Complex analysis is an extraordinarily elegant branch of mathematics, with applications ranging from quantum mechanics and signal processing to number theory and algebraic geometry. Yet it can be challenging to develop an intuition for the subject, given that the graph of even a humble function of a single complex variable is four-dimensional:

$$f: \mathbb{C} \to \mathbb{C}$$
$$\big(z, f(z)\big) = \big(x + iy, u(x,y) + iv(x,y)\big)$$

where $x, y \in \mathbb{R}$ and $u, v: \mathbb{R}^2 \to \mathbb{R}$.

While four-dimensional Euclidean space holds its charms for the intrepid geometer, there are better ways to visualize complex functions. We can, for example, represent the two dimensions of the range f(z) not as spatial coordinates, but as points on a two-dimensional color wheel. For each point *z* in the domain, we simply plot a pixel according to the color value of *f(z)*. This approach to complex visualization is known as *domain mapping*.

For this to work, we just need an invertible mapping between the complex plane and the chosen color palette. There are numerous systems for representing colors in digital format. Most of these employ either three or four independent coordinates. For example, the well-known RGB model, which is the model actually used by most display hardware, specifies each color by decomposing it into a precise combination of the additive primaries red, green and blue.

For our purposes, the HSV (hue, saturation, value) and HSL (hue, saturation, lightness) models are preferable to RGB. In these systems, one coordinate represents the *hue* of a color (starting with red and ranging through orange, yellow, green, cyan, blue, violet, magenta, and finally back to red), while a second cooordinate represents the *saturation* and a thrid coordinate represents *lightness* or *value*:

To represent the complex number $f(z) = u + iv$, we recast it in polar coordinates

$$f(z) = re^{i\theta}$$

and associate the argument θ with hue and the magnitude $|r|$ with value or lightness. Explicitly,

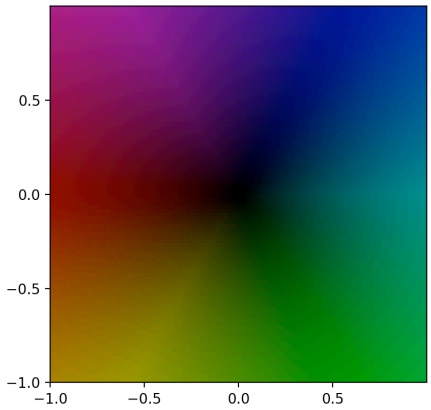
$$\text{hue} = \arg(f(z)) = \theta$$
$$\text{value or lightness} = \tan^{-1}|f(z)| = \tan^{-1}(r)$$

where we have used $\tan^{-1}|f(z)|$, but any sigmoid function $f(z)$: $(-\infty, \infty) \rightarrow (0,1)$ will do. (As for saturation, we only require two dimensions to represent f(z), so we will peg the saturation to a constant value s = 1.0 and leave it there.)

It is straightforward to implement this approach in the region $z \in [-1,1] \times [-1,1]$ using matplotlib:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import hsv_to_rgb

x = np.linspace(-1, 1, 1024)
y = np.linspace(-1, 1, 1024)
x,y = np.meshgrid(x,y)
z = x + 1j*y

h = (np.angle(z) + np.pi)/(2 * np.pi)
v = (2 / np.pi) * (np.arctan(np.absolute(z)))
s = np.ones(z.shape)

hsv = np.dstack((h, s, v))
rgb = hsv_to_rgb(hsv)

plt.imshow(rgb, origin="lower", extent=[-1, 1, -1, 1])
plt.xticks(np.arange(-1, 1, step=0.5))
plt.yticks(np.arange(-1, 1, step=0.5))
plt.show()
```

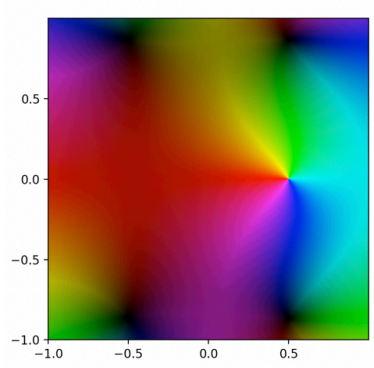
This yields the following domain-coloring plot:



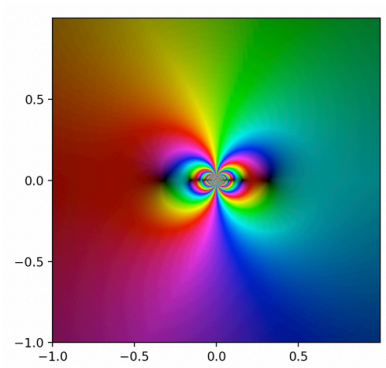
Note that the zero at (0,0) is shaded black.

To represent an arbitrary complex function, we now just need to pass z through the desired function before calculating the hue and value of each point. A few examples:

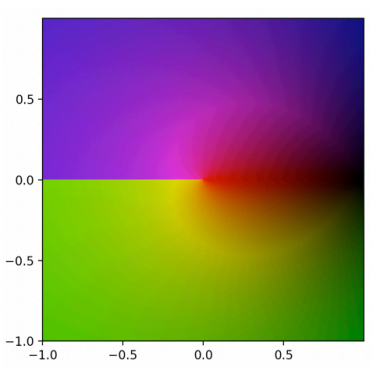
$f(z) = (z^4 + z^2 + 1)/(z - 1/2)$



$f(z) = \sin(1/z)$



$f(z) = log(z)$

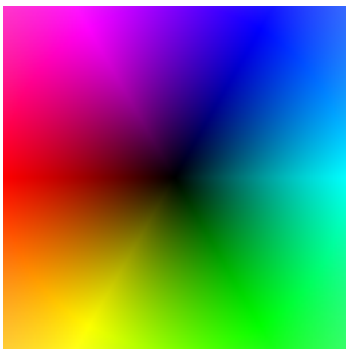


If we want to use the HSL model, which allows us to represent poles $f(z) \to \infty$ as white and zeros $f(z) \to 0$ as black, we can do so with PIL, the Python Imaging Library, and a little extra work:

```
from PIL import Image
import numpy as np
import colorsys

resolution = 256

#Make the grid over [-1,1] x [-1,1]
x = np.linspace(-1, 1, resolution)
y = np.linspace(-1, 1, resolution)
x, y = np.meshgrid(x, y)
```

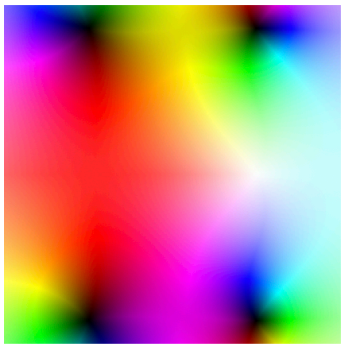
```
#Convert the grid to the complex plane
z = x + 1j*y

#Get the hls values for each z = r*e^(i*theta)
H = (np.angle(z) + np.pi)/(2 * np.pi)
L = (2 / np.pi) * (np.arctan(np.absolute(z)))
S = np.ones(z.shape)

#Stack the HLS values to make a (x,iy,3) array
HLS = np.dstack((H, L, S))

#Make a new grid for corresponding RGB values
rgb_array = np.empty_like(HLS)
rows = HLS.shape[0]
cols = HLS.shape[1]
for i in range(0, rows):
    for j in range(0, cols):
        hls = HLS[i,j]
        rgbs = 255 * np.array(colorsys.hls_to_rgb(hls[0],
hls[1], hls[2]))
        rgb_array[i,j] = rgbs

#Convert any NaNs (infinity) to 255s (white)
rgb_array[np.isnan(rgb_array)] = 255

#Make the new image and display it
newimg = Image.new('RGB', (rgb_array.shape[0],
rgb_array.shape[1]))
pixels = newimg.load()
for i in range(newimg.size[0]):
    for j in range(newimg.size[1]):
        hls_vals = tuple(rgb_array[i, j])
        pixels[i, j] = tuple([int(k) for k in hls_vals])
newimg.show()
```

The region $z \in [-1,1] \times [-i, i]$ is then rendered as follows:

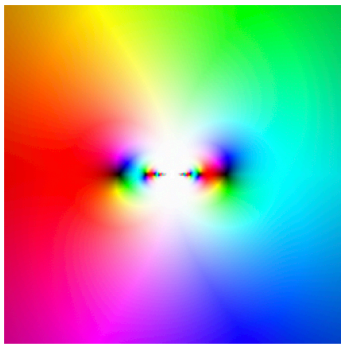


For the sake of comparison, we re-render the functions cited above:

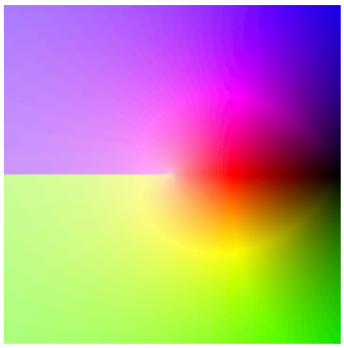
$\boldsymbol{f(z) = (z^4 + z^2 + 1)/(z - 1/2)}$



$\boldsymbol{f(z) = \sin(1/z)}$



$\boldsymbol{f(z) = log(z)}$



Note that in this format, poles (white, $f(z) \to \infty$) and zeros (black, $f(z) \to 0$) are both indicated.

An alternative approach to visualizing changes in $|f(z)|$ with respect to *z* is to introduce contour lines similar to those found in topographical maps. Specifically, we create a greyscale mask based on the fractional part of the log of f(z),

$$\text{greyscale mask value} = log(|f(z)|) - int(log(|f(z)|),$$

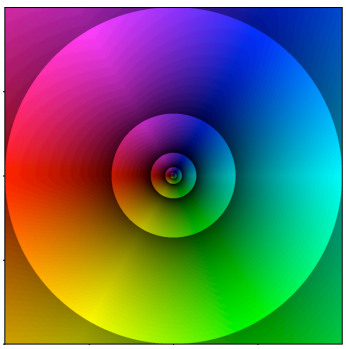
which then ranges from *0* (black) to *1* (white), and add this number to the *V* parameter (in the HSV system) of each pixel. In Python, we have

```
greyscale = 0.3 * np.mod(np.log(np.absolute(fz)),1)
V = V + greyscale
```

where the 0.3 value here is simply a scaling factor for aesthetic purposes. This alteration produces contour lines at

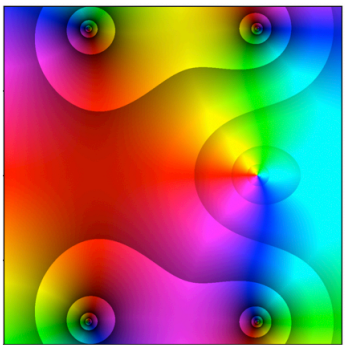
$$|f(z)| = \{\dots e^{-2}, e^{-1}, 1, e, e^2, \dots\}.$$

For example, the region $z \in [-1,1] \times [-i, i]$ is rendered as follows:

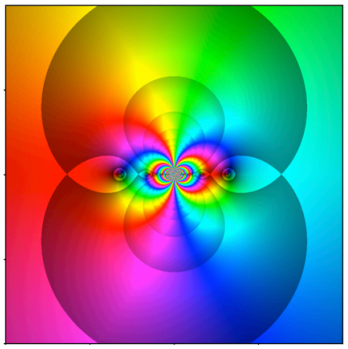


And the three functions we saw above are as follows:

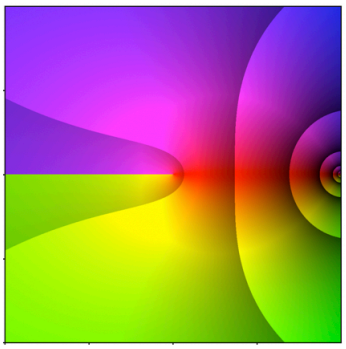
$f(z) = (z^4 + z^2 + 1)/(z - 1/2)$



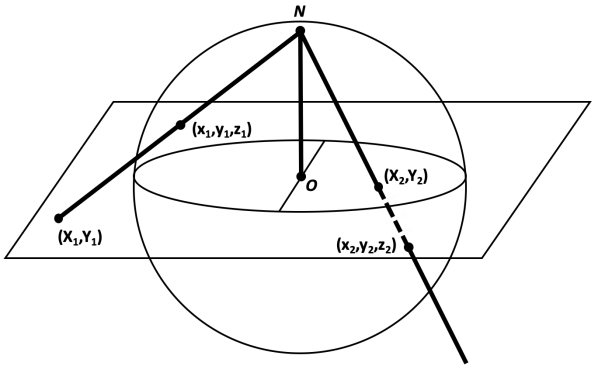
$f(z) = \sin(1/z)$



$f(z) = log(z)$



While plots over a finite rectangular region provide useful insight into the behavior of complex functions, they depict only a limited region of a complex plane (in the above examples, the region $z \in [-1,1] \times [-i, i]$). While it is possible to enlarge the plot window, and thus capture more of the function, there is a better approach: add the point "at infinity" to the complex plane, define $z/\infty = 0$ and $z/0 = \infty$, and use stereogrphic projection to map the resulting "extended" complex plane $\mathbb{C} \cup \{\infty\}$ onto the unit sphere centered at the origin:



Stereographic projection is visualized by aligning the equator of the unit sphere with the real plane $\mathbb{R}^2$ and then extending a line from the North pole through an arbitrary point on the sphere (x, y, z) where $x^2 + y^2 + z^2 = 1$. This line will then intersect the plane at a unique point (X, Y), as shown in the figure above. The point (X, Y) is given explicitly by

$$(X, Y) = \left(\frac{x}{1-z}, \frac{y}{1-z}\right),$$

or in polar coordinates by

$$(R, \Theta) = \left(\frac{\sin\varphi}{1-\cos\varphi}, \theta\right)$$

where these can be complexified as $Z = X + iY$ or $Z = Re^{i\Theta}$ to give a map from the unit sphere to $\mathbb{C}$. In fact, this projection is both one-to-one and onto, and thus invertible:

$$(x, y, z) = \left(\frac{2X}{1+X^2+Y^2}, \frac{2Y}{1+X^2+Y^2}, \frac{-1+X^2+Y^2}{1+X^2+Y^2}\right)$$

We see that the inverse projection maps the circle $|Z| = 1$ onto the equator of the sphere, the region $|Z| < 1$ onto the southern hemisphere, and the region $|Z| > 1$ onto the northern hemisphere. (Due to its curvature, the Riemann sphere cannot be covered by a single patch *Z*. However, it *can* be covered by adding a second patch *W* computed through stereographic projection from the south pole. In this case, the transition functions $Z = 1/W$ and $W = 1/Z$ ar both holomorphic, and so the Riemann sphere is a complex manifold).

To plot a complex function on the Riemann sphere, we will use the data visualization package mayavi.

We begin with a two-dimensional grid corresponding to the spherical coordinates $\{(\theta, \varphi) \mid \theta \in [0, 2\pi], \varphi \in [0, \pi]\}$ on the surface of the unit sphere (*r = 1*):

```
theta = np.linspace(0, 2*np.pi, steps)
phi = np.linspace(np.pi, 0, int(steps/2))
phi, theta = np.meshgrid(phi, theta)
```

Next we use stereographic projection to find the corresponding *Z* on the complex plane for each (θ, φ):

```
R = np.sin(phi)/(1 - np.cos(phi))
THETA = theta
x = R * np.cos(THETA)
y = R * np.sin(THETA)
z = x + 1j*y
```

We then run each *Z* through the complex function we want to visualize, for example:

```
fz = np.log((2*z**5 - z**3 - 1)/(z + 1))
```

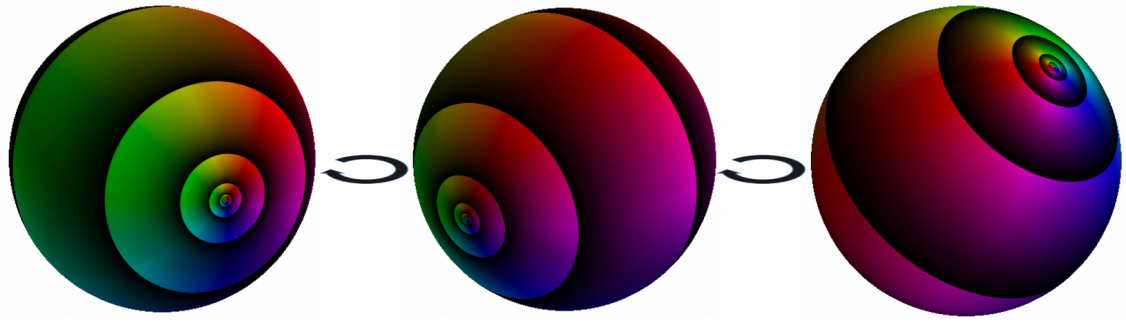
Finally, we calculate the hue and value (or lightness) of each fz, using one of the approaches demonstrated above. These steps yield a three-dimensional array: a two-dimensional array of (θ, φ) points, to each of which is assigned the RGB value (a 1,3-array) of the corresponding f(z). This array can be rendered and saved using either PIL or matplotlib.

To render a graph on the Riemann sphere with mayavi is then straightforward:

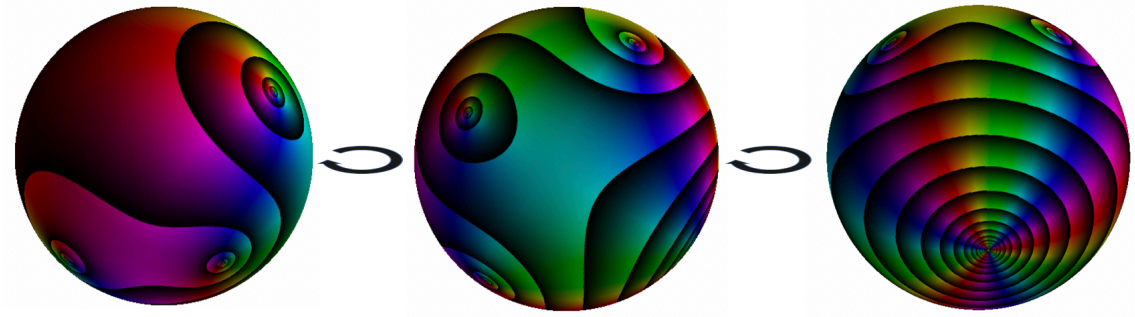
```
from mayavi import mlab
from tvtk.api import tvtk

# create a figure window (and scene)
fig = mlab.figure(size=(1200, 1200))

# load and map the texture
img = tvtk.JPEGReader()
img.file_name = imagefile_name
texture = tvtk.Texture(input_connection=img.output_port,\
                       interpolate=1)

# use a TexturedSphereSource
R = 1
Nrad = 180

# create the sphere source with a given radius and angular
# resolution
sphere = tvtk.TexturedSphereSource(radius=R,\
                                   theta_resolution=Nrad,\
                                   phi_resolution=Nrad)

# assemble rest of the pipeline, assign texture
sphere_mapper = \
     tvtk.PolyDataMapper(input_connection=sphere.output_port)
sphere_actor = tvtk.Actor(mapper=sphere_mapper, texture=texture)
fig.scene.add_actor(sphere_actor)
mlab.show()
```

Here again are the four equations we've been graphing throughout this demo, rendered this time with contour lines on the Riemann sphere:

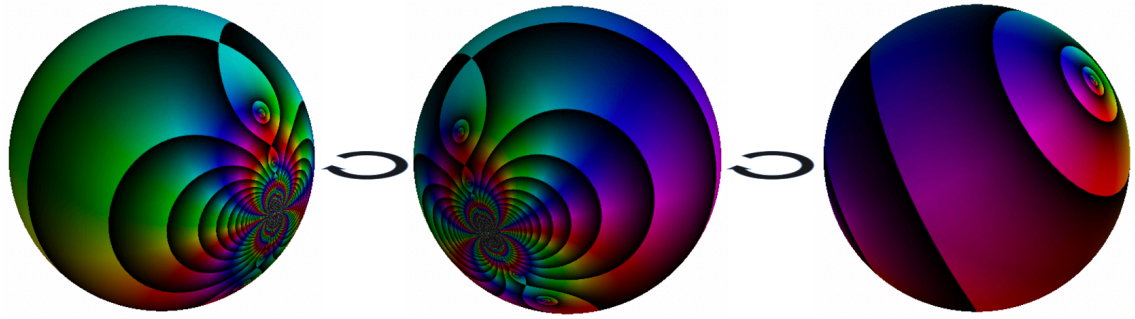
$$f(z) = z$$



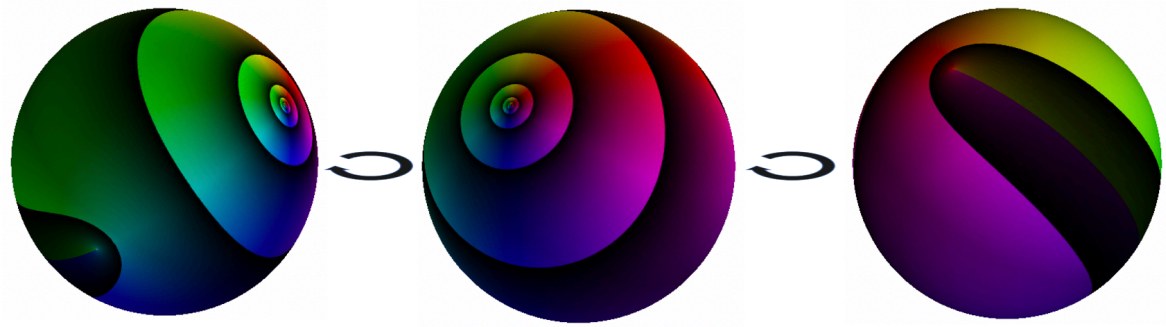
$$f(z) = \left(z^4 + z^2 + 1\right)/(z - 1/2)$$



$$f(z) = sin(1/z)$$

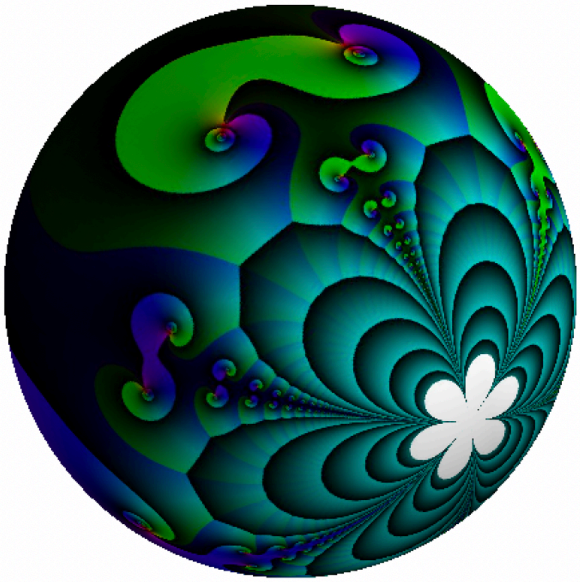


$$f(z) = log(z)$$

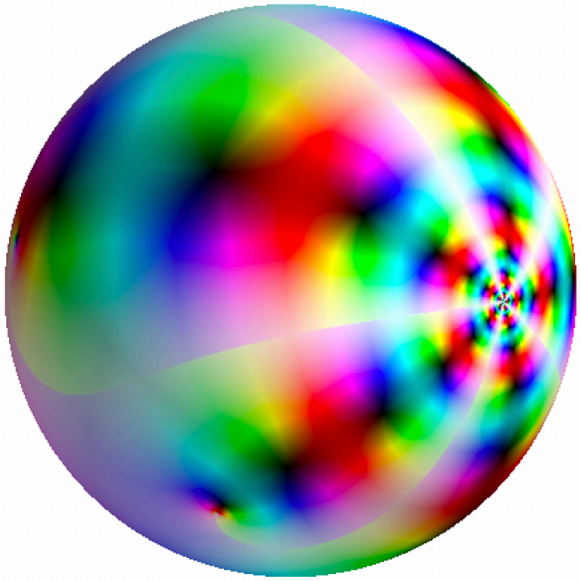


And to conclude, a miscellaneous selection of complex functions (with and without contour lines) on the Riemann sphere:

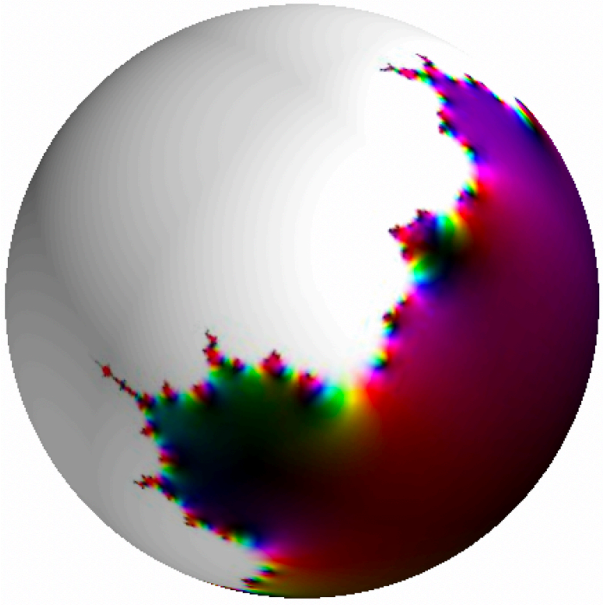
$$f(z) = \log\left(\sin\left(z^3\right) + iz - 1\right)$$



$$f(z) = \sin\left(\log\left(z^5 - 2z^3 - iz\right)\right)$$



The Mandelbrot set
(8 iterations of $f(z) = z^2 + c$)



$$f(z) = \sin\left(\log\left(\left(\frac{z^2+1}{z}\right)^3 - 1\right)\right)$$