Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский политехнический университет»

**факультет информационных технологий**

**Кафедра СМАРТ-технологий**

Дисциплина: Технологии визуализации данных систем управления

Отчёт по лабораторной работе №2

«Применение проекций и матриц трансляции при визуализации данных»

Работа выполнена:

<ФИО>

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<ФИО>

**Цель работы**

Подготовить приложение на языке C# для визуализации собранных наборов пространственных данных методом проекции на плоскость.

**Задачи**

* Реализовать загрузку набора данных, заданных как тройки чисел (X, Y, Z) из файлов формата CSV (разделитель – точка с запятой) и генерацию заданного (через текстовое поле) количества случайных точек, где X,Y и Z – равномерно распределенные случайные величины на диапазоне [ -1 ÷ 1].
* Разработать генератор случайных величин на основе пространственной параметрической функции (от номера точки) с добавлением случайного смещения по трем координатам, распределенного по нормальному закону с заданным параметром σ (СКО) [функция определяется индивидуальным заданием к лабораторной работе].
* Разработать систему отображения данных в виде облака точек на битовой карте (bitmap) с использованием прямой (ортогональной) проекции на координатные плоскости. Размер битовой карты – не менее 600 х 600 px. Применять изменение яркости точек для отображения удаленности точек от наблюдателя.
* Разработать систему визуализации с использованием преспективной проекции облака точек, полагая, что область распределения точек можно принять за куб с длиной грани в 1 м, размещенной на расстоянии 4 метров от наблюдателя.
* Разработать систему поворота базовой системы координат размещения облака точек (относительно плоскости экрана) с помощью матрицы трансляцииповорота. Реализовать инструмент редактирования матрицы трансляцииповорота через DataGrid или текстовые поля.
* Реализовать инструмент инкрементального поворота базовой системы коор- динат путем использования механизма Drag&Drop позиционным манипулятором. Каждое перетаскивание интерпретировать как локальное приращение угла поворота вокруг осей 0X и 0Y в системе координат, связанной с плоскостью экрана, но с центром, размещенным в начале системы координат набора точек.
* Реализовать отображение точек с учетом накопленного поворота, а также отображение базовых векторов системы координат.
* Реализовать сохранение полученного изображения в файл.

**Индивидуальное задание**

L02\_04: использовать в качестве пространственной функции параметрическое

представление функции.

При необходимости выполнить масштабирование в интервал [ -1 ÷ 1] по

каждому измерению.

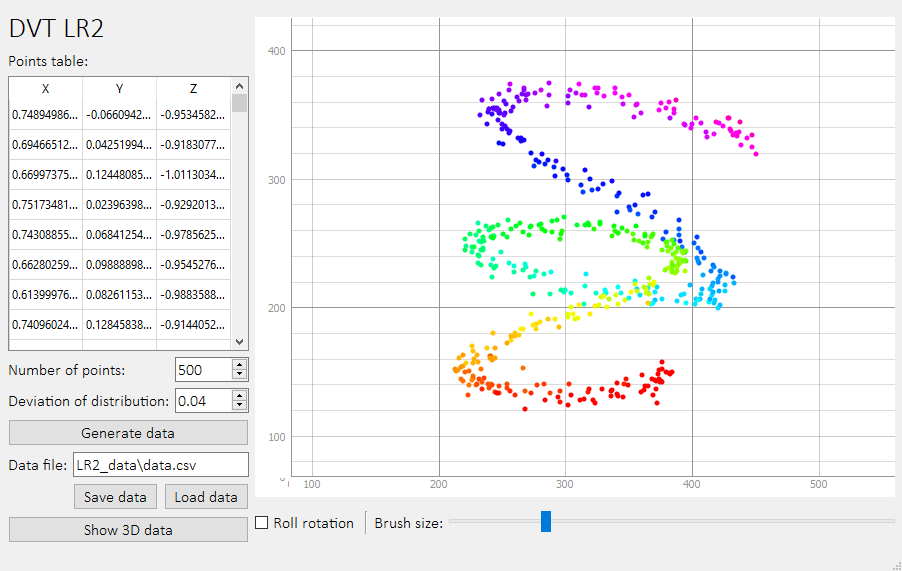
x = 0.7 \* cos (6 \* pi \* (i / N))

y = 0.5 \* sin (4 \* pi \* (i / N))

z = -1 + 2 \* i / N (0 <= i <= N)

**Ход работы**

Для генерации данных на форме имеется возможность выбрать количество точек и среднеквадратическое отклонение координат точки от вычисленного положения.



