Higher-Dimensional (4D+) Geometry and Fractals

31c3

What we'll talk about today

31c3, December 29th:

Higher Dimensional (4D+) Geometry and Fractals

Speaker: Magnus

magnus@ef.gy

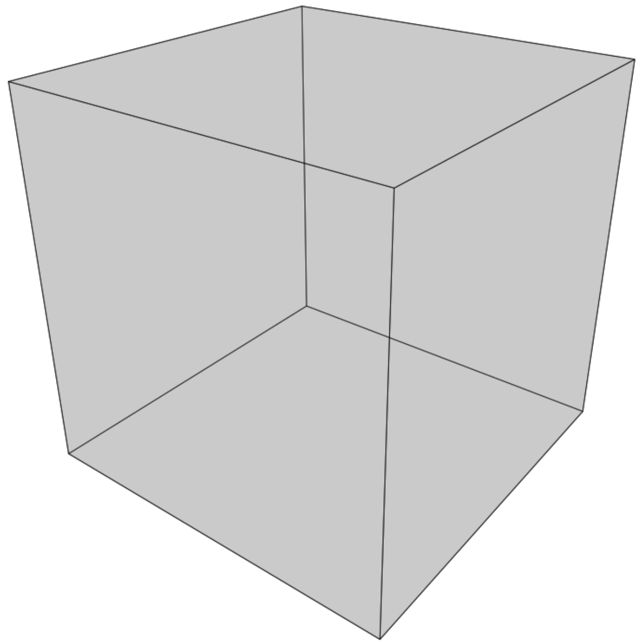
@jyujinX

This talk is split into 3 sections:

1. Perspective Projections (4D+)
2. Iterated Function Systems
3. Fractal Flames

Intermission slides will demo the preceding part's content hands on.

Follow along at: https://dee.pe/r



Part 0: What your linear algebra teacher already told you.

What you learned in LinAlg 101

- Vector-matrix and matrix-matrix products
- Cross product
- Homogeneous coordinates
- Rotations
- Affine transformations
- Look-at transformation
- Perspective transformation
- Projections

Cross product

We multiply two 3D vectors to obtain a normal to these two vectors.

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} a_0 & b_0 & \mathbf{right} \\ a_1 & b_1 & \mathbf{up} \\ a_2 & b_2 & \mathbf{back} \end{vmatrix}$$

Homogeneous matrix expansion

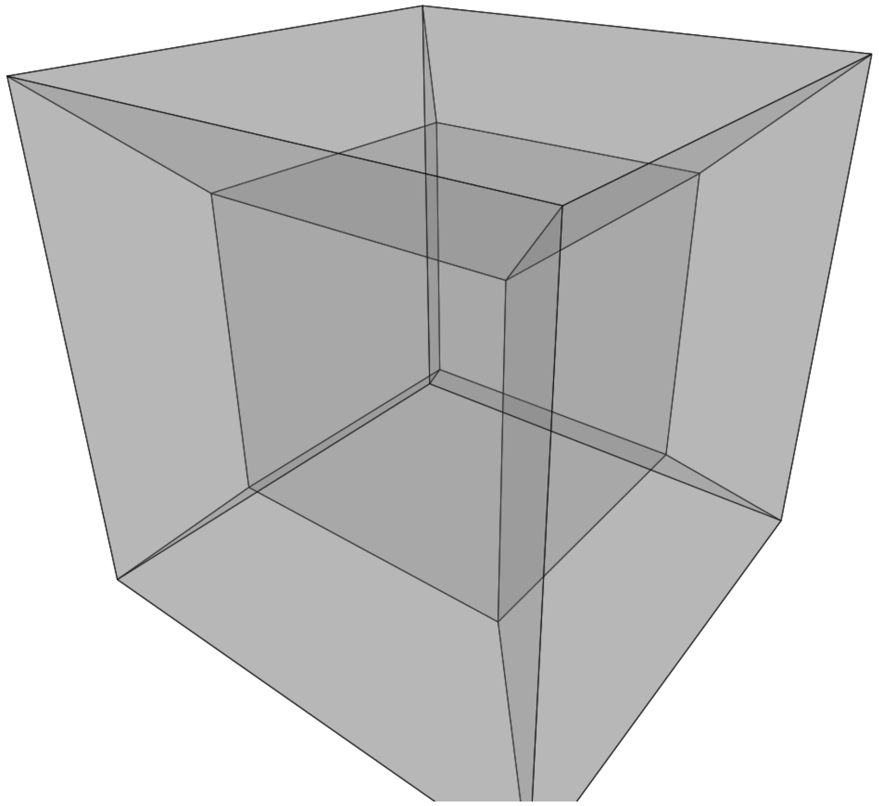
To use a linear transformation as part of an affine transform we need to expand its matrix.

$$\begin{pmatrix} m_{0,0} & \dots & m_{0,n-1} \\ \dots & \dots & \dots \\ m_{n-1,0} & \dots & m_{n-1,n-1} \end{pmatrix} \to \begin{pmatrix} m_{0,0} & \dots & m_{0,n-1} & 0 \\ \dots & \dots & \dots & \dots \\ m_{n-1,0} & \dots & m_{n-1,n-1} & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix}$$

