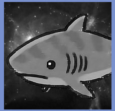
Duke PRATT SCHOOL *of* ENGINEERING

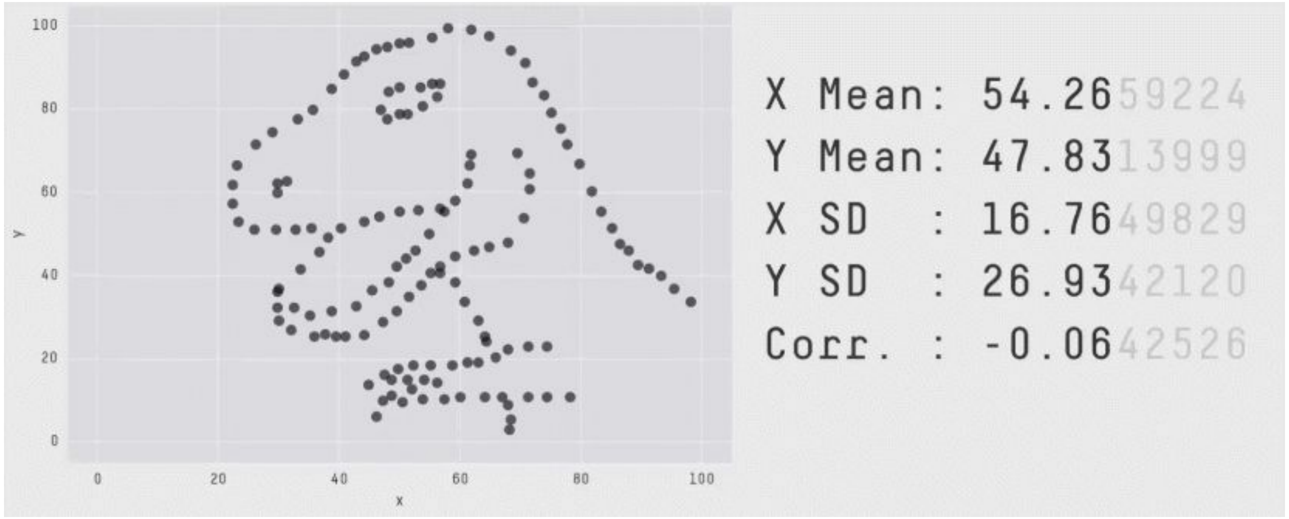
fastplotlib

Augmenting our scientific imagination

	Caitlin Lewis	@caitlinllewis	clewis7
	Kushal Kolar	@kushalkolar	kushalkolar

It is important to look at your data!

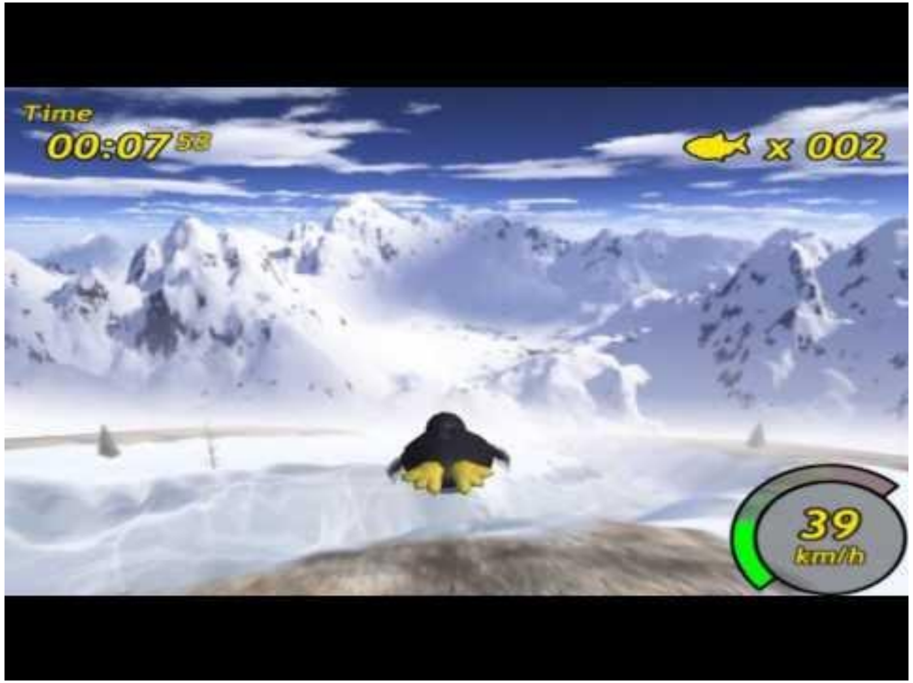
- Statistics are not sufficient
- “All models are wrong, some are useful”
- All algorithms are approximations



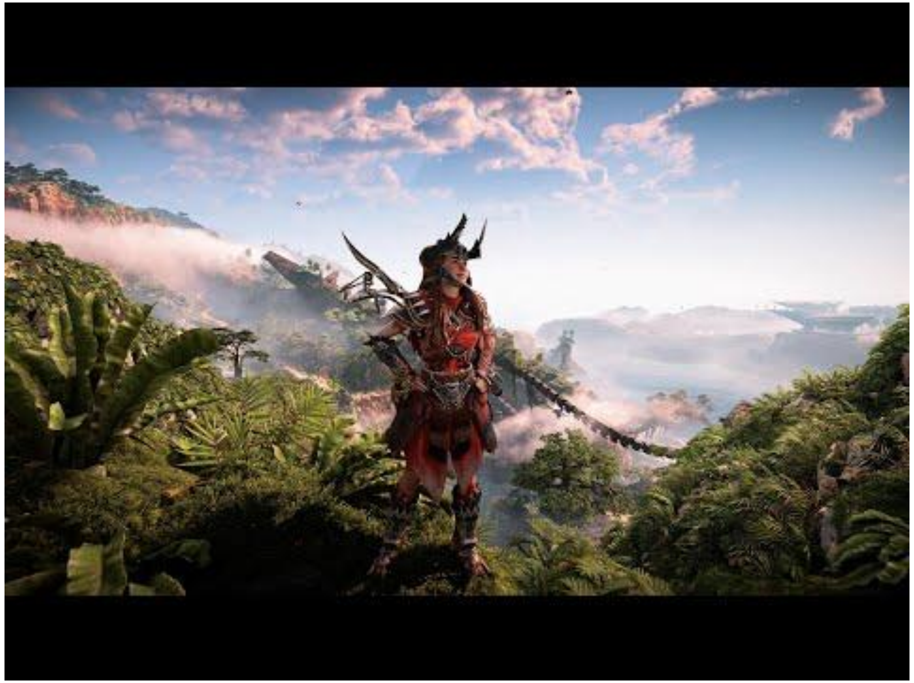
Matejka, Justin, and George Fitzmaurice. "Same stats, different graphs: generating datasets with varied appearance and identical statistics through simulated annealing." *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017.

