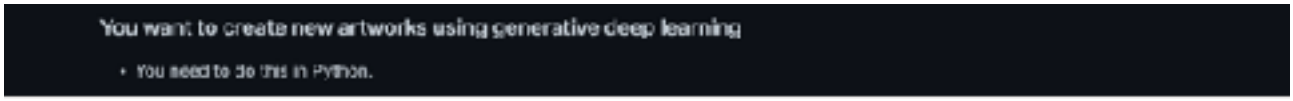
Coding two – Final work

Pinsi Wang

For coding 2's final work, there has a lot of options for us, and I decided to choose 'create new artworks using generative deep learning.'.



And I create two art works in python, here are the statements.

Art work 1:

Coding part:

Firstly, let's import everything we need.

```
In [5]: import matplotlib.pyplot as plt
        import random
        import math
        from semiln import GenerativeImage, Projection
```

Secondly, let's define f1 and f2.
We can play with it, different parameter will create different image.

```
In [13]: def f1(x, y):
             result = random.uniform(-1,1) * x**3 - math.cos(y**2) + abs(y-x)
             return result

         def f2(x, y):
             result = random.uniform(-1,1) * y**3 - math.cos(x**2) + 2*x
             return result

         g = GenerativeImage(f1, f2)
         g.generate()
         g.seed
Out[13]: 1033700
```

Let's see the image.

```
In [16]: g.plot(projection= Projection.POLAR, color = 'black',bgcolor = 'white')
```

Projection

We can use the projection attribute to define the coordinate system to transform our functions

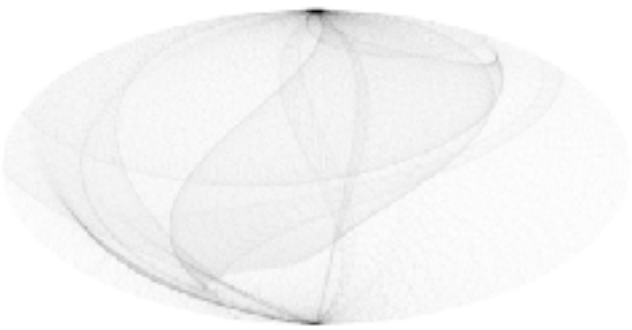
The avaliable projections are

RECTILINEAR, POLAR, AITOFF, HAMMER, LAMBERT, MOLLWEIDE and RANDOM

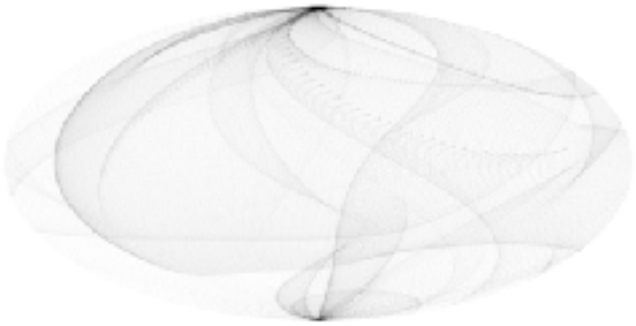
Let's see other projections:

RECTILINEAR:

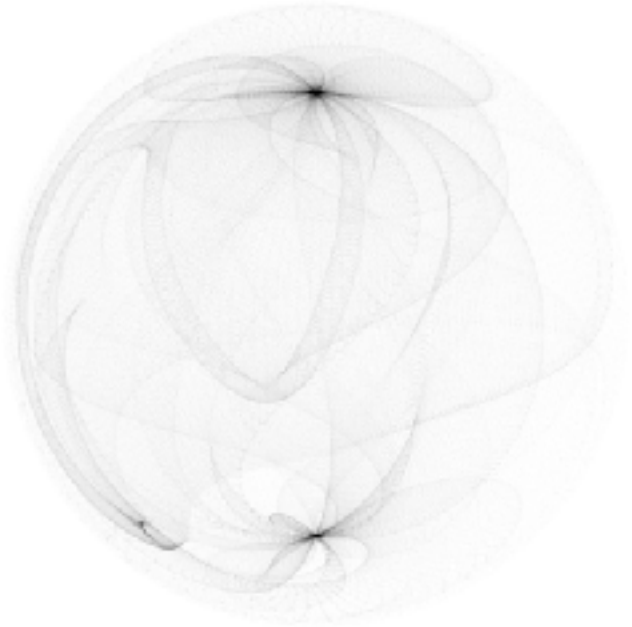
AITOFF:



HAMMER:



LAMBERT:



MOLLWEIDE:

RANDOM:

And let’s change cos to sin and see what will happen.

