Neutron Science Visualization Software

1. September 2011

69. Koordinierungstreffen | J. Heinen
Contents

✓ Visualization Software Requirements in Neutron Science
✓ Available Software
✓ Review
✓ Possible Approaches
✓ Pros and Cons
✓ GR Basics
✓ Real-life-examples
✓ Conclusions
✓ Future plans
# Visualization Software Requirements in Neutron Science

<table>
<thead>
<tr>
<th>User Aspects</th>
<th>Software Engineering Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of use</td>
<td>Easy to install</td>
</tr>
<tr>
<td>Fully-customizable</td>
<td>High quality graphics output in various formats</td>
</tr>
<tr>
<td>Produce fast and reliable results</td>
<td>Standard APIs for the creation of 2D- or 3D-graphs</td>
</tr>
<tr>
<td>Publication-quality printing and capturing capabilities</td>
<td>Embeddable in other applications, especially GUIs and Web Apps</td>
</tr>
<tr>
<td>Native look and feel</td>
<td>Scripting capabilities</td>
</tr>
<tr>
<td>Usability over slow network connections</td>
<td>Long-time support</td>
</tr>
<tr>
<td>Well-documented</td>
<td>Open Source based</td>
</tr>
<tr>
<td></td>
<td>Cross platform interoperability</td>
</tr>
</tbody>
</table>
### Available Software

<table>
<thead>
<tr>
<th>Graphics Application Programming Interfaces</th>
<th>Integrated Visualization Systems</th>
<th>Graphical User Interface Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓Cairo Graphics</td>
<td>✓Gnuplot</td>
<td>✓FLTK</td>
</tr>
<tr>
<td>✓DISLIN</td>
<td>✓GLI</td>
<td>✓Gtk / Gtk+</td>
</tr>
<tr>
<td>✓GR / GKS</td>
<td>✓grpython</td>
<td>✓Motif</td>
</tr>
<tr>
<td>✓Matplotlib</td>
<td>✓IDL</td>
<td>✓Qt4 / Qt3</td>
</tr>
<tr>
<td>✓OpenGL</td>
<td>✓IGOR Pro</td>
<td>✓Tk</td>
</tr>
<tr>
<td>✓PGPlot</td>
<td>✓MantidPlot</td>
<td>✓Ultimate++</td>
</tr>
<tr>
<td>✓Qwt3D</td>
<td>✓Octave</td>
<td>✓wxWidgets</td>
</tr>
<tr>
<td>✓SDL</td>
<td>✓Origin</td>
<td>✓XForms</td>
</tr>
<tr>
<td>✓VTK</td>
<td>✓PlotScript</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓QtKWS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓QtiPlot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓XmGrace</td>
<td></td>
</tr>
</tbody>
</table>
Review

“Jülich-Munich Standard”
- Software environment: Qt / C++, Python
- Middleware: TACO
- Operating System: Linux

Lessons learned
- Qt3 to Qt4 migration (complex and time-consuming)
- Code-reuse (one of the main goals of OOP) as an end in itself
- TACO - TANGO migration certainly unavoidable
- Linux distributions divergent
Review (continued)

Software developers are maintaining three principles:
✓ Reusability (classes, objects, methods)
✓ Extensibility (inheritance)
✓ Simplicity (abstraction, encapsulation, polymorphism)

The above mentioned principles might have improved maintainability, but did they really help users?
✓ Linux distribution spins, flavors, …
✓ Desktop environments: GNOME, KDE, Unity, …
✓ Office software: OpenOffice, LibreOffice, KOffice, …
## Possible Approaches

