MODERN & INTERACTIVE SCIENTIFIC VISUALIZATION
USING SHADE BASED RENDERING

Nicolas P. Rougier
Luke Campagola - Almar Klein - Cyrille Rossant - Eric Larson
X.Org Developer Conference 2014
Bordeaux, October 8–10
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... 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)
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

Proprietary solution


→ OpenVG API, Standard for Vector Graphics Acceleration, Khronos group.

But this can also be fixed freely!
Python/OpenGL frameworks

Rendering framework

• Pyglet
  www.pyglet.org

• PyOpenGL
  pyopengl.sourceforge.net

• Nodebox for OpenGL
  www.cityinabottle.org/nodebox

• PyProcessing
  code.google.com/p/pyprocessing

Visualization framework

• mayavi 2 (Enthought)
  github.com/enthought/mayavi

• VTK (Kitware)
  www.vtk.org

• galry (Cyrille Rossant)
  rossant.github.io/galry/

• visvis (Almar Klein)
  code.google.com/p/visvis/

• glumpy (Nicolas Rougier)
  code.google.com/p/glumpy/

• pyqtgraph (Luke Campagnola)
  www.pyqtgraph.org
OpenGL history

ES 1.0
ES 1.1
ES 2.0
ES 3.0

1.0
1.1
1.2
1.3
1.4
1.5
2.0
2.1
3.0
3.1
3.2
4.0
4.1
4.2
4.3

1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012

Fixed pipeline
(no shaders)

Programmable pipeline
(vertex/fragment/geometry shaders)

Legacy OpenGL

Core Profile
(deprecation model)

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)

Around 350 constants and 150 functions.
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.
Where do we start?

Scalable Vector Graphics (SVG) 2

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

...
Different techniques
Bitmap, stroke, texture, sdf, vector...

Higher quality text rendering

Vertical vs Horizontal hinting

- No hinting
- Native hinting
- Auto hinting
- Vertical hinting


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

<table>
<thead>
<tr>
<th>PATTERN</th>
<th>FACTOR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00FF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x00FF</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0x0C0F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x0C0F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0xA0AA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0xA0AA</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0xA0AA</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0xA0AA</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
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.

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.)

Grids, markers and arrows

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

Still lot of problems ahead...
...but work is in progress

Shader composition
How to define a shader format that allow easy composition/templating?

Level of details
How to set automatic level of details?

Very big data
How to render data that doesn’t even fit into GPU memory?

Complex data transformation
How to handle user-supplied exotic transformation?

From DesktopGL to WebGL
How to render in browser from a python session?

...
Conclusion

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

**Projects page**
- vispy.org
- vispy.org/gallery.html
- glumpy.github.io
- glumpy.github.io/gallery.html

**Code repositories**
- github.com/vispy/vispy
- github.com/glumpy/glumpy
- github.com/rougier/gl-agg
- github.com/rougier/freetype-gl

**Demo pages**
- Markers: https://www.shadertoy.com/view/XsXXDX
- Arrows: https://www.shadertoy.com/view/ldlSWj
- Transverse Mercator grid: https://www.shadertoy.com/view/lsSXzm
- Cartesian grid: https://www.shadertoy.com/view/MdSXRm
- Polar grid: https://www.shadertoy.com/view/MsBSRm