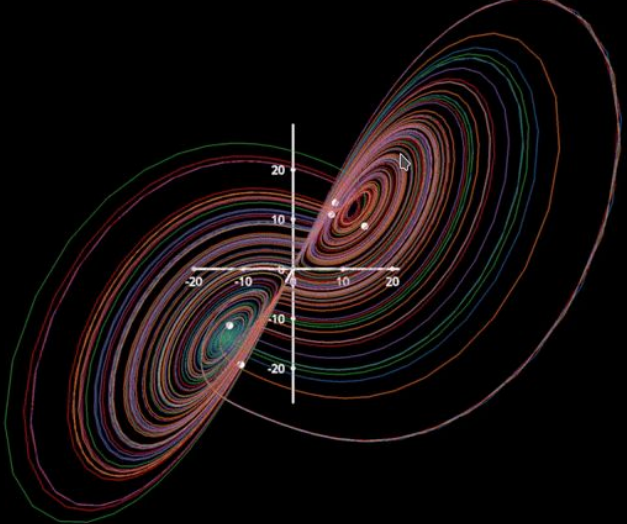
Why don’t scientific plots look as good as modern video games?

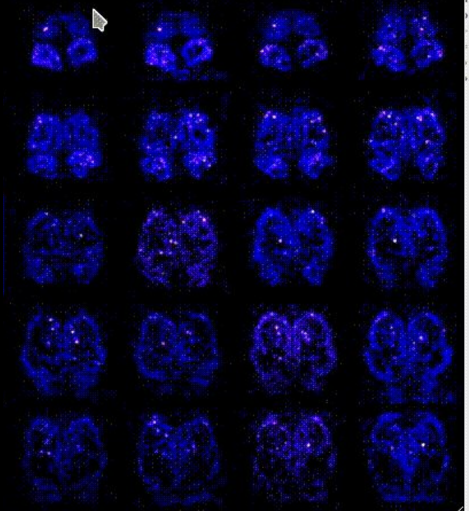
Graphics ~20 years ago

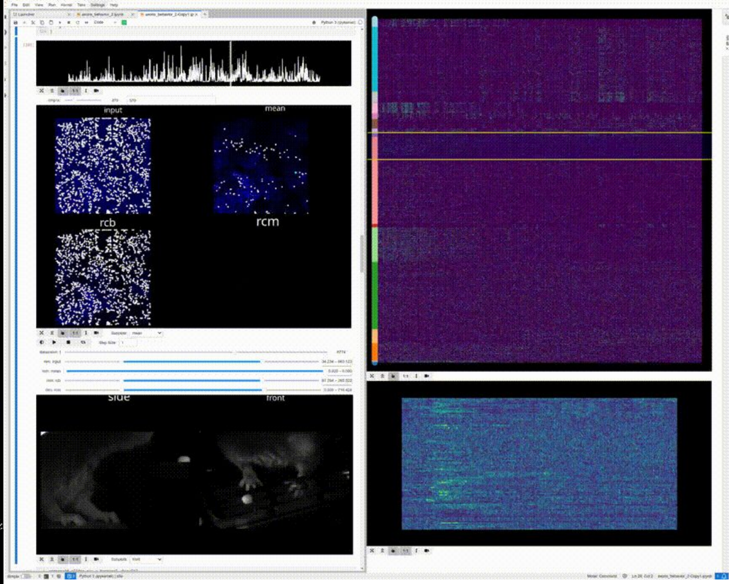


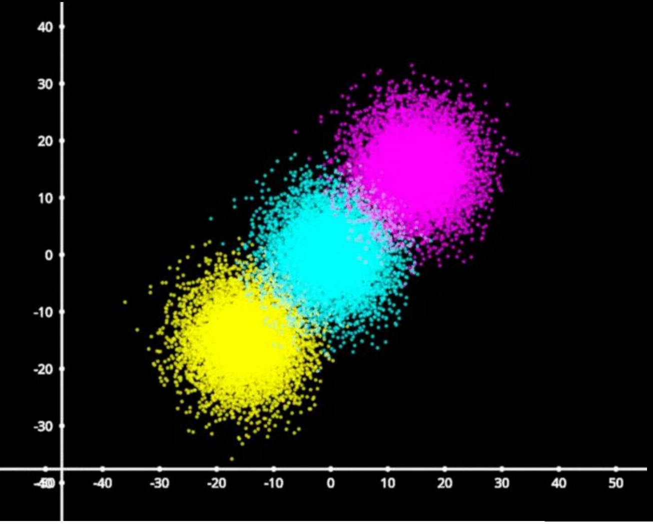
Graphics today

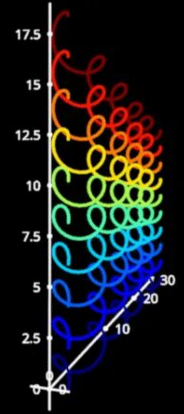



20
10
0
-20
-10
10
20
-10
-20




input
mean
rcb
rcm
side
front

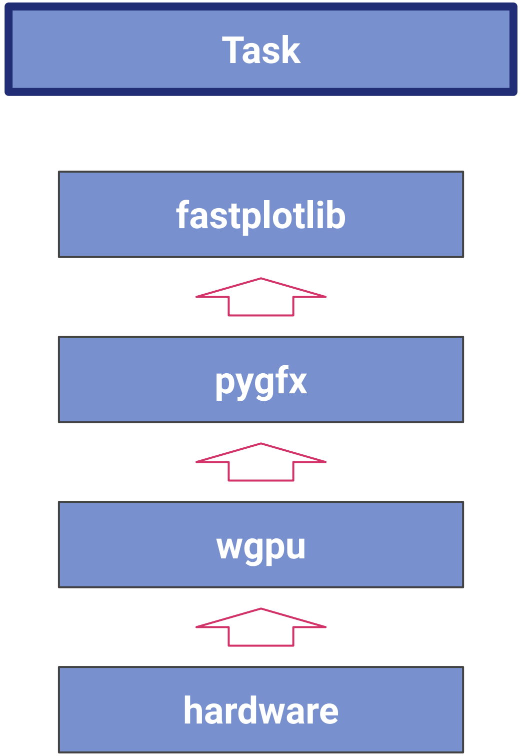

40
30
20
10
0
-10
-20
-30
-50
-40
-30
-20
-10
0
10
20
30
40
50


17.5
15
12.5
10
7.5
5
2.5
0
0
10
20
30

1:1
0

fastplotlib-pygfx-wgpu Stack

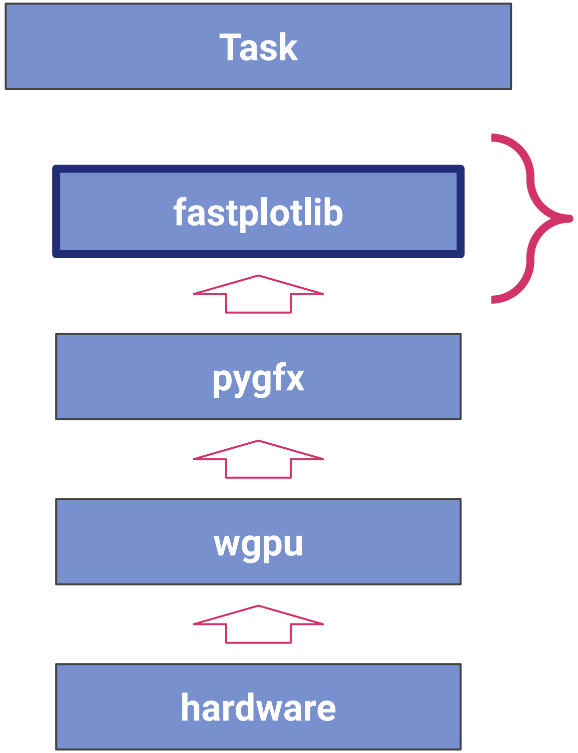
Image



fastplotlib-pygfx-wgpu Stack

Image

```
fig = fpl.Figure() # create a figure

data = iio.imread("imageio:astronaut.png") # data

fig[0, 0].add_image(data=data) # plot an image

fig.show() # show the figure :D
```

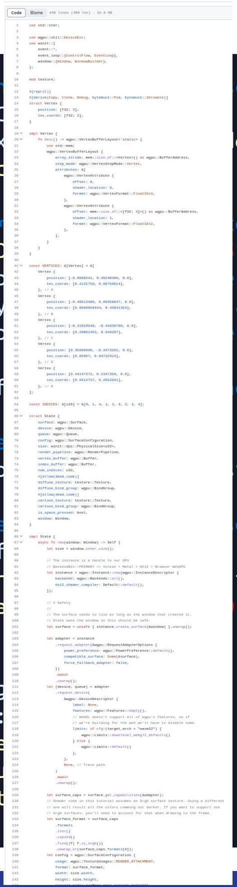


Core devs:

- Kushal Kolar
- Caitlin Lewis
- Almar Klein
- Amol Pasarkar

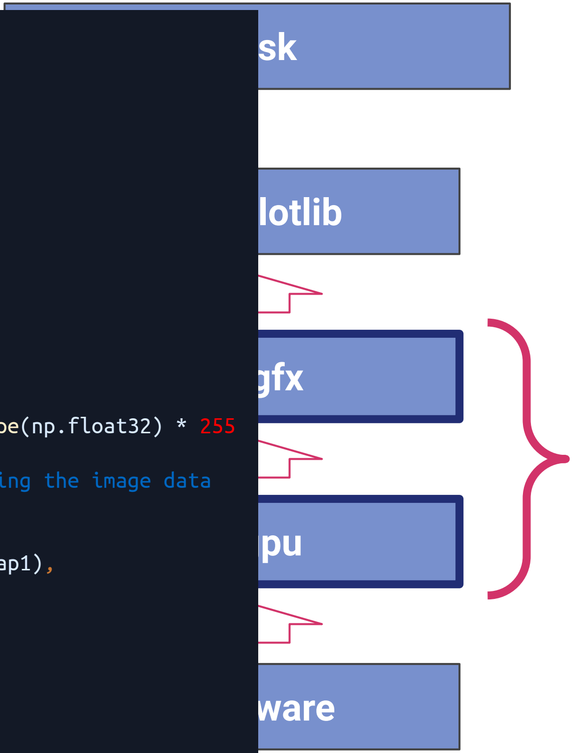
fastplotlib-pygfx-wgpu Stack

```
# create a car                 cene
canvas = WgpuC
renderer = gfx                 erer(canvas)
scene = gfx.Sc

# create a cam                 o center of image
camera = gfx.O                 12, 512)
camera.positio                 r of the scene
camera.scale.y                 axis
camera.positio                 r of the scene

colormap1 = gf                 e a colormap

# 512x512 arra
img_data = iio                 tronaut.png").astype(np.float32) * 255

# define Geome                 et as a Texture using the image data
image_obj = gf
    gfx.Geometr                img_data, dim=2)),
    gfx.ImageBa                , 255), map=colormap1),
)

scene.add(imag
def animate():
    renderer.re
    canvas.requ
canvas.request
canvas
```



Code Blame 446 lines (400 loc) · 15.9 KB

```rust
use std::iter;

use wgpu::util::DeviceExt;
use winit::{
    event::*,
    event_loop::{ControlFlow, EventLoop},
    window::{Window, WindowBuilder},
};

mod texture;

#[repr(C)]
#[derive(Copy, Clone, Debug, bytemuck::Pod, bytemuck::Zeroable)]
struct Vertex {
    position: [f32; 3],
    tex_coords: [f32; 2],
}

impl Vertex {
    fn desc() -> wgpu::VertexBufferLayout<'static> {
        use std::mem;
        wgpu::VertexBufferLayout {
            array_stride: mem::size_of::<Vertex>() as wgpu::BufferAddress,
            step_mode: wgpu::VertexStepMode::Vertex,
            attributes: &[
                wgpu::VertexAttribute {
                    offset: 0,
                    shader_location: 0,
                    format: wgpu::VertexFormat::Float32x3,
                },
                wgpu::VertexAttribute {
                    offset: mem::size_of::<[f32; 3]>() as wgpu::BufferAddress,
                    shader_location: 1,
                    format: wgpu::VertexFormat::Float32x2,
                },
            ],
        }
    }
}

const VERTICES: &[Vertex] = &[
    Vertex {
        position: [-0.0868241, 0.49240386, 0.0],
        tex_coords: [0.4131759, 0.00759614],
    }, // A
    Vertex {
        position: [-0.49513406, 0.06958647, 0.0],
        tex_coords: [0.0048659444, 0.43041354],
    }, // B
    Vertex {
        position: [-0.21918549, -0.44939706, 0.0],
        tex_coords: [0.28081453, 0.949397],
    }, // C
    Vertex {
        position: [0.35966998, -0.3473291, 0.0],
        tex_coords: [0.85967, 0.84732914],
    }, // D
    Vertex {
        position: [0.44147372, 0.2347359, 0.0],
        tex_coords: [0.9414737, 0.2652641],
    }, // E
];

const INDICES: &[u16] = &[0, 1, 4, 1, 2, 4, 2, 3, 4];

struct State {
    surface: wgpu::Surface,
    device: wgpu::Device,
    queue: wgpu::Queue,
    config: wgpu::SurfaceConfiguration,
    size: winit::dpi::PhysicalSize<u32>,
    render_pipeline: wgpu::RenderPipeline,
    vertex_buffer: wgpu::Buffer,
    index_buffer: wgpu::Buffer,
    num_indices: u32,
    #[allow(dead_code)]
    diffuse_texture: texture::Texture,
    diffuse_bind_group: wgpu::BindGroup,
    #[allow(dead_code)]
    cartoon_texture: texture::Texture,
    cartoon_bind_group: wgpu::BindGroup,
    is_space_pressed: bool,
    window: Window,
}

impl State {
    async fn new(window: Window) -> Self {
        let size = window.inner_size();

        // The instance is a handle to our GPU
        // BackendBit::PRIMARY => Vulkan + Metal + DX12 + Browser WebGPU
        let instance = wgpu::Instance::new(wgpu::InstanceDescriptor {
            backends: wgpu::Backends::all(),
            dx12_shader_compiler: Default::default(),
        });

        // # Safety
        //
        // The surface needs to live as long as the window that created it.
        // State owns the window so this should be safe.
        let surface = unsafe { instance.create_surface(&window) }.unwrap();

        let adapter = instance
            .request_adapter(&wgpu::RequestAdapterOptions {
                power_preference: wgpu::PowerPreference::default(),
                compatible_surface: Some(&surface),
                force_fallback_adapter: false,
            })
            .await
            .unwrap();
        let (device, queue) = adapter
            .request_device(
                &wgpu::DeviceDescriptor {
                    label: None,
                    features: wgpu::Features::empty(),
                    // WebGL doesn't support all of wgpu's features, so if
                    // we're building for the web we'll have to disable some.
                    limits: if cfg!(target_arch = "wasm32") {
                        wgpu::Limits::downlevel_webgl2_defaults()
                    } else {
                        wgpu::Limits::default()
                    },
                },
                None, // Trace path
            )
            .await
            .unwrap();

        let surface_caps = surface.get_capabilities(&adapter);
        // Shader code in this tutorial assumes an Srgb surface texture. Using a different
        // one will result all the colors comming out darker. If you want to support non
        // Srgb surfaces, you'll need to account for that when drawing to the frame.
        let surface_format = surface_caps
            .formats
            .iter()
            .copied()
            .find(|f| f.is_srgb())
            .unwrap_or(surface_caps.formats[0]);
        let config = wgpu::SurfaceConfiguration {
            usage: wgpu::TextureUsages::RENDER_ATTACHMENT,
            format: surface_format,
            width: size.width,
            height: size.height,
```



Core devs:

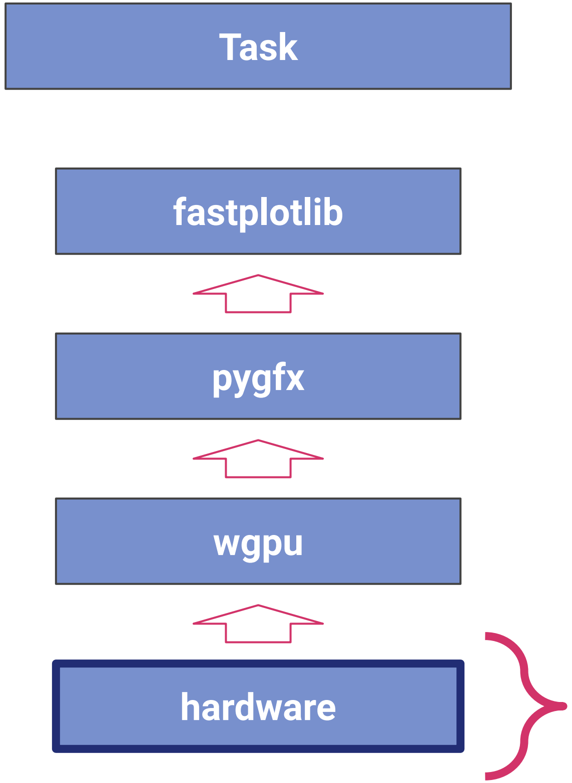
pygfx

- Almar Klein
- Korijn van Golen

gpu

fastplotlib-pygfx-wgpu Stack

Image



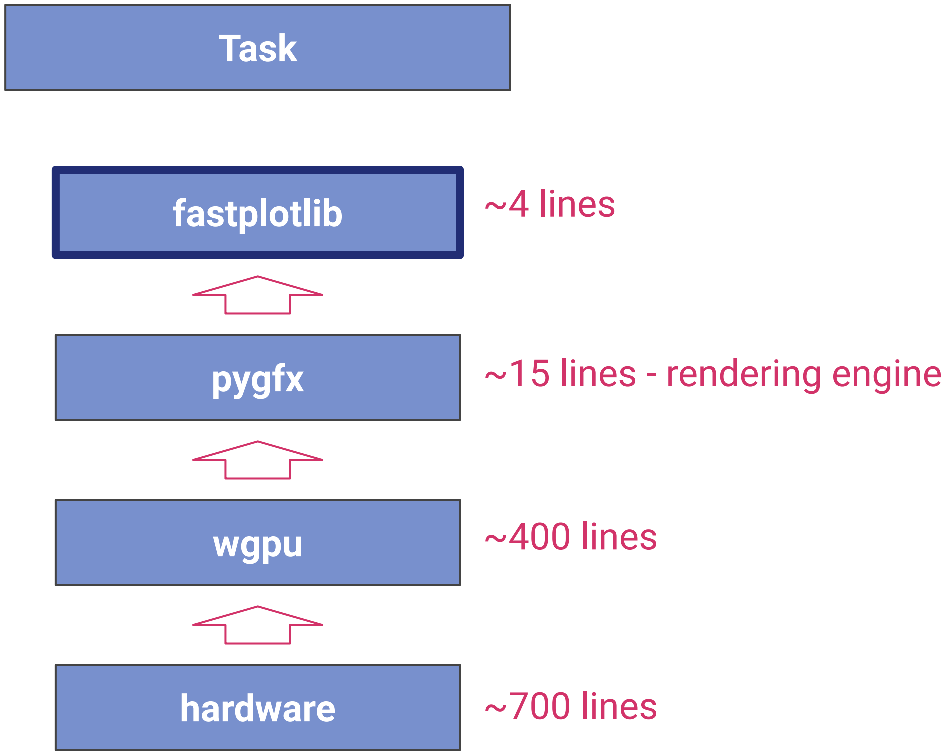
- Vulkan
- Metal (Mac)
- DX12 (Windows)

