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 №3

«Аппаратная поддержка отображения пространственных данных»

Работа выполнена (ФИО):

Ф И О

Научный руководитель

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

Ф И О

**Цель работы**

Подготовить приложение на языке C# для визуализации данных с использованием библиотеки OpenGL (через обертку SharpGL)**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].

• Выполнить статистический анализ набора точек, выполнив построение частотной диаграммы (гистограммы) 10х10 ячеек в координатной плоскости XY.

• Выполнить статистический анализ набора точек, выполнив расчет плотности заполнения вокселей (voxel) как пространственной матрицы 10х10х10 вокселей (пространственных ячеек). • Разработать систему отображения данных в виде облака точек средствами OpenGL (SharpGL) с указанием степени прореживания (отображать с шагом по номеру) и функцией анимации со сдвигом по номеру отображения.

• Разработать систему отображения данных в виде пространства вокселей, значение плотности заполнения каждого из которых отображать размером кубика, помещенного в центр вокселя.

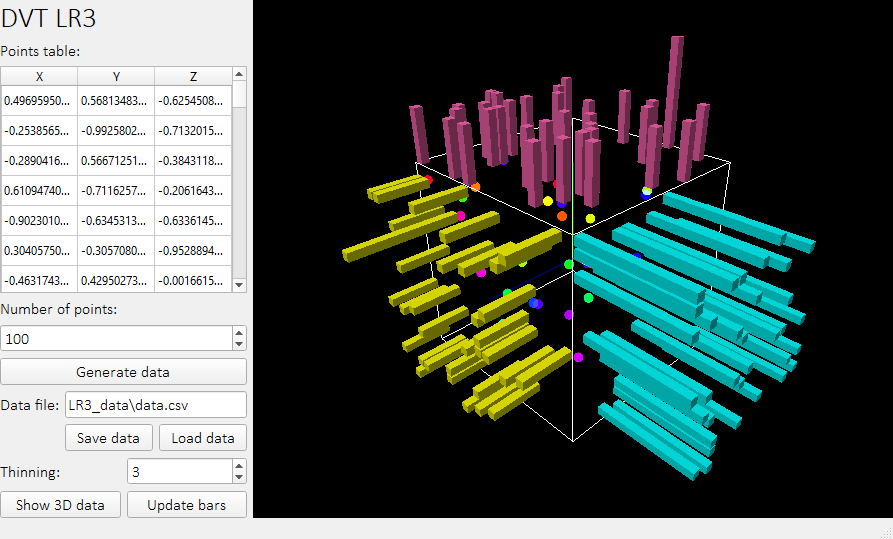
• Разработать систему отображения гистограммы распределения точек в виде поверхности, где координаты вычисляются следующим образом: координаты X и Y соответствуют координатам ячеек (центрам) гистограммы, а координата Z вычисляется как доля частоты попадания точек в данную ячейку по отношению к самому большому значению среди всех ячеек.

• Реализовать систему поворота базовой системы координат при отображении данных с помощью матрицы трансляции-поворота, управляемого позиционным манипулятором (мышью).

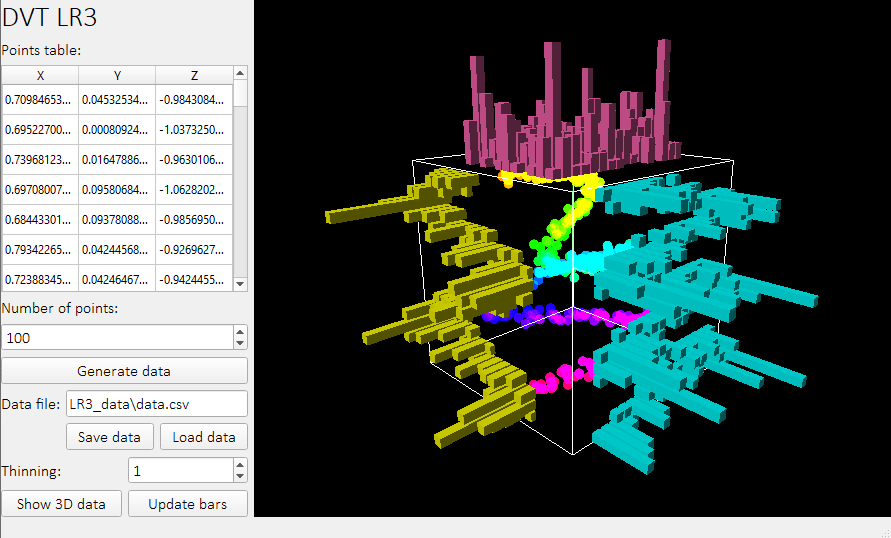
• Реализовать отображение системы координат и ребер описывающего куба (стороны: -1 и 1 по каждой координате).