<table>
<thead>
<tr>
<th>Graphics Application Programming Interfaces</th>
<th>Integrated Visualization Systems</th>
<th>Graphical User Interface Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Cairo Graphics</td>
<td>✓ Gnuplot</td>
<td>✓ FLTK</td>
</tr>
<tr>
<td>✓ DISLIN</td>
<td>✓ GLI</td>
<td>✓ Gtk / Gtk+</td>
</tr>
<tr>
<td>✓ GR / GKS</td>
<td>✓ grpython</td>
<td>✓ Qt4 / Qt3</td>
</tr>
<tr>
<td>✓ Matplotlib</td>
<td>✓ IDL</td>
<td>✓ Tk</td>
</tr>
<tr>
<td>✓ OpenGL</td>
<td>✓ IGOR Pro</td>
<td>✓ Ultimate++</td>
</tr>
<tr>
<td>✓ PGPlot</td>
<td>✓ MantidPlot</td>
<td>✓ wxWidgets</td>
</tr>
<tr>
<td>✓ Qwt3D</td>
<td>✓ Octave</td>
<td>✓ XForms</td>
</tr>
<tr>
<td>✓ SDL</td>
<td>✓ Origin</td>
<td></td>
</tr>
<tr>
<td>✓ VTK</td>
<td>✓ PlotScript</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ QtiKWS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ QtiPlot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ XmGrace</td>
<td></td>
</tr>
</tbody>
</table>
## Pros and Cons

<table>
<thead>
<tr>
<th></th>
<th>Pro</th>
<th>Contra</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PlotScript</strong></td>
<td>+ Instrument specific menus</td>
<td>– GTK/GTK+ (or wxWidgets) based GUI</td>
</tr>
<tr>
<td></td>
<td>+ Adapted to scientists’ needs</td>
<td>– Gnuplot graphics (separated)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– no API</td>
</tr>
<tr>
<td><strong>QtiKWS</strong></td>
<td>+ Uses existing QtiPlot framework</td>
<td>– Qt3 based GUI</td>
</tr>
<tr>
<td></td>
<td>+ Adapted to KWS instruments</td>
<td>– Qt/Qwt3D graphics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– no API</td>
</tr>
<tr>
<td><strong>GR</strong></td>
<td>+ Standardized graphics (GKS)</td>
<td>– Documentation out-dated</td>
</tr>
<tr>
<td></td>
<td>+ Embeddable into Qt4 based GUIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ High quality graphics output in various formats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Cross language (Fortran, C/C++, Python, …)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Cross-platform</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Conforms to the “Jülich–Munich Standard”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Web ready</td>
<td></td>
</tr>
</tbody>
</table>
GR from a developer point of view

Graphical Kernel System (GKS)

- Fortran
- C / C++
- Objective-C
- Python
- Psh
- ...

Additional drivers / plugins:
- CGM, GKSM, GIF, RF, UIL
- WMF, Xfig
- GS (BMP, JPEG, PNG, TIFF)
GR from user point of view

Physics Computer

Qt GUI

OpenGL enabled Host / iOS Device

GLGr

Socket communication

GKS Logical Device Drivers

GKS

GR

C

C / C++

Python

C / Objective-C

Qt event loop

Additional drivers / plugins:
- CGM, GKSM, GIF, RF, UIL
- WMF, Xfig
- GS (BMP, JPEG, PNG, TIFF)
% gkswebapp
% export GKS_WSTYPE=410
...

% glgr
% export GKS_WSTYPE=410
...

Browser Window

OpenGL Window
Real-life-examples: KWSlive ...

✓ Written in C++
✓ Requires Qt4, TACO and GR / GKS
✓ Native GUI completely rewritten using Qt SDK (Qt Creator, Qt Designer)
✓ Visualization based on GKS Qt logical device driver
  ✓ Platform independent graphics output (GrWidget)
  ✓ Smooth integration into Qt event mechanism
✓ Features:
  3D surface plots, contour plots, 2D line plots, radial averages, ROI selection, least-square fitting, sensitivity masks, TOF channel selection
KWSlive Main Window