Perspective transformation

$$\text{perspective}_3\left(\text{eye-angle}, \text{aspect}, \text{near}, \text{far}\right) :=$$

$$\begin{pmatrix} \dfrac{\frac{1}{\tan\left(\frac{\text{eye-angle}}{2}\right)}}{\text{aspect}} & 0 & 0 & 0 \\ 0 & \dfrac{1}{\tan\left(\dfrac{\text{eye-angle}}{2}\right)} & 0 & 0 \\ 0 & 0 & \dfrac{\text{near} + \text{far}}{\text{near} - \text{far}} & \text{-1} \\ 0 & 0 & \text{-2} \times \dfrac{\text{near} \times \text{far}}{\text{near} - \text{far}} & 0 \end{pmatrix}$$



Part I: Perspective Projections in 4D+

What do we need?

- Vector-matrix and matrix-matrix products
- Cross product
- Homogeneous coordinates
- Rotations
- Affine transformations
- Look-at transformation
- Perspective transformation
- Projections

What do we need?

- Vector-matrix and matrix-matrix products
- ~~Cross product~~
- Homogeneous coordinates
- Rotations
- Affine transformations
- Look-at transformation
- Perspective transformation
- Projections

What’s different?

- ~~Cross product~~ normals
- Look-at transformation
- Perspective transformation
- Projections

Normals

There is no 4D cross product, but we really only need normals of three vectors:

$$\mathbf{a} \times \mathbf{b} \times \mathbf{c} = \begin{vmatrix} a_0 & b_0 & c_0 & \mathbf{right} \\ a_1 & b_1 & c_1 & \mathbf{up} \\ a_2 & b_2 & c_2 & \mathbf{back} \\ a_3 & b_3 & c_3 & \mathbf{charm} \end{vmatrix}$$

Normals

In general, we calculate an n-D normal with *n-1* vectors using the determinant of a matrix:

$$\mathbf{n} = \mathbf{v^1} \times \mathbf{v^2} \times \ldots \times \mathbf{v^{n-1}} = \begin{vmatrix} v_0^1 & v_0^2 & \ldots & v_0^{n-1} & \mathbf{b^1} \\ v_1^1 & v_1^2 & \ldots & v_1^{n-1} & \mathbf{b^2} \\ v_2^1 & v_2^2 & \ldots & v_2^{n-1} & \mathbf{b^3} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ v_{n-1}^1 & v_{n-1}^2 & \ldots & v_{n-1}^{n-1} & \mathbf{b^n} \end{vmatrix}$$

Look-at transformation

$$\text{look-at}_4\left(\mathbf{to}, \mathbf{from}, \mathbf{up}, \mathbf{back}\right) :=$$

$$\left(\mathbf{col}_1 \times \mathbf{col}_2 \times \mathbf{col}_3, \mathbf{back} \times \mathbf{col}_2 \times \mathbf{col}_3, \mathbf{up} \times \mathbf{back} \times \mathbf{col}_3, \mathbf{to} - \mathbf{from}\right)$$

Look-at transformation

$$\text{look-at}_n\left(\mathbf{to}, \mathbf{from}, \mathbf{b}^{n-2}\right) :=$$

$$\left(\mathbf{c}_1 \times ... \times \mathbf{c}_{n-1}, ..., \mathbf{b}^1 \times ... \times \mathbf{b}^{n-4} \times \mathbf{c}_{n-2} \times \mathbf{c}_{n-1}, \mathbf{b}^0 \times ... \times \mathbf{b}^{n-3} \times \mathbf{c}_{n-1}, \mathbf{to} - \mathbf{from}\right)$$

Perspective transformation

We can rely on an underlying 3D perspective transformation’s correction for far and near clipping and aspect ratio correction.

The much simpler matrix is then...

Perspective transformation

$$\text{perspective}_4\left(\text{eye-angle}\right) :=$$

$$\begin{pmatrix} \frac{1}{\tan\left(\frac{\text{eye-angle}}{2}\right)} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\text{eye-angle}}{2}\right)} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\tan\left(\frac{\text{eye-angle}}{2}\right)} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspective transformation

$$\text{perspective}_n\left(\text{eye-angle}\right) :=$$

$$\begin{pmatrix} \dfrac{1}{\tan\left(\frac{\text{eye-angle}}{2}\right)} & 0 & \ldots & 0 & 0 \\ 0 & \dfrac{1}{\tan\left(\frac{\text{eye-angle}}{2}\right)} & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}$$

View Matrix

$$\text{view-matrix}_4\left(\mathbf{to}, \mathbf{from}, \mathbf{up}, \mathbf{back}, \text{eye-angle}\right) :=$$

$$\text{translate}_4\left(-\,\mathbf{from}\right) \times \text{look-at}_4\left(\mathbf{to}, \mathbf{from}, \mathbf{up}, \mathbf{back}\right) \times \text{perspective}_4\left(\text{eye-angle}\right)$$

View Matrix

$$\text{view-matrix}_n\left(\mathbf{to}, \mathbf{from}, \mathbf{base}^{n-2}, \text{eye-angle}\right) :=$$

$$\text{translate}_n\left(-\,\mathbf{from}\right) \times \text{look-at}_n\left(\mathbf{to}, \mathbf{from}, \mathbf{base}^{n-2}\right) \times \text{perspective}_n\left(\text{eye-angle}\right)$$

Projections

We convert to non-homogeneous coordinates and divide by the last coordinate.