Art work 2:

For art work 2, I want to create a generative painting that can make audience play with it. I will make this painting interactive.

Coding part:

Import everything.

```
In [34]: import numpy as np
         from scipy.spatial import Voronoi, voronoi_plot_2d

         import matplotlib.pyplot as plt
         from matplotlib.collections import LineCollection

         from ipywidgets import widgets
         from ipywidgets import interactive
```

Create a canvas.

```
In [2]: x_bounds = np.array([0, 13])
        y_bounds = np.array([0, 16])

        x_buffer, y_buffer = 1, 1

        x_plot = x_bounds + np.array([x_buffer, -x_buffer])
        y_plot = y_bounds + np.array([y_buffer, -y_buffer])
```

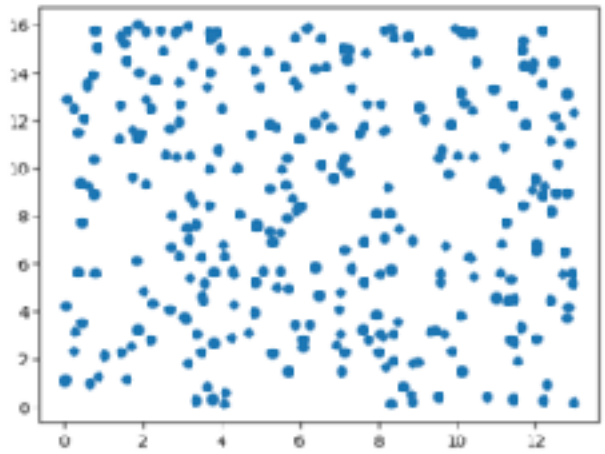
Voronoi diagram contains points and gives polygons. Let's show all the points first.

```
In [3]: num_points = 300

In [4]: x = np.random.uniform(*x_bounds, size=num_points).reshape((num_points, 1))
        y = np.random.uniform(*y_bounds, size=num_points).reshape((num_points, 1))

        pts = np.hstack([x, y])

In [5]: plt.scatter(*pts.transpose())
Out[5]: <matplotlib.collections.PathCollection at 0x11d0460>
```



This is a list of all the points that are the verticies in are diagram.