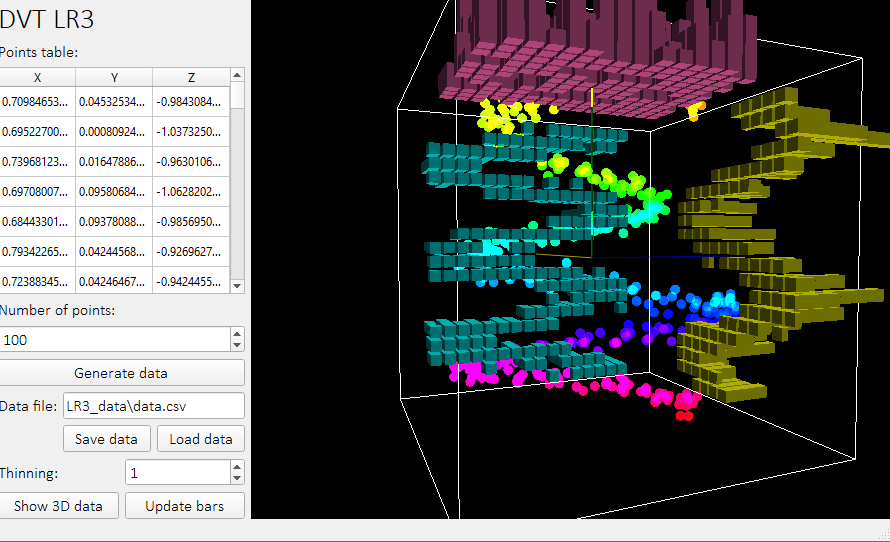
**Ход работы**

Для генерации данных на форме имеется возможность выбрать количество точек. После нажатия кнопки Generate, будет сгенерирован массив случайных точек требуемого размера. Для удобства, на форме также имеется таблица, в которой отображаются сгенерированные или считанные их файла точки.



Данные можно сохранить в файл или загрузить из него. Для этого предусмотрены соответствующие кнопки Save data / Load data. При нажатии кнопки Show 3D data, точки будут отрисованы в OpenGL с учётом прореживания, которое задаётся через отдельное поле «Thinning»





Для отрисовки диаграмм плотности распределения, на форме предусмотрена кнопка «Update bars». При нажатии на неё вызывается функция

count\_blocks(self, data\_x, data\_y, orientation)

вычисляющая плотность точек по каждой из плоскостей. Выход функции нормализуется, чтобы размах (высота) гистограмм была в диапазоне 0-1.

После нормализации, сгенерированные массивы блоков передаются объектам GLBarGraphItem для отрисовки гистограмм на соответствующих плоскостях.

**Вывод**

В ходе данной работы было создано приложение приложение для визуализации данных, а именно, облака точек, с использованием библиотеки OpenGL

**Исходный код**

*"""  
This is free and unencumbered software released into the public domain.  
  
Anyone is free to copy, modify, publish, use, compile, sell, or  
distribute this software, either in source code form or as a compiled  
binary, for any purpose, commercial or non-commercial, and by any  
means.  
  
In jurisdictions that recognize copyright laws, the author or authors  
of this software dedicate any and all copyright interest in the  
software to the public domain. We make this dedication for the benefit  
of the public at large and to the detriment of our heirs and  
successors. We intend this dedication to be an overt act of  
relinquishment in perpetuity of all present and future rights to this  
software under copyright law.  
  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR  
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR  
OTHER DEALINGS IN THE SOFTWARE.  
  
For more information, please refer to <https://unlicense.org>  
"""*import os  
import sys  
  
import matplotlib.pyplot as plt  
import numpy as np  
import pyqtgraph.opengl as gl  
from PyQt5 import uic, QtWidgets  
from PyQt5.QtWidgets import QApplication, QMainWindow, QTableWidgetItem  
  
  
class Window(QMainWindow):  
 def \_\_init\_\_(self):  
 super(Window, self).\_\_init\_\_()  
 *# Load GUI file* uic.loadUi(**'LR3.ui'**, self)  
  
 *# Settings* self.bars\_width = 0.08  
 self.bar\_z\_color = np.array(np.array([1., 0.4, 0.7, 1.]))  
 self.bar\_x\_color = np.array(np.array([1., 1., 0., 1.]))  
 self.bar\_y\_color = np.array(np.array([0., 1., 1., 1.]))  
  