Projections

$$\text{normalise-reduce}\left(\mathbf{V}\right) := \begin{pmatrix} \dfrac{\mathbf{V}_0}{\mathbf{V}_{n-1}} \\ \dfrac{\mathbf{V}_1}{\mathbf{V}_{n-1}} \\ \dots \\ \dfrac{\mathbf{V}_{n-2}}{\mathbf{V}_{n-1}} \end{pmatrix}$$

$$\text{project}_n\left(\mathbf{V}, M\right) := \text{normalise-reduce}\left(\text{normalise-reduce}\left(\left(\begin{pmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \dots \\ \mathbf{V}_{n-1} \\ 1 \end{pmatrix} \times M\right)\right)\right)$$

Projections

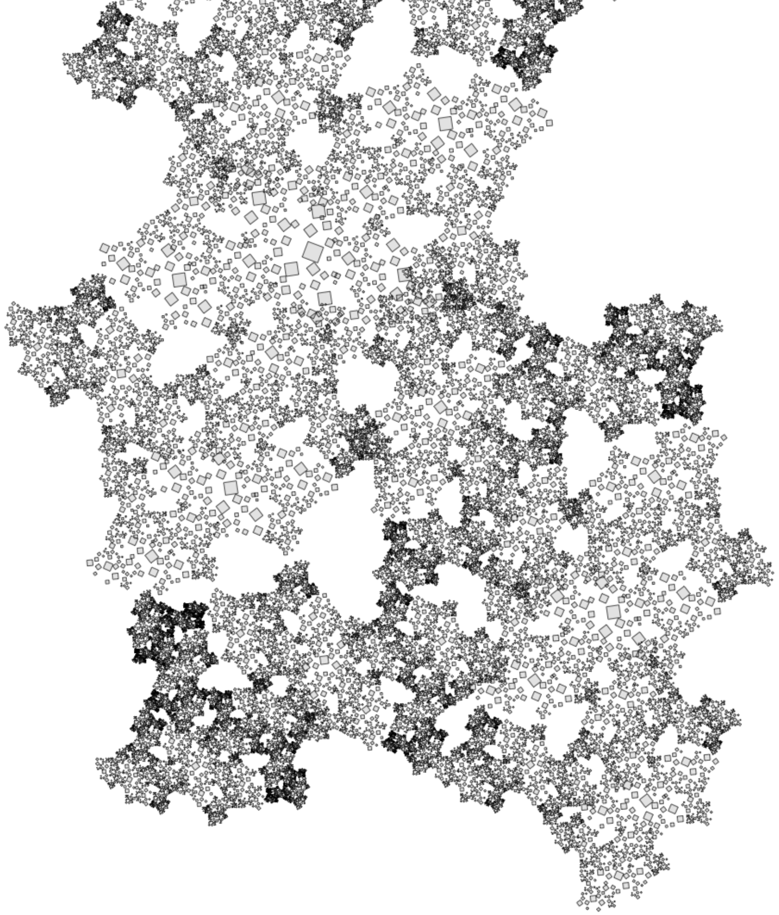
However, this only reduces the dimension by one. Therefore, after this step we need additional projections until we get to 2D, which we can plot on-screen.

Projections

Each dimension thus gets its own "camera" - a set of "from" and "to" points, eye angles, transformation matrices, etc.

All of these are independent of each other.

Intermission



Part II: (Affine) Iterated Function Systems

What is an IFS?

An Iterated Function System is a collection of functions on vectors. We're usually interested in the set of points that result when applying these to points on a plane.

To get any kind of fractal effect, we need at least 2 functions. The functions should also be contractive.

What is an IFS?

$$S = \bigcup_{i=0}^{n-1} F_i(S)$$

What is an affine IFS?

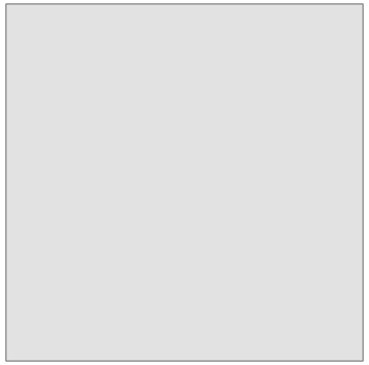
In principle, we can choose any kind of function domain for an IFS, so long as they operate on vectors.

If we choose affine transformations, we can represent the IFS using a set of (d+1)x(d+1) matrices.

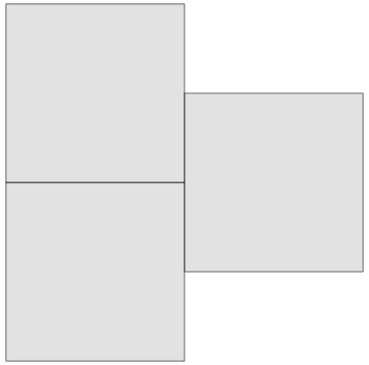
Discrete Renders

We can plot this by taking a sufficiently subdivided primitive and applying all of the transformations to the vertices recursively.