Data

Source

Refresh rate

Display

Mode

Colormap

Contour

Uniform

Detector

Resolution

X: 1

Y: 1

TOF-Channel: 0

View

Tilt

Rotation

Flip

X

Y

Z

Control

Update

Refresh

Clear

Print

Quit

Select points

Linear fit

Calculate

From: -inf

To: inf

y = m * x + b

Delete point

Finish point list
#!/usr/bin/env python

import sys, string, os

from PyQt4 import *
from PyQt4 import QtCore, QtGui
import sip
import gr

def readData(file, data):
    stream = open(file, "r")
    num_items = -1
    for line in stream.readlines():
        if line == "":
            break
        if line[0] == "$":
            num_items = 0
        else:
            if num_items >= 0:
                line = line[:-1]
                for s in string.split(line):
                    data[num_items] = int(s)
                num_items = num_items + 1
    stream.close()

class GrWidget(QtGui.QWidget):
    def __init__(self, *args):
        QtGui.QWidget.__init__(self)
        self.setupUi(self)
        os.environ["GKS_WSTYPE"] = "381"
        os.environ["GKS_DOUBLE_BUF"] = "True"

        timer = QtCore.QTimer(self)
        self.connect(timer, QtCore.SIGNAL("timeout()"), self.draw)
        timer.start(1000)

    def setupUi(self, Form):
        Form.setWindowTitle("GrWidget")
        Form.resize(QtCore.QSize(600, 600).expandedTo( Form.minimumSizeHint()))

        def draw(self):
            x = y = range(128)
            z = range(128 * 128)
            readData('simple.dat', z)
            z_max = max(z)

            gr.setviewport(0.1, 0.95, 0.1, 0.95)
            gr.setwindow(0, 127, 0, 127)
            gr.setspace(0, z_max, 0, 90)
            gr.setlinewidth(0.75)
            gr.setcharheight(18 / 600.)
            gr.axes(5, 5, 1, 1, 4, 4, -0.0075)
            gr.surface(128, 128, x, y, z, 5)
            h = range(0, 1)
            gr.contour(128, 128, 0, x, y, h, z, -1)
            self.update()

        def paintEvent(self, ev):
            self.painter = QtGui.QPainter()
            self.painter.begin(self)
            os.environ["GKS_CONID"] = "%x%!%x" % (sip.unwrapinstance(self),
                                                     sip.unwrapinstance(self.painter))
            gr.updatews()
            self.painter.end()

        def quit(self):
            gr.emergencyclosegks()
            self.close()

    app = QtGui.QApplication(sys.argv)
    w = GrWidget()
    w.show()
    sys.exit(app.exec_())
Fortran

```fortran
real x(30), y(30), z(30,30), tbx(4), tby(4)
real r1, r2, z1, z2

do 1, i = 1, 30
  x(i) = -2.0 + (i - 1) * 0.5
  y(i) = -7.0 + (i - 1) * 0.5
1  continue

do 2, i = 1, 30
  do 3, j = 1, 30
    r1 = sqrt((x(i)-5)**2+y(j)**2)
    r2 = sqrt((x(i)+5)**2+y(j)**2)
    z1 = exp(cos(r1))
    z2 = exp(cos(r2))
    z(i,j) = (z1 + z2 - 0.9) * 25
 3  continue
2  continue

call gr_beginprint('face.pdf')
call gr_setcharheight(24.0/500)
call gr_settextalign(2, 1)
call gr_textext(0.5, 0.9, 'Surface Example')
call gr_inqtextext(0.5, 0.9, 'Surface Example', tbx, tby)
call gr_fillarea(4, tbx, tby)
call gr_setcharheight(14.0/500)
call gr_setviewport(0.1, 0.9, 0.1, 0.9)
call gr_setwindow(-3.0, 13.0, -8.0, 8.0)
call gr_settextfontprec(3, 0)
call gr_setspace(-20.0, 200.0, 45, 70)
call gr_axes3d(1.0, 0.0, 10.0, -3.0, -8.0, -20.0, 2, 2, 4, -0.01)
call gr_axes3d(0.0, 1.0, 0.0, 13.0, -8.0, -20.0, 2, 2, 4, -0.01)
call gr_titles3d('X-Axis', 'Y-Axis', 'Z-Axis')
call gr_surface(30, 30, x, y, z, 3)
call gr_surface(30, 30, x, y, z, 1)
call gr_updatews()
call gr_endprint()
read(*, *)
call gr_emergencyclosegks()
end
```

