



# GRAPHICS AND ANIMATIONS IN PYTHON

## USING MATPLOTLIB AND OPENGL

**Nicolas P. Rougier**

PyConFr Conference 2014

Lyon, October 24–25

# Graphics and Animations in Python

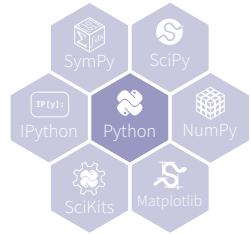
## Where do we start ?



## A Bit of Context

### The Python Scientific Stack

- Python, modern computing script language
- IPython, an advanced Python shell
- Numpy, powerful numerical arrays objects.
- Scipy, high-level data processing routines.
- **Matplotlib**, 2-D visualization plots

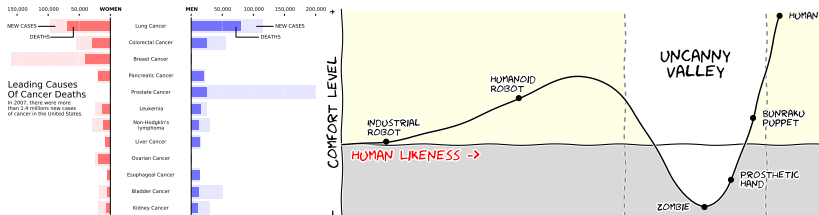


### Versatile & beautiful (but a bit slow)

Matplotlib is a python plotting library, primarily for 2-D plotting, but with some 3-D support, which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

- **Antigrain geometry**, High Fidelity 2D Graphics ([www.antigrain.com](http://www.antigrain.com))
- Matplotlib can draw virtually anything

Matplotlib allows to draw any kind (plot, scatter, quiver, etc.) of visualization for virtually any scientific domain.



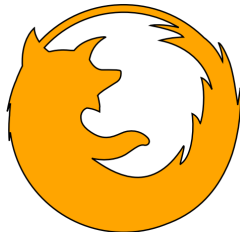
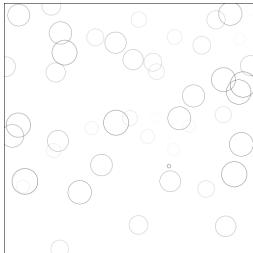
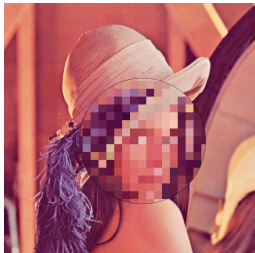
From **Ten Simple Rules for Better Figures**

N.P. Rougier, M.Droettboom, P.E. Bourne PLoS Computational Biology, Vol. 10, No. 9.

Code at [github.com/rougier/ten-rules](https://github.com/rougier/ten-rules)



Matplotlib can also be used as a regular graphic package allowing to draw or animate anything.



From **Introduction à Matplotlib**

N.P. Rougier, Linux Mag HS, August 2014

Code at [github.com/rougier/LinuxMag-HS-2014](https://github.com/rougier/LinuxMag-HS-2014)

# Matplotlib

Example: display an image

```
import matplotlib.pyplot as plt
from matplotlib._png import read_png

RGB = read_png('lena.png')
height, width = RGB.shape[:2]
dpi = 72.0

figsize= width/float(dpi), height/float(dpi)
fig = plt.figure(figsize=figsize, dpi=dpi, facecolor="white")
ax = fig.add_axes([0.0, 0.0, 1.0, 1.0], frameon=False)

ax.imshow(RGB)
ax.set_xticks([]), ax.set_yticks([])

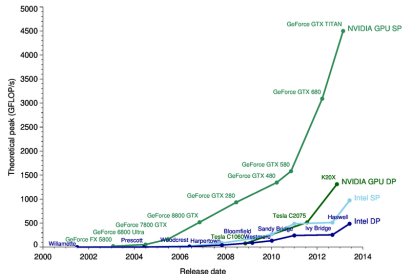
plt.show()
```

## What about OpenGL ?

Powerful, fast but... **ugly** !

- No decent anti-aliasing
- Only two image filters
- No native text handling
- No markers, no arrows
- No paths, no curves

But this can be fixed !