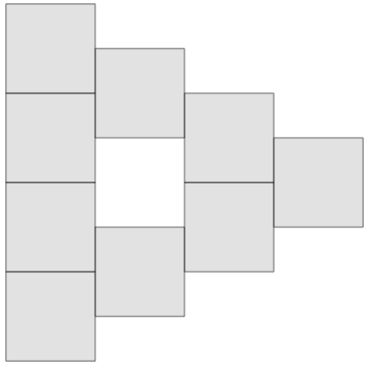
Sierpinski Gasket



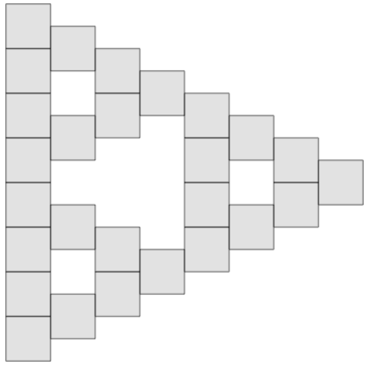
Sierpinski Gasket



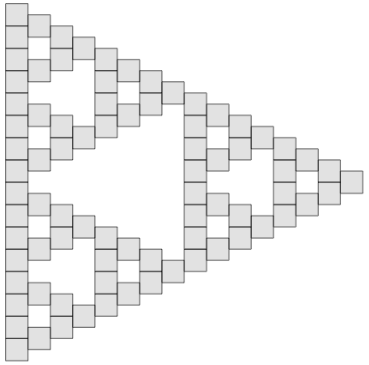
Sierpinski Gasket



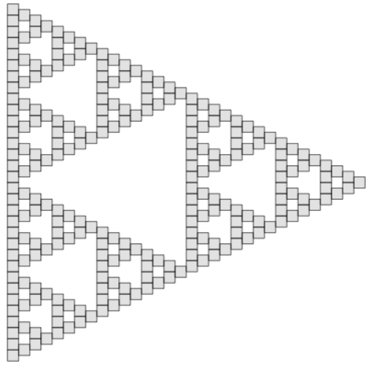
Sierpinski Gasket



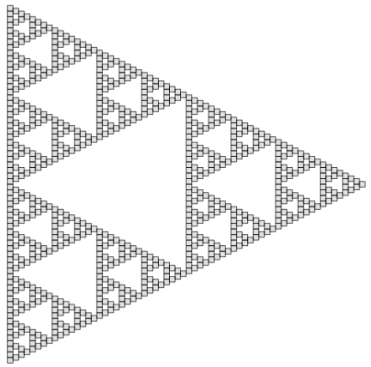
Sierpinski Gasket



Sierpinski Gasket



Sierpinski Gasket



Does this work in 3D and up?

Well...

Intermission

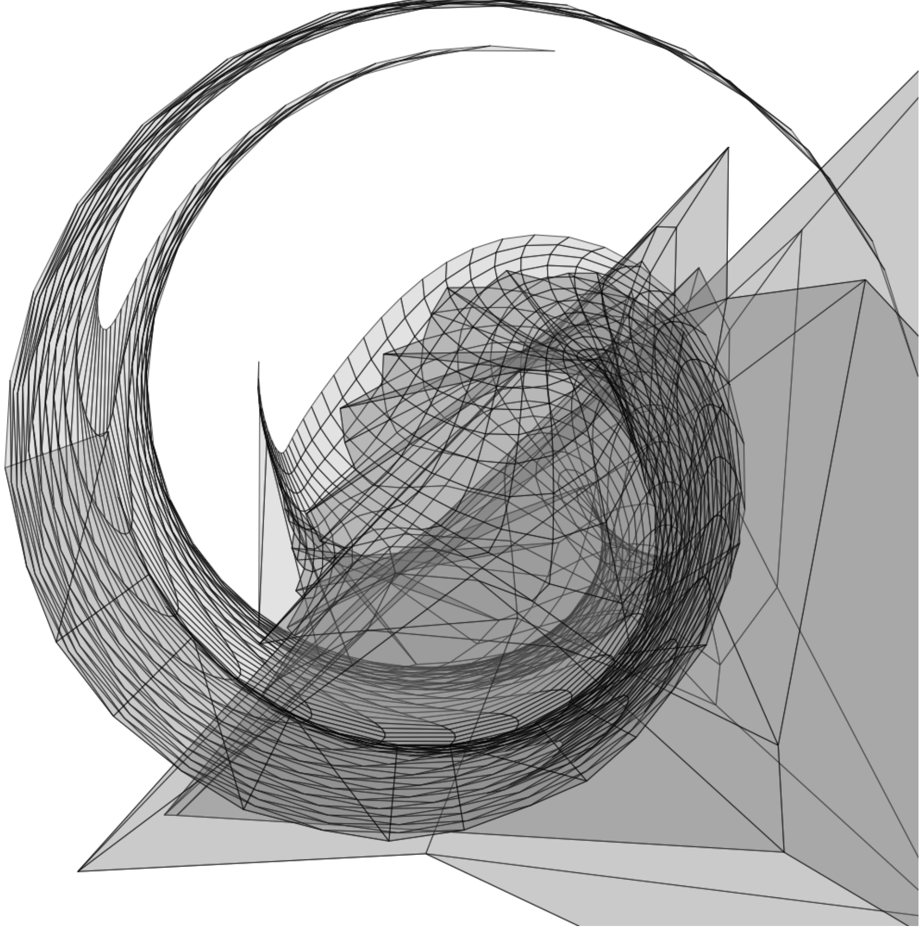
The Chaos Game

```
repeat until satisfied:
1. Take a random point on the plane.
2. iterations++
3. Pick a random function and apply to the point.
4. if (iterations > cutoff) plot(point);
5. if (iterations < maxIterations) repeat from 2;
```