C

```c
#include <math.h>
#include "gr.h"

int main(void)
{
  float x[30], y[30], z[30][30], tbx[4], tby[4];
  float r1, r2, z1, z2;
  int i, j;

  for (i = 0; i < 30; i++) {
    x[i] = -2.0 + (i - 1) * 0.5;
    y[i] = -7.0 + (i - 1) * 0.5;
  }

  for (i = 0; i < 30; i++) {
    for (j = 0; j < 30; j++) {
      r1 = sqrt(pow(x[i]-5,2)+pow(y[j],2));
      r2 = sqrt(pow(x[i]+5,2)+pow(y[j],2));
      z1 = exp(cos(r1));
      z2 = exp(cos(r2));
      z[i][j] = (z1 + z2 - 0.9) * 25;
    }
  }

call gr_beginprint("face.pdf");
call gr_setcharheight(24.0/500);
call gr_settextalign(2, 1);
call gr_textext(0.5, 0.9, "Surface Example");
call gr_inqtextext(0.5, 0.9, "Surface Example", tbx, tby);
call gr_fillarea(4, tbx, tby);
call gr_setcharheight(14.0/500);
call gr_setviewport(0.1, 0.9, 0.1, 0.9);
call gr_setwindow(-3.0, 13.0, -8.0, 8.0);
call gr_settextfontprec(3, 0);
call gr_setspace(-20.0, 200.0, 45, 70);
call gr_axes3d(1.0, 0.0, 10.0, -3.0, -8.0, -20.0, 2, 2, 4, -0.01);
call gr_axes3d(0.0, 1.0, 0.0, 13.0, -8.0, -20.0, 2, 2, 4, -0.01);
call gr_titles3d("X-Axis", "Y-Axis", "Z-Axis");
call gr_surface(30, 30, x, y, (float*)z, 3);
call gr_surface(30, 30, x, y, (float*)z, 1);
call gr_updatews();
call gr_endprint();
sleep(5);
call gr_emergencyclosegks();
}```
… Python + NumPy + GR
Conclusions

GR is not an all-in-one solution, but it might help to make a lot of things easier:

- keeps things as simple as possible and reduces complexity
- increases maintainability and extensibility
- makes use of abstraction layers
- loosely couples system-specific components
- allows code to be reused in other applications
- minimizes use of external code

“Less is more”
Future plans

- Move GR code into a code management system (Subversion, Git, Trac)
- Setup a Wiki (documentation)
- Add more 3D functions (OpenGL)
- Add a GKS logical device driver for wxWidgets
- Migrate SIGHT (Simple Interactive Graphics Handling Tool) to Qt4
Thanks to …

Dück, Marcel (GKS WebApp)
Bachelor Thesis: “Entwicklung eines GKS-Gerätetreibers als Java-basierte Client/Server Webanwendung”

Felder, Christian (GLGr OpenGL ES Programming)
Master Thesis: “Implementierung einer Visualisierungsanwendung für Apples iOS”

Goblet, Marvin (GKSTerm)
Bachelor Thesis: “Entwicklung eines GKS Gerätetreibers auf der Basis des Mac OS X Core Graphics Frameworks”

Neßelrath, Robert (GLGr OpenGL Programming)
Diploma Thesis: “Plattformunabhängige Visualisierung von Neutronenspektren auf der Basis von OpenGL”

Schätzler, Liane (KWSlive Qt4 Migration)