

Express Yourself: Creative Coding and Generative Art

School of Computing // Seminar Series

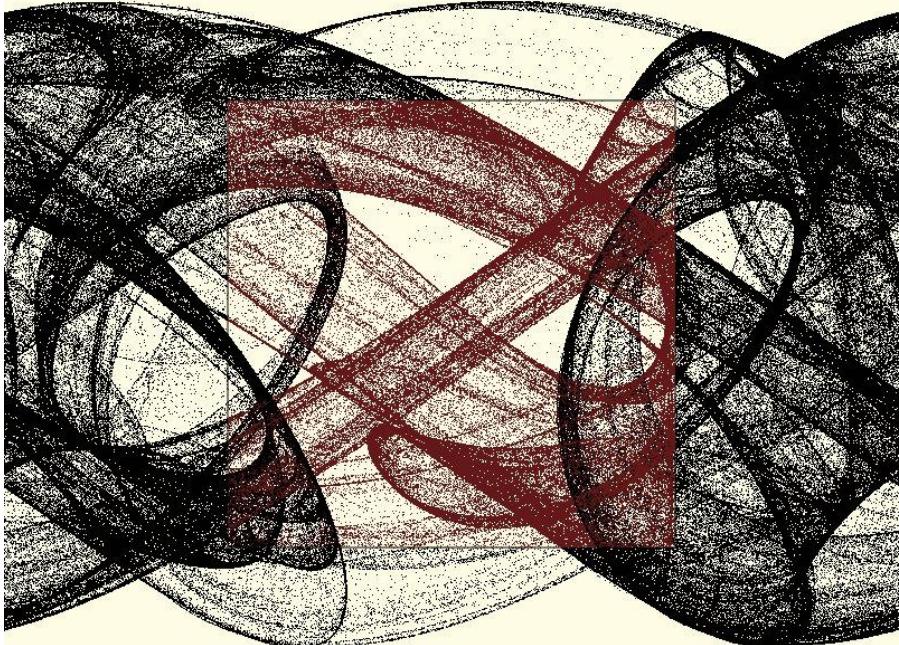
Fredericks / 2024

<http://efredericks.net>

<https://discord.gg/pNaTDKH>

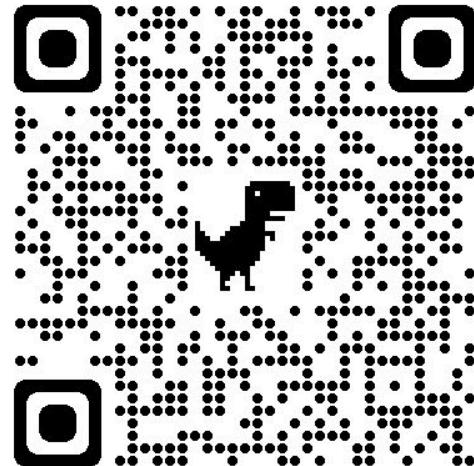






Strange attractor - colored based on location

<https://editor.p5js.org/frederer/sketches/2HV0XJKmJ>



Things to talk about

Creative coding

Generative art

Generative art research



FIRST

What comes to your mind when you think of
creative coding?

How about **generative art?**



Creative coding

All about discovery and exploration!

Or → writing algorithms and having
an **immediate and visual output**

Often used to learn coding!

Characterized by abstractions to
facilitate learning or **understanding**

Popularized by this guy →
Daniel Shiffman - TheCodingTrain



Examples of Creative Coding

<https://p5js.org/examples/>

p5.js

- Game of life
- L-systems
- Cellular automata
- Particle systems

Possibilities are endless!