 *# System variables* self.points = []  
 self.actual\_points = []  
 self.points\_surface = gl.GLScatterPlotItem(pos=np.array([[0, 0, 0]]))  
 self.bar\_graph\_x = gl.GLBarGraphItem(pos=np.array([[0, 0, 0]]), size=np.array([0, 0, 0]))  
 self.bar\_graph\_y = gl.GLBarGraphItem(pos=np.array([[0, 0, 0]]), size=np.array([0, 0, 0]))  
 self.bar\_graph\_z = gl.GLBarGraphItem(pos=np.array([[0, 0, 0]]), size=np.array([0, 0, 0]))  
  
 *# Connect GUI controls* self.btn\_generate\_data.clicked.connect(self.generate\_data)  
 self.btn\_save\_data.clicked.connect(self.save\_data)  
 self.btn\_load\_data.clicked.connect(self.load\_data)  
 self.btn\_show\_3d.clicked.connect(self.draw\_points)  
 self.btn\_update\_bars.clicked.connect(self.update\_bars)  
  
 *# Initialize table* self.init\_table()  
  
 *# Initialize OpenGL widget* self.init\_opengl()  
  
 *# Show GUI* self.show()  
  
 def init\_table(self):  
 *"""  
 Initializes table of points  
 :return:  
 """* self.points\_table.setColumnCount(3)  
 self.points\_table.verticalHeader().setVisible(False)  
 self.points\_table.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
 self.points\_table.setHorizontalHeaderItem(0, QtWidgets.QTableWidgetItem(**'X'**))  
 self.points\_table.setHorizontalHeaderItem(1, QtWidgets.QTableWidgetItem(**'Y'**))  
 self.points\_table.setHorizontalHeaderItem(2, QtWidgets.QTableWidgetItem(**'Z'**))  
 header = self.points\_table.horizontalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 header.setSectionResizeMode(1, QtWidgets.QHeaderView.Stretch)  
 header.setSectionResizeMode(2, QtWidgets.QHeaderView.Stretch)  
  
 def init\_opengl(self):  
 *"""  
 Initializes OpenGL Widget  
 :return:  
 """  
  
 # Cube bottom square* cube\_bottom\_square = gl.GLLinePlotItem(  
 pos=np.array([[-1, -1, -1], [1, -1, -1], [1, 1, -1], [-1, 1, -1], [-1, -1, -1]]),  
 color=[1, 1, 1, 1])  
 self.openGLWidget.addItem(cube\_bottom\_square)  
  
 *# Cube top square* cube\_top\_square = gl.GLLinePlotItem(  
 pos=np.array([[-1, -1, 1], [1, -1, 1], [1, 1, 1], [-1, 1, 1], [-1, -1, 1]]),  
 color=[1, 1, 1, 1])  
 self.openGLWidget.addItem(cube\_top\_square)  
  
 *# Cube sides* cube\_line\_bl = gl.GLLinePlotItem(  
 pos=np.array([[-1, -1, -1], [-1, -1, 1]]),  
 color=[1, 1, 1, 1])  
 self.openGLWidget.addItem(cube\_line\_bl)  
 cube\_line\_br = gl.GLLinePlotItem(  
 pos=np.array([[-1, 1, -1], [-1, 1, 1]]),  
 color=[1, 1, 1, 1])  
 self.openGLWidget.addItem(cube\_line\_br)  
 cube\_line\_tr = gl.GLLinePlotItem(  
 pos=np.array([[1, 1, -1], [1, 1, 1]]),  
 color=[1, 1, 1, 1])  
 self.openGLWidget.addItem(cube\_line\_tr)  
 cube\_line\_tl = gl.GLLinePlotItem(  
 pos=np.array([[1, -1, -1], [1, -1, 1]]),  
 color=[1, 1, 1, 1])  
 self.openGLWidget.addItem(cube\_line\_tl)  
  
