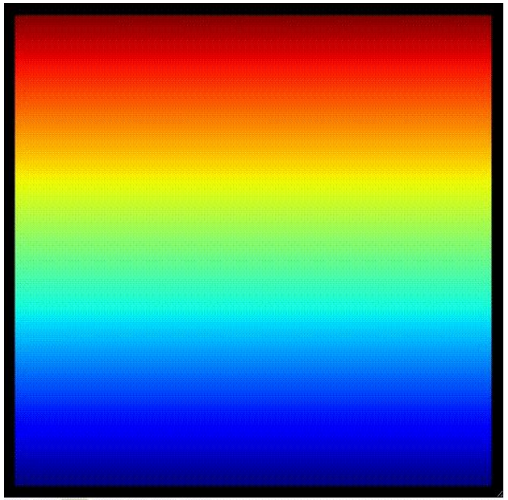
fastplotlib

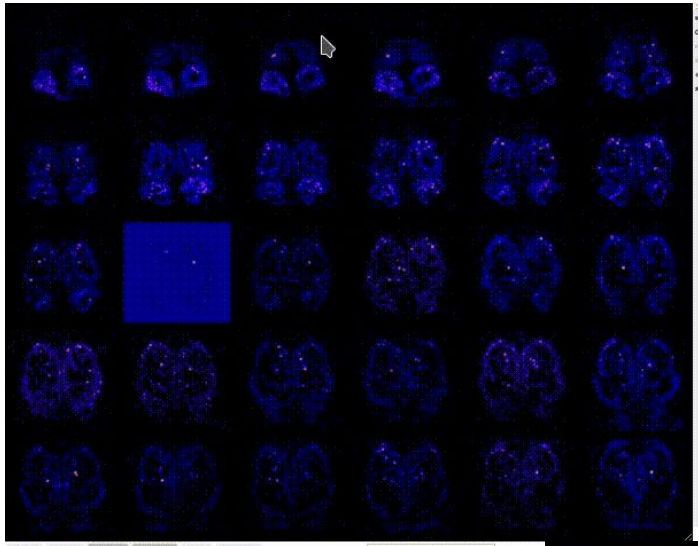
https://github.com/kushalkolar/fastplotlib

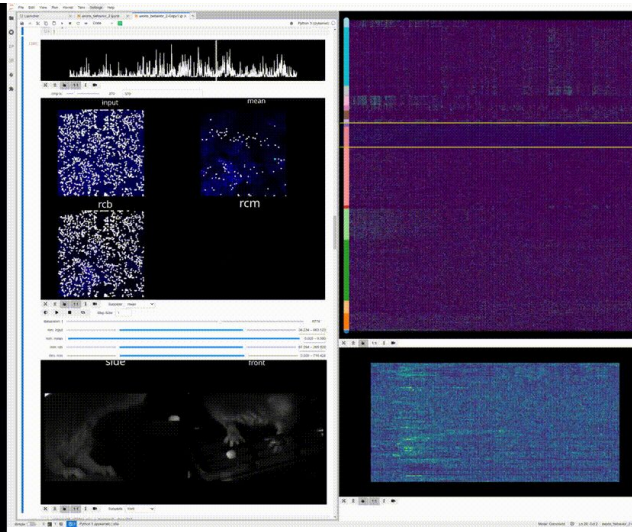
Ultrafast interactive visualizations

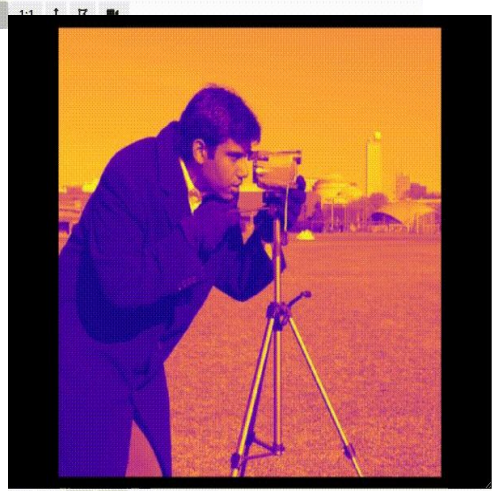
Caitlin Lewis @caitlinllewis clewis7

Kushal Kolar @kushalkolar kushalkolar









1:1 Subplots: (0, 0)

Step Size: 1 Speed 100

dimension: t

1:1

0

New Graphics APIs

Vulkan / Metal (Mac) / DX12 (Windows)

Image

Wikipedia:

> "Vulkan is a low-overhead, **cross-platform API, open standard** for 3D graphics and computing … **higher performance** and **more efficient CPU and GPU** usage **compared to older OpenGL**"

Basically: New APIs: very fast, efficient, & leverage modern GPU hardware better than OpenGL