Home

Editor

Download

Donate

Get Started

Reference

Libraries

Learn

Teach

Examples

Contribute

Books

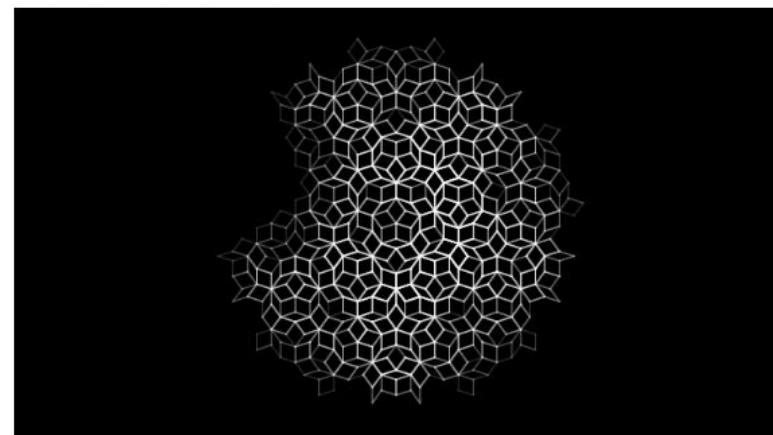
Community

Showcase

Penrose Tiles

< Back to Examples

This is a port by David Blitz of the "Penrose Tile" example from processing.org/examples



run reset copy

Creative coding outputs

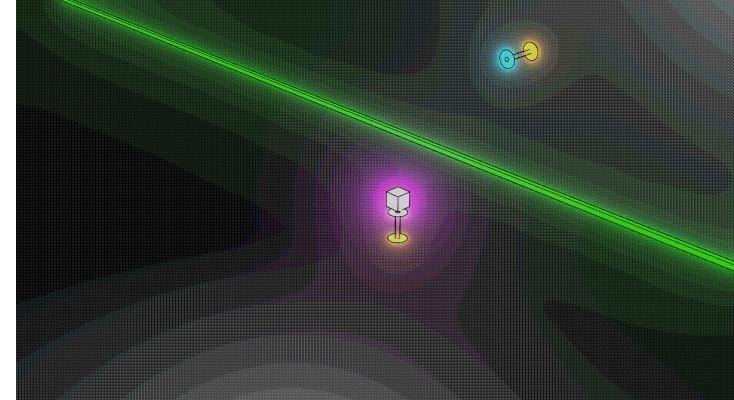
Is a digital display our only form of output?

Absolutely not!

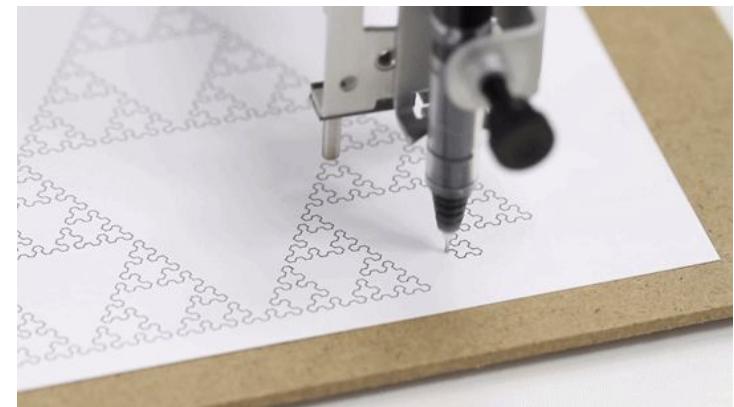
- Possible to take your creations and make them a reality!
 - Pen plotting, 3D printing, etc.



https://mary.codes/blog/art/3d_printing_generative_art_with_p5_and_blender/



Blender - physics + post-processing



<https://prostheticknowledge.tumblr.com/post/153869265156/axidraw-v3-new-version-of-precise-plotter-drawing>

How can we do this?

Really, any language that supports visuals can be used

Application-based:

- **Python, C++, Java** → all support writing to windows

Web-based:

- JavaScript + HTML → write to <canvas>
 - Shaders → write to <canvas> *much faster*
- **p5js** → abstraction over JavaScript (to write to <canvas>)
 - Horribly slow, but much quicker to prototype!
 - Also supports shaders
 - And very well-documented

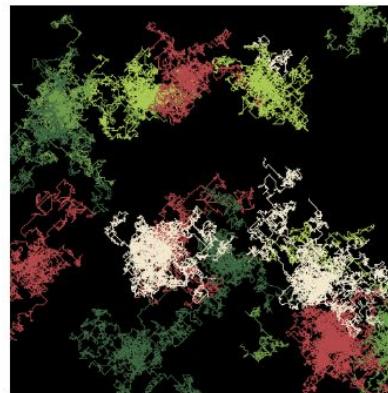
All you really need to do is:

Have an algorithm, output, or dataset you want to *visualize*

- What is this visualization going to show?
- To represent?

Have a method for representing it in a visual manner

- How are we going to draw this *thing* in such a way that it will have **meaning**?



