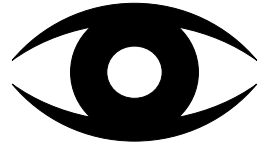
Vispy

A Modern and Interactive Visualization Framework

Luke Campagnola, Almar Klein, Cyrille Rossant
& **Nicolas Rougier** (it's me)

EuroSciPy 2013



Our goal is to create a **interactive** visualization software in Python based on OpenGL/WebGL.

We do not aim at replacing matplotlib.
We aim at living by its side.

Python/OpenGL frameworks

Rendering framework

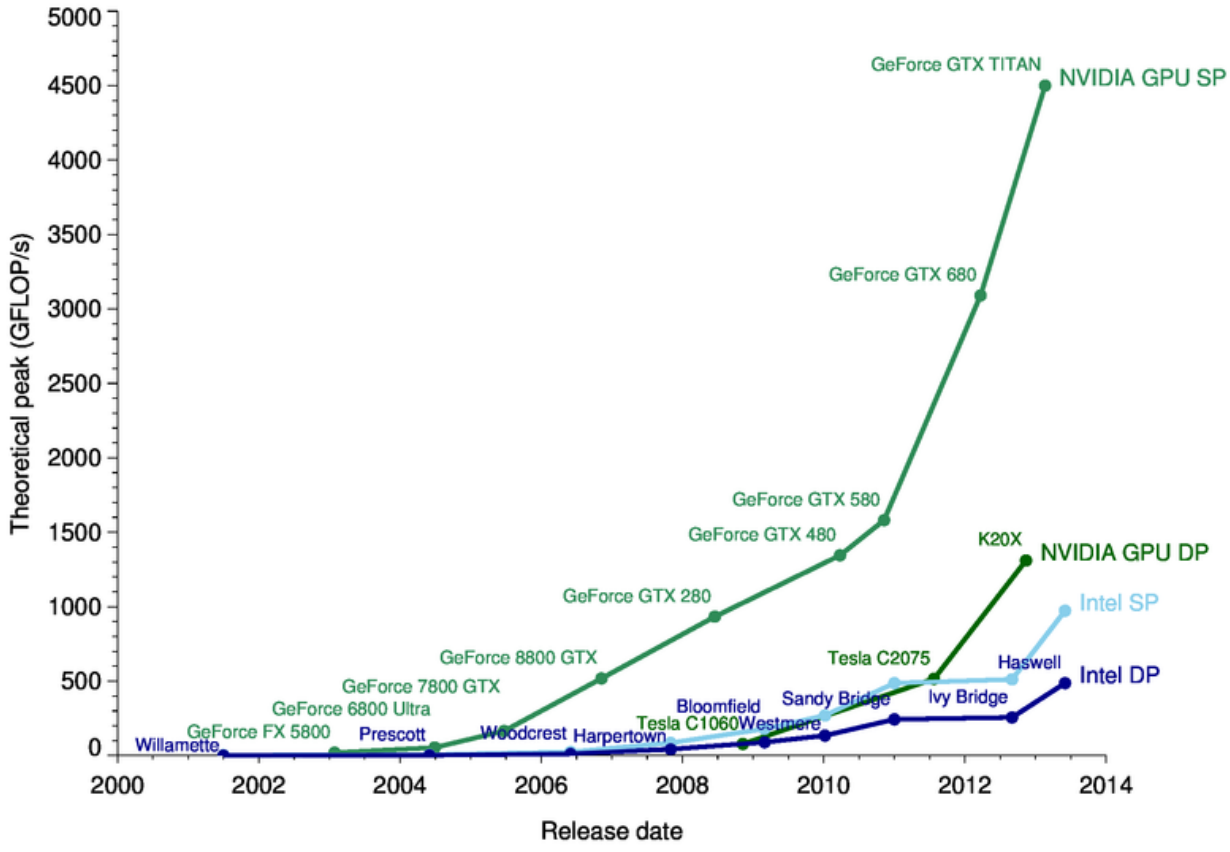
- Pyglet
  `www.pyglet.org`
- PyOpenGL
  `pyopengl.sourceforge.net`
- Nodebox for OpenGL
  `www.cityinabottle.org/nodebox`
- PyProcessing
  `code.google.com/p/pyprocessing`

