DearCyGui

A fast and Python-centric GUI Framework

Building Interactive Image Processing Applications

By Axel Davy

Why Do We Need a GUI?

Problement statement

Workflow 1:

- Running portions of code in notebook/console with specific inputs
- visualize the data (via print, vpv, matplotlib, etc)

Back and forth needed !

Workflow 2:

- Looking at a lot of images with vpv
- Relaunch vpv after having run an algorithm that couldn't be expressed in plambda

I only we had embedded Python

Workflow 3:

- You use vpv and you wish you could draw without using an svg and interact with the data

Vpv plugin are useful but that's not Python

DearCyGui is a home-made library written in *Cython*.

It can:

- Make UIs (buttons, sliders, etc)
- Draw plots
- Add interactable items to the plot
- Enable to create custom objects
- Show data from any other python library

A lot of effort has been spent for:

- Good documentation
- Typing, Autocompletion
- Performance

Available GUI Solutions

Traditional GUI Frameworks

Qt / GTK / wxWidgets:

+ Mature and feature-rich

- Complex Python bindings

- Not designed for scientific computing

DearCygui

+ Developped by someone you can talk to

- Not mature

+ Well integrated with Python

+ Developped for interactive data

Scientific Visualization

Matplotlib / Plotly / VTK:

+ Great for static visualization

+ Scientific computing focus

- Limited interactive capabilities

- Not suited for full applications

DearCygui

Can show results from Matplotlib, etc

+ Has core tools for data visualization

+ Developed for me and for you

+ Interactive: Python callbacks

+ Suited for a pro application

Understanding RMGUI vs IMGUI

Traditional Retained Mode GUI (RMGUI):

- Your code create items with the library.
- The library handles item rendering and management.
- For complex interactions, a lot of boilerplate is needed.

DearCygui provides a *Retained Mode* API,

but uses *Immediate Mode* underneath.

Immediate Mode GUI (IMGUI):

- Your code manages itself its items
- EVERY frame, you call the library to render your items
- The library is optimized to render as fast as possible
- The library may be slower in cases where having a precomputed result would be useful.

Your code:

- Create items
- Handle events
- Take action

Library:

- Render items
- Trigger events
- Fill items states

Your code:

- Create items
- Check events
- Render items

Library (Toolbox):

- check events
- render items
- organize items

Array Operations Benchmark

```
import numpy as np
import time

# Regular Python
def fill_array_python(size):
    arr = np.zeros(size)
    for i in range(size):
        arr[i] = i
    return arr

t1 = time.time()
fill_array_python(1000000)
t2 = time.time()
print(f"Python: {1000*(t2-t1):.2f}ms")

t1 = time.time()
np.arange(1000000)
t2 = time.time()
print(f"Numpy: {1000*(t2-t1):.2f}ms")
```

Run Success

Some Python IMGUI Libraries exist:

- Direct C++ API mapping - error prone
- Unsafe API calls with limited type checking
- High Python overhead from frequent calls