(e) Drunkard's Walk



Generative art

<https://editor.p5js.org/frederer/sketches/DbWjEErKy>

Creative coding - but aesthetically pleasing

Take those visualizations and make them works of art!

Tangentially - there is also generative AI

- Midjourney, DALL-E, etc.
- Not talking about that today



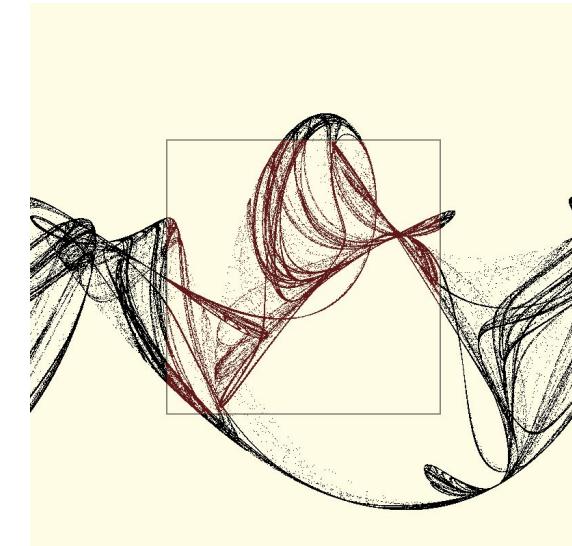
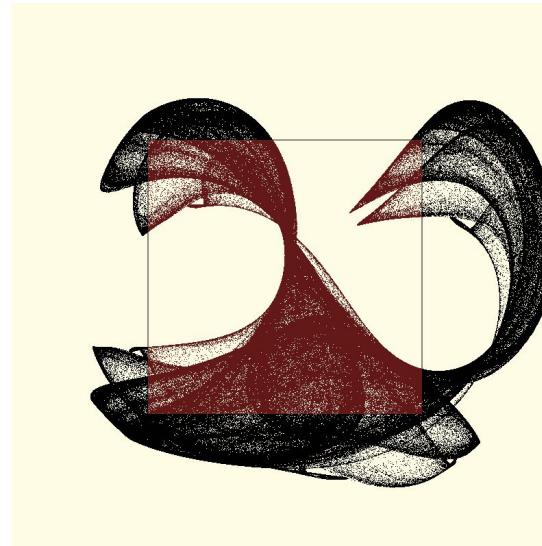
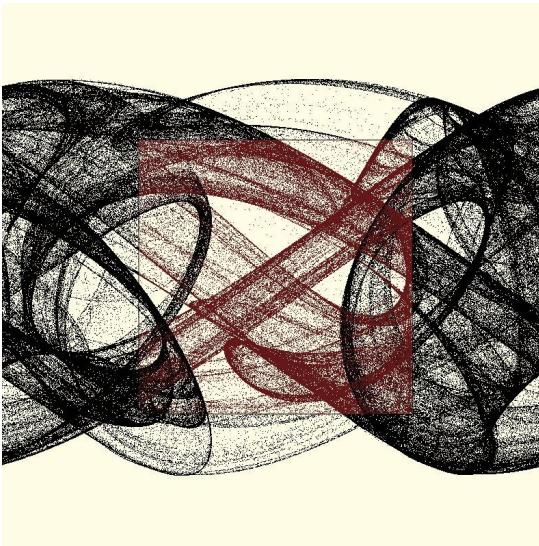
Flow field visualization

And then ... random numbers

Same algorithm → different outputs

How do we know which parameters to use?

- *More importantly, how do we revisit old outputs?*



```
function clifford(x, y) {  
    let _x = sin(a * y) + c * cos(a * x);  
    let _y = sin(b * x) + d * cos(b * y);  
    return { x: _x, y: _y };  
}
```

