Expanding the *curl*-function into C^3

Georg Kiehne

May 29, 2015

1 Introduction

This document describes the expansion of an existing complex polynomial named *curl*. *curl* was written as a so called "variation" for an image generation algorithm named "Fractal Flames" [1]
One particular implementation of the fractal flame algorithm is resided in the software Apophysis[2] in which I have a history of contributions to both, the program itself and several external packages ("plugins") which add additional variations to the software.

2 Approach

The aim is to create a variation working in a higher dimension than *curl*. It is important that one doesn't understand "higher dimension" as something like *curl3D* (which also already exists), but rather a higher order in a polynomial equation within the complex number space.

Before starting, it is essential to understand how complex numbers are utilized in Apophysis and fractal flame implementations in general. The reader should also have basic knowledge about what a complex number is and how it works.

A complex number z can be written as:

$$\mathrm{z} = \mathrm{a} + \mathrm{bi}$$

where a is the real part, b the imaginary part and i the imaginary unit $i = \sqrt{1}$.

In Apophysis, it is habit to interpret the real part as the x-coordinate and the imaginary part as the y-coordinate. This is why the document refers to x and y, not a and b from now on.
Currently, *curl* is defined as:

$$f(\mathrm{z}) = \frac{z}{c_2 z^2 + c_1 z + 1} \text{ where } z = x + iy$$

[1] `http://flam3.com/flame_draves.pdf`
[2] `http://apophysis.org`

As it is obvious, the denominator of the fraction represents a polynomial equation of second order similar to $y = ax^2 + bx + c$. The denominator is raised to a polynomial equation of third order:

$$y = ax^3 + bx^2 + cx + d$$

Resulting in:

$$f(\mathrm{z}) = \frac{z}{c_3 z^3 + c_2 z^2 + c_1 z + 1} \text{ where } z = x + iy$$

3 Derivation

Essentially, the polynomial equation is now to be broken down. This is done by first determining z^2, then z^3 and finally, composing it into a complete variation equation.

To square a complex number z, the first binomial formula is useful:

$$(a + b)^2 = (a + b)(a + b) = a^2 + 2ab + b$$

Therefore:

$$z^2 = (x + iy)^2 = x^2 + 2xyi + (yi)^2$$

Assuming $i^2 = -1$ because $i = \sqrt{-1}$ and simplifying to:

$$z^2 = x^2 + 2xyi - y^2$$

Cubing a complex number is performed using the same principle:

$$(a + b)^3 = (a + b)(a + b)(a + b) = a^3 + 3a^2b + 3ab^2 + b^3$$

Therefore:

$$z^3 = (x + iy)^3 = x^3 + 3x^2yi + 3x(yi)^2 + (yi)^3$$

Assuming $i^3 = -i$ because $i^2 = -1$ and $i^3 = i^2 \cdot i$:

$$z^3 = x^3 + 3x^2yi - 3xy^2 - y^3i$$

As all necessary terms are collected, it would now be possible to compose the complete variation formula. Since the numerator won't change, the focus is on the denominator. Therefore substituting:

$$\mathrm{f(z)} = \frac{z}{c_3 z^3 + c_2 z^2 + c_1 z + 1} = \frac{z}{q}$$

Then continuing on q:

$$q = c_3 z^3 + c_2 z^2 + c_1 z + 1$$
$$q = c_3(x^3 + 3x^2yi - 3xy^2 - y^3i) + c_2(x^2 + 2xyi - y^2) + c_1(x + yi) + 1$$
$$q = c_3x^3 + c_3x^2yi - 3c_3xy^2 - c_3y^3i + c_2x^2 + 2c_2xyi - c_2y^2 + c_1x + c_1yi + 1$$

Substituting back into $f(z) = \frac{z}{q}$:

$$f(z) = \frac{x+yi}{(c_3x^3-3c_3xy^2+c_2x^2-c_2y^2+c_1x+1)+(c_3x^2y-c_3y^3+2c_2xy+c_1y)*i}$$

The algorithm requires a separate treatment of real and imaginary part. In order to gain two separate terms, it is necessary to factor out i. The most straightforward approach is to view the above term as a division of two arbitrary complex numbers, factor out i. Therefore:

$$z_1 = a + bi$$

$$z_2 = c + di:$$

Assuming $a = x$ and $b = y$ because $f(z) = \frac{z}{q} = \frac{x+iy}{q}$. Therefore

$$f(z) = \frac{a+bi}{c+di}$$

Conjugating the denominator:

$$\overline{c + di} = c - di$$

Multiplying numerator and denominator with the conjugate:

$$\begin{aligned} f(z) &= \frac{a+bi}{c+di} \cdot \frac{c-di}{c-di} \\ &= \frac{ac-adi+bci-bdi^2}{c^2-cdi+cdi-d^2i^2} \\ &= \frac{ac-adi+bci+bd}{c^2+d^2} \\ &= \frac{ac+bd}{c^2+d^2} + i \cdot \frac{bc-ad}{c^2+d^2} \end{aligned}$$

Resulting in:

$$\begin{aligned} &a = x, \\ &b = y, \\ &c = c_3x^3 - 3c_3xy^2 + c_2x^2 - c_2y^2 + c_1x + 1, \\ &d = c_3x^2y - c_3y^3 + 2c_2xy + c_1y \end{aligned}$$

Substituting back into $f(z) = \frac{z}{q}$ is left optional, as computer programs can utilize the intermediate results of a, b, c and d stored in memory to perform the calculation from there on.
In order to extract a meaningful output for eventual plotting and/or the input of the next iteration, the following quasi-standard approach can be used:

$$x' = x + \omega * R(z')$$

$$y' = y + \omega * I(z')$$

$\omega :=$ being a constant representing the variation's "density" parameter and R, I being functions as described in *Inverse Geometry*[3]:

[3]Frank Morley and son, 1933

$$R(z') = \frac{z' + \overline{z'}}{2}$$

$$I(z') = \frac{z' - \overline{z'}}{2i}$$

Inserting into the above term:

$$z' = f(z) = \frac{ac+bd}{c^2+d^2} + i \cdot \frac{bc-ad}{c^2+d^2}$$

Resulting in:

$$R(z') = \frac{ac+bd}{c^2+d^2}$$

$$I(z') = \frac{bc+ad}{c^2+d^2}$$

Therefore:

$$x' = x + \omega \cdot \frac{ac+bd}{c^2+d^2}$$

$$y' = y + \omega \cdot \frac{bc+ad}{c^2+d^2}$$

The terms (together with the correct substitutions for a, b, c, d) are fit to be used in the source code of an Apophysis extension.

4 Optimizations

The resulting variation is trivial to implement. Like with *curl*, several optimizations can be made. The most straightforward approach is to separately treat the following special cases:

a.) $c_1 = 0$
b.) $c_2 = 0$
c.) $c_3 = 0$
d.) Any combination of the above

Precalculating $3c_3$ and $2c_2$ is recommendable. While being simple terms, the repeated and unnecessary multiplication has a reasonable weight, considering that they would be potentially executed billion million times per second.

5 Conclusions

After sufficiently simplifying the terms, an implementation of a 3rd-order polynomial *curl*-variation becomes trivial enough to promise high performance batch calculations, assuming that the host code is efficient enough. The only requirement posed to the host is the ability to provide floating-point parameters as the calculation would otherwise reduce to $f(z) = z$.

6 Appendix: Implementation for Apophysis

An implementation in C can be found at:

`https://gist.github.com/xyrus02/2e1b5c2777d1f2b1b8a084d0b747bcd2`