New Graphics APIs

Image

Vulkan / Metal / DX12

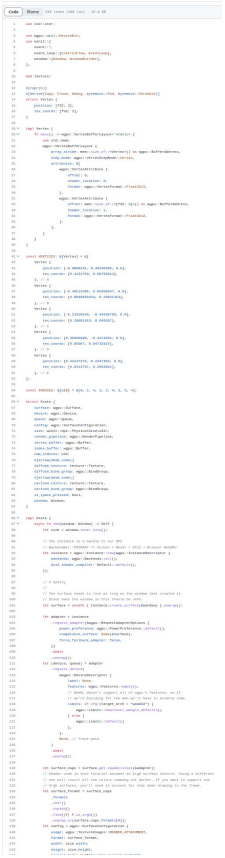
wgpu

wgpu-py

Still low level, manage:

- Buffers
- Render pipeline
- Textures
- Shaders

wgpu - 400+ lines



New Graphics APIs

Image

Vulkan / Metal / DX12

wgpu

wgpu-py

pygfx

pygfx - ~15 lines

```python
import imageio.v3 as iio
from wgpu.gui.auto import WgpuCanvas, run
import pygfx as gfx


canvas = WgpuCanvas()
renderer = gfx.renderers.WgpuRenderer(canvas)
scene = gfx.Scene()

im = iio.imread("imageio:astronaut.png")[:, :, 1]

image = gfx.Image(
    gfx.Geometry(grid=gfx.Texture(im, dim=2)),
    gfx.ImageBasicMaterial(clim=(0, 255)),
)
scene.add(image)

camera = gfx.OrthographicCamera(512, 512)
camera.local.position = (256, 256, 0)
camera.local.scale_y = -1


if __name__ == "__main__":
    canvas.request_draw(lambda: renderer.render(scene, camera))
    run()
```

Higher level, manage:

- Renderer, Canvas
- Scene
- Camera
- Controller
- Objects
  - Geometry
  - Material

pygfx - object model

developers:

Almar Klein
Korijn van Golen
others, Zimmer Biomet
https://github.com/pygfx/pygfx

WorldObject

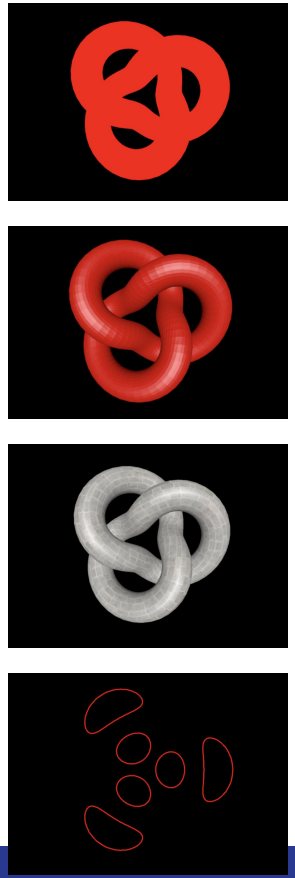
- transform
- visibility
- parent & children

Geometry

Intrinsic data:
- positions
- normals
- colors
- etc.

Material

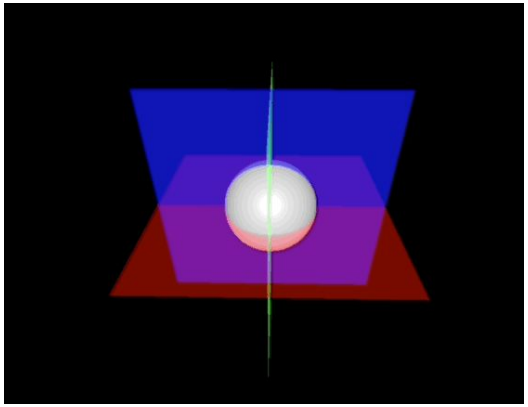
Appearance:
- color
- thickness
- colormap
- etc.



pygfx - some features

- Antialiasing
- Transparency
- Picking
- Event system
- Cameras
- Controllers

- Notebooks
  - jupyter-rfb
  - non-blocking



fastplotlib

- High-level Python API for scientific plotting - inspiration from pyqtgraph, bokeh, etc.
- Built with pygfx rendering engine
- **Interactive in jupyter notebooks - cloud computing and remote infrastructure**
- Goals: fast visualization, **expressive & elegant API**
- Core developers:
  - **Kushal Kolar**
    Caitlin Lewis
  - **Key users:**
    - **Arjun Putcha - streaming data**
    - **Eric Thomson - CaImAn**

Demos!