- Python is a scripting API. Not adapted for many calls

  Python: 541.65ms
  Numpy: 0.87ms
  Numpy (but Python objects): 139.28ms

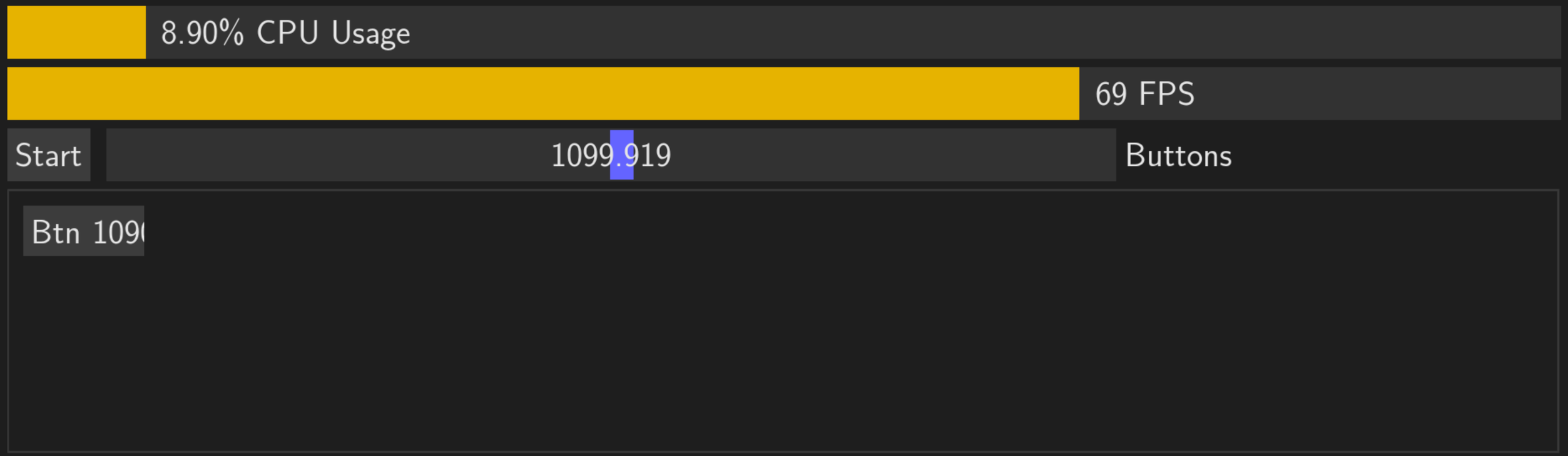
DearCyGui is Cython (C++) and Python friendly

- Type-safe Python API
- Optimized Cython implementation
- Better error handling and debugging

Can it be fast?

This laptop, on battery, can:

- Render hello world at 2441 fps
- Render 20000 buttons at 55 fps
- Render 20000 anti-aliased lines at 134 fps
- Create and configure a simple button in Python at 77K fps
- Basically you can create and configure 1000-10000 items per frame
- Does not use CPU/GPU if content doesn't need to change.



Building a minimal application

1. Create a Context
2. Display an Image
3. Add Controls
4. Handle Events

Denoise 0.500 Strength


```
import dearcygui as dcg
import numpy as np

C = dcg.Context()
C.viewport.initialize()
with dcg.Window(C, primary=True):
    with dcg.Plot(C, width=-1, height=-1):
        with dcg.DrawInPlot(C):
            image = imageio.imread("lenapashm
            texture = dcg.Texture(C, image)
            dcg.DrawImage(C,
                          texture=texture,
                          pmin=(0, 1),
                          pmax=(1, 0))
while C.running:
    C.viewport.render_frame()
```

OpenCV Integration:

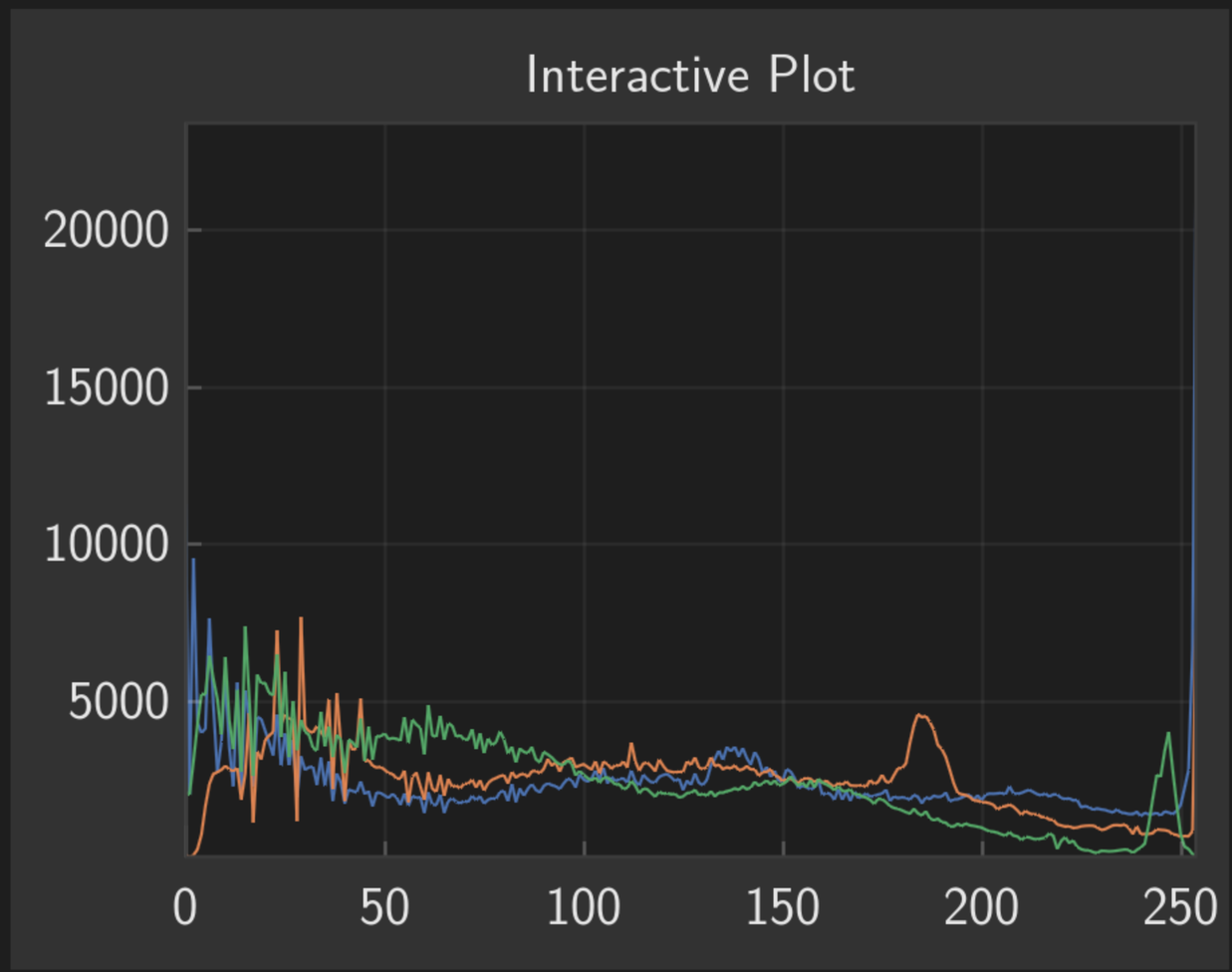
```
# Edge detection with OpenCV
import numpy as np
import cv2
import imageio

# Load and convert image to grayscale
image = imageio.imread("lenapashm.png")
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY

# Apply Canny edge detection
edges = cv2.Canny(gray, 100, 200)

# Convert back to RGB for display
edges_rgb = cv2.cvtColor(edges, cv2.COLOR_GRA

return edges_rgb
```

Run Success

Image: [800, 800, 3]