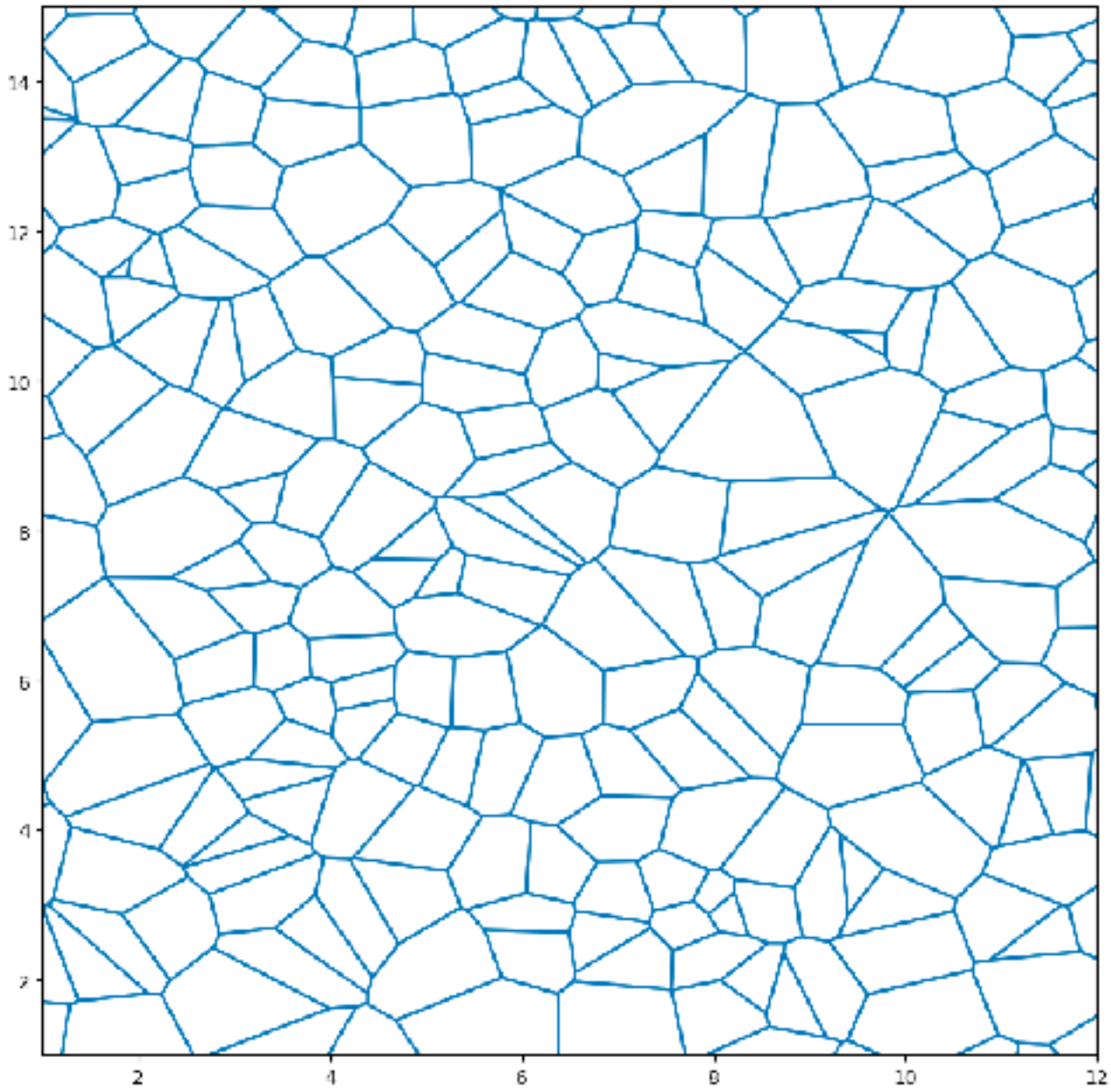
```
In [7]: verts[0:5, :]
Out[7]: array([[ 1.35189576e+01,  5.64628192e-03],
               [-9.19505731e+00,  1.01071223e-01],
               [ 9.18456591e-01,  1.16501876e-01],
               [ 1.23302513e+01,  1.51206584e-01],
               [ 1.32462114e+01,  8.61204433e-02]])
```

Filtering out any empty shapes (len(s) == 0)
Filtering out any shapes that go out of bounds (then it has an index of –1)
Closing the polygon by adding the last point back (so [1,2,3]–>[1,2,3,1])

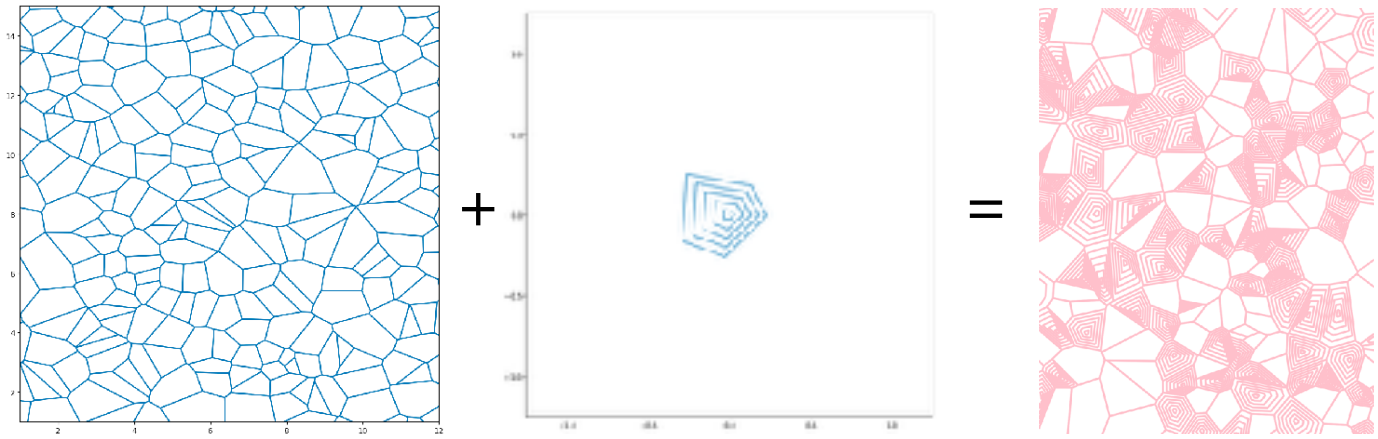
```
In [10]: shapes_ind = [s+s[0:1] for s in shapes_ind if len(s)>0 and -1 not in s]
         shapes = [verts[s] for s in shapes_ind]

In [11]: fig, ax = plt.subplots(figsize=(10,10))
         ax.set_xlim(*x_plot)
         ax.set_ylim(*y_plot)
         lc = PolyCollection(shapes)
         ax.add_collection(lc)
```

Then we can get the diagram.



And what I want to do is:



Let's add interactivity.

```
In [50]: def make_some_art(num_points=200, percent_to_fill = 0.5, n_fill_lines=5, min_scalar = 0.1,  debug=False, toggle_for_ne

    x = np.random.uniform(*x_bounds, size=num_points).reshape(num_points, 1))
    y = np.random.uniform(*y_bounds, size=num_points).reshape(num_points, 1))
    pts = np.hstack([x, y])

    vor = Voronoi(pts)
    verts = vor.vertices
    shapes_ind = vor.regions

    shapes_ind = [s[0:1] for s in shapes_ind if len(s)>0 and -1 not in s]
    shapes = [verts[s] for s in shapes_ind]

    n_shapes_to_fill = int(percent_to_fill*len(shapes))
    shapes_to_fill = np.random.choice(shapes, size=n_shapes_to_fill, replace=False)

    fill = []

    for s in shapes_to_fill:
        center = np.mean(s, axis=0)
        for scaler in np.linspace(min_scalar, 1, num=n_fill_lines, endpoint=False):
            scaled = scaler*(s - center) + center
            fill.append(scaled)


    fig, ax = plt.subplots(figsize=(20,20))
    ax.set_aspect('equal')

    if not debug:
        plt.grid(False)
        plt.axis('off')


    ax.set_xlim(*x_plot)
    ax.set_ylim(*y_plot)
    lc = LineCollection(shapes+fill,color='pink',linewidths=5)
    ax.add_collection(lc)

    return fig, ax

w = interactive(make_some_art,
                num_points=(10,2000,25),
                percent_to_fill=(0., 1., 0.05),
                n_fill_lines=(1, 20, 1),
                min_scalar=(0,1,0.01))
display(w)
```

Then we will get this.

Audience can change the image thorough those sliders.

demo video:
https://youtu.be/V87XBMaRcdY

At the end, audience can add filters for the painting.

```
In [ ]: from PIL import Image, ImageFilter

        # image
        image = Image.open('image.png')

        # filter
        def apply_filter(filter_name):
            if filter_name == 'BLUR':
                image_filtered = image.filter(ImageFilter.BLUR)
            elif filter_name == 'CONTOUR':
                image_filtered = image.filter(ImageFilter.CONTOUR)
            elif filter_name == 'DETAIL':
                image_filtered = image.filter(ImageFilter.DETAIL)
            elif filter_name == 'EDGE_ENHANCE':
                image_filtered = image.filter(ImageFilter.EDGE_ENHANCE)
            elif filter_name == 'EMBOSS':
                image_filtered = image.filter(ImageFilter.EMBOSS)
            elif filter_name == 'FIND_EDGES':
                image_filtered = image.filter(ImageFilter.FIND_EDGES)
            elif filter_name == 'SHARPEN':
                image_filtered = image.filter(ImageFilter.SHARPEN)
            else:
                image_filtered = image

            # image after filter
            image_filtered.show()

        # game
        while True:
            # choose one filter
            filter_name = input("Please enter a filter name (BLUR, CONTOUR, DETAIL, EDGE_ENHANCE, EMBOSS, FIND_EDGES, SHARPEN) ,
            if filter_name == 'QUIT':
                break

            # use filter
            apply_filter(filter_name)
```

Conclusion

We know that Python, as a programming language, is rigorous and logical, while art seems to be in another dimension. However I want to find the intersection of the two. I hope you have fun with my work and enjoy it.