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

This is also what newer games use!

fastplotlib-pygfx-wgpu Stack

Image

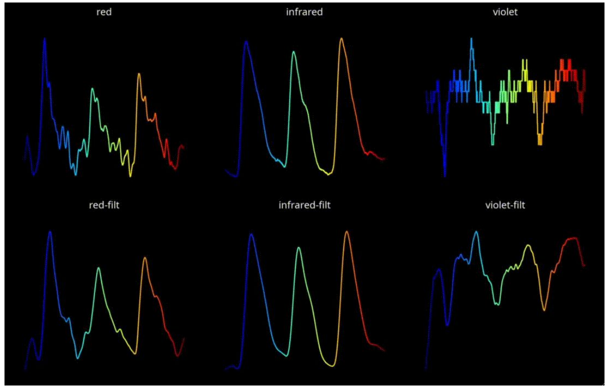


fastplotlib

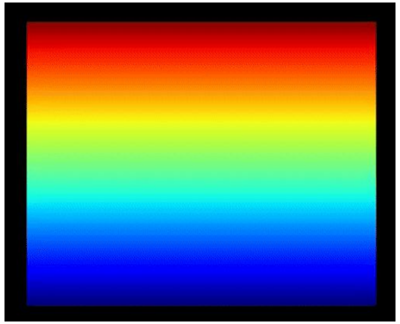
- High-level API for scientific plotting - inspiration from *pyqtgraph* and other libs
  - ❤️ pyqtgraph
- Uses the *pygfx* rendering engine
- Very new - 2 years old
- Goals: fast visualization, **expressive & elegant API -** we'll tell you what this means!
- Core developers & leadership:
  - **Kushal Kolar - Flatiron Institute/NYU**
  - **Caitlin Lewis - Duke University**
- Major developers:
  - **Almar Klein - Independent/funded by Flatiron Institute**
  - **Amol Pasarkar - Columbia University**

What can I do with fastplotlib?

- GPU accelerated visualization
  - Modern integrated graphics is sufficient for many use-cases!
- Rapid prototyping and algorithm design
  - Examples: matrix decompositions, time series exploration
  - Design, develop, evaluate and ship machine learning models
- Exploration and fast rendering of large-scale data
- Create real-time acquisition systems for instruments



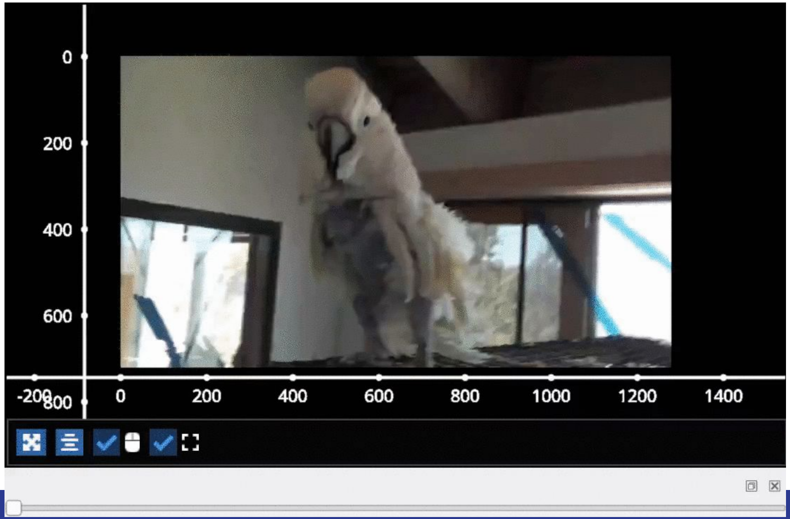
3 million points - fastplotlib
midrange 2017 GPU

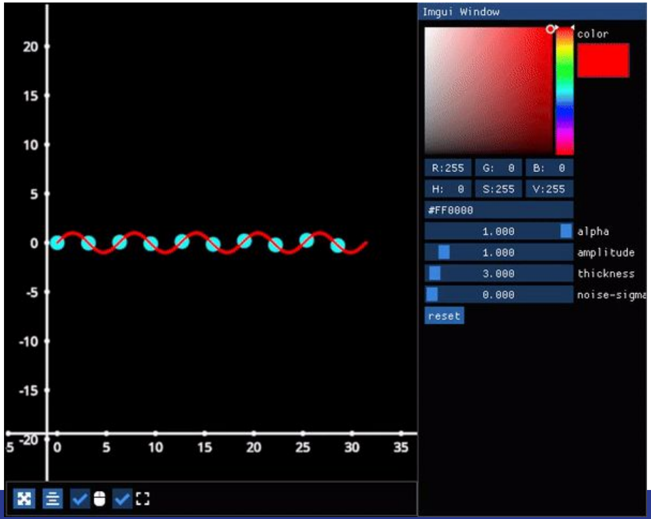


Where can I use fastplotlib?

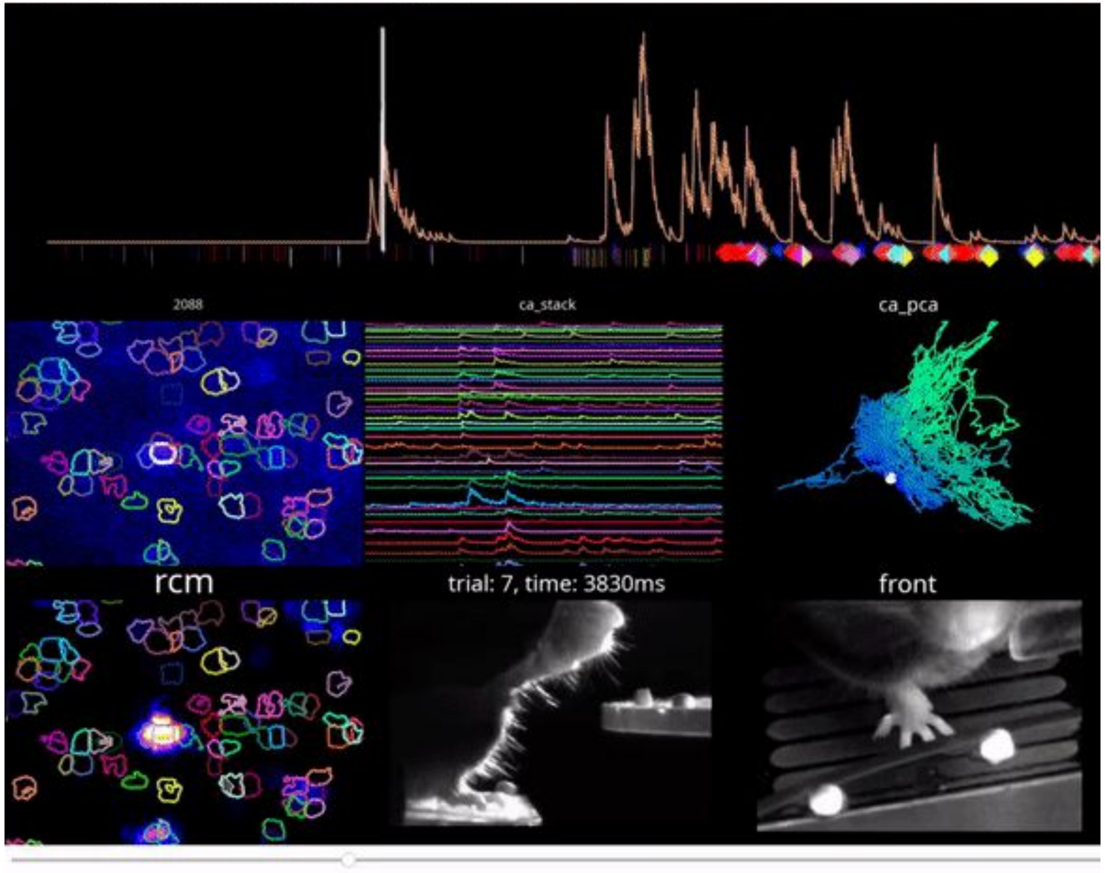
- Identical code across Qt, glfw, and jupyter lab. (and wx?)
  - **cloud computing, remote infrastructure**
  - Prototype in jupyter → ship Qt, glfw or web app!
- Imgui-integration (next release)

"Write once, run everywhere", ~~a.k.a. "Write once debug everywhere"~~

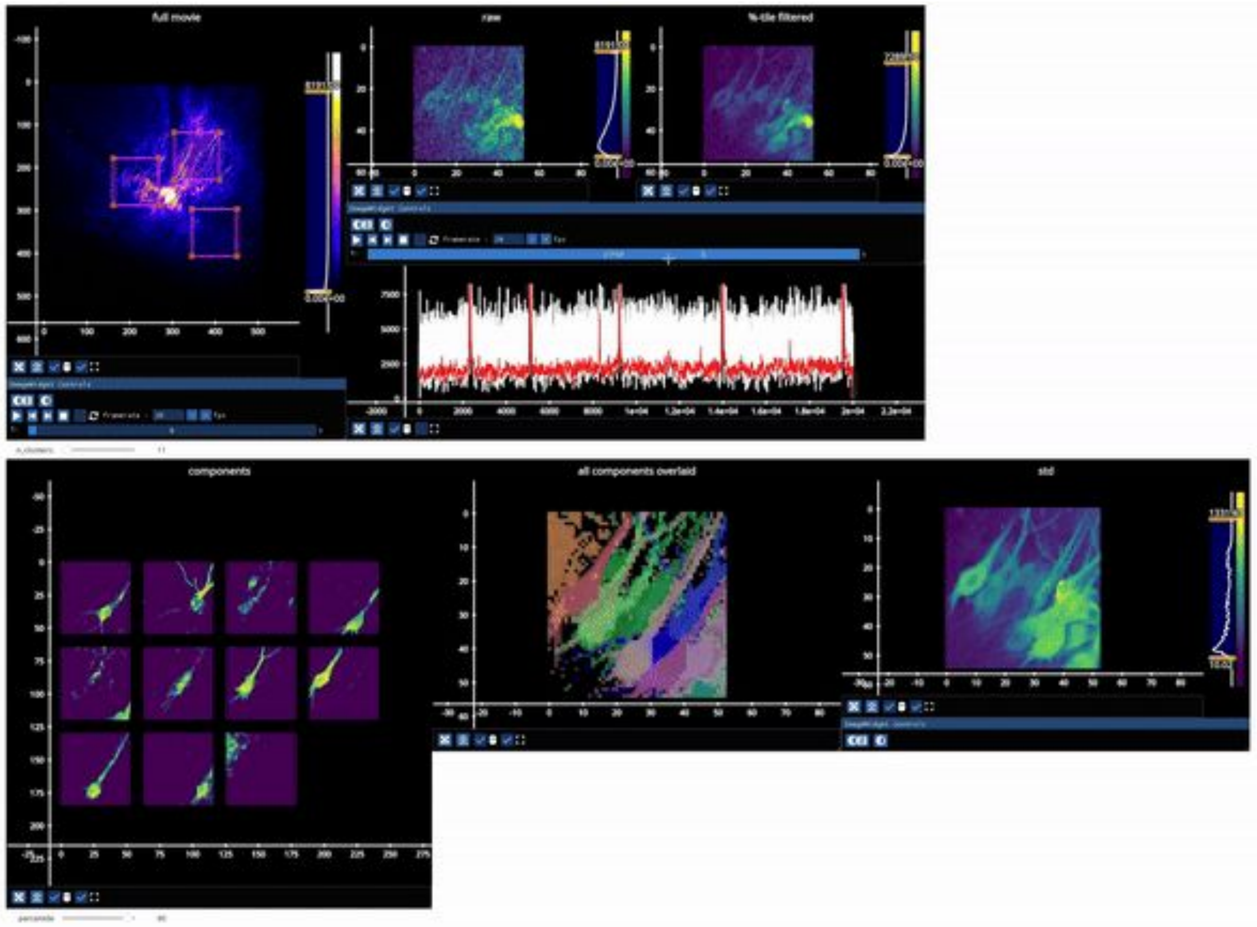




Exploration of large multi-modal experimental datasets

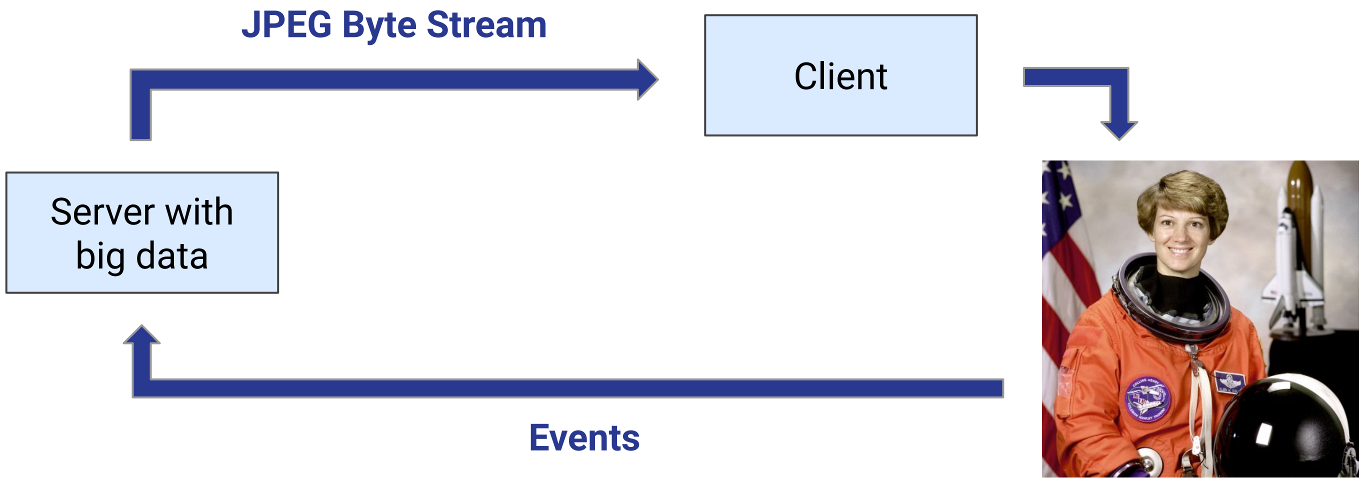


Matrix decomposition algorithm development

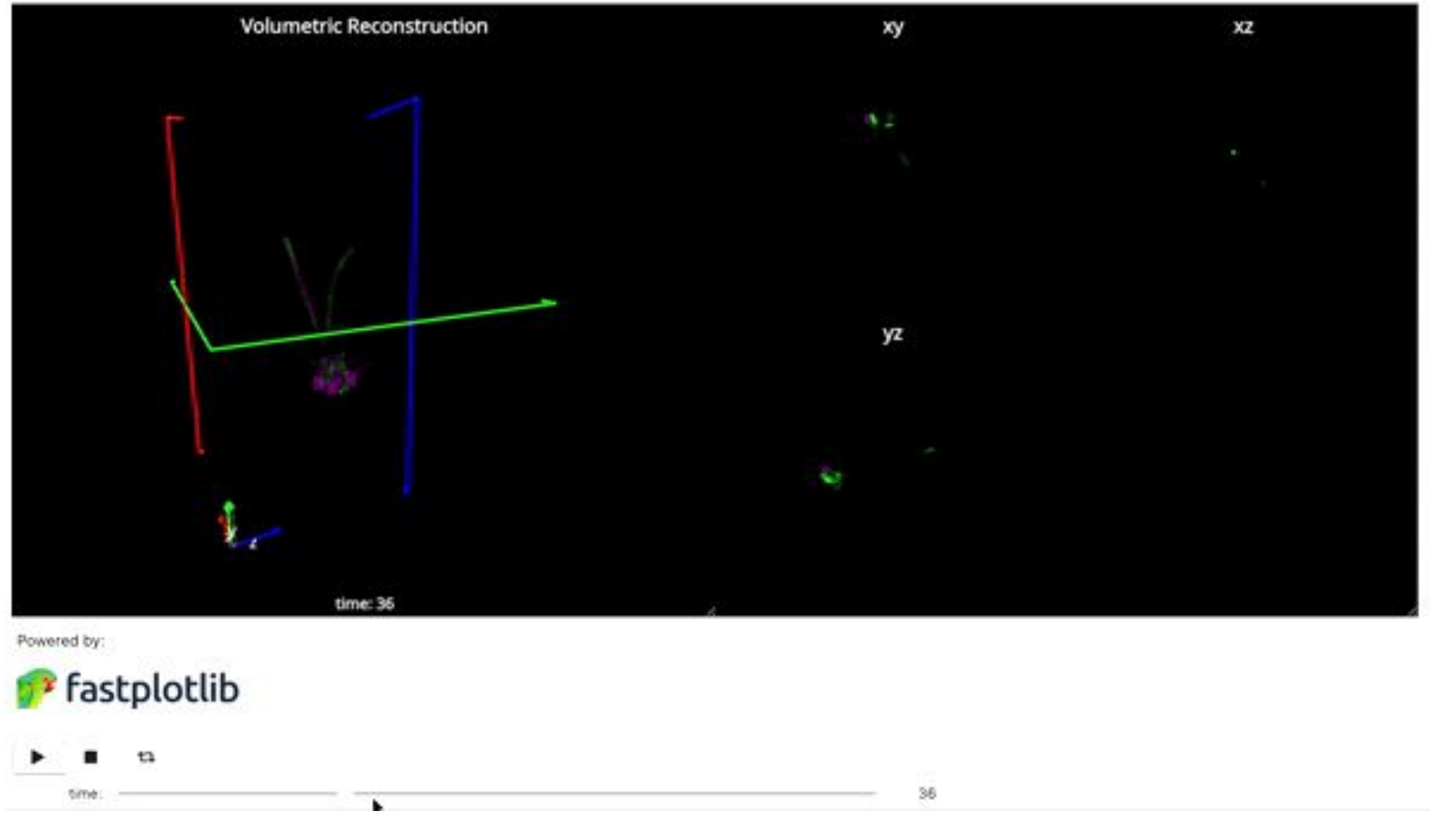


Remote rendering

- Server-side rendering, client only receives a jpeg byte stream
- Faster than client libs - bokeh, dash, plotly, etc.
  - Render big data on server/cloud, client only gets small jpeg stream!

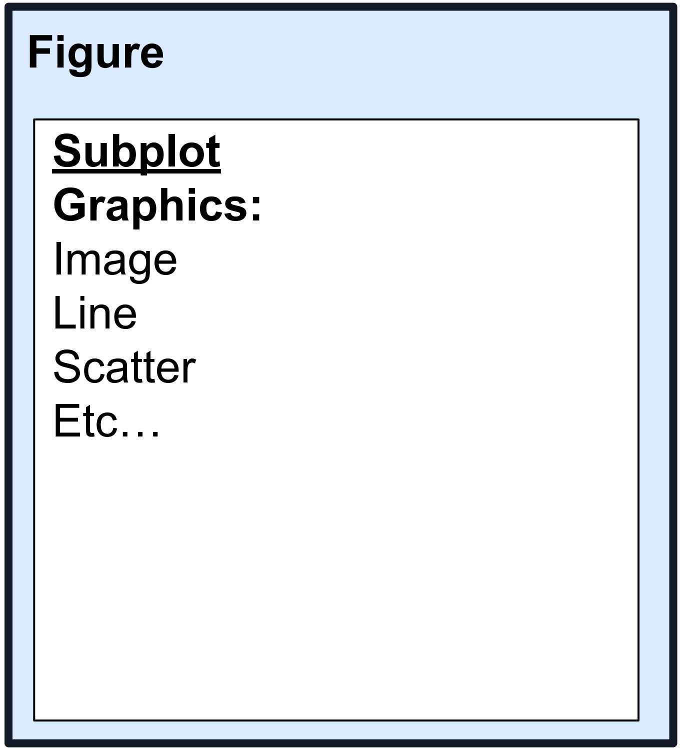


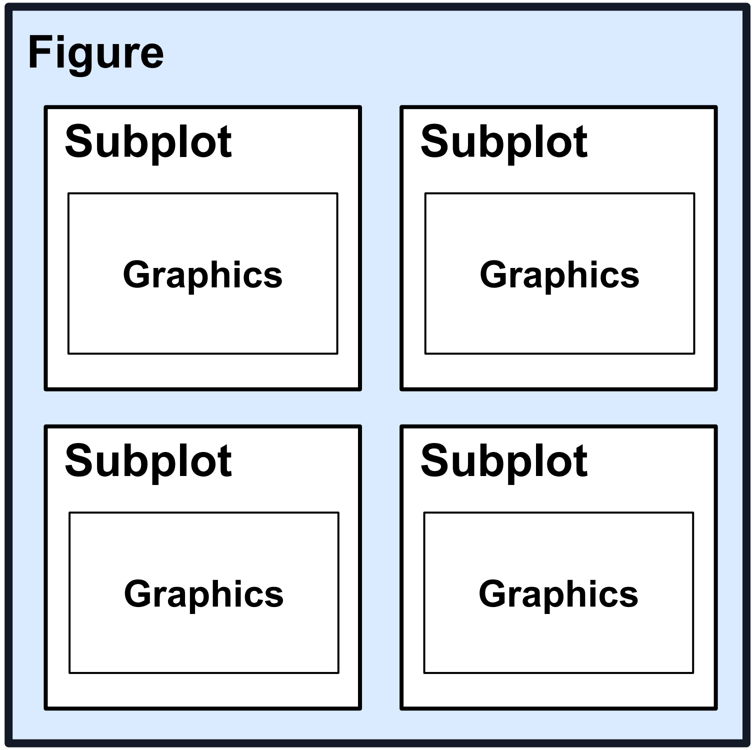
Reflective Fourier Light field Computed Tomography (ReFLeCT)



https://www.biorxiv.org/content/10.1101/2024.09.16.609432v1

API Walkthrough!





Demo!

Documentation & Examples

https://www.fastplotlib.org/ver/dev/

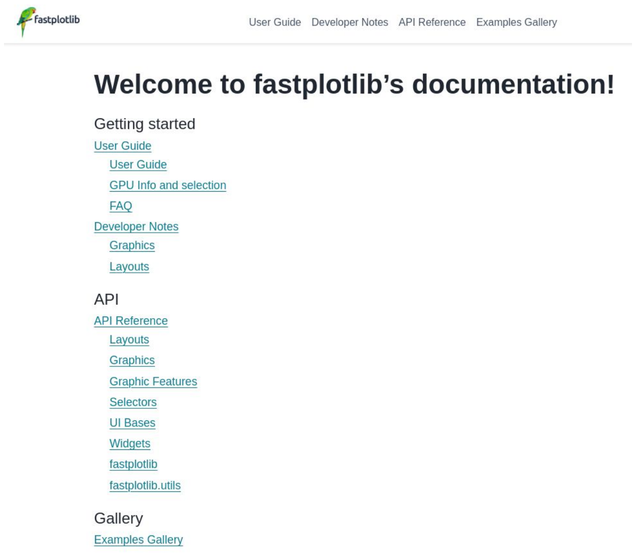
Includes a user guide and how-to on getting started using fastplotlib!

We also have an extensive examples gallery that we are always adding to :D

- We would love to add more examples for your use-case

We are also always happy to help you with visualizing your data!!

- Feel free to reach out on our GitHub via an issue or discussion post
- You can also come find me downstairs in John's lab :)



Welcome to fastplotlib's documentation!

Getting started

- User Guide
  - User Guide
  - GPU Info and selection
  - FAQ
- Developer Notes
  - Graphics
  - Layouts

API

- API Reference
  - Layouts
  - Graphics
  - Graphic Features
  - Selectors
  - UI Bases
  - Widgets
  - fastplotlib
  - fastplotlib.utils

Gallery

- Examples Gallery

Thank you!

What fastplotlib is not

- Fastplotlib is NOT related to matplotlib in any way!
    - Different use cases, different APIs!
    - Fastplotlib is not aimed towards the creation of static publication figures
- Fastplotlib does not handle data loading
    - Numpy-like data arrays which support **memoryview()** should work
- Fastplotlib is a plotting library, not a viewer, GUI, or application
    - You can use it to build viewers and GUI applications