Improvements to the Chaos Game

- Assign weights to the individual functions
- Keep track of the number of hits to produce grayscale images
- Assign colours to the individual functions and mix them with the current pixel colour

Does this work in 3D and up?



Part III: Fractal Flames

What are Fractal Flames?

Structurally they are Iterated Function Systems with a special type of function composed of a finite number of *variations*.

They also use a special colouring algorithm to highlight details in the functions. Rendering is done with a slightly modified chaos game.

Variations

Instead of plain affine transformations, we start with an affine transformation and apply *variation* functions to the result.

$$F_i(x, y) = V_j(a_i x + b_i y + c_i, d_i x + e_i y + f_i)$$

(There is a list of 49 canonical variations in the Fractal Flame paper.)

Variations

$$\begin{array}{rcll}
V_0(x,y) & = & (x,y) & \textit{linear} \\
V_1(x,y) & = & (\sin x, \sin y) & \textit{sinusoidal} \\
V_2(x,y) & = & \frac{1}{r^2} \cdot (x,y) & \textit{spherical} \\
V_3(x,y) & = & \big(x\sin(r^2) - y\cos(r^2), x\cos(r^2) + y\sin(r^2)\big) & \textit{swirl} \\
V_4(x,y) & = & \frac{1}{r} \cdot ((x-y)(x+y), 2xy) & \textit{horseshoe} \\
V_{17}(x,y) & = & (x + c\sin(\tan 3y), y + f\sin(\tan 3x)) & \textit{popcorn} \\
V_{24}(x,y) & = & (\sin(p_1 y) - \cos(p_2 x), \sin(p_3 x) - \cos(p_4 y)) & \textit{pdj}
\end{array}$$

Variation Blending

For even more impressive results, we blend several of these variations together.

$$F_i(x, y) = \sum_j v_{ij} V_j(a_i x + b_i y + c_i, d_i x + e_i y + f_i)$$

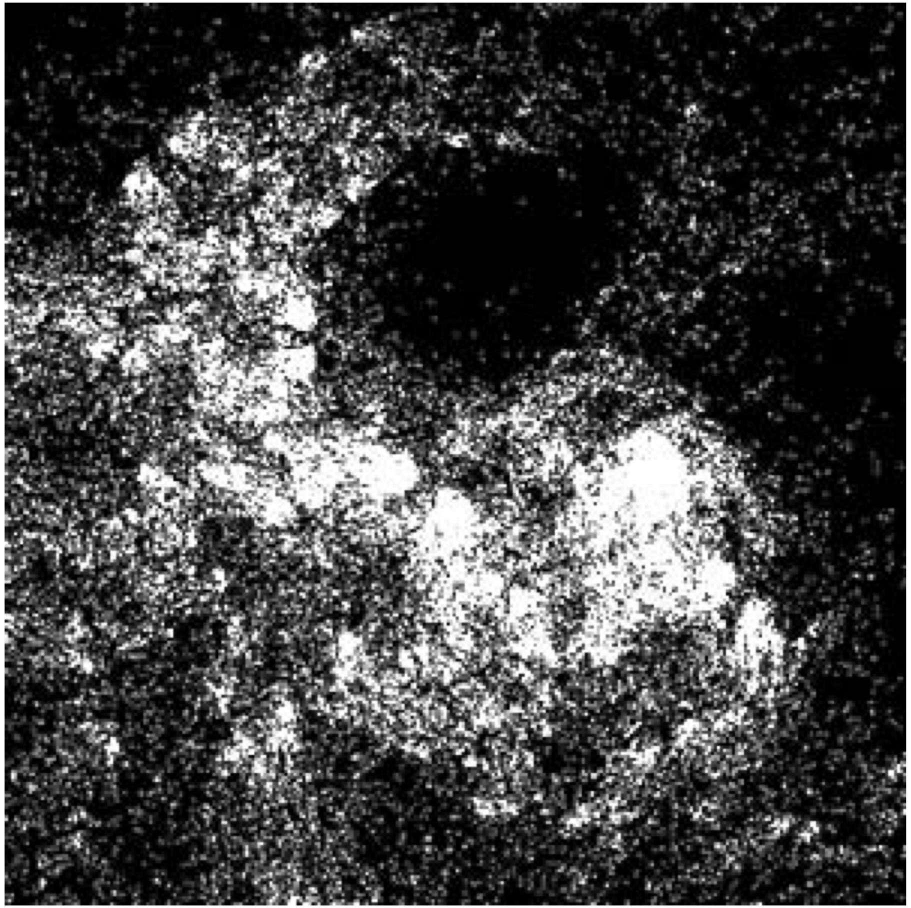
Post Transforms

For added picture control, we may apply another variation as a post-transform.