 *# Add data elements* self.openGLWidget.addItem(gl.GLAxisItem())  
 self.openGLWidget.addItem(self.points\_surface)  
 self.openGLWidget.addItem(self.bar\_graph\_x)  
 self.openGLWidget.addItem(self.bar\_graph\_y)  
 self.openGLWidget.addItem(self.bar\_graph\_z)  
  
 def generate\_data(self):  
 *"""  
 Generates points cloud  
 :return:  
 """* print(**'Generating data...'**)  
 points\_num = self.points\_num.value()  
  
 self.points = []  
 for \_ in range(points\_num):  
 self.points.append(np.random.uniform(-1, 1, 3))  
  
 for i in range(points\_num):  
 pass  
 self.points = np.array(self.points)  
  
 self.show\_on\_table()  
 self.btn\_save\_data.setEnabled(True)  
 print(**'Data generated.'**)  
  
 def show\_on\_table(self):  
 *"""  
 Shows points in table  
 :return:  
 """* self.points\_table.setRowCount(0)  
 for point in self.points:  
 row\_position = self.points\_table.rowCount()  
 self.points\_table.insertRow(row\_position)  
 self.points\_table.setItem(row\_position, 0, QTableWidgetItem(str(point[0])))  
 self.points\_table.setItem(row\_position, 1, QTableWidgetItem(str(point[1])))  
 self.points\_table.setItem(row\_position, 2, QTableWidgetItem(str(point[2])))  
 self.btn\_save\_data.setEnabled(True)  
  
 def update\_bars(self):  
 *"""  
 Updates and draws 3D - bar charts  
 """  
 # Z (x:y)* bar\_pos, bar\_size = self.count\_blocks([item[0] for item in self.actual\_points],  
 [item[1] for item in self.actual\_points], 2)  
 self.openGLWidget.removeItem(self.bar\_graph\_z)  
 self.bar\_graph\_z = gl.GLBarGraphItem(pos=np.array(bar\_pos), size=np.array(bar\_size))  
 self.bar\_graph\_z.setColor(self.bar\_z\_color)  
 self.openGLWidget.addItem(self.bar\_graph\_z)  
  
 *# Y (x:z)* bar\_pos, bar\_size = self.count\_blocks([item[0] for item in self.actual\_points],  
 [item[2] for item in self.actual\_points], 1)  
 self.openGLWidget.removeItem(self.bar\_graph\_y)  
 self.bar\_graph\_y = gl.GLBarGraphItem(pos=np.array(bar\_pos), size=np.array(bar\_size))  
 self.bar\_graph\_y.setColor(self.bar\_y\_color)  
 self.openGLWidget.addItem(self.bar\_graph\_y)  
  
 *# X (y:z)* bar\_pos, bar\_size = self.count\_blocks([item[1] for item in self.actual\_points],  
 [item[2] for item in self.actual\_points], 0)  
 self.openGLWidget.removeItem(self.bar\_graph\_x)  
 self.bar\_graph\_x = gl.GLBarGraphItem(pos=np.array(bar\_pos), size=np.array(bar\_size))  
 self.bar\_graph\_x.setColor(self.bar\_x\_color)  
 self.openGLWidget.addItem(self.bar\_graph\_x)  
  
 def count\_blocks(self, data\_x, data\_y, orientation):  
 *"""  
 Counts how many points in each block and returns bars data  
 """* max\_block\_value = 0  
 bar\_pos = []  
 bar\_size = []  
 for x in range(-10, 10):  
 for y in range(-10, 10):  
 points\_in\_block = 0  
 for i in range(len(data\_x)):  
 if x / 10 <= data\_x[i] < (x / 10 + 0.1) \  
 and y / 10 <= data\_y[i] < (y / 10 + 0.1):  
 points\_in\_block += 1  
 if points\_in\_block > max\_block\_value:  
 max\_block\_value = points\_in\_block  
  
 if points\_in\_block > 0:  
 if orientation == 0:  
 pos = [1, x / 10 + ((0.1 - self.bars\_width) / 2), y / 10 + ((0.1 - self.bars\_width) / 2)]  
 elif orientation == 1:  
 pos = [x / 10 + ((0.1 - self.bars\_width) / 2), 1, y / 10 + ((0.1 - self.bars\_width) / 2)]  
 else:  
 pos = [x / 10 + ((0.1 - self.bars\_width) / 2), y / 10 + ((0.1 - self.bars\_width) / 2), 1]  
 size = [self.bars\_width] \* 3  
 size[orientation] = points\_in\_block  
 bar\_pos.append(pos)  
 bar\_size.append(size)  
  
 bar\_size\_normalized = []  
 for bar in bar\_size:  
 bar[orientation] /= max\_block\_value  
 bar\_size\_normalized.append(bar)  
 return bar\_pos, bar\_size\_normalized  
  
 def draw\_points(self):  
 *"""  
 Draws 3D points with OpenGL  
 :return:  
 """* thinning = self.thinning.value()  
 if self.points is not None and len(self.points) > 0:  
 *# 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)  
 rgba\_img = cmap(1.0 - (z - min\_z) / (max\_z - min\_z))  
  