Значения генерируются по заданной формуле, в данном случае, генерация координат выглядит следующим образом:

**for** i **in** range(points\_num):  
 self.points.append([0.7 \* math.cos(6 \* math.pi \* (i / points\_num)) +  
 random.normalvariate(0, deviation\_val),  
  
 0.5 \* math.sin(4 \* math.pi \* (i / points\_num)) +  
 random.normalvariate(0, deviation\_val),  
  
 -1 + 2 \* i / points\_num + random.normalvariate(0, deviation\_val)])

Сгенерированные данные хранятся в массиве. После генерации или загрузки из CSV-файла, координаты отображаются в таблице. Также, сгенерированные данные можно сохранить в CSV-файл. В качестве разделителя используется символ пробела, как описано в задании.

Для отрисовки 3D точек на 2D – плоскость использется элемент PlotWidget (Двух-мерный график класса pyqtgraph) и математические преобразования для поворота точек.

Функция поворота точек:

def rotate2d(pos, rad):  
 x, y = pos  
 s, c = math.sin(rad), math.cos(rad)  
 return x \* c - y \* s, y \* c + x \* s

Для отрисовки, точки последовательно прочитываются в цикле из массива, и к парам x-y x-z и y-z применяются функции поворота по осям Roll, Yaw и Pitch соответсвенно. Поврот происходит при помощи зажатия колёсика или правой кнопки мыши и движения по графику. После окончания поворота или перемещения графиков предыдущая позиция указателя сохраняется, чтобы иметь возможность удобного неограниченного перемещния / вращения.

Для пермещения необходимо зажать левую кнопку мыши и перемещать курсор.

**Вывод**

В ходе данной работы было создано приложение приложение для визуализации собранных наборов пространственных данных методом проекции на плоскость.

**Исходный код**

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*import math  
import os  
import random  
import sys  
  
import matplotlib.pyplot as plt  
import numpy as np  
from PyQt5 import uic, QtGui, QtWidgets, QtCore  
from PyQt5.QtWidgets import QApplication, QMainWindow, QTableWidgetItem  
  
**"""  
Rotates points by angle  
"""**def rotate2d(pos, rad):  
 x, y = pos  
 s, c = math.sin(rad), math.cos(rad)  
 return x \* c - y \* s, y \* c + x \* s  
  
  
**"""  
Arduino map function  
"""**def valmap(value, istart, istop, ostart, ostop):  
 return ostart + (ostop - ostart) \* ((value - istart) / (istop - istart))  
  
  
class Window(QMainWindow):  
 def \_\_init\_\_(self):  
 super(Window, self).\_\_init\_\_()  
 *# Load GUI file* self.gui = uic.loadUi(**'LR2.ui'**)  
 self.gui.show()  
  
 self.points = []  
 self.width = self.gui.graphWidget.width()  
 self.height = self.gui.graphWidget.height()  
 self.center = np.array([int(self.width / 2), int(self.height / 2)])  
  
 *# Mouse variables* self.mouse = self.center  
 self.mouse\_move = np.array([0, 0])  
 self.mouse\_rot = np.array([0, 0])  
 self.mouse\_move\_released = self.center  
 self.mouse\_rot\_released = self.center  
 self.mouse\_pressed = self.center  
 self.rot\_flag = False  
 self.move\_flag = False  
  
 *# Connect GUI controls* self.gui.btn\_generate\_data.clicked.connect(self.generate\_data)  
 self.gui.btn\_show\_3d.clicked.connect(self.draw\_points)  
 self.gui.btn\_save\_data.clicked.connect(self.save\_data)  
 self.gui.btn\_load\_data.clicked.connect(self.load\_data)  
 self.gui.brush\_size.valueChanged.connect(self.draw\_points)  
  