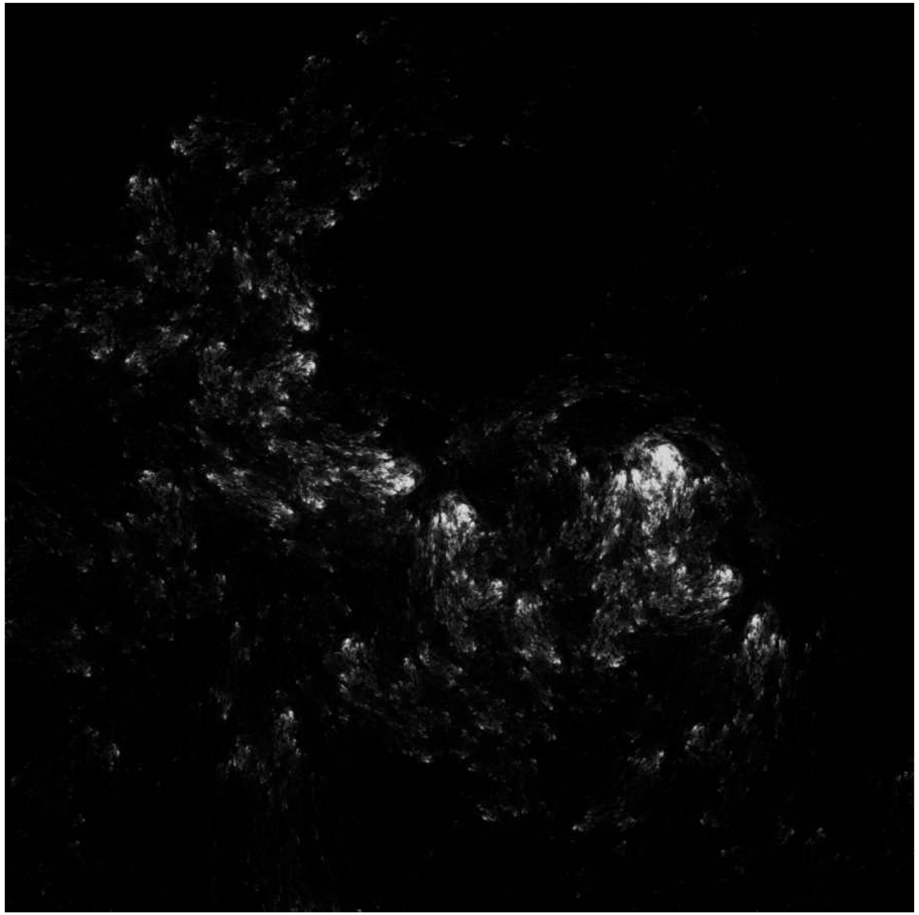
$$F_i(x, y) = P_i(\sum_j v_{ij} V_j(a_i x + b_i y + c_i, d_i x + e_i y + f_i))$$

Final Transform

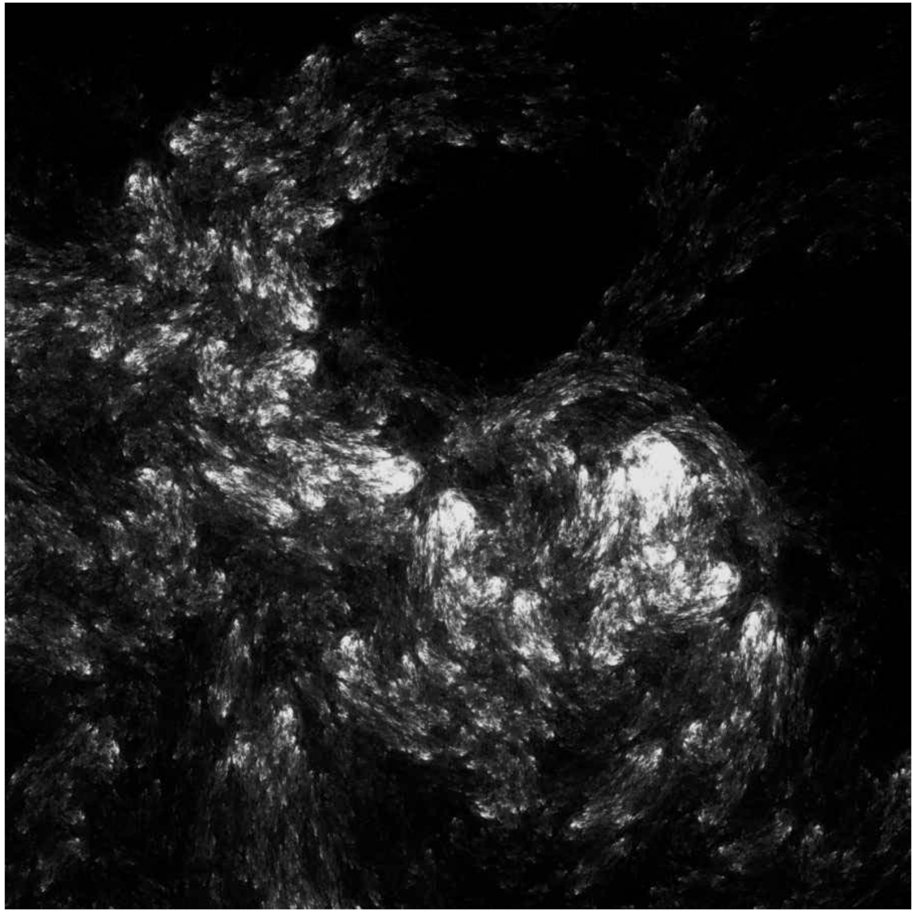
Another transform may be applied to all output vertices for certain picture effects.