 *# Draw points using defined thinning* self.actual\_points = []  
 for i in range(len(self.points)):  
 if i % thinning == 0:  
 self.actual\_points.append(self.points[i])  
 self.points\_surface.setData(pos=np.array(self.actual\_points), color=rgba\_img)  
  
 def save\_data(self):  
 *"""  
 Saves points to CSV file  
 :return:  
 """* print(**'Saving data...'**)  
 np.savetxt(self.data\_file.text(), self.points, delimiter=**' '**)  
 print(**'File'**, self.data\_file.text(), **'saved.'**)  
  
 def load\_data(self):  
 *"""  
 Loads points to CSV file  
 :return:  
 """* if os.path.exists(self.data\_file.text()):  
 print(**'Loading data...'**)  
 self.points = np.loadtxt(self.data\_file.text(), delimiter=**' '**)  
 self.show\_on\_table()  
 print(**'File'**, self.data\_file.text(), **'loaded.'**)  
 else:  
 print(**'File'**, self.data\_file.text(), **'doesn**\'**t exist!'**)  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QApplication(sys.argv)  
 app.setStyle(**'fusion'**)  
 win = Window()  
 sys.exit(app.exec\_())

*<?*xml version="1.0" encoding="UTF-8"*?>  
<!--  
This is free and unencumbered software released into the public domain.  
  
Anyone is free to copy, modify, publish, use, compile, sell, or  
distribute this software, either in source code form or as a compiled  
binary, for any purpose, commercial or non-commercial, and by any  
means.  
  
In jurisdictions that recognize copyright laws, the author or authors  
of this software dedicate any and all copyright interest in the  
software to the public domain. We make this dedication for the benefit  
of the public at large and to the detriment of our heirs and  
successors. We intend this dedication to be an overt act of  
relinquishment in perpetuity of all present and future rights to this  
software under copyright law.  
  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR  
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR  
OTHER DEALINGS IN THE SOFTWARE.  
  
For more information, please refer to <https://unlicense.org>  
 -->*<ui version="4.0">  
 <class>MainWindow</class>  
 <widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>893</width>  
 <height>541</height>  
 </rect>  
 </property>  
 <property name="font">  
 <font>  
 <family>Calibri Light</family>  
 <pointsize>12</pointsize>  
 </font>  
 </property>  
 <property name="windowTitle">  
 <string>DVT LR3</string>  
 </property>  
 <widget class="QWidget" name="centralwidget">  
 <layout class="QGridLayout" name="gridLayout">  
 <item row="0" column="0">  
 <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 LR3</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>  
 <widget class="QLabel" name="label\_2">  
 <property name="text">  
 <string>Number of points:</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QSpinBox" name="points\_num">  
 <property name="minimum">  
 <number>1</number>  
 </property>  
 <property name="maximum">  
 <number>99999</number>  
 </property>  
 <property name="value">  
 <number>100</number>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <layout class="QVBoxLayout" name="verticalLayout\_2">  
 <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>LR3\_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="QGridLayout" name="gridLayout\_2">  
 <item row="0" column="0">  
 <widget class="QLabel" name="label\_3">  
 <property name="text">  
 <string>Thinning:</string>  
 </property>  
 </widget>  
 </item>  
 <item row="0" column="1">  
 <widget class="QSpinBox" name="thinning">  
 <property name="value">  
 <number>3</number>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="0">  
 <widget class="QPushButton" name="btn\_show\_3d">  
 <property name="text">  
 <string>Show 3D data</string>  
 </property>  
 </widget>  
 </item>  
 <item row="1" column="1">  
 <widget class="QPushButton" name="btn\_update\_bars">  
 <property name="text">  
 <string>Update bars</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </item>  
 <item row="0" column="1">  
 <layout class="QVBoxLayout" name="verticalLayout\_4">  
 <item>  
 <widget class="GLViewWidget" name="openGLWidget">  
 <property name="minimumSize">  
 <size>  
 <width>640</width>  
 <height>480</height>  
 </size>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </widget>  
 <widget class="QMenuBar" name="menubar">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>893</width>  
 <height>21</height>  
 </rect>  
 </property>  
 </widget>  
 <widget class="QStatusBar" name="statusbar"/>  
 </widget>  
 <customwidgets>  
 <customwidget>  
 <class>GLViewWidget</class>  
 <extends>QOpenGLWidget</extends>  
 <header>pyqtgraph.opengl</header>  
 </customwidget>  
 </customwidgets>  
 <resources/>  
 <connections/>  
</ui>