Generative <X>

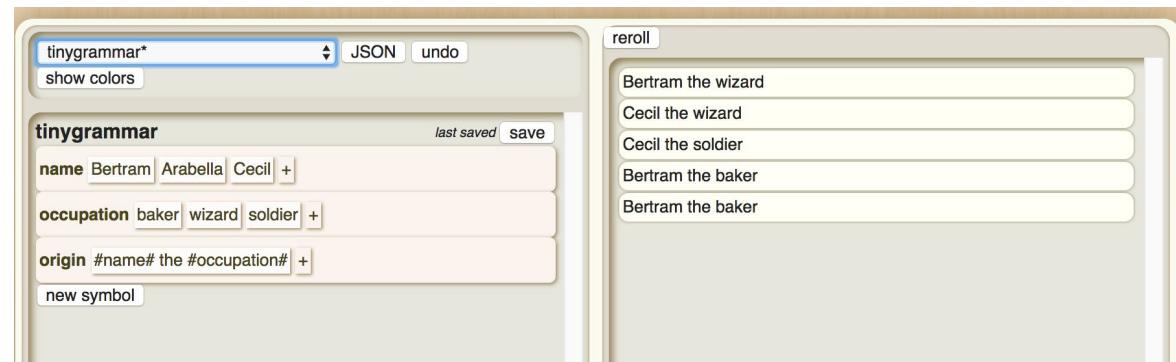
We don't need to simply be making nifty looking things on a screen...

Could be generative music (or visualizers)

- <https://junshern.github.io/algorithmic-music-tutorial/part1.html>

Could be storytelling

A screenshot of the Tracery website. At the top, there's a navigation bar with 'Tutorial: Tracery' and social links. Below it is a main content area with a pink header titled 'Tutorial: Tracery'. It explains what Tracery is, its history, and how to use it. A 'Generating text!' section shows a code editor with a grammar and a preview window showing generated text like 'animal: ["unicorn", "raven", "sparrow", "scorpion", "coyote", "eagle", "owl", "izard"]'. There are also sections for 'symbols' and 'rules'.

A screenshot of the tinygrammar interface. It has a top bar with 'tinygrammar*', 'JSON', 'undo', and a 'show colors' button. The main area is titled 'tinygrammar' and contains three rows of rules: 'name' (with values 'Bertram', 'Arabella', 'Cecil'), 'occupation' (with values 'baker', 'wizard', 'soldier'), and 'origin' (with a '#name# #occupation#'). Below these is a 'new symbol' button. To the right, a 'reroll' button is followed by a list of generated names and occupations: 'Bertram the wizard', 'Cecil the wizard', 'Cecil the soldier', 'Bertram the baker', and 'Bertram the baker'.

Onto the research...



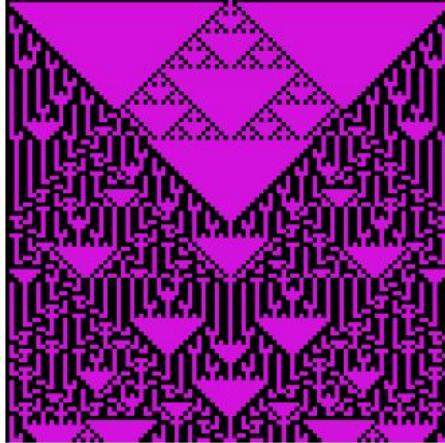
Art + genetic algorithms = research

Main concept:

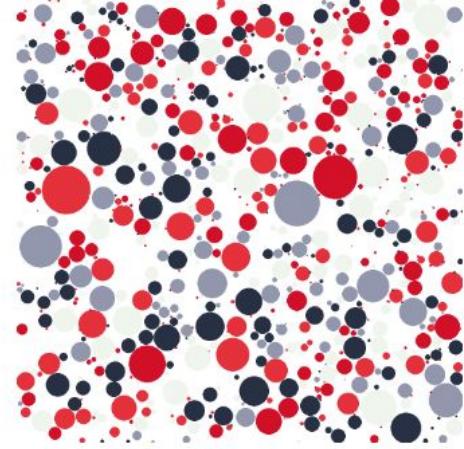
- Take all of these disparate generative art techniques that we like to make
- Combine them with a search algorithm
- ???
- New art!