# Python/OpenGL frameworks

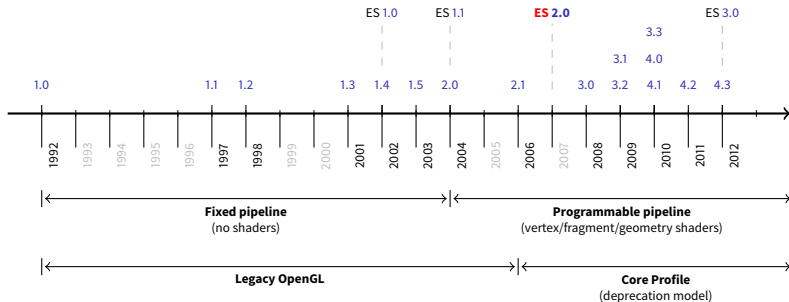
## Rendering framework

- Pyglet  
[www.pyglet.org](http://www.pyglet.org)
- PyOpenGL  
[pyopengl.sourceforge.net](http://pyopengl.sourceforge.net)
- Nodebox for OpenGL  
[www.cityinabottle.org/nodebox](http://www.cityinabottle.org/nodebox)
- PyProcessing  
[code.google.com/p/pyprocessing](http://code.google.com/p/pyprocessing)
- Kivy (**see tomorrow**)  
<http://kivy.org/>
- PyOpenGLng (**see tomorrow**)  
<https://github.com/FabriceSalvaire/PyOpenGLng>

## Visualization framework

- mayavi 2 (Enthought)  
[github.com/enthought/mayavi](https://github.com/enthought/mayavi)
- VTK (Kitware)  
[www.vtk.org](http://www.vtk.org)
- galry (Cyrille Rossant)  
[rossant.github.io/galry/](http://rossant.github.io/galry/)
- visvis (Almar Klein)  
[code.google.com/p/visvis/](http://code.google.com/p/visvis/)
- glumpy (Nicolas Rougier)  
[code.google.com/p/glumpy/](http://code.google.com/p/glumpy/)
- pyqtgraph (Luke Campagnola)  
[www.pyqtgraph.org](http://www.pyqtgraph.org)

# OpenGL history



Doom (1993)

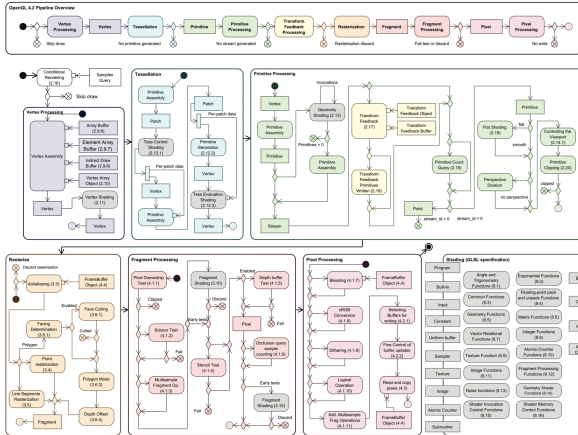


Rage (2011)

# OpenGL 4.2 pipeline overview

(could have been worse...)

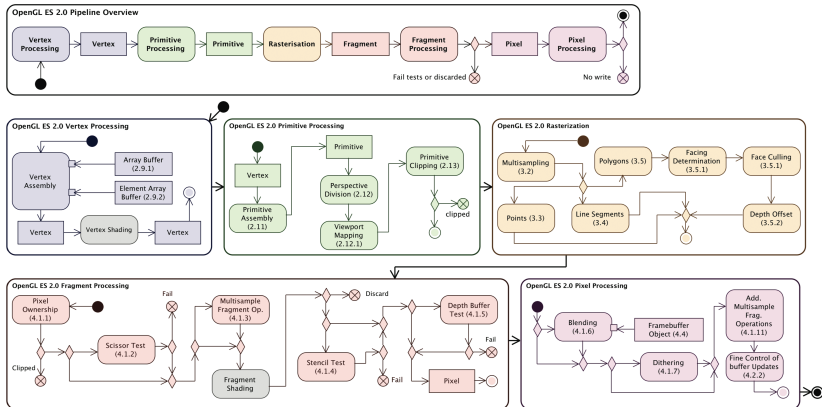
Around 2000 constants and 1000 functions.



# OpenGL ES 2.0 pipeline overview

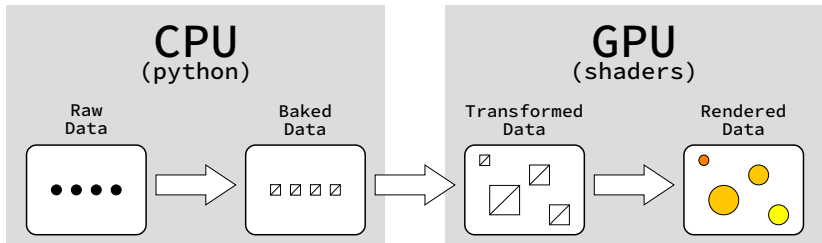
([openglinsights.com](http://openglinsights.com))

Around 350 constants and 150 functions.