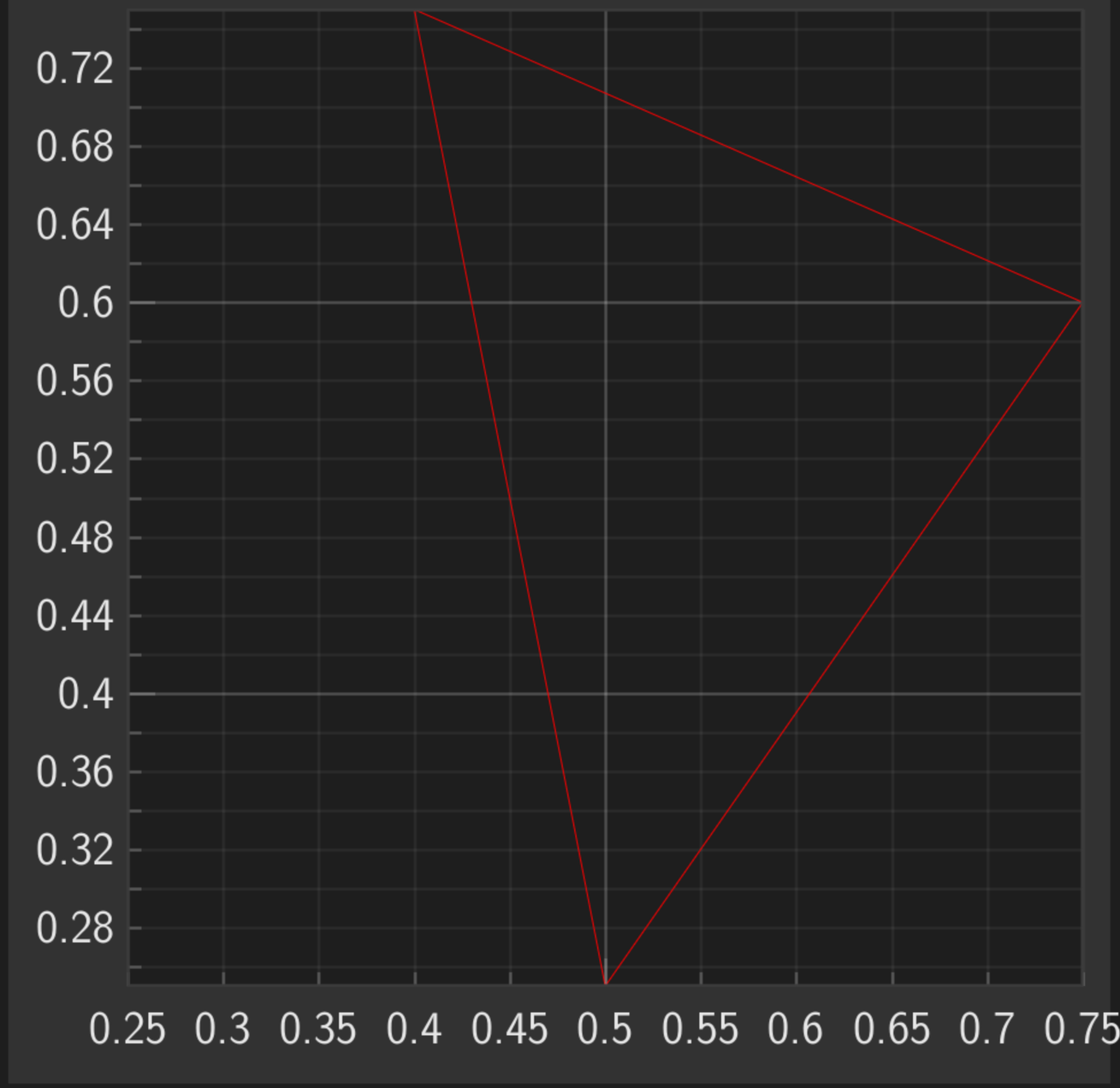
Color Channel Analysis:

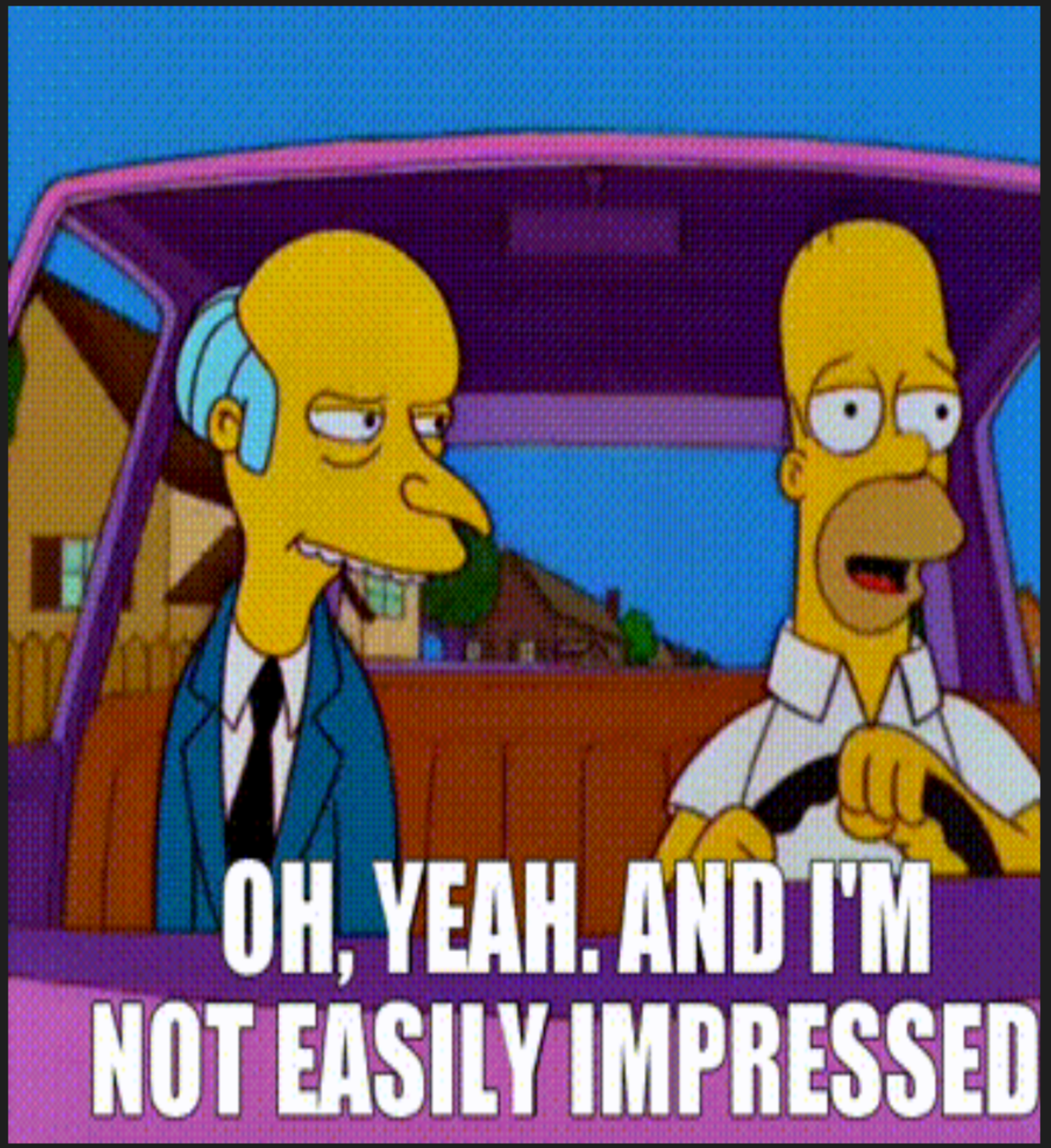
```
# Plot RGB channel histograms
import numpy as np
import imageio

# Load image
image = imageio.imread("lenapashm.png")

# Calculate histograms for each channel
bins = np.arange(256)
r_hist = np.histogram(image[:,:,0], bins=bins
g_hist = np.histogram(image[:,:,1], bins=bins
b_hist = np.histogram(image[:,:,2], bins=bins

# Return data for plotting
x = bins[:-1]  # Bin centers
return x, (r_hist, g_hist, b_hist)
```