 *# Initialize charts and tables* self.init\_chart()  
 self.init\_table()  
  
 **"""  
 Initializes table of points  
 """** def init\_table(self):  
 self.gui.points\_table.setColumnCount(3)  
 self.gui.points\_table.verticalHeader().setVisible(False)  
 self.gui.points\_table.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
 self.gui.points\_table.setHorizontalHeaderItem(0, QtWidgets.QTableWidgetItem(**'X'**))  
 self.gui.points\_table.setHorizontalHeaderItem(1, QtWidgets.QTableWidgetItem(**'Y'**))  
 self.gui.points\_table.setHorizontalHeaderItem(2, QtWidgets.QTableWidgetItem(**'Z'**))  
 header = self.gui.points\_table.horizontalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 header.setSectionResizeMode(1, QtWidgets.QHeaderView.Stretch)  
 header.setSectionResizeMode(2, QtWidgets.QHeaderView.Stretch)  
  
 **"""  
 Initializes main chart  
 """** def init\_chart(self):  
 self.gui.graphWidget.setBackground(QtGui.QColor(**'white'**))  
 *# self.gui.graphWidget.getAxis('left').setPen(QtGui.QColor('black'))  
 # self.gui.graphWidget.getAxis('left').setTextPen(QtGui.QColor('black'))  
 # self.gui.graphWidget.getAxis('bottom').setPen(QtGui.QColor('black'))  
 # self.gui.graphWidget.getAxis('bottom').setTextPen(QtGui.QColor('black'))* self.gui.graphWidget.setRange(xRange=(0, 640), yRange=(0, 480), padding=0)  
 self.gui.graphWidget.showGrid(x=True, y=True, alpha=1.0)  
  
 self.gui.graphWidget.mouseMoveEvent = self.mouse\_move\_event  
 self.gui.graphWidget.mouseReleaseEvent = self.mouse\_release\_event  
 self.gui.graphWidget.mousePressEvent = self.mouse\_press\_event  
  
 **"""  
 Generates data  
 """** def generate\_data(self):  
 print(**'Generating data...'**)  
 points\_num = self.gui.points\_num.value()  
 deviation\_val = self.gui.deviation\_val.value()  
 self.points = []  
 for i in range(points\_num):  
 self.points.append([0.5 \* math.cos(4 \* math.pi \* (i / points\_num)) + *# 0.7, 6* random.normalvariate(0, deviation\_val),  
  
 0.5 \* math.sin(4 \* math.pi \* (i / points\_num)) + *# 0.5, 4* random.normalvariate(0, deviation\_val),  
  
 -1 + 2 \* i / points\_num + random.normalvariate(0, deviation\_val)])  
 self.points = np.array(self.points)  
  
 self.show\_on\_table()  
 self.gui.btn\_save\_data.setEnabled(True)  
 print(**'Data generated.'**)  
  
 **"""  
 Shows points in table  
 """** def show\_on\_table(self):  
 self.gui.points\_table.setRowCount(0)  
 for point in self.points:  
 row\_position = self.gui.points\_table.rowCount()  
 self.gui.points\_table.insertRow(row\_position)  
 self.gui.points\_table.setItem(row\_position, 0, QTableWidgetItem(str(point[0])))  
 self.gui.points\_table.setItem(row\_position, 1, QTableWidgetItem(str(point[1])))  
 self.gui.points\_table.setItem(row\_position, 2, QTableWidgetItem(str(point[2])))  
 self.gui.btn\_save\_data.setEnabled(True)  
  
 **"""  
 Draws 3D points over 2D chart  
 """** def draw\_points(self):  
 if self.points is not None and len(self.points) > 0:  
 *# Clear old points* self.gui.graphWidget.clear()  
  
 *# Create color map* z = np.array(np.array([item[2] for item in self.points]))  
 cmap = plt.get\_cmap(**'hsv'**)  
 min\_z = np.min(z)  
 max\_z = np.max(z)  
  