Visualization framework

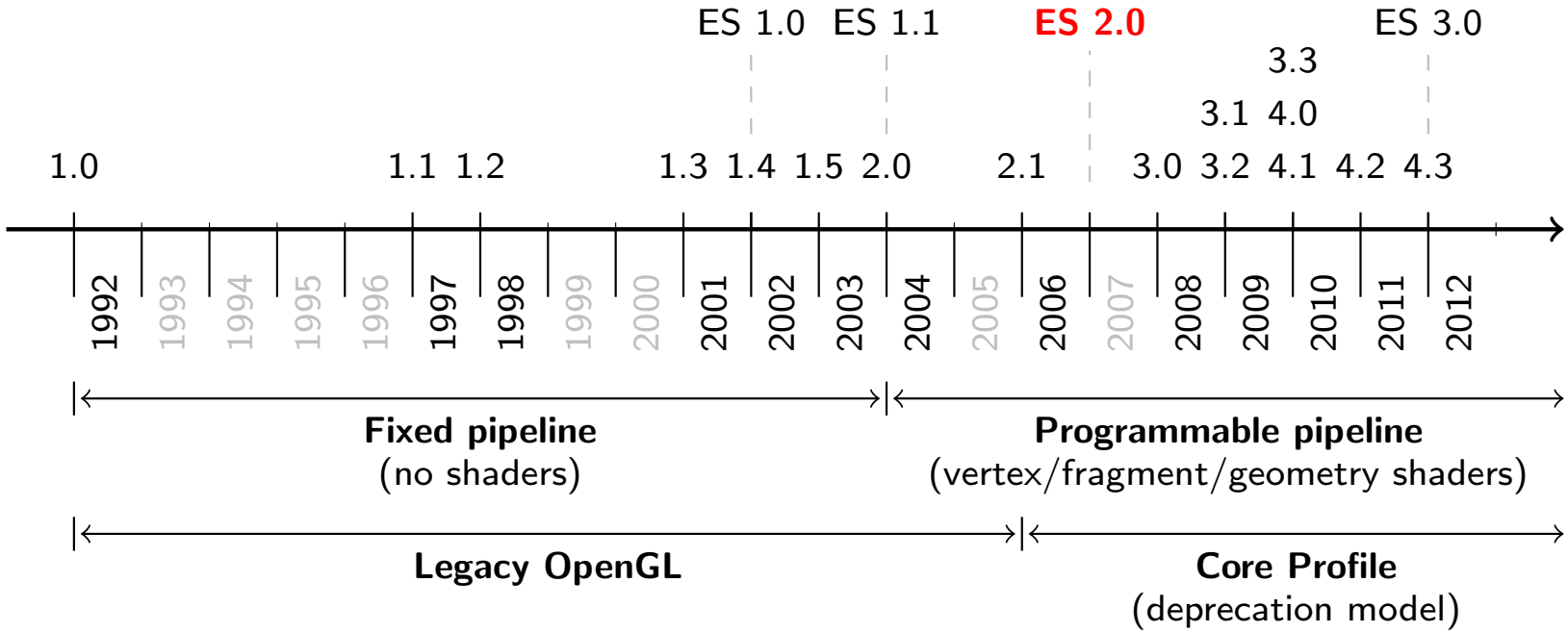
- mayavi 2 (Enthought)
  `github.com/enthought/mayavi`
- VTK (Kitware)
  `www.vtk.org`
- galry (Cyrille Rossant)
  `rossant.github.io/galry/`
- visvis (Almar Klein)
  `code.google.com/p/visvis/`
- glumpy (Nicolas Rougier)
  `code.google.com/p/glumpy/`
- pyqtgraph (Luke Campagnola)
  `www.pyqtgraph.org`

Graphic cards

video-gaming industry is working for us !



OpenGL history



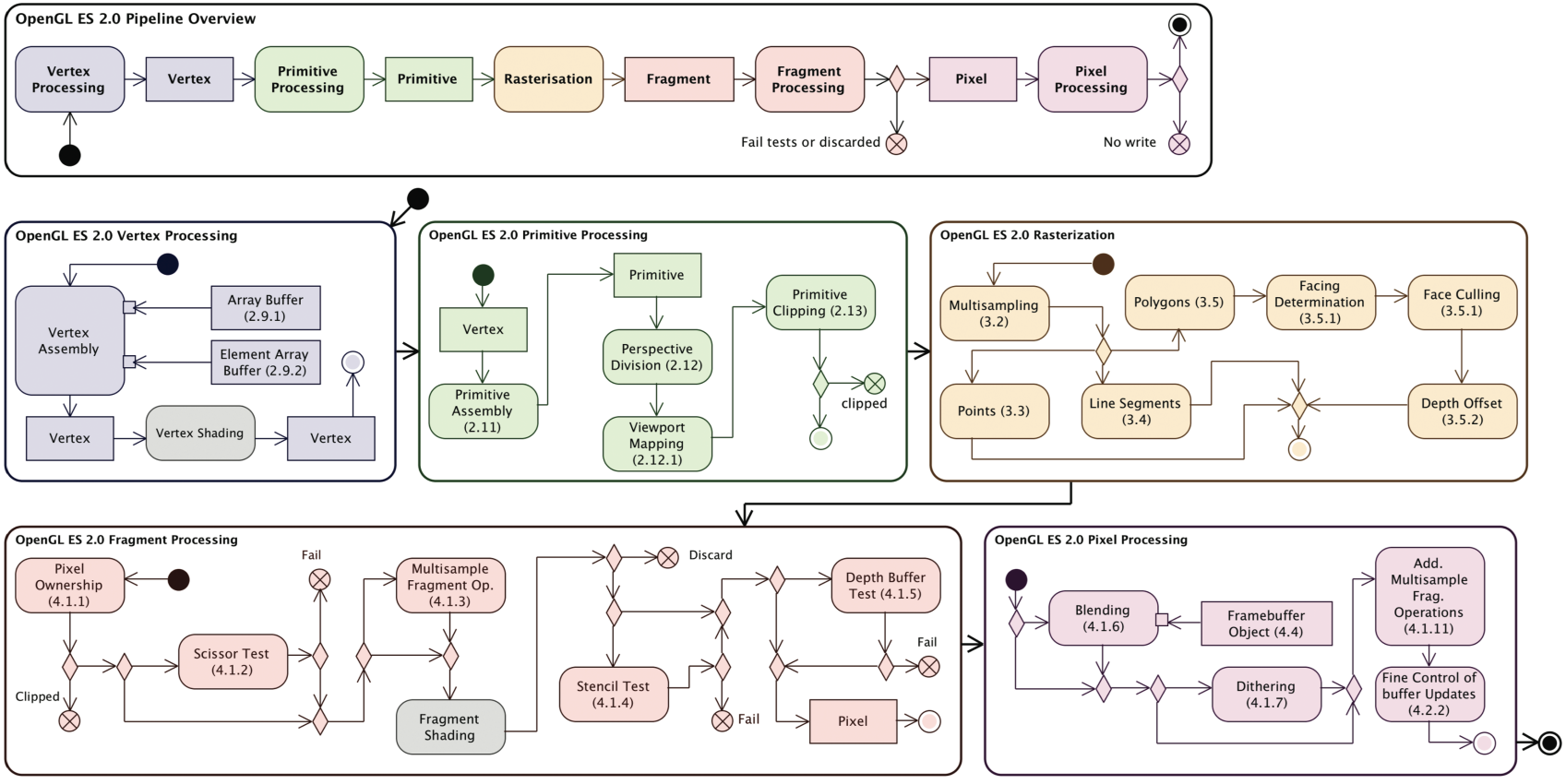
Doom (1993)

Rage (2011)

OpenGL ES 2.0 pipeline overview

(openglinsights.com)

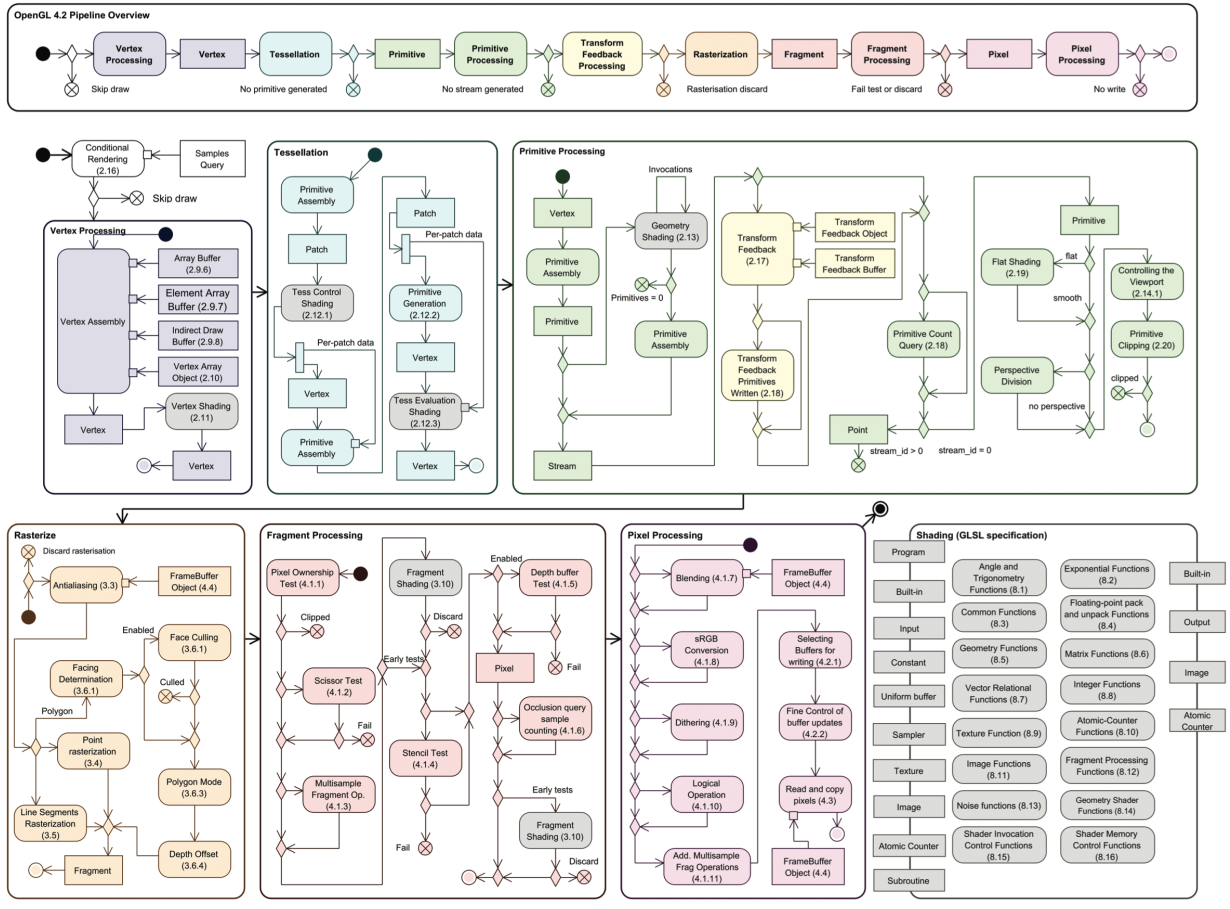
Around 350 constants and 150 functions.



OpenGL 4.2 pipeline overview

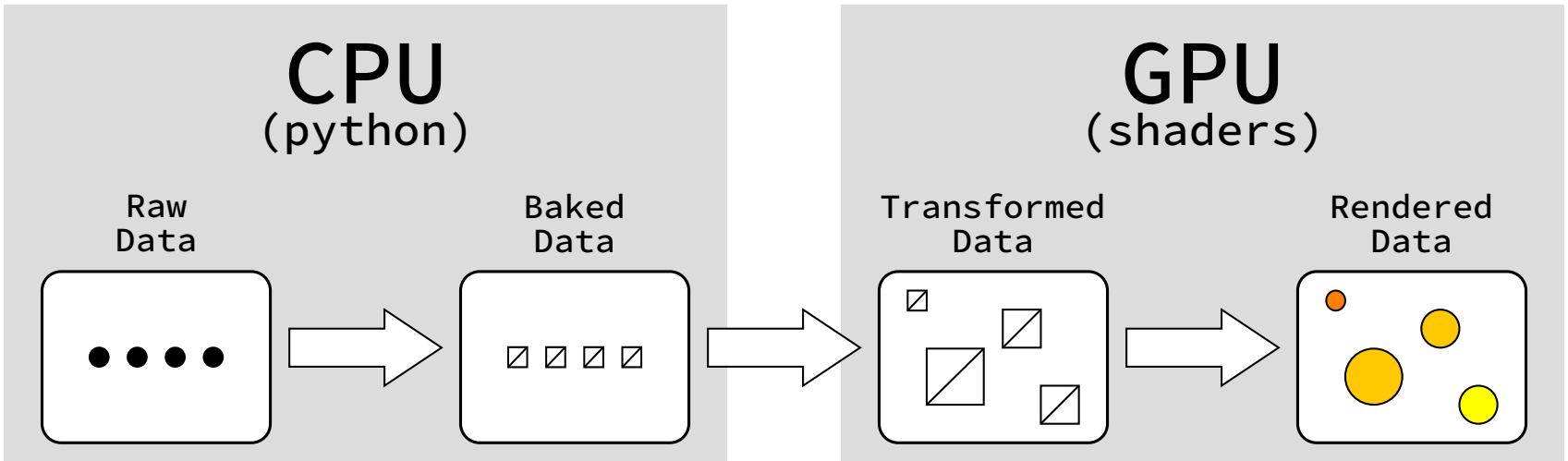
(could have been worse...)

Around 2000 constants and 1000 functions.



Vispy pipeline overview

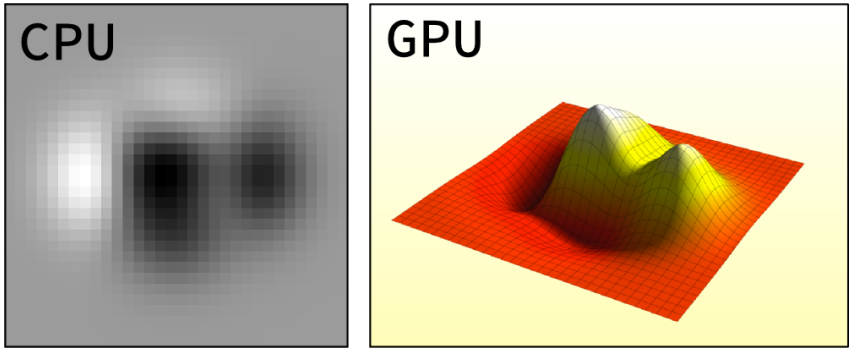
Data centered



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

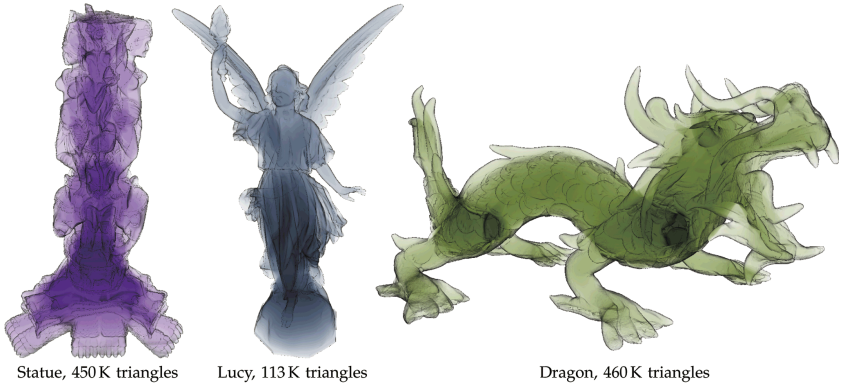
Baking process

Ideal case: no baking



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

Hard case: baking depends on transformation



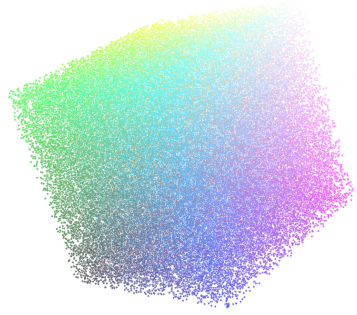
Statue, 450 K triangles Lucy, 113 K triangles Dragon, 460 K triangles

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

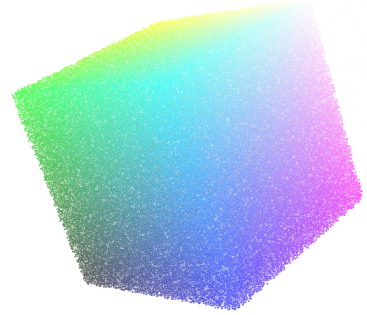
Performances and Quality

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

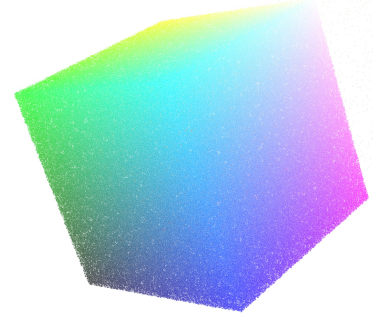
10,000 pts - 403 FPS



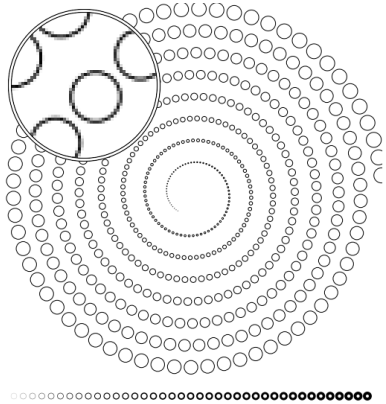
100,000 pts - 140 FPS



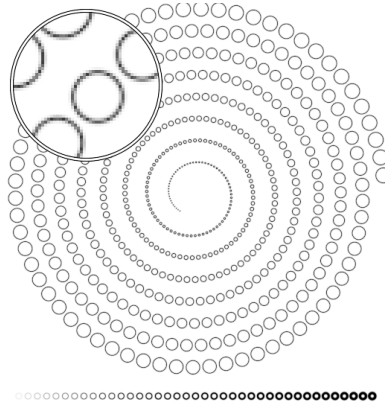
1,000,000 pts - 40 FPS



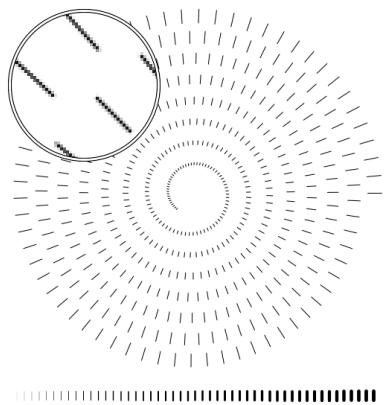
10,000,000 pts - 1.5 FPS



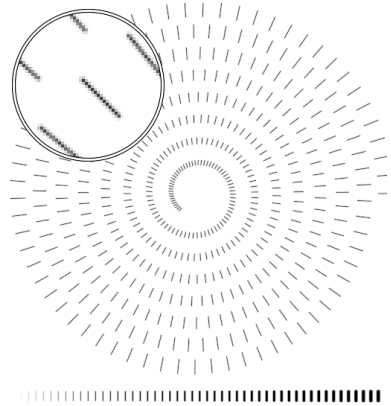
AntiGrain Geometry
(matplotlib agg backend)



OpenGL AntiGrain
(using dedicated shaders)



AntiGrain Geometry
(matplotlib agg backend)



OpenGL AntiGrain
(using dedicated shaders)

Architecture

Modules

- app - Window & events
- gl - OpenGL ES 2.0 (emulation)
- io - input/output
- oogl - Object Oriented OpenGL
- visual - Low-level interface
- plot - High-level interface

Backends

- GLUT (historical, packaged with OpenGL)
- Pyglet
- PyQT
- WebGL (not yet done)

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;
}
```

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 ?

Supersampling or antialiasing ?

Full 2D anti-aliasing possible, but unlikely for complex 3D scene.

...


SHOWTIME

Informations

Project page

`vispy.org`

Code repository

`github.com/vispy/vispy`

Sprint

A sprint is organised on Sunday. Interested people are encouraged to come to share ideas, comments, code, and even request features !