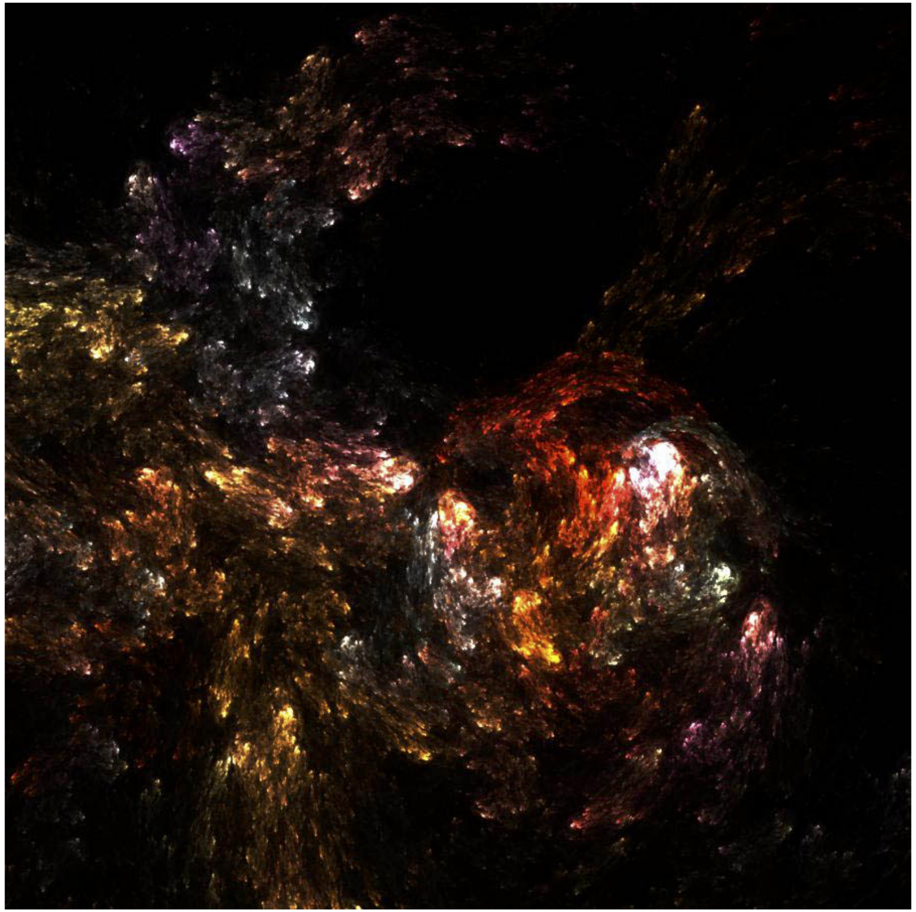
By default, the chaos game only results in a binary, black and white image.



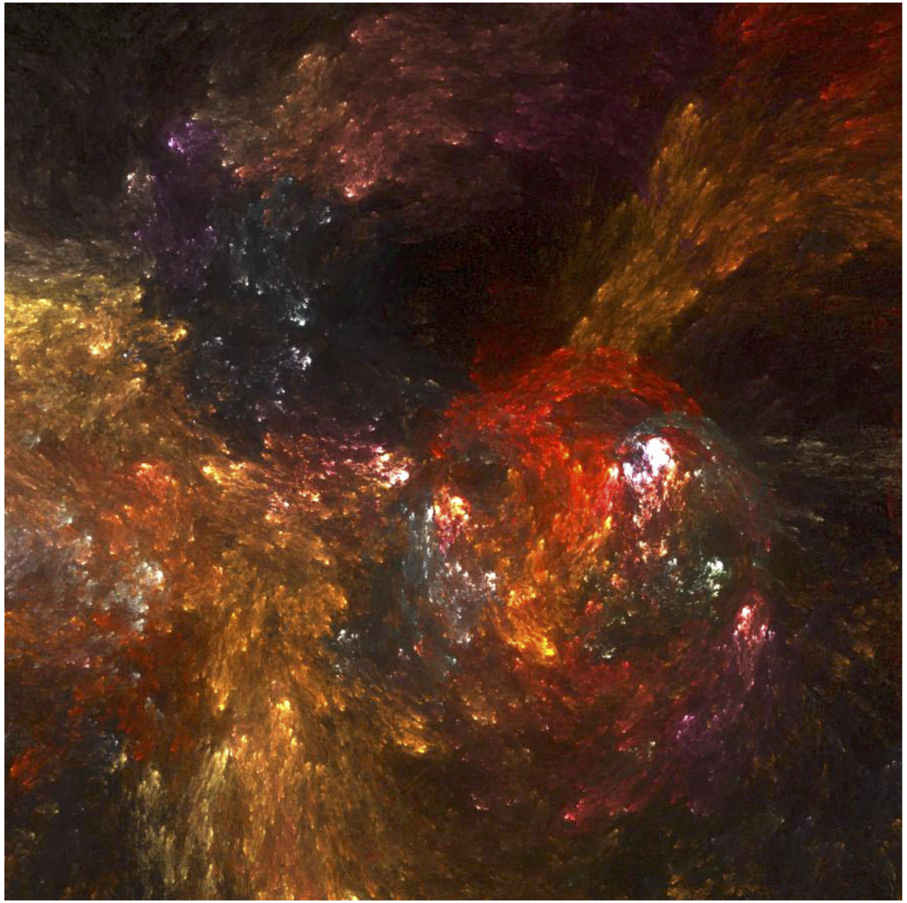
To improve this, we count the number of hits on each pixel.