(a) Stippled



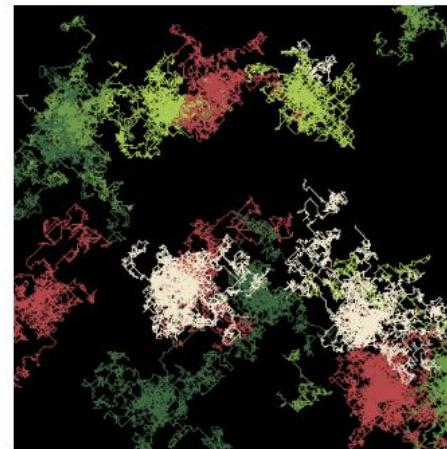
(b) Cellular Automata



(c) Circle Packing

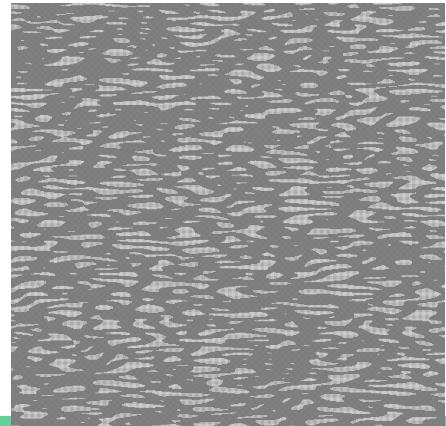
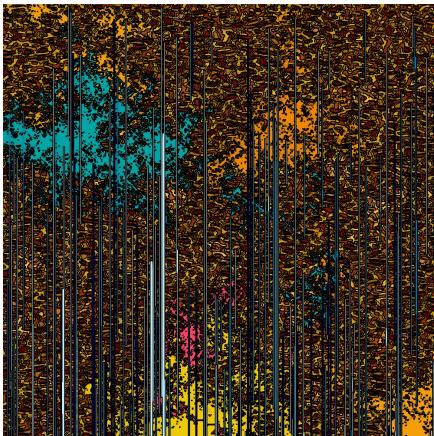
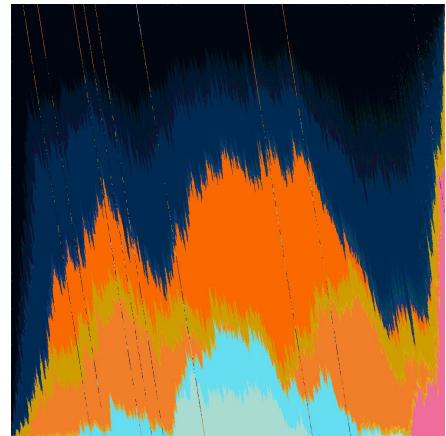


(d) Flow Field



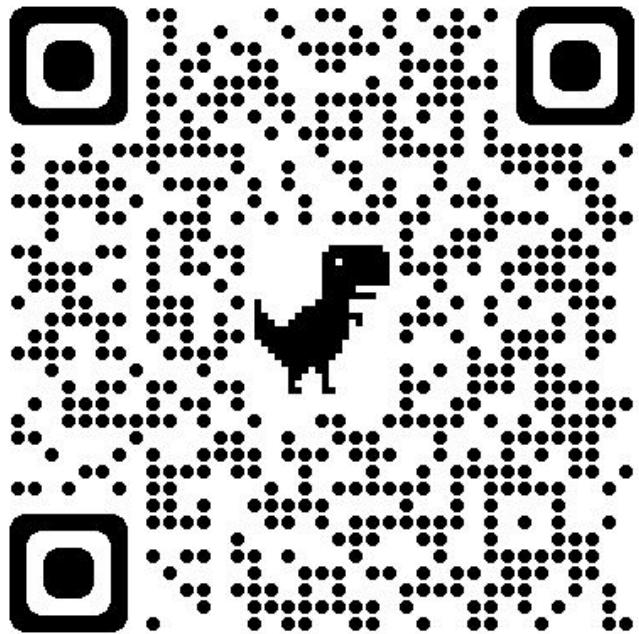
(e) Drunkard's Walk

Generative art research - GenerativeGI



1 workshop paper (best research award)
1 invited journal extension (accepted for pub)

link to blog post →



Generative GI

Generative art + GI (genetic improvement)

What is GI then?

- Evolutionary computation applied to *source code* to improve program
 - EC: see last week's talk
- Genome is the source code itself

What is our source code?

- A "drawing program"

What is our fitness function?

- **The hard part**

```
rules = {
    'ordered_pattern': ['#techniques#'],
    'techniques': ['#technique#', '#techniques#,#technique'],
    'technique': [
        'stippled:', 'wolfram-ca:#palette#',
        'flow-field:#flow-field-type#:#flow-field-zoom#',
        '#flow-field-type#:#palette#:#flow-field-zoom#',
        'flow-field-type': ['edgy', 'curves'],
        'flow-field-zