgreSQL в языке Python выполняется с использованием драйвера psycopg2 [6], устанавливаемого командой conda install psycopg2. Пример создания соединения с БД имеет следующий вид:

```
conn = psycopg2.connect("dbname=test user=postgres")
curs = conn.cursor()
```

Для иллюстрации техники работы с БД геометрических данных приведем пример таблицы *edge* для хранения бровок уступов горного карьера. Операторы для создания таблицы имеют следующий вид:

```
CREATE TABLE edge
(
  id_edge serial NOT NULL,
  hor integer NOT NULL,
  edge_type integer NOT NULL,
  CONSTRAINT "PK_edge" PRIMARY KEY (id_edge),
  CONSTRAINT edg_typ FOREIGN KEY (edge_type)
      REFERENCES edge_type (id_edge_type) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT "edge-hor" FOREIGN KEY (hor)
      REFERENCES horizons (id_hor) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
ALTER TABLE edge ADD COLUMN geom geometry(LINESTRINGZ,0);
```

Поля *hor* и *edge_type* являются внешними ключами описания горизонта и типа бровки. Основное назначение таблицы – хранить геометрию бровок уступа в поле *geom*.

Для работы с геометрическими объектами в период исполнения приложения ОСС необходима структура данных в памяти ЭВМ, содержащая текущую информацию об объекте. Для этого используется родительский класс *Geom* со следующими методами экземпляров:

getPnts / setPnts - запрос/установка точек линии;

getColor / setColor - запрос/установка цвета объекта;

getWidth / setWidth - запрос/установка толщины линии;

show / hide - показать/скрыть объект;

activate / deactivate - разрешить/запретить выбор объекта;

modified - признак изменений в объекте.

Остальные классы бровок, рудных тел съездов, тополгии поверхности и других геометрических объектов наследуют свойства родительского класса *Geom*, дополняя его специфическими атрибутами. Например, класс бровок содержит дополнительный метод *save* для сохранения геометрии и атрибутов бровки в БД.

```
def save(self, curs):
        global SubstId
```

```
        if self.modified:
            if (self.id_obj > -1)and(not self.shape):
                query = "DELETE FROM edge WHERE id_edge=" +
        str(self.id_obj) + ";"
                curs.execute(query)
            elif (self.id_obj < -1) and self.shape:
                # Добавить в БД
                edge_type = 1
                geom = makeLINESTRING(self.pnts)
                query = "INSERT INTO edge (hor,edge_type,geom) VALUES
        ("+\

        str(self.id_hor)+","+str(edge_type)+","+geom+")    RETURNING
        id_edge;"
                curs.execute(query)
                id_obj=curs.fetchone()[0]
                SubstId[self.id_obj]=id_obj
                self.id_obj=id_obj
            elif ((self.id_obj > -1) and self.shape):
                # Модифицировать в БД
                geom = makeLINESTRING(self.pnts)
                query = "UPDATE edge SET geom=" + geom + " WHERE
        id_edge=" + str(self.id_obj) + ";"
                curs.execute(query)
            self.modified = False
```

Фрагмент загрузки геометрических объектов из БД имеет следующий вид:

```
query = "select
         id_edge,hor,edge_type,ST_AsEWKT(geom),point,color,thickness
         from edge,horizons,edge_type "
        query = query + "where (id_hor in " + setHorIds + ") and
         (edge.hor=horizons.id_hor)                                  and
         (edge.edge_type=edge_type.id_edge_type);"
        self.msgWin.AppendText("Query = " + query + "\n")
        curs.execute(query)
        rows = curs.fetchall()
        for rec in rows:
            id_edge = int(rec[0])
            id_horBr = int(rec[1])
            edge_type = int(rec[2])
            if (not rec[3]): continue  # нет геометрии
            coordsPLine = parsGeometry(str(rec[3]))
            point = float(rec[4])  # отметка горизонта
            for pnt in coordsPLine:
                if len(pnt)<3:
                    pnt.append(point)
            id_color=rec[5]
            thickness = int(rec[6])
            colorLine = Color.getQuantity_ColorById(id_color)
            obj=EdgeDn(id_edge,  coordsPLine,  id_horBr,  color=col-
         orLine, width=thickness,closed=False)
```

Аналогичные средства предоставляет сам пакет PyQt5. Пример фрагмента для работы с БД имеет следующий вид:

```
# Открываем базу данных
con1 = QtSql.QSqlDatabase.addDatabase('QPSQL7')
con1.setDatabaseName('gisDB')
con1.setUserName('postgres')
con1.setPassword('postgres')
con1.open()
query = QtSql.QSqlQuery()
query.exec("select * from edge_type")
lst = []
if query.isActive():
    query.first()
    while query.isValid():
        lst.append(query.value('name') + ': ' +
        str(query.value('color')))
        query.next()
    for p in lst: print(p)
con1.close()
```

Программирование модулей ГИС для работы с объектами

Главный модуль программы определяет класс *MainWindow* с конструктором __init__, где описаны основные органы управления приложения (меню и палитры инструментов), связанные с соответствующими обработчиками событий. Обработчики событий – методы того же класса, описанные в тексте модуля.

Модуль *QtWidgets.QMainWindow* отвечает за взаимодействие с графической оболочкой PyQt5 и содержит обработчики таких событий, как перемещение указателя (мышки), нажатие и отпускание кнопок мышки и клавиатуры, изменение размеров окна и другие.

Главный модуль имеет классическую для технологии PyQt5 структуру, включающую объявление главного оконного класса и головную программу.

```
class MainWindow(QtWidgets.QMainWindow):
    """ Главная программа САПР Карьер Qt5
    """
    def __init__(self, arg=None):
        """ Главное окно приложения
        """
        QtWidgets.QMainWindow.__init__(self, arg)
        self.canva = qtViewer3d(self)
        self.setWindowTitle("САПР Карьер %s" % VERSION)
        self.setCentralWidget(self.canva)
        self.layout = QtWidgets.QHBoxLayout()
        self.optDlg = ParamsDlg()
        self.menu_bar = self.menuBar()
                ...
```

Создание статусной строки и панели инструментов имеют следующий вид:

```
self.stbMain = self.statusBar()
self.stbMain.setSizeGripEnabled(False)
self.lblReady = QtWidgets.QLabel("Готов")
self.stbMain.addWidget(self.lblReady, 3)
self.lblCursor = QtWidgets.QLabel("Курсор")
self.stbMain.addWidget(self.lblCursor, 3)
self.lblXY = QtWidgets.QLabel("X,Y[,Z]")
self.stbMain.addWidget(self.lblXY, 3)
self.canva.coordXYZ = QtWidgets.QLineEdit()
self.canva.coordXYZ.resize(100,20)
self.stbMain.addWidget(self.canva.coordXYZ, 3)
self.yes = QtWidgets.QPushButton("Ввод")
self.stbMain.addWidget(self.yes, 3)
self.yes.clicked.connect(self.onCoord_yes)
self.canva.snap = QtWidgets.QComboBox()
self.canva.snap.addItems(['None',    'End',    'Near',    'Center',
        'Tangent'])
self.stbMain.addWidget(self.canva.snap, 3)
self.lblHoriz = QtWidgets.QLabel("Горизонт")
self.stbMain.addWidget(self.lblHoriz, 3)
```

Будем считать, что мировая система координат является правосторонней прямоугольной евклидовой системой с направлением осей: X – вправо, Y – вверх, Z – на наблюдателя. При вводе и редактировании геометрических координат объектов важным является возможность их интерактивного задания с использованием указателя мышки на экране. Для этого добавим в текст обработчика события *mouseMoveEvent* в модуле qtDisplay следующий фрагмент.

```
# Расчет позиции точки в мировых координатах
pt = point(evt.pos())
(x, y, z, vx, vy, vz) = self._display.View.ConvertWithProj(pt.x,
        pt.y)
self.frame.lblCursor.setText("%.2f, %.2f, %.2f"%(x,y,z))
```

При вводе координат левой кнопкой мышки следует исходить из того, что точное прицеливание практически невозможно и пользователю нужно дать возможность вручную откорректировать координаты указателя в обычном окне редактирования при дальнейшем их использовании. Фрагмент обработчика *mousePressEvent* имеет следующий вид:

```
if event.button() == QtCore.Qt.LeftButton:
   pt = point(event.pos())
   (x, y, z, vx, vy, vz) = self._display.View.ConvertWithProj(pt.x,
        pt.y)
   self.frame.canva.coordXYZ.setText("%.2f, %.2f, %.2f"%(x,y,z))
```

Пользователь может вручную редактировать все три координаты точки перед передачей их на исполнение командам построения и изменения геометрических объектов. Использование координат для геометрических построений можно разрешить, добавив поле редактора и кнопку "Ввод" в статусную строку приложения:

```
self.canva.coordXYZ = QtWidgets.QLineEdit()
self.yes = QtWidgets.QPushButton("Ввод")
self.stbMain.addWidget(self.canva.coordXYZ, 1)
self.stbMain.addWidget(self.yes, 0)
self.yes.clicked.connect(self.onCoord_yes)
```

Обработчик события *onCoord_yes* кнопки "Ввод" будет выполнять манипуляции с координатами из окна редактора в зависимости от исполняемой команды, ранее задаваемой пользователем средствами меню и кнопок. Обработчик кнопки «Ввод» будет выглядеть так:

```
def onCoord_yes(self):
    """ Ввод координат из окна coordZ по кнопке Ввод """
    Z = float(self.canva.coordZ.text())
    coordStr = self.canva.coord.text()
    coord1 = [float(v) for v in coordStr.split(',')]
  ...
```

Если пользователь ввел не менее двух точек и решил закончить полилинию, повторив координаты или введя символ "*C*", то на экране строится объект средствами OCC. Иначе новая точка добавляется в список вершин *lstPnt*. Вспомогательное окно ввода *coordZ* позволяет сократить ввод третьей координаты в окне координат "*X,Y,Z"* за счет использования значения *Z* по умолчанию из окна *coordZ.*

Булевские операции над геометрическими объектами

Булевские операции объединения, пересечения и вычитания геометрических фигур часто возникают при выполнении анализа обстановки или метрических характеристик объектов. Разработчику программ предоставляется возможность использовать для этих целей инструментарий, предоставляемый как OCC, так и PostGIS. Упомянутые функционально эквивалентные средства предоставляют различные интерфейсы для их использования. Более удобными, а следовательно, и целесообразными, являются интерфейсы функций PostGIS. В приведенном ниже методе используются инструменты выполнения булевских операций PostGIS. Исходные геометрии задаются списками точек lstPnt1 и lstPnt2, а выполняемая операция параметром op='*Union*' для объединения, '*Intersection*' для пересечения и '*Difference*' для разности фигур. Результатом является список полигонов (возможно, нескольких), задаваемых координатами их вершин.

```
def boolOp(lstPnt1, lstPnt2, op='Union'):
    # op ::=  Difference | Intersection | Union
    # результат [[p1,...,pn], ...]
    geom1 = "ST_AsEWKT('POLYGON(("
    cnt = len(lstPnt1); i = 0
    for pnt in lstPnt1:
        i = i + 1; geom1 = geom1 + "%.0f %.0f" % (pnt[0], pnt[1])
        if (i < cnt):geom1 = geom1 + ","
    geom1 = geom1 + "))')"; geom2 = "ST_AsEWKT('POLYGON(("
    cnt = len(lstPnt2); i = 0
    for pnt in lstPnt2:
        i = i + 1; geom2 = geom2 + "%.0f %.0f" % (pnt[0], pnt[1])
        if (i < cnt): geom2 = geom2 + ","
    geom2 = geom2 + "))')"
    query = "SELECT ST_AsEWKT(ST_" + op + "(" + geom1 + "," +
          geom2 + "));"
    conn = psycopg2.connect("dbname="+POSTGR_DBN+"
          user="+POSTGR_USR)
    curs = conn.cursor(); curs.execute(query)
```

```
    rows = curs.fetchall(); result = []
    for rec in rows:
        geom = rec[0]
        if 'MULTIPOLYGON' in geom:
            dlt = 3; strgeom = geom[13:]; startpos = 2
            while 1:
                endpos = strgeom.find('))')
                if endpos < 0: break
                coords = strgeom[startpos:endpos]
                lstCoords = coords.split(',')
                lstXYZ=[]
                for pnt in lstCoords:
                    pntXYZ=[]; xyz=pnt.split(' ')
                    for val in xyz:
                       pntXYZ = pntXYZ + [float(val)]
                    lstXYZ = lstXYZ + [pntXYZ]
                result = result + [lstXYZ]
                strgeom = strgeom[endpos+3:]; startpos = 2
        elif 'POLYGON' in geom:
            dlt = 2; plgn = parsGeometry(geom,dlt)
            result = result + [plgn]
    curs.close()
    conn.close()
    return result
```

Редактирование вершин полилиний

Редактирование вершин полилиний связано со вставкой новой, удалением или переносом существующей. Учитывая необходимость контроля правильности задания координат, такое преобразование следует выполнять при вводе положения редактируемой вершины.

Фрагмент обработчика кнопки «Ввод», выполняющий добавление, удаление и перемещения вершины полилинии, задаваемой выбранным пользователем объектом *self.canva.selShape*:

```
if (self.canva.Operation in ["InsPnt","DelPnt","MovePnt"]):
      # self.canva.lstPnt[-1] - Координаты точки
      # self.canva.selShape   - Объект для редактирования
      resPnt = lstXY                # Координаты в окне X,Y,Z
      if (self.canva.Operation in ["MovePnt"]) and
                  (len(self.canva.lstPnt)<2):
          self.SetStatusText("Нет точки", 2)
          return
      pnts = getPoints(self.canva.selShape)
      p1 = pnts[0]; newPnts = [p1]
      for p2 in pnts[1:]:
          if (self.canva.Operation in ["InsPnt"]):
              if inline(resPnt, p1, p2, eps=0.5):
                  newPnts.append(resPnt)
              newPnts.append(p2)
          elif (self.canva.Operation in ["DelPnt"]):
              if not near(self.canva.lstPnt[-1],p2):
                  newPnts.append(p2)
          elif (self.canva.Operation in ["MovePnt"]):
              if near(self.canva.lstPnt[-2],p2, eps=0.5):
                  newPnts.append(self.canva.lstPnt[-1])
              else: newPnts.append(p2)
          p1 = p2
```

Булевские функции *inline* и *near* производят проверку принадлежности точки отрезку и ее близости к точке.

Заключение

Материал статьи описывает основные понятия и требования при создании геоинформационных приложений, согласованных с рекомендациями OGC и поддерживающих их программных продуктов для хранения и отображения геоданных. Приведенные примеры реализации основных функций хранения и манипулирования геоданными позволяют малым коллективам самостоятельно вести разработку ГИС без привлечения дорогостоящих коммерческих продуктов, гибко адаптируя к требованиям предметной области их использования. Эл. почта автора cyx-fat@mail.ru.

Литература

1. Стандарты OGC [Электронный ресурс] - Режим доступа: http://gis-lab.info/wiki/Стандарты_OGC. 12.05.2014

2. PostGIS 2.0 Manual [Электронный ресурс] - Режим доступа: http://postgis.org. 20.09.2013

3. GeoServer User Manual . Release 2.0.1 [Электронный ресурс] - Режим доступа: http://docs.geoserver.org/stable/en/user/. 20.09.2013

4. Open CASCADE Technology [Электронный ресурс] - Режим доступа: http://www.opencascade.org/. 20.09.2013

5. Обзор возможностей геометрических ядер с открытым кодом для задач построения геоинформационного обеспечения горного производства / В.И. Суханов и др. // Известия вузов. Горный журнал. – 2010. - № 8. - С. 87 – 102.

6. Psycopg [Электронный ресурс] - Режим доступа: http://initd.org/psycopg/. 20.09.2013

7. pythonOCC – 3D CAD for python. [Электронный ресурс] - Режим доступа: http://www.pythonocc.org/. 14.06.2018

8. Суханов В.И. Разработка горно-геологической информационной системы по стандартам открытых систем / В.И. Суханов, В.М. Аленичев // Горный информационно-аналитический бюллетень. – 2015. - № 11 - С. 320 – 329.

9. Съедина С.А. Разработка 3D геомеханических моделей для подземных рудников и карьеров РАН / С.А. Съедина, А.А. Балтиева, Л.С. Шамганова // Проблемы недропользования. – 2018. – № 1. – С. 60 – 65. DOI: 10.25635/2313-1586.2018.01.060

10. Хоютанов Е.А. Моделирование угольных месторождений заполярной зоны Якутии / Е.А. Хоютанов, В.Л. Гаврилов // Проблемы недропользования. – 2017. – №4. – С. 53 – 60. DOI: 10.18454/2313-1586.2017.04.053