... which looks a lot better if you plot it after applying a logarithm.



To add colour, we mix the function numbers used and apply a colour map.

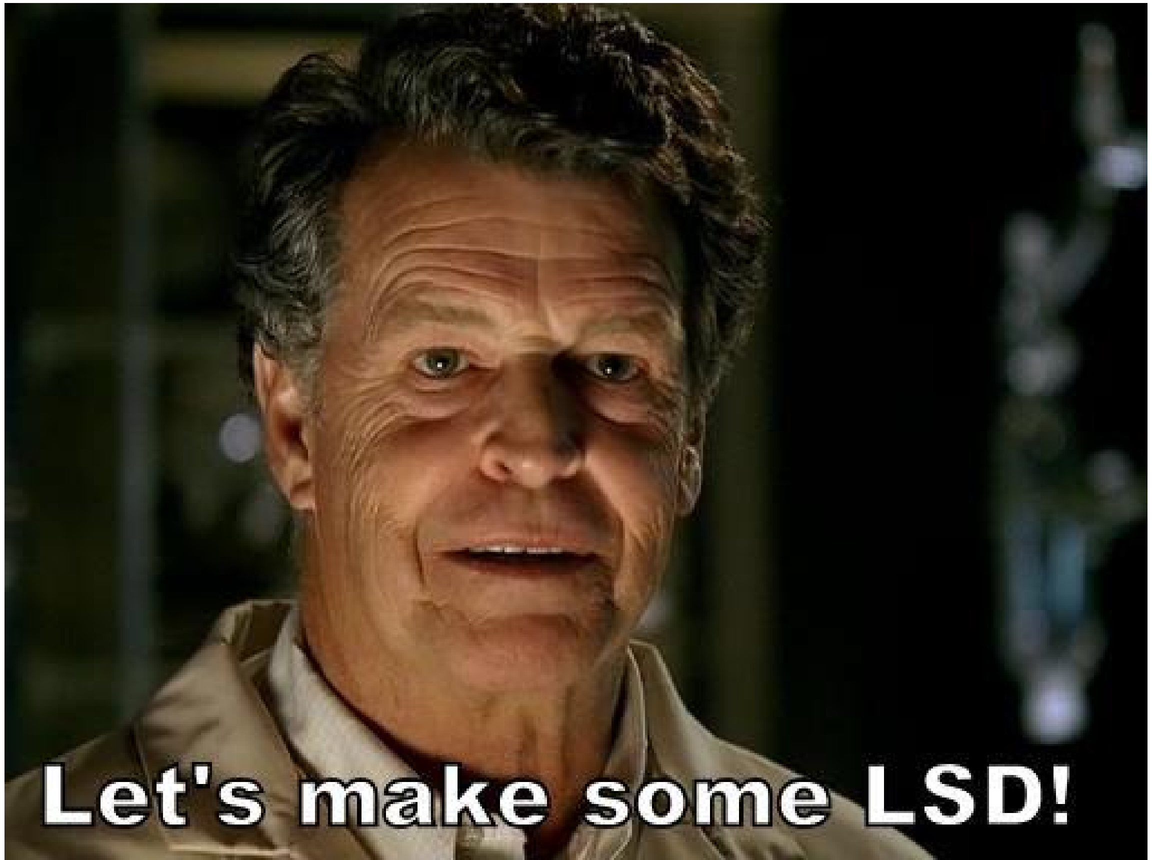


And your favourite colour vibrancy algorithm never hurt, either.

However...

The original fractal flame algorithm is strictly 2D. All of the 3D effects seen in, e.g. *Electric Sheep* are illusions.

Does this work in 3D and up?



Intermission

Questions?

31c3, December 29th:

Higher Dimensional (4D+) Geometry and Fractals

Speaker: Magnus

magnus@ef.gy

@jyujinX

C++ sources for demo segments:

https://github.com/ef-gy/topologic

Blog series:

https://ef.gy/linear-algebra

Scott Draves, Erik Reckase,

The Fractal Flames Algorithm: http://flam3.com/flame_draves.pdf

Motivationals from *Fringe*.