 *# Draw each point* for point in self.points:  
 x, y, z = point  
 rgb\_z = cmap(0.9 - ((z - min\_z) / (max\_z - min\_z)))  
  
 *# Roll* if self.gui.roll\_enabled.isChecked():  
 x, y = rotate2d((x, y), math.radians((self.mouse\_rot[0] + self.mouse\_rot[1]) / 2))  
 *# Yaw* x, z = rotate2d((x, z), math.radians(self.mouse\_rot[0]))  
 *# Pitch* y, z = rotate2d((y, z), math.radians(self.mouse\_rot[1]))  
  
 z += 5  
 f = self.width / z  
 x, y = x \* f, y \* f  
  
 self.gui.graphWidget.plot([(self.center[0] + x) + self.mouse\_move[0]],  
 [(self.center[1] + y) + self.mouse\_move[1]],  
 pen=None,  
 symbolBrush=(rgb\_z[0] \* 255, rgb\_z[1] \* 255, rgb\_z[2] \* 255),  
 symbolSize=self.gui.brush\_size.value(),  
 symbolPen=None)  
  
 **"""  
 Moves or rotates points with mouse  
 """** def mouse\_move\_event(self, event):  
 if event.buttons():  
 self.mouse = np.array([event.pos().x(), (event.pos().y() - 480) \* -1])  
 if event.buttons() & QtCore.Qt.LeftButton:  
 *# Move* self.move\_flag = True  
 self.rot\_flag = False  
 self.mouse -= self.mouse\_pressed - self.mouse\_move\_released  
 self.mouse\_move = self.mouse - self.center  
 else:  
 *# Rotate* self.move\_flag = False  
 self.rot\_flag = True  
 self.mouse -= self.mouse\_pressed - self.mouse\_rot\_released  
 self.mouse\_rot = self.mouse - self.center  
 np.floor\_divide(self.mouse\_rot, 2)  
  
 self.draw\_points()  
  
 **"""  
 Remembers final position (when stop moving)  
 """** def mouse\_release\_event(self, event):  
 if self.move\_flag:  
 self.mouse\_move\_released = self.mouse  
 if self.rot\_flag:  
 self.mouse\_rot\_released = self.mouse  
  
 **"""  
 Remembers entry position (when start moving)  
 """** def mouse\_press\_event(self, event):  
 self.mouse\_pressed = np.array([event.pos().x(), (event.pos().y() - 480) \* -1])  
  
 **"""  
 Saves points to CSV file  
 """** def save\_data(self):  
 print(**'Saving data...'**)  
 np.savetxt(self.gui.data\_file.text(), self.points, delimiter=**' '**)  
 print(**'File'**, self.gui.data\_file.text(), **'saved.'**)  
  
 **"""  
 Loads points to CSV file  
 """** def load\_data(self):  
 if os.path.exists(self.gui.data\_file.text()):  
 print(**'Loading data...'**)  
 self.points = np.loadtxt(self.gui.data\_file.text(), delimiter=**' '**)  
 self.show\_on\_table()  
 print(**'File'**, self.gui.data\_file.text(), **'loaded.'**)  
 else:  
 print(**'File'**, self.gui.data\_file.text(), **'doesn**\'**t exist!'**)  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QApplication(sys.argv)  
 win = Window()  
 sys.exit(app.exec\_())

<?xml version="1.0" encoding="UTF-8"?>

<!--

Licensed under the Unlicense License;

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

https://unlicense.org

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

-->

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>905</width>

<height>572</height>

</rect>

</property>

<property name="font">

<font>

<family>Calibri Light</family>

<pointsize>12</pointsize>

</font>

</property>

<property name="windowTitle">

<string>DVT LR2</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QHBoxLayout" name="horizontalLayout\_3">

