Computer Graphics

Assignment 1: Exploration of Graphics Libraries
By Romrawin Chumpu 639506093 (Jinpu)

Notebook: [CG] Assignment 1 - Python Graphics Library.ipynb - Colaboratory

For the graphical illustration, there are two typical perspectives.

1. Two dimensional (2D) graphics - PyCairo
2. Three dimensional (3D) graphics - Numpy and Matplotlib, PyTorch3D

Table of Content

PyCairo

Source documentation: https://pycairo.readthedocs.io/

PyCairo is the *python 2D graphics* library implemented from the Cairo graphics library (http://cairographics.org) in C language.

Install PyCairo in Colab

```
!apt install libcairo2-dev libgif-dev
!pip install pycairo
```

PyCairo Code Examples

Import cairo library and IPython display tools

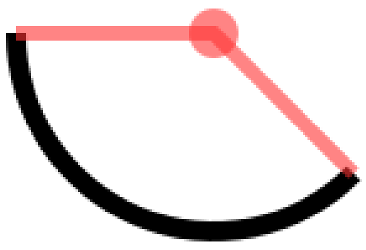
```
# https://github.com/pygobject/pycairo/blob/master/examples/pycairo_examples.ipynb
import cairo
from IPython.display import Image, display
```

```python
from io import BytesIO

def disp(draw_func):
    surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, 200, 200)
    ctx = cairo.Context(surface)
    draw_func(ctx, 200, 200)
    with BytesIO() as fileobj:
        surface.write_to_png(fileobj)
        display(Image(fileobj.getvalue(), width=200))
```

Draw 2D arch

```python
from math import pi
@disp
def draw(cr, width, height):
    cr.scale(width, height)
    cr.set_line_width(0.04)
    xc = 0.5
    yc = 0.5
    radius = 0.4
    angle1 = 45.0 * (pi / 180.0)  # angles are specified
    angle2 = 180.0 * (pi / 180.0)  # in radians

    cr.arc(xc, yc, radius, angle1, angle2)
    cr.stroke()

    # draw helping lines
    cr.set_source_rgba(1, 0.2, 0.2, 0.6)
    cr.arc(xc, yc, 0.05, 0, 2 * pi)
    cr.fill()
    cr.set_line_width(0.03)
    cr.arc(xc, yc, radius, angle1, angle1)
    cr.line_to(xc, yc)
    cr.arc(xc, yc, radius, angle2, angle2)
    cr.line_to(xc, yc)
    cr.stroke()
```

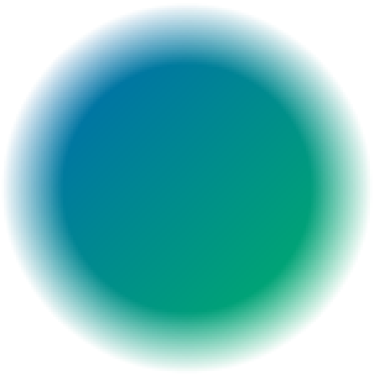
Output:



Draw gradient circle

```
@disp
def draw(cr, width, height):
  cr.scale(width, height)
  cr.set_line_width(0.04)

  pattern = cairo.LinearGradient(0, 0, 1, 1)
  pattern.add_color_stop_rgb(0, 0, 0.3, 0.8)
  pattern.add_color_stop_rgb(1, 0, 0.8, 0.3)

  mask = cairo.RadialGradient(0.5, 0.5, 0.25, 0.5, 0.5, 0.5)
  mask.add_color_stop_rgba(0, 0, 0, 0, 1)
  mask.add_color_stop_rgba(0.5, 0, 0, 0, 0)

  cr.set_source(pattern)
  cr.mask(mask)
```

Output:



NumPy and Matplotlib

NumPy is a python library for large high-dimensional mathematical operations. Matplotlib is a visualization library for python programming.

NumPy and Matplotlib for 3D illustration source:
https://www.geeksforgeeks.org/displaying-3d-images-in-python/

Since numpy and matplotlib are pre installed in google colab. So, we can import the library straightway.