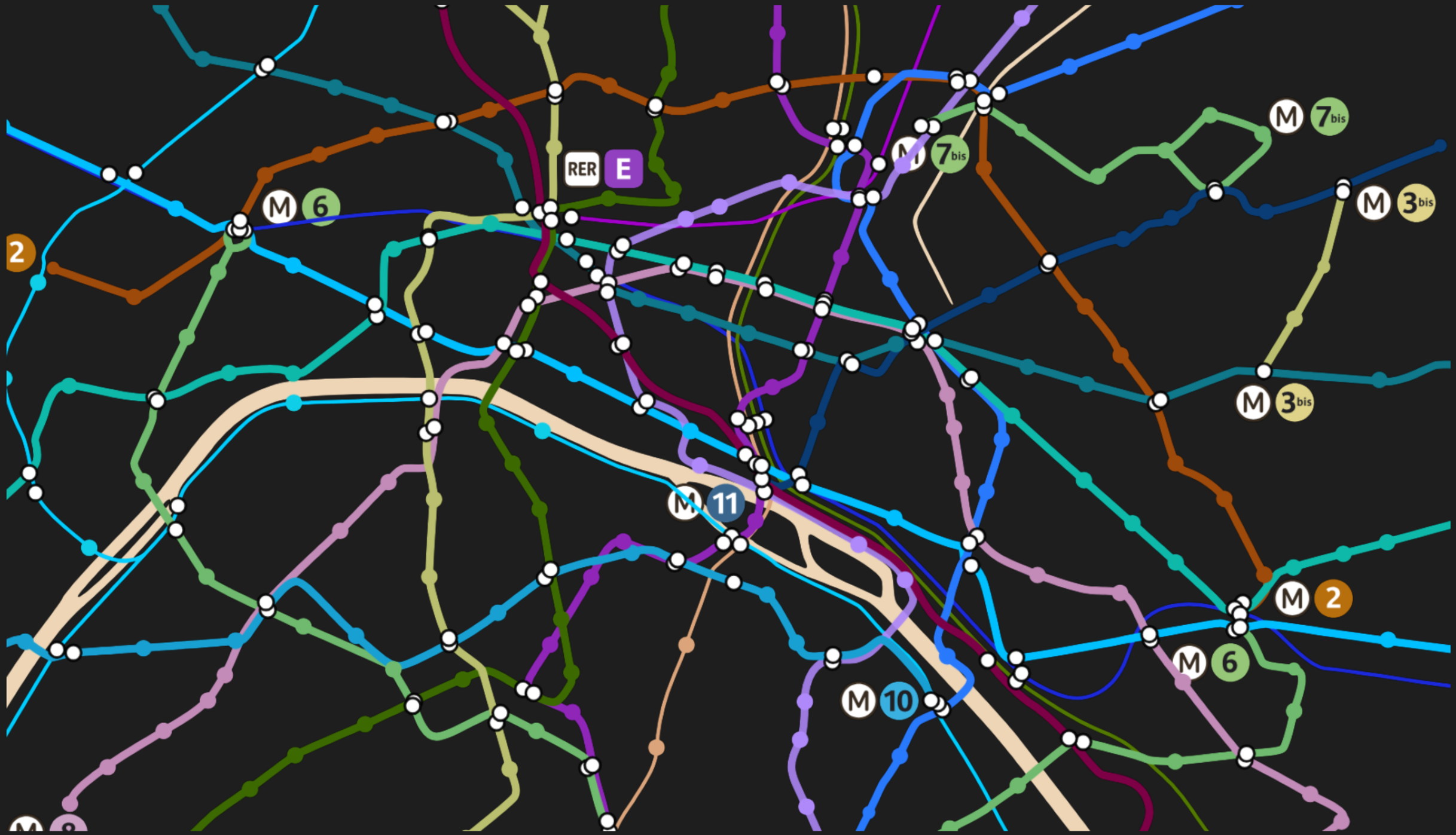
Run Success

Example of interactable item





Interactive maps ?



Learn More:

Documentation: github.com/DearCyGui/DearCyGui/docs

GitHub Repository: github.com/DearCyGui/DearCyGui

Demos: github.com/DearCyGui/Demos

Thank you!