<item>

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QLabel" name="label">

<property name="font">

<font>

<pointsize>22</pointsize>

</font>

</property>

<property name="text">

<string>DVT LR2</string>

</property>

</widget>

</item>

<item>

<widget class="QLabel" name="label\_4">

<property name="text">

<string>Points table:</string>

</property>

</widget>

</item>

<item>

<widget class="QTableWidget" name="points\_table">

<property name="sortingEnabled">

<bool>true</bool>

</property>

<attribute name="horizontalHeaderShowSortIndicator" stdset="0">

<bool>true</bool>

</attribute>

<attribute name="verticalHeaderShowSortIndicator" stdset="0">

<bool>false</bool>

</attribute>

</widget>

</item>

<item>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<layout class="QFormLayout" name="formLayout">

<item row="0" column="0">

<widget class="QLabel" name="label\_2">

<property name="text">

<string>Number of points:</string>

</property>

</widget>

</item>

<item row="0" column="1">

<widget class="QSpinBox" name="points\_num">

<property name="minimum">

<number>1</number>

</property>

<property name="maximum">

<number>99999</number>

</property>

<property name="value">

<number>50</number>

</property>

</widget>

</item>

<item row="1" column="0">

<widget class="QLabel" name="label\_3">

<property name="text">

<string>Deviation of distribution:</string>

</property>

</widget>

</item>

<item row="1" column="1">

<widget class="QDoubleSpinBox" name="deviation\_val">

<property name="value">

<double>0.040000000000000</double>

</property>

</widget>

</item>

</layout>

</item>

<item>

<widget class="QPushButton" name="btn\_generate\_data">

<property name="text">

<string>Generate data</string>

</property>

</widget>

</item>

</layout>

</item>

<item>

<layout class="QGridLayout" name="gridLayout\_4">

<item row="1" column="1">

<widget class="QLineEdit" name="data\_file">

<property name="text">

<string>LR2\_data\data.csv</string>

</property>

</widget>

</item>

<item row="1" column="0">

<widget class="QLabel" name="label\_19">

<property name="text">

<string>Data file:</string>

</property>

</widget>

</item>

<item row="2" column="1">

<layout class="QGridLayout" name="gridLayout\_5">

<item row="0" column="1">

<widget class="QPushButton" name="btn\_save\_data">

<property name="enabled">

<bool>false</bool>

</property>

<property name="text">

<string>Save data</string>

</property>

</widget>

</item>

<item row="0" column="2">

<widget class="QPushButton" name="btn\_load\_data">

<property name="text">

<string>Load data</string>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</item>

<item>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<widget class="QPushButton" name="btn\_show\_3d">

<property name="text">

<string>Show 3D data</string>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</item>

</layout>

</item>

<item>

<layout class="QVBoxLayout" name="verticalLayout\_5">

<item>

<widget class="PlotWidget" name="graphWidget" native="true">

<property name="minimumSize">

<size>

<width>640</width>

<height>480</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>640</width>

<height>480</height>

</size>

</property>

</widget>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_5" stretch="0,0,0,0">

<property name="spacing">

<number>6</number>

</property>

<property name="leftMargin">

<number>0</number>

</property>

<property name="topMargin">

<number>0</number>

</property>

<property name="rightMargin">

<number>0</number>

</property>

<property name="bottomMargin">

<number>0</number>

</property>

<item>

<widget class="QCheckBox" name="roll\_enabled">

<property name="text">

<string>Roll rotation</string>

</property>

<property name="checked">

<bool>false</bool>

</property>

</widget>

</item>

<item>

<widget class="Line" name="line">

<property name="orientation">

<enum>Qt::Vertical</enum>

</property>

</widget>

</item>

<item>

<widget class="QLabel" name="label\_5">

<property name="text">

<string>Brush size:</string>

</property>

</widget>

</item>

<item>

<widget class="QSlider" name="brush\_size">

<property name="minimum">

<number>1</number>

</property>

<property name="maximum">

<number>20</number>

</property>

<property name="value">

<number>5</number>

</property>

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>905</width>

<height>21</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<customwidgets>

<customwidget>

<class>PlotWidget</class>

<extends>QWidget</extends>

<header>pyqtgraph</header>

<container>1</container>

</customwidget>

</customwidgets>

<resources/>

<connections/>

</ui>