https://github.com/kushalkolar/fastplotlib-scipy2023

Reduce rendering engine boilerplate

pygfx

```
# create a canvas and renderer
canvas = WgpuCanvas()
renderer = gfx.renderers.WgpuRenderer(canvas)

# create a scene
scene = gfx.Scene()

# create a camera, set position to center of image
camera = gfx.OrthographicCamera(512, 512)
camera.position.y = 256
camera.scale.y = -1
camera.position.x = 256
colormap1 = gfx.cm.plasma
# 512x512 array of random data
rand_img_data = np.random.rand(512, 512).astype(np.float32) * 255
# create an image graphic
# define Geometry with the grid set as a Texture using the random data
img_graphic = gfx.Image(
   gfx.Geometry(grid=gfx.Texture(rand_img_data, dim=2)),
   gfx.ImageBasicMaterial(clim=(0, 255), map=colormap1),
)
scene.add(img_graphic)
def animate():
   renderer.render(scene, camera)
   canvas.request_draw()
canvas.request_draw(animate)
canvas
```

fastplotlib

```
plot = Plot()

data = np.random.rand(512, 512)

plot.add_image(data=data)

plot.show()
```

Focus on scientific data!

canvas
renderer
camera
viewports
geometry
material
buffers

} fastplotlib manages these for you!

GridPlot, Subplot interface

pygfx

```
canvas = WgpuCanvas()
renderer = gfx.renderers.WgpuRenderer(canvas)
dims = (512, 512) # image dimensions
center_cam_pos = (256, 256, 0)
cmaps = [gfx.cm.inferno, gfx.cm.plasma, gfx.cm.magma, gfx.cm.viridis]
scenes = list()
cameras = list()
images = list()
controllers = list()
cntl_defaults = list()
viewports = list()
for i in range(4):
    # create scene for this subplot
    scene = gfx.Scene()
    scenes.append(scene)
    # create a Image WorldObject
    img = gfx.Image(
        gfx.Geometry(           grid=gfx.Texture(np.random.rand(*dims).astype(np.float32) * 255, dim=2)
        ),
        gfx.ImageBasicMaterial(clim=(0, 255), map=cmaps[i]),
    )
    images.append(img)
    scene.add(img)
    camera = gfx.OrthographicCamera(*dims)
    camera.position.set(*center_cam_pos)
    cameras.append(camera)
    viewport = gfx.Viewport(renderer)
    viewports.append(viewport)
    controller = gfx.PanZoomController(camera.position.clone())
controller.add_default_event_handlers(viewport, camera)   controllers.append(controller)
cntl_defaults.append(controller.save_state())
@renderer.add_event_handler("resize")
def layout(event=None):
    """
    Update the viewports when the canvas is resized
    """
    w, h = renderer.logical_size
    w2, h2 = w / 2, h / 2
    viewports[0].rect = 10, 10, w2, h2
    viewports[1].rect = w / 2 + 5, 10, w2, h2
    viewports[2].rect = 10, h / 2 + 5, w2, h2
    viewports[3].rect = w / 2 + 5, h / 2 + 5, w2, h2
def animate():
    for img in images:
        img.geometry.grid.data[:] = np.random.rand(*dims).astype(np.float32) * 255
        img.geometry.grid.update_range((0, 0, 0), img.geometry.grid.size)
    for camera, controller in zip(cameras, controllers):
        controller.update_camera(camera)
    for viewport, s, c in zip(viewports, scenes, cameras):
        viewport.render(s, c)
    renderer.flush()
    canvas.request_draw()
def center_objects(ev):
    for con, cam, img in zip(controllers, cameras, images):
        con.show_object(cam, img)
        con.zoom(1.3)
renderer.add_event_handler(center_objects, "double_click")
layout()
if __name__ == "__main__":
    canvas.request_draw(animate)
    run()
```

fastplotlib

```
# GridPlot of shape 2 x 3
grid_plot = GridPlot(shape=(2, 3))

# Make a random image graphic for each subplot
for subplot in grid_plot:
    img_data = np.random.rand(512, 512)
    # add image to the subplot
    subplot.add_image(img_data)

# update the image graphics with new generated data
def set_random_frame(gp):
    for sp in gp.subplots:
        new_data = np.random.rand(512, 512)
        sp.graphics[0].data = new_data

# add the animation
grid_plot.add_animations(set_random_frame)
grid_plot.show()
```

Roadmap for 2023

Contributions and ideas are welcome from people with all levels of experience!

There are several items highlighted with 🟢 that are perfect for newcomers!

https://github.com/kushalkolar/fastplotlib/issues/55

Come to our sprint on the weekend!

- **fastplotlib for your use case**
- **Contributions**

Things we would like help with:

- **pyimgui integration**

SciPy Sprint

PyScript for fastplotlib in the browser

- Get pygfx to use wgpu
- Browser already has wgpu, would no longer need wgpu-py

magicgui integration instead of ipywidgets

- Uniform support between notebook and desktop using Qt