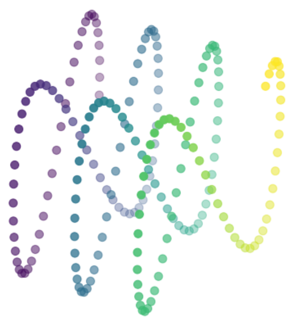
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

NumPy and Matplotlib Code Examples

Draw 3D Scatter

```
# Define and set figure size
fig = plt.figure(figsize=(10, 10))

# Define 3D axes to "ax"
ax = plt.axes(projection='3d')

# Create the point of values using numpy
x = np.arange(0, 20, 0.1)
y = np.cos(x)
z = y*np.sin(x)
c = x + y

# Plot the points from previous calculation as 3D scatter plot
ax.scatter(x, y, z, c=c, s=100)

# Turn off axes
plt.axis('off')

# Show the plot
plt.show()
```

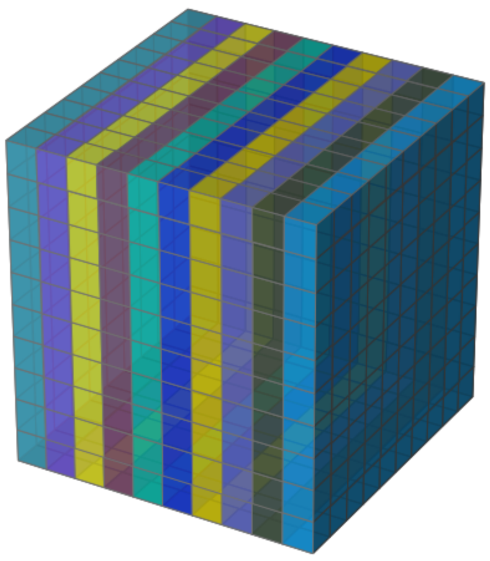
Output:



Draw voxels

```
# Define and set figure size
fig = plt.figure(figsize=(10, 10))

# Generate a 3D sine wave
ax = plt.axes(projection='3d')

# Generate axes of 10x10x10
axes = [10, 10, 10]

# Create data
data = np.ones(axes)

# Control transparency
alpha = 0.8

# set colour
colors = np.empty(axes + [4])

# Set color in each row of voxels
for i in range(10):
  r, g, b = np.random.uniform(), np.random.uniform(), np.random.uniform()
  colors[i] = [r, g, b, alpha]

# Turn off axes
plt.axis('off')

# Create voxels representation
ax.voxels(data, facecolors=colors, edgecolors='grey')
plt.show()
```

Output:

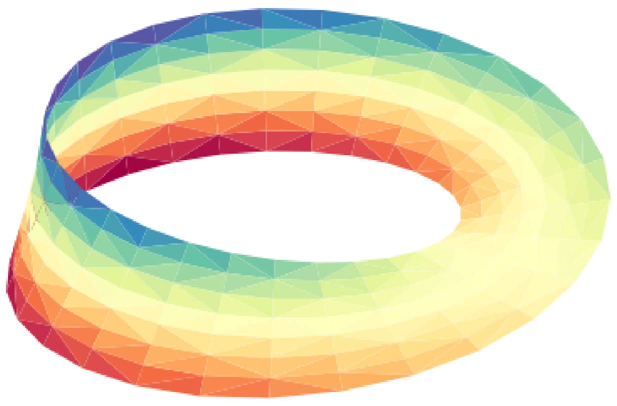


Draw Möbius strip

Source: Three-Dimensional Plotting in Matplotlib | Python Data Science Handbook (jakevdp.github.io)

```
# Define parameters
theta = np.linspace(0, 2 * np.pi, 30)
w = np.linspace(-0.3, 0.3, 8)
w, theta = np.meshgrid(w, theta)
phi = 0.5 * theta

# radius in x-y plane
r = 1 + w * np.cos(phi)

x = np.ravel(r * np.cos(theta))
y = np.ravel(r * np.sin(theta))
z = np.ravel(w * np.sin(phi))

# triangulate in the underlying parametrization
from matplotlib.tri import Triangulation
tri = Triangulation(np.ravel(w), np.ravel(theta))

ax = plt.axes(projection='3d')
ax.plot_trisurf(x, y, z, triangles=trig.triangles,
                cmap='Spectral', linewidths=0.5);

ax.set_xlim(-1, 1); ax.set_ylim(-1, 1); ax.set_zlim(-1, 1);
plt.axis("off")
plt.show()
```

Output:



PyTorch3D

PyTorch3D is a python package based on PyTorch for deep learning with 3Dl data. We will go though only the renderer of 3D objects.

Source: [PyTorch3D · A library for deep learning with 3D data]()

Install PyTorch3D in Google Colab

```
import os
import sys
import torch
need_pytorch3d=False
try:
    import pytorch3d
except ModuleNotFoundError:
    need_pytorch3d=True
if need_pytorch3d:
    if torch.__version__.startswith("1.9") and sys.platform.startswith("linux"):
        # We try to install PyTorch3D via a released wheel.
        version_str="".join([
            f"py3{sys.version_info.minor}_cu",
            torch.version.cuda.replace(".",""),
            f"_pyt{torch.__version__[0:5:2]}"
        ])
        !pip install pytorch3d -f
https://dl.fbaipublicfiles.com/pytorch3d/packaging/wheels/{version_str}/download.html
    else:
        # We try to install PyTorch3D from source.
        !curl -LO https://github.com/NVIDIA/cub/archive/1.10.0.tar.gz
        !tar xzf 1.10.0.tar.gz
        os.environ["CUB_HOME"] = os.getcwd() + "/cub-1.10.0"
        !pip install 'git+https://github.com/facebookresearch/pytorch3d.git@stable'
```

Import libraries

```
import os
import torch
import matplotlib.pyplot as plt

# Util function for loading meshes
from pytorch3d.io import load_objs_as_meshes, load_obj

# Data structures and functions for rendering
from pytorch3d.structures import Meshes
from pytorch3d.vis.plotly_vis import AxisArgs, plot_batch_individually, plot_scene
from pytorch3d.vis.texture_vis import texturesuv_image_matplotlib
from pytorch3d.renderer import (
    look_at_view_transform,
    FoVPerspectiveCameras,
    PointLights,
    DirectionalLights,
    Materials,
    RasterizationSettings,
    MeshRenderer,
    MeshRasterizer,
    SoftPhongShader,
    TexturesUV,
    TexturesVertex
)

# add path for demo utils functions
import sys
import os
sys.path.append(os.path.abspath(''))
```

Download 3D files

```
!mkdir -p data/cow_mesh
!wget -P data/cow_mesh https://dl.fbaipublicfiles.com/pytorch3d/data/cow_mesh/cow.obj
!wget -P data/cow_mesh https://dl.fbaipublicfiles.com/pytorch3d/data/cow_mesh/cow.mtl
!wget -P data/cow_mesh
https://dl.fbaipublicfiles.com/pytorch3d/data/cow_mesh/cow_texture.png
```

Load 3D .obj file

```
# Setup
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    torch.cuda.set_device(device)
else:
    device = torch.device("cpu")

# Set paths
DATA_DIR = "./data"
obj_filename = os.path.join(DATA_DIR, "cow_mesh/cow.obj")

# Load obj file
mesh = load_objs_as_meshes([obj_filename], device=device)
```

```
# Set batch size - this is the number of different viewpoints from which we want to
render the mesh.
batch_size = 20

# Create a batch of meshes by repeating the cow mesh and associated textures.
# Meshes has a useful `extend` method which allows us do this very easily.
# This also extends the textures.
meshes = mesh.extend(batch_size)

# Get a batch of viewing angles.
elev = torch.linspace(0, 180, batch_size)
azim = torch.linspace(-180, 180, batch_size)

# All the cameras helper methods support mixed type inputs and broadcasting. So we can
# view the camera from the same distance and specify dist=2.7 as a float,
# and then specify elevation and azimuth angles for each viewpoint as tensors.
R, T = look_at_view_transform(dist=2.7, elev=elev, azim=azim)
cameras = FoVPerspectiveCameras(device=device, R=R, T=T)

# Move the light back in front of the cow which is facing the -z direction.
lights.location = torch.tensor([[0.0, 0.0, -3.0]], device=device)
```

```
# We can pass arbitrary keyword arguments to the rasterizer/shader via the renderer
# so the renderer does not need to be reinitialized if any of the settings change.
images = renderer(meshes, cameras=cameras, lights=lights)
```

Plot 3D Object

```
image_grid(images.cpu().numpy(), rows=4, cols=5, rgb=True)
```