## Pipeline overview

Data centered

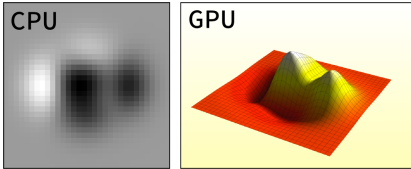


Critical parts are the **baking** process and the **transfer** to GPU memory.



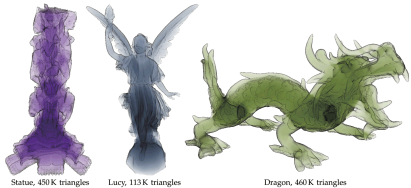
## Baking process

### Ideal case: no baking



Interpolation, colorization, leveling, gridding, scaling, lighting, aliasing, rendering entirely done on GPU.

### Hard case: baking depends on transformation



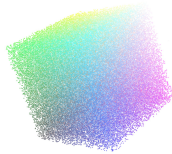
Transparency implies lot of CPU processing (sorting) or multi-pass rendering.

# Performance and Quality

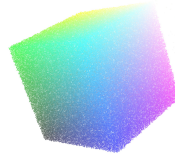
We do not have to (always) trade quality for speed



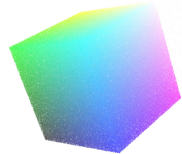
10,000 pts - 403 FPS



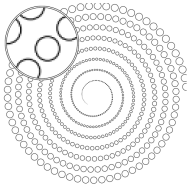
100,000 pts - 140 FPS



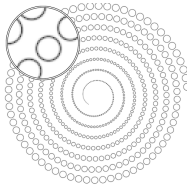
1,000,000 pts - 40 FPS



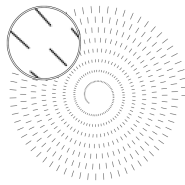
10,000,000 pts - 1.5 FPS



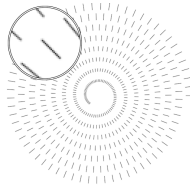
AntiGrain Geometry  
(matplotlib agg backend)



OpenGL AntiGrain  
(using dedicated shaders)



AntiGrain Geometry  
(matplotlib agg backend)



OpenGL AntiGrain  
(using dedicated shaders)

# Application Programming Interface

## Raw GLUT

```
import sys
import OpenGL.GL as gl
import OpenGL.GLUT as glut

def display():
    gl.glClear (gl.GL_COLOR_BUFFER_BIT
               | gl.GL_DEPTH_BUFFER_BIT)
    # draw something
    glut.glutSwapBuffers()

glut.glutInit(sys.argv)
glut.glutInitDisplayMode(glut.GLUT_DOUBLE |
                        glut.GLUT_RGBA |
                        glut.GLUT_DEPTH)
glut.glutCreateWindow(sys.argv[0])
glut.glutDisplayFunc(display)
gl.glClearColor(1,1,1,1)
glut.glutMainLoop()
```

## Low-level (current)

```
from vispy import app

@app.connect
def on_paint(event):
    # draw something

app.run()
```

## High-level (future)

```
import numpy as np
import vispy as vp

P = np.random.random((1000,3))
vp.scatter(P), vp.show()
```

# Shaders

What do they look like ?

## Vertex shader

```
attribute vec3 position;  
void main()  
{  
    gl_Position = vec4(position,1.0);  
}
```

## Fragment shader

```
uniform vec4 color;  
void main()  
{  
    gl_FragmentColor = color;  
}
```

## Scalable Vector Graphics (SVG)

Starting from these basic shaders, we need to handle

- ✓ Text
- ✓ Paths
- ✓ Basic shapes
- ✓ Painting: Filling, Stroking and Marker Symbols
- ✓ Clipping, Masking and Compositing
- ✓ Filter Effects

...

## Different techniques

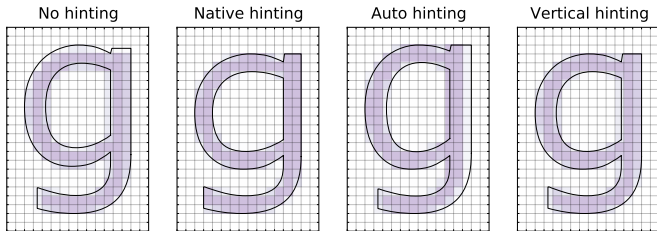
Bitmap, stroke, texture, sdf, vector...



→ Nicolas P. Rougier, **Higher Quality 2D Text Rendering**, Journal of Computer Graphics Techniques (JCGT), vol. 2, no. 1, 50-64, 2013.

## Higher quality text rendering

### Vertical vs Horizontal hinting



→ Maxim Shemarev, **Texts Rasterization Exposures**, An attempt to improve text rasterization algorithms, 2007

### Implementation ([github.com/rougier/freetype-gl](https://github.com/rougier/freetype-gl))

- Subpixel positioning & kerning
- Per pixel gamma correction
- Signed Distance Fields









## Dashed stroked polyline

### GL line width (fixed pipeline)

- Limited in thickness
- No control over joins and caps
- Deprecated & ugly

### GL Stipple (fixed pipeline)

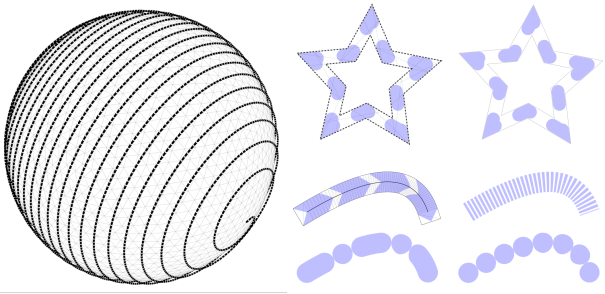
- Limited in pattern
- No control over dash caps
- Deprecated & ugly

PATTERN	FACTOR	
0x00FF	1	
0x00FF	2	
0x0C0F	1	
0x0C0F	3	
0xAAAA	1	
0xAAAA	2	
0xAAAA	3	
0xAAAA	4	

# Higher quality dashed stroked polyline

## Shader based approach

A new method for rendering arbitrary dash patterns along any continuous polyline (smooth or broken). The proposed method does not tessellate individual dash patterns and allows for fast and accurate rendering of any user-defined dash pattern and caps

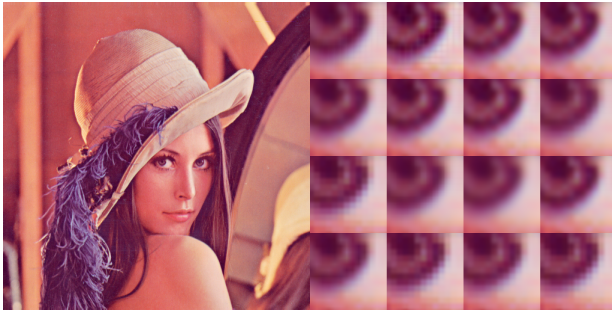


→ Nicolas P. Rougier, **Shader-Based Antialiased, Dashed, Stroked Polylines**, Journal of Computer Graphics Techniques (JCGT), vol. 2, no. 2, 105–121, 2013.



## Image interpolation & filters

OpenGL offers only nearest and linear filters while much more are needed for scientific visualization (Hanning, Hamming, Hermite, Kaiser, Quadric, Bicubic, CatRom, Mitchell, Spline16, Spline36, Gaussian, Bessel, Sinc, Lanczos, Blackman, etc.)

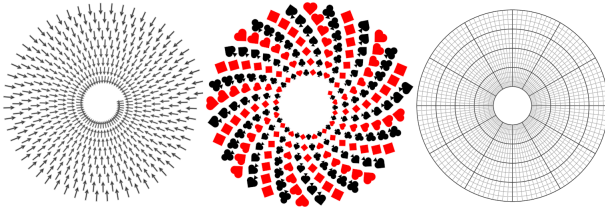


→ Kevin Bjorke, **High-Quality Filtering** in GPU gems 2 : programming techniques for high-performance graphics and general-purpose computation / edited by Matt Pharr ; Randima Fernando (2007).

## Grids, markers and arrows

### Point based approach

A new method for drawing grids, markers, and arrows using implicit functions such that it is possible to draw pixel-perfect antialiased objects with very good performances.



→ Nicolas P. Rougier, **Shader Based Antialiased 2D Grids, Markers, and Arrows**, Journal of Computer Graphics Techniques (JCGT), to appear, 2014.

**SHOWTIME**

## Conclusion

The code is spread in several projects but should be soon integrated in the master **vispy** project.

### Projects page

- [vispy.org](https://vispy.org)
- [vispy.org/gallery.html](https://vispy.org/gallery.html)
- [glumpy.github.io](https://glumpy.github.io)
- [glumpy.github.io/gallery.html](https://glumpy.github.io/gallery.html)

### Code repositories

- [github.com/vispy/vispy](https://github.com/vispy/vispy)
- [github.com/glumpy/glumpy](https://github.com/glumpy/glumpy)
- [github.com/rougier/gl-agg](https://github.com/rougier/gl-agg)
- [github.com/rougier/freetype-gl](https://github.com/rougier/freetype-gl)

And we should get soon WebGL backend...

Questions ?