P2 - New Methods from Mathematics to visual Art

Stefan Eger

June 24, 2024

1 Webcam BPM

In order to make mathematical Art a bit more interactive it was decided to use the pulse of a person as an Input. This is achieved via AI, which measure your pulse with a Webcam or smartphone camera. At first the code (`https://github.com/thearn/webcam-pulse-detector`) only supported webcams and not IP cameras. After I have fixed the HTTP request and some minor bugs, everything ran smoothly. The camera can be accessed via IP address with the following terminal example command:

```
source pyenv/bin/activate
python3 get_pulse_IP_cam.py --ip http://192.168.43.1:8080/shot.jpg
    --user Stefan --pwd pwd123 --udp 127.0.0.1:5005
```

The App "IP Webcam" (`https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en_US`) was used to make the webcam available to the python code. After the pulse is measured the pulse it is sent via UDP to the specified address (e.g. 127.0.0.1:5005).

2 Phase Portraits

I have implemented phase portraits in a generalised way that any complex function and exponential function can be displayed. The Input from the camera is used to change the color as Input for exponential complex function e.g $f(z) = z^c$. Also any coloring map from python can be used as a basis for domain coloring.

3 Newton Fractal

I have implemented the newton fractal, but it did not yield pretty results so it was abandoned along with 2d Fractals.

4 3d Fractals

3d Fractals are fairly new and yield interesting and interactive landscapes/3d models, but are quiet difficult to implement and understand. A great reference for understanding and implementation have been `http://blog.hvidtfeldts.net/index.php/2011/06/distance-estimated-3d-fractals-part-i/` and the SDFs and Raymarching part `https://iquilezles.org/articles/`

4.1 Implementation

The raymarching algorithm was implemented (from scratch with OpenGL) in the fragment shader. Similar to raytracing through every pixel a ray is shot, but unlike raytracing it uses a distance function which determines how close an object is to a calculated point on the ray. If it is close then the next step along the ray is smaller until a certain epsilon is reached. If not then the object is being searched further away (along the ray). This is especially good for fractals, which are defined mathematically and do not have to be build with triangles or very complicated ray fractal intersection calculations. Raymarching also allows for space folding, which makes it very performant to calculate fractals, and that is necessary for an interactive art scene. The following 3d fractals have been been implemented:

- Mandelbulb
- Modified Juliaset Quaternion with Orbital Traps
- Mandelbox
- Sierpinski Tetrahedrons

Many more 3d Fractals can be implemented but these are a good starting point for interesting scenes (except Sierpinski). It is also possible to move around and through each 3d Fractal to find a good angle/images. The Input of the camera is used to change the color of the fractal or the fractal itself. The coloring of the fractals can be tricky and has to be evaluated for each fractal to make an interesting scene.

5 Code

The code can be found here: `https://github.com/Stefan-Eger/P2`

6 Videos

Domain Coloring: `https://youtu.be/u4y7OHOu0-s`
Mandelbulb: `https://youtu.be/N3wixvq-aSU`
Mandelbox: `https://youtu.be/kEjKIpORqMc`
Juliaset Quaternion : `https://youtu.be/aLuOZHGXi5w`