Getting more out of Matplotlib with GR

July 20th – 26th, 2015

Bilbao | EuroPython 2015 | Josef Heinen | @josef_heinen

Visualization needs

- ✓ visualize and analyzing two- and three-dimensional data sets
- ✓ plot 2D data for real-time monitoring purposes (signal processing)
- ✓ visualize large data sets, probably with a dynamic component, preferably in real-time
- ✓ create publication-quality and web-ready graphics
- ✓ create animations or videos on the fly

Python visualization solutions

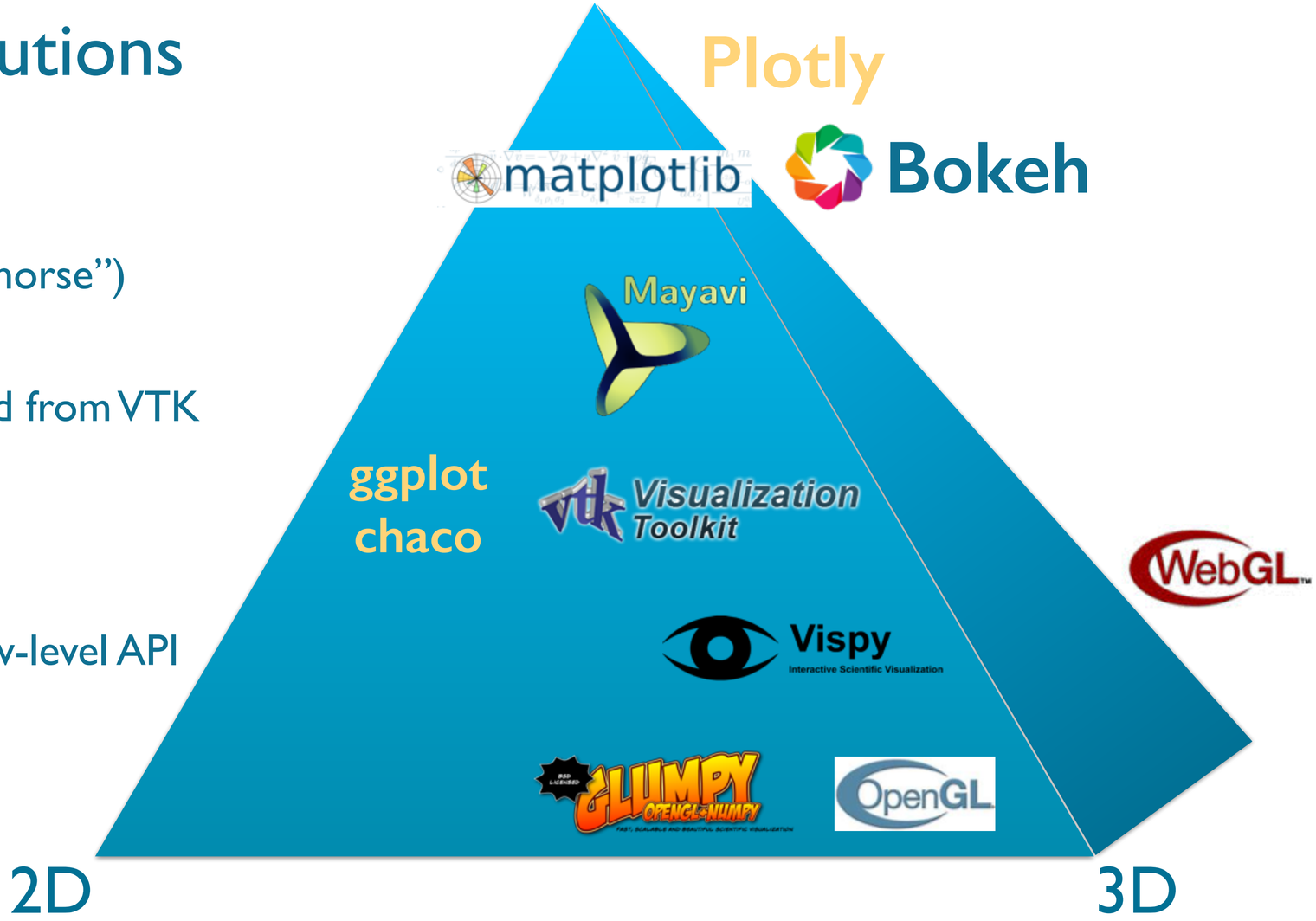
Matplotlib — de-facto standard ("workhorse")
➟ Browser solutions: Bokeh, plot.ly

Mayavi (mlab) — powerful, but overhead from VTK

ggplot, chaco — statistical, 2D graphics

VTK — versatile, but difficult to learn

Vispy, Glumpy, OpenGL — fast, but low-level API

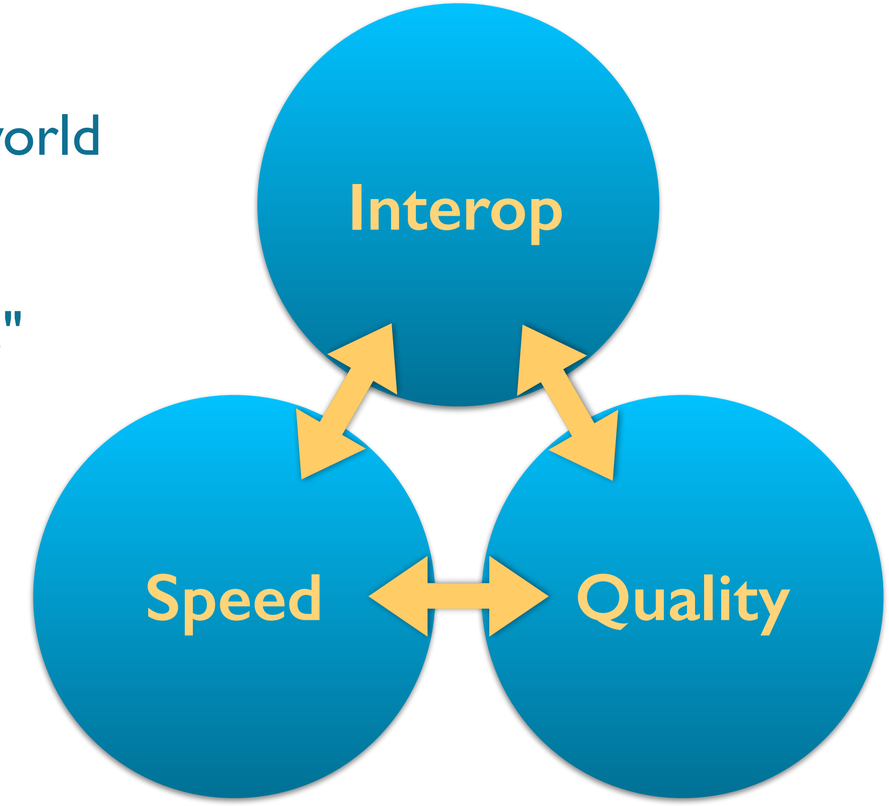


Problems so far — Crux of the matter

separated 2D and (hardware accelerated) 3D world

some graphics backends "only" produce "figures"
➟ no presentation of continuous data streams

speed up “only” by means of backend specific code ➟ poor performance on large data sets



Where to go from here?

How can we improve the performance?

- ✓ Several Python modules can be compiled into native code, making them much faster (Cython)
- ✓ Compiling hotspots on the fly (Numba, PyPy) can significantly speed up numerical code segments
- ✓ Use hardware acceleration, but …

… these approaches cannot easily be applied to visualization software! 

➟ **Could another backend speedup Matplotlib and improve interop ?**

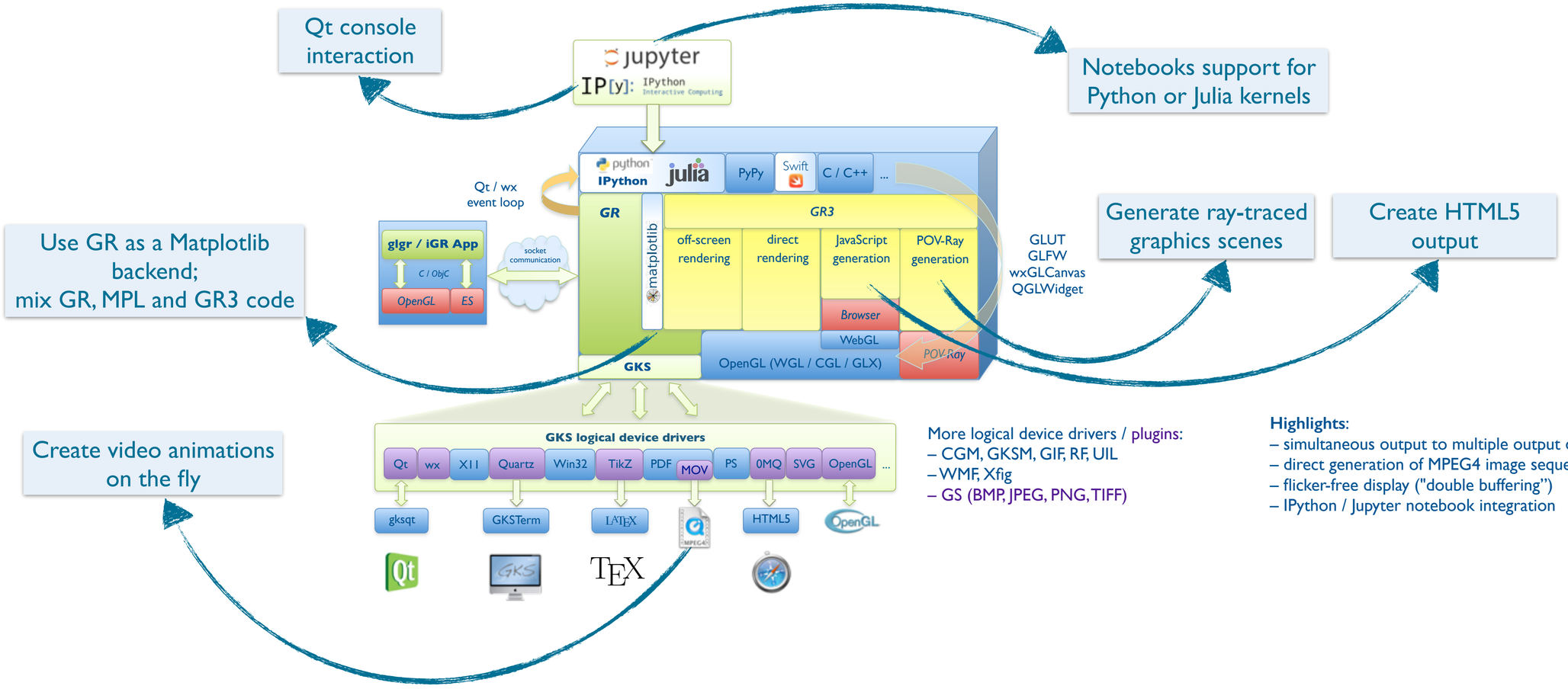
Use GR to achieve more graphics performance

- ✓ procedural graphics backend (completely written in C)
  - ➠ presentation of continuous data streams
- ✓ builtin support for 2D plotting and OpenGL (GR3)
  - ➠ coexistent 2D and 3D world
- ✓ interoperability with GUI toolkits and Web frameworks
  - ➠ good user interaction

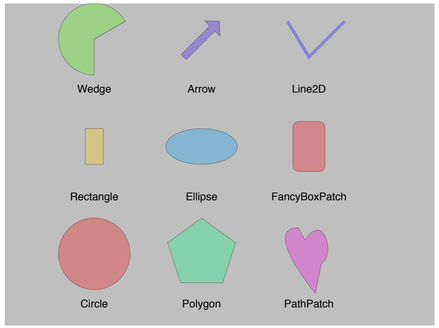
Use GR to extend Matplotlib's capabilities

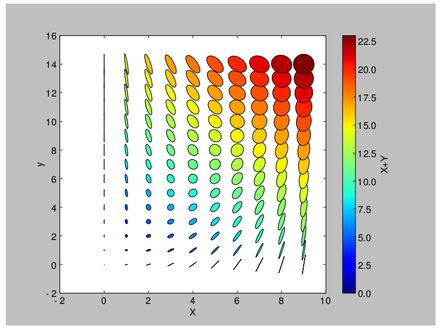
- ✓ combine the power of Matplotlib and GR
  ➟ next Matplotlib release will allow selecting the backend by setting the environment variable `MPLBACKEND`
- ✓ produce video contents on the fly by adding a single line of code
  ➟ no need to import an animation module or write extra code
- ✓ create plots containing both 2D and 3D graphics elements

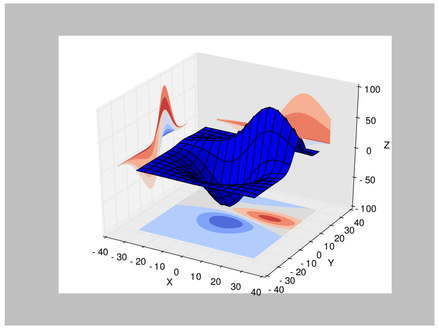
How it works: GR layer architecture

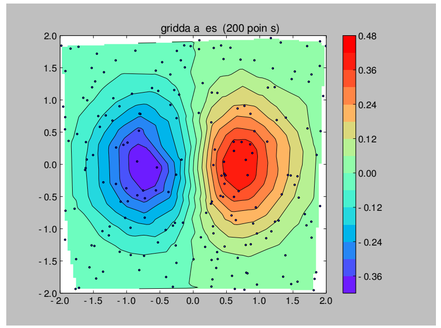


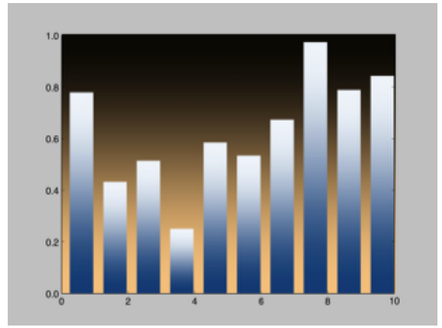
Matplotlib using the GR backend

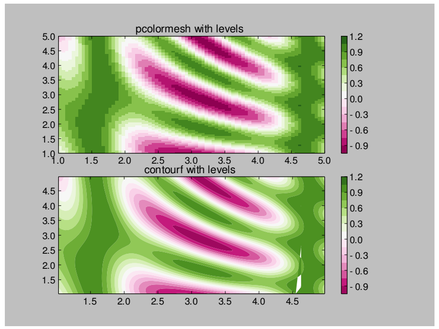


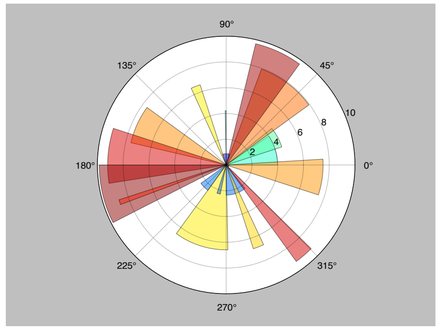


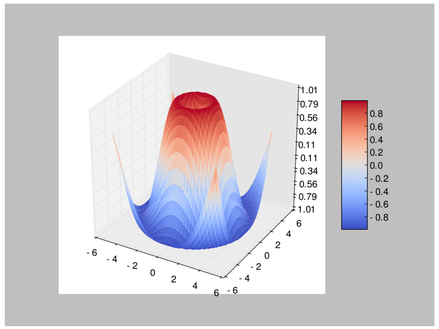


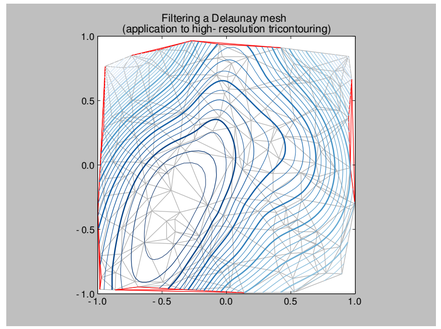


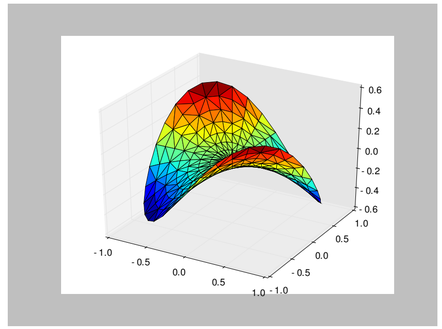




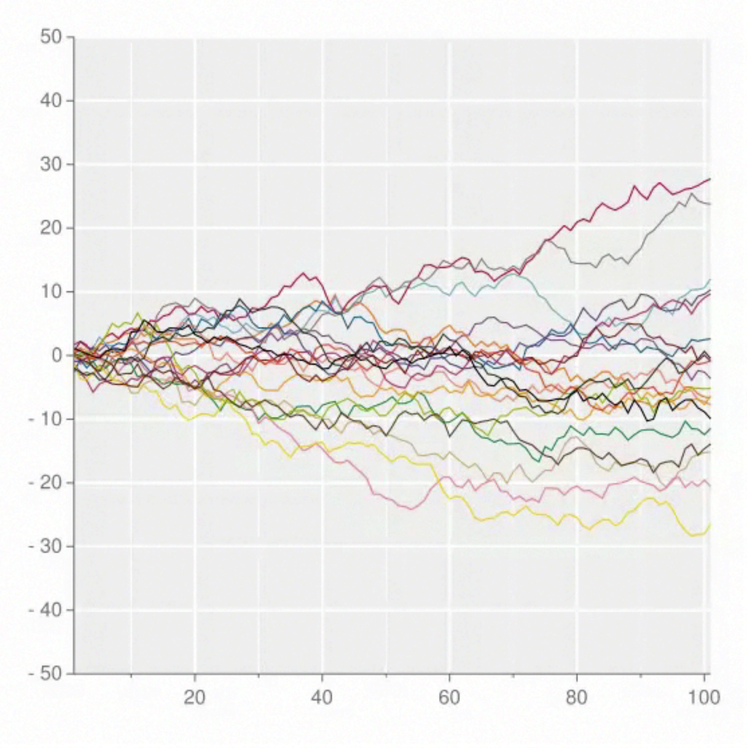


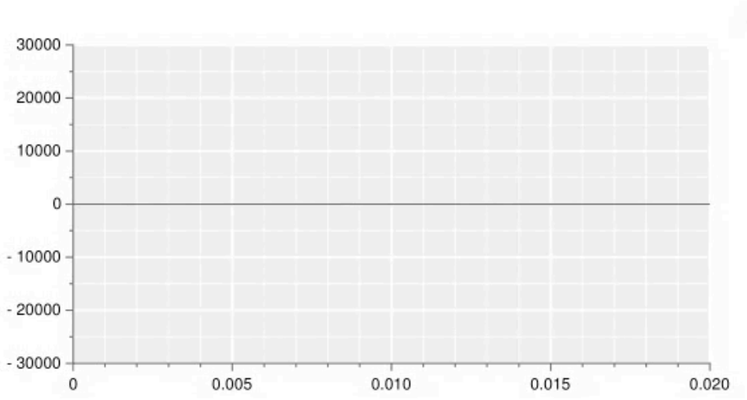


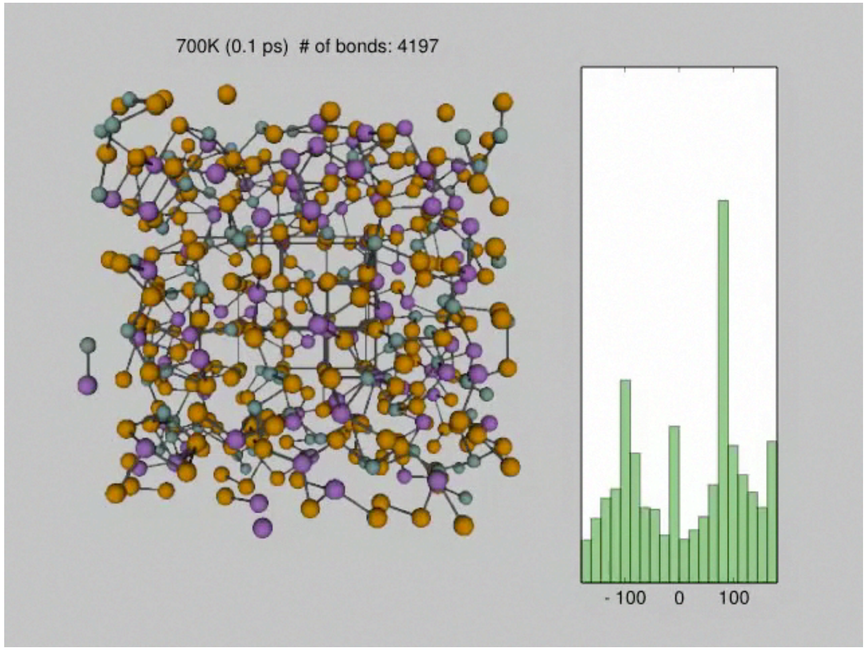




GR in action …

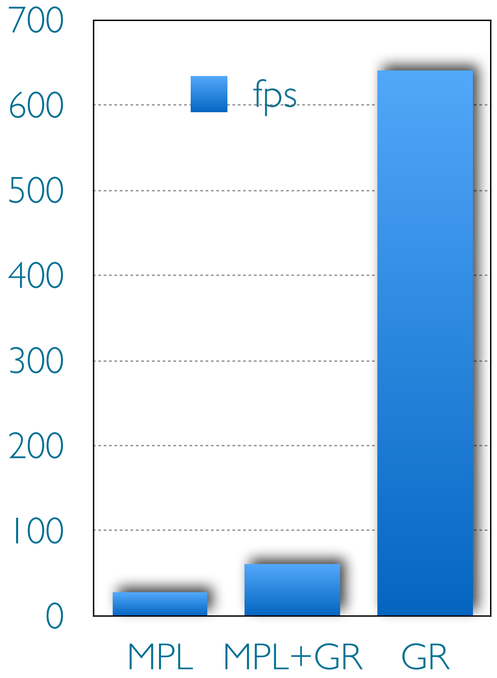

50
40
30
20
10
0
- 10
- 20
- 30
- 40
- 50
20
40
60
80
100

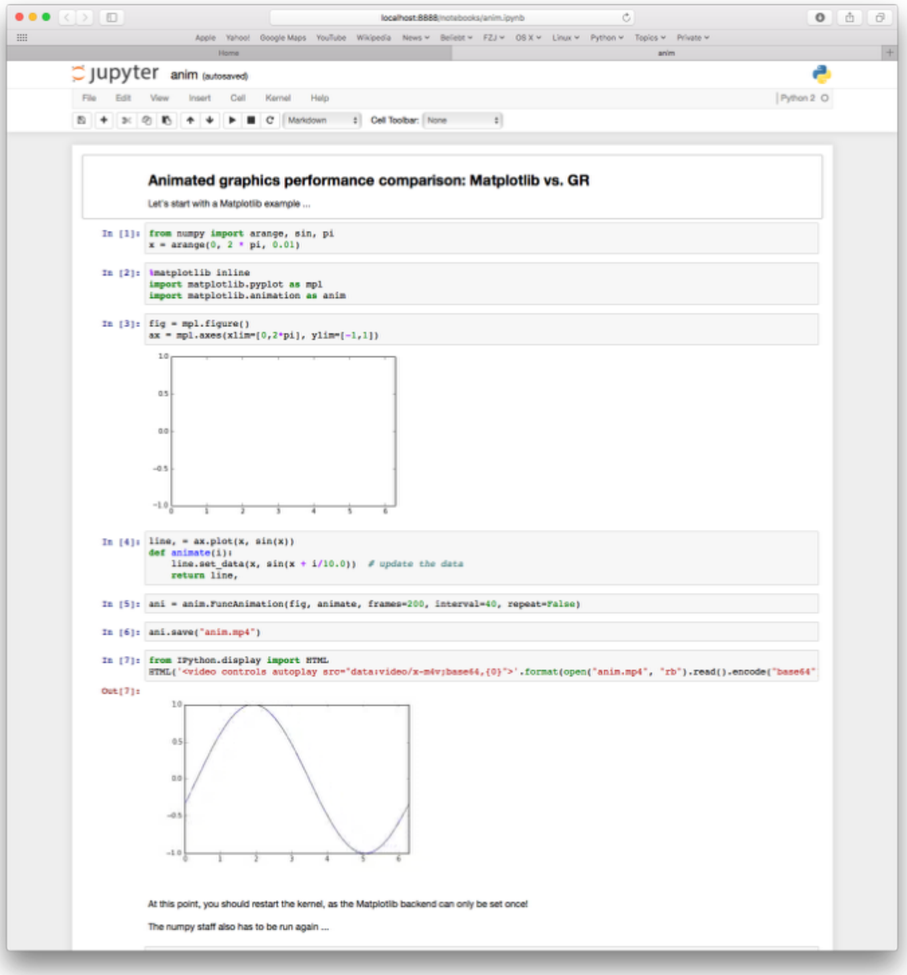

30000
20000
10000
0
- 10000
- 20000
- 30000
0
0.005
0.010
0.015
0.020

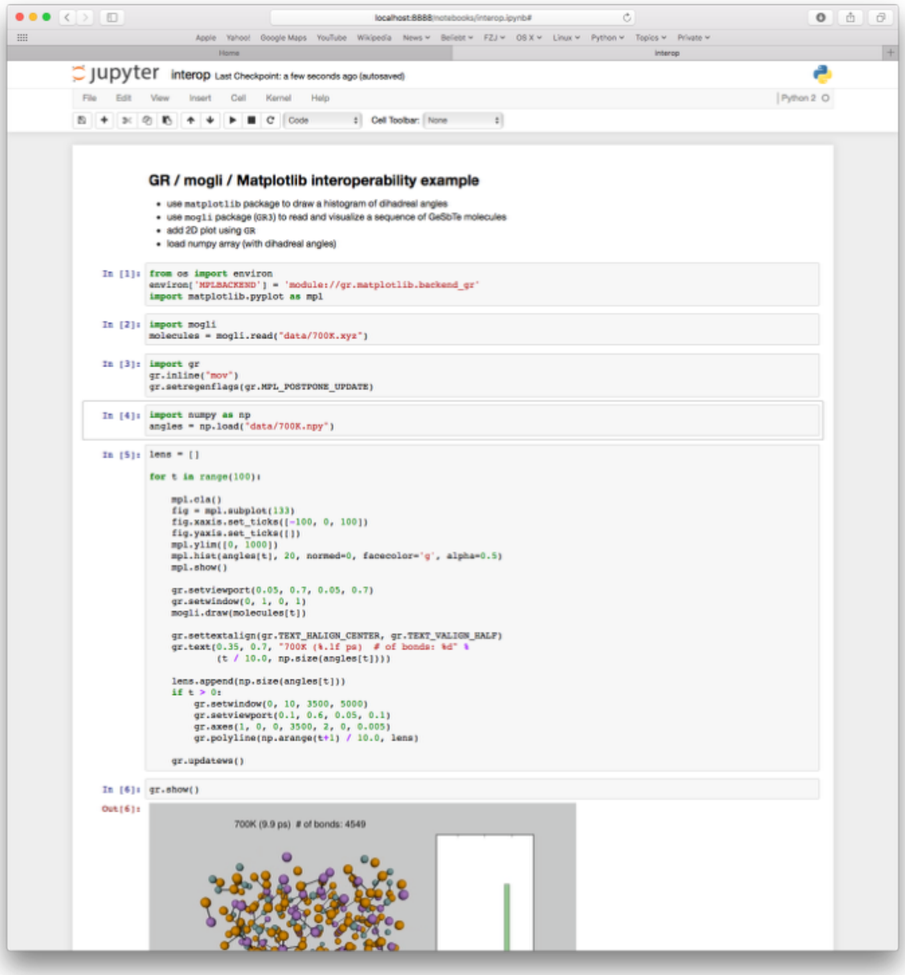

700K (0.1 ps) # of bonds: 4197
- 100
0
100

GR / Jupyter

Performance (anim.py)

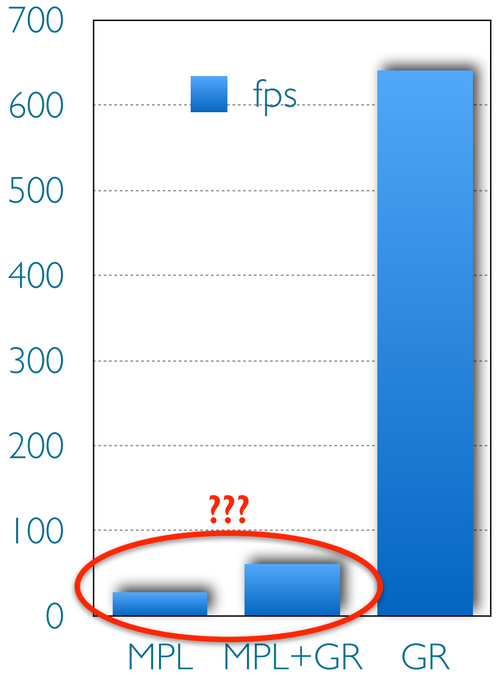






click images to view notebooks …

Performance analysis



MPL

ncalls	cumtime	filename:lineno(function)
398	**6.852**	**{method'draw'of'_macosx.FigureCanvas'objects}**
29378/397	6.771	artist.py:57(draw_wrapper)
397	6.769	figure.py:1004(draw)
397	6.574	_base.py:1989(draw)
794	5.894	axis.py:1106(draw)
5161	4.601	axis.py:232(draw)
199	3.616	pyplot.py:175(pause)
10719	3.609	lines.py:661(draw)
199	3.480	pyplot.py:551(draw)
7940	1.044	text.py:581(draw)

MPL + GR

ncalls	cumtime	filename:lineno(function)
199	4.412	pyplot.py:551(draw)
199	**4.410**	**backend_gr.py:227(draw)**
14726/199	4.237	artist.py:57(draw_wrapper)
199	4.236	figure.py:1004(draw)
199	4.138	_base.py:1989(draw)
398	3.770	axis.py:1106(draw)
2587	3.073	axis.py:232(draw)
5373	2.642	lines.py:661(draw)
5174	1.202	backend_bases.py:237(draw_markers)

GR

ncalls	cumtime	filename:lineno(function)
199	**3.263**	**__init__.py:1910(plot)**
199	**3.184**	**__init__.py:250(updatews)**

most time spent in backend wrapper

No room for further optimizations on the backend side

GR + GR3 + Matplotlib interop

```
from os import environ
environ['MPLBACKEND'] = 'module://gr.matplotlib.backend_gr'
import matplotlib.pyplot as mpl
```

```
import mogli
molecules = mogli.read("data/700K.xyz")
```

```
import gr
gr.inline("mov")
gr.setregenflags(gr.MPL_POSTPONE_UPDATE)
```

Important:
tells MPL backend **not** to update

```
import numpy as np
angles = np.load("data/700K.npy")
```

```
for t in range(100):
    mpl.cla()
    fig = mpl.subplot(133)
    fig.xaxis.set_ticks([-100, 0, 100])
    fig.yaxis.set_ticks([])
    mpl.ylim([0, 1000])
    mpl.hist(angles[t], 20, normed=0, facecolor='g', alpha=0.5)
    mpl.show()
```

Matplotlib

```
    gr.setviewport(0.05, 0.7, 0.05, 0.7)
    gr.setwindow(0, 1, 0, 1)
    mogli.draw(molecules[t])
```

GR3

```
    gr.settextalign(gr.TEXT_HALIGN_CENTER, gr.TEXT_VALIGN_HALF)
    gr.text(0.35, 0.7, "700K (%.1f ps)  # of bonds: %d" %
            (t / 10.0, np.size(angles[t])))

    lens.append(np.size(angles[t]))
    if t > 0:
        gr.setwindow(0, 10, 3500, 5000)
        gr.setviewport(0.1, 0.6, 0.05, 0.1)
        gr.axes(1, 0, 0, 3500, 2, 0, 0.005)
        gr.polyline(np.arange(t+1) / 10.0, lens)

    gr.updatews()
```

GR

MPEG4

```
import gr3
gr3.export("data/700K.html", 600, 600)
```

WebGL

Inline graphics

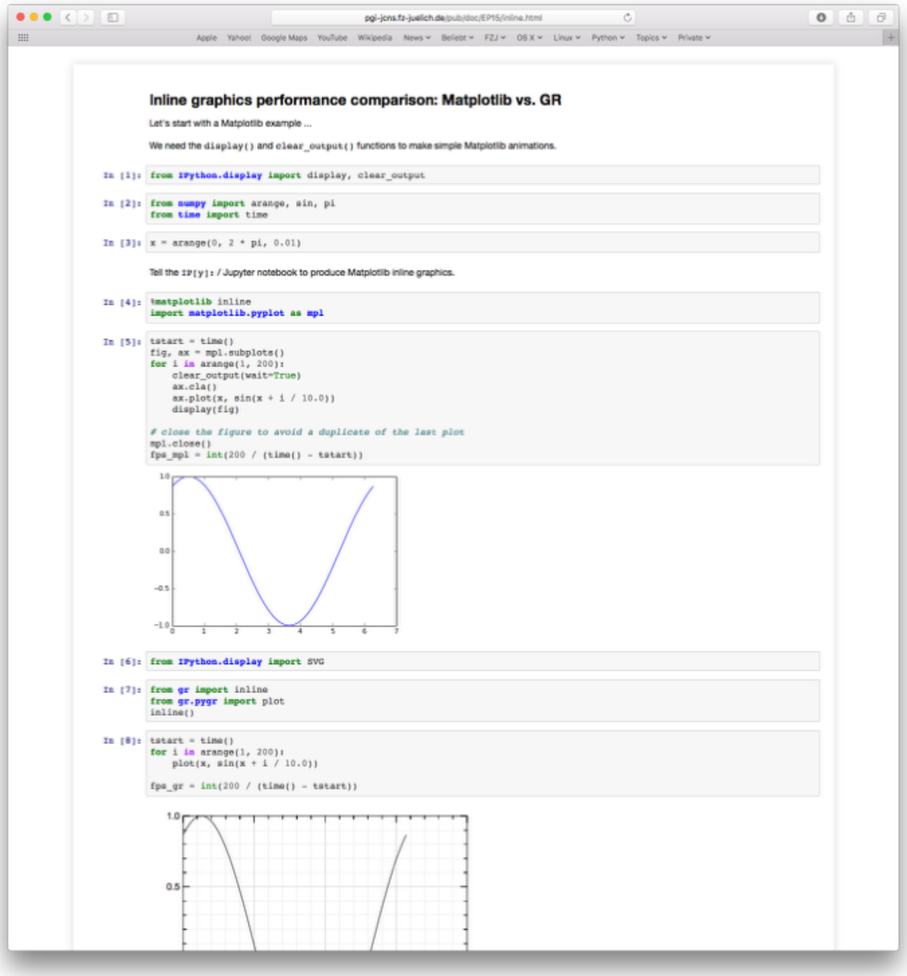
Matplotlib

```
%matplotlib inline
import matplotlib.pyplot as mpl

fig, ax = mpl.subplots()
for i in arange(1, 200):
    clear_output(wait=True)
    ax.cla()
    ax.plot(x, sin(x + i / 10.0))
    display(fig)

mpl.close()
```

GR

```
from gr import inline
from gr.pygr import plot
inline()

for i in arange(1, 200):
    plot(x, sin(x + i / 10.0))
```

~ 10 times faster



Demos

- ✓ Animated graphics performance comparison: Matplotlib vs. GR (`anim.ipynb`)
- ✓ GR / mogli / Matplotlib interoperability example (`interop.ipynb`)
- ✓ Inline graphics performance comparison: Matplotlib vs. GR (`inline.ipynb`)
- ✓ Simple spectral (`specgram.ipynb`)

Outlook (GR release v0.15.0)

GR + GKS can be transpiled to JS
(Emscripten: LLVM-to-JavaScript compiler)

➠ Use cases:

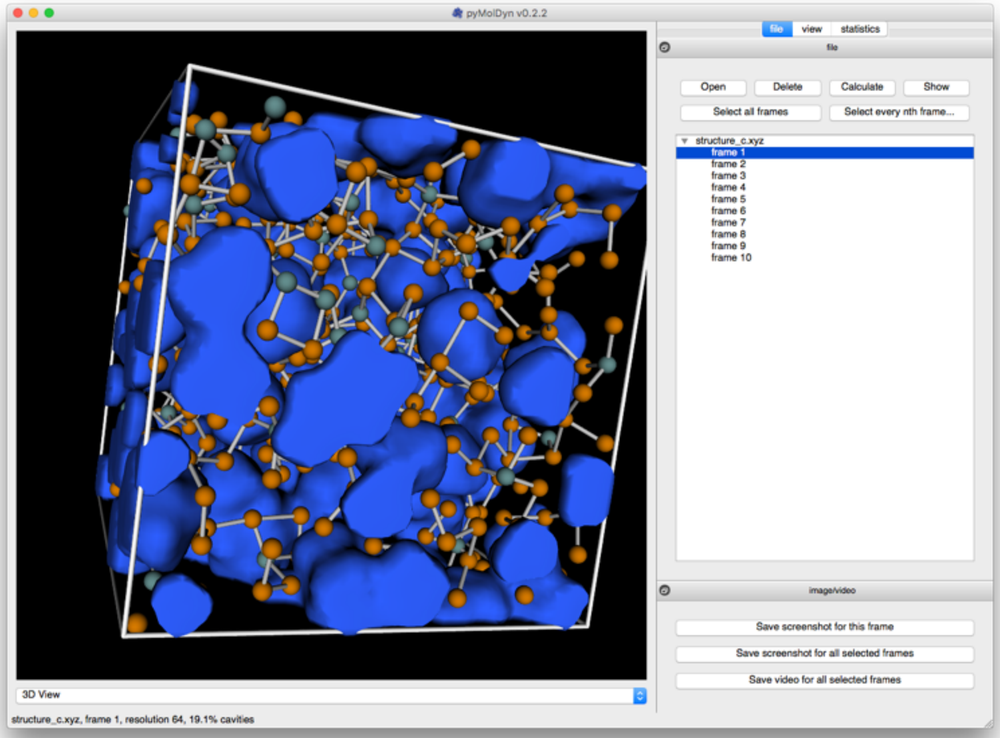
- ✓ embed JS code in IP[y]: or IJulia (Jypyter)
- ✓ parse GKS JavaScript logical device driver generated display list in browser

JavaScript

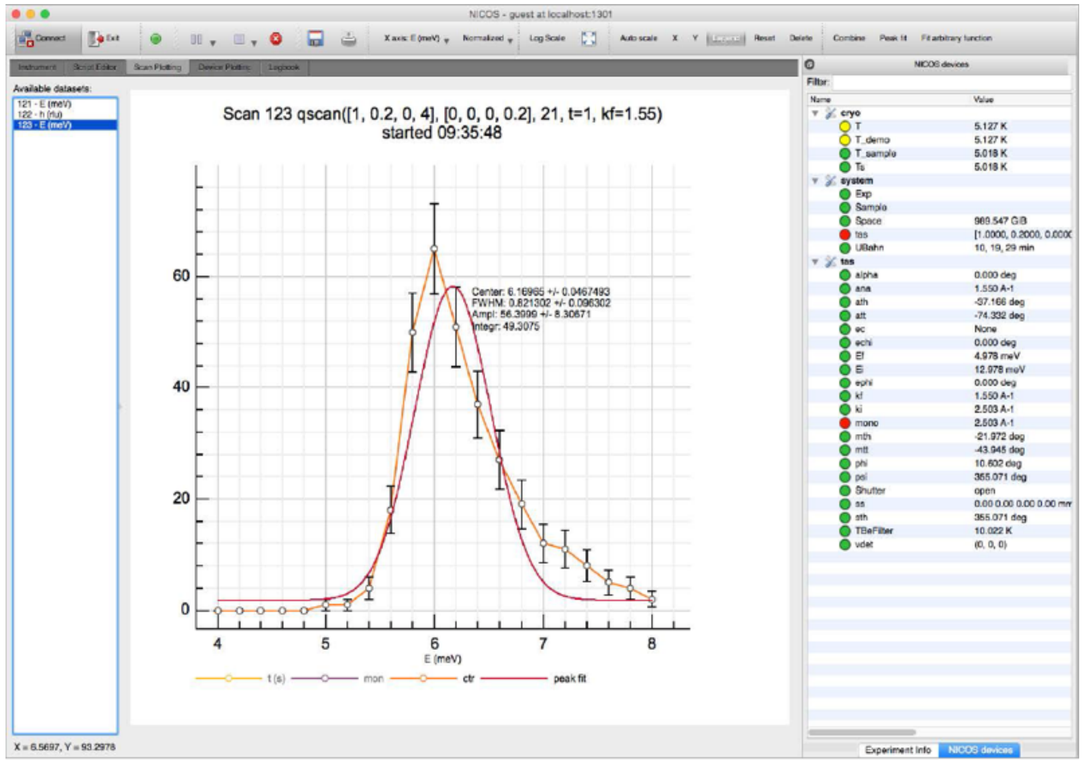
```
var gr = new GR();
var t = 0;
var x = new Array(629);
var y = new Array(629);

var draw = function() {
    gr_clearws();
    var i;
    for (i = 0; i < 629; i++) {
        x[i] = i / 630.0 * 2 * Math.PI;
        y[i] = Math.sin(x[i] + t / 10.0);
    }

    gr_setviewport(0.1, 0.95, 0.1, 0.95);
    gr_setwindow(0, 8, -1, 1);
    gr_setcharheight(0.020);
    gr_grid(0.5, 0.1, 0, -1, 4, 5);
    gr_axes(0.5, 0.1, 0, -1, 4, 5, 0.01);
    gr_polyline(629, x, y);
    gr_updatews();
    t = t + 1;
    if (t < 200) {
        setTimeout(draw, 1);
    }
};

draw();
```

What else can GR be used for?

pyMolDyn



NICOS



see Poster session: *Embedding visualization applications with pygr* by *Christian Felder*

Conclusions

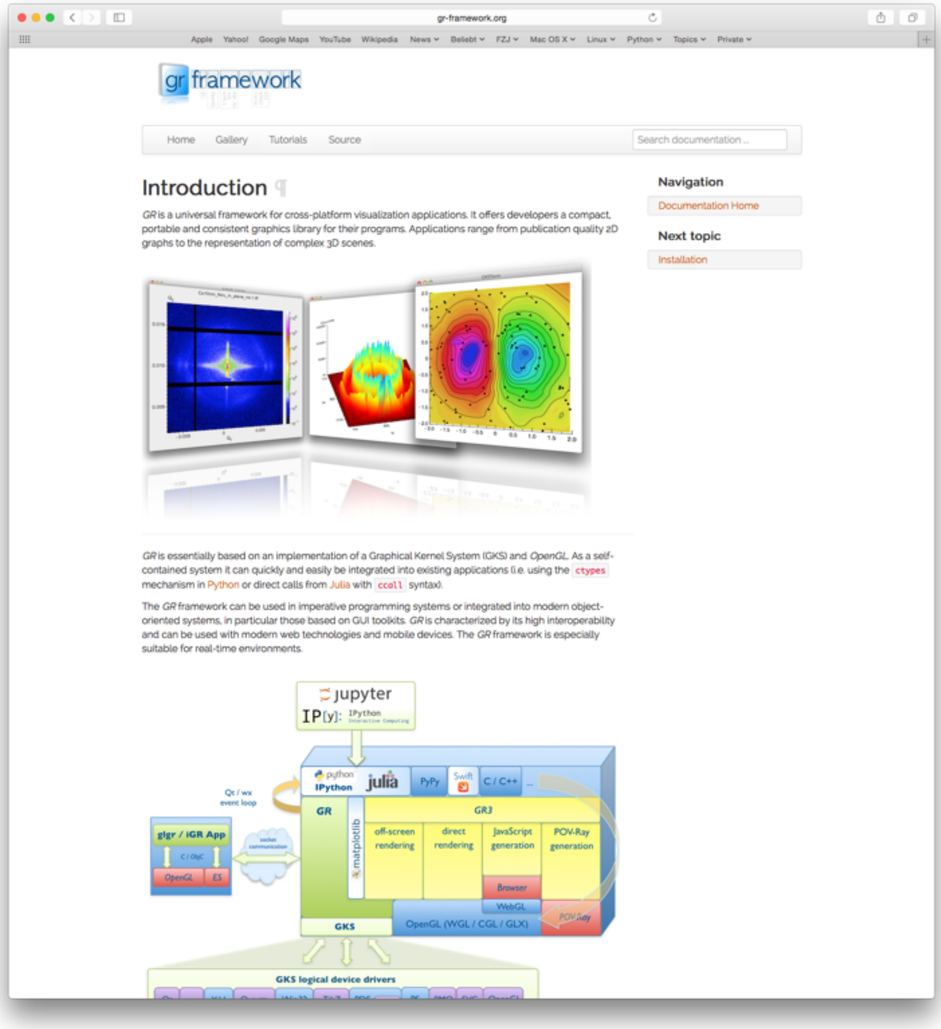
- ✓ Using the GR Matplotlib backend has not turned out satisfactory as the speedups were not as expected
- ✓ GR adds more plotting capabilities to Matplotlib allowing to mix 2D drawings and 3D graphics scenes or create movies on the fly
- ✓ Producing plots / figures is much faster with the GR framework (speedup for plots > 20, > 100 respectively)

What happens next?

- ✓ integrate JavaScript GKS logical device driver
- ✓ provide more convenience function
- ✓ migrate the GR3 library to *modern OpenGL* (using OpenGL shader language)
  - ➠ visualize millions of vertices / faces
- ✓ simplify the installation

Resources

- ✓ Website: http://gr-framework.org
- ✓ GR framework: https://github.com/jheinen/gr
- ✓ PyPI: https://pypi.python.org/pypi/gr
- ✓ Talk material: Getting more out of Matplotlib with GR



Thank you for your attention

Questions?

Contact:

j.heinen@fz-juelich.de
@josef_heinen

Thanks to:

Fabian Beule, Steffen Drossard, Christian Felder, Marvin Goblet, Ingo Heimbach, Daniel Kaiser, Philip Klinkhammer, David Knodt, Florian Rhiem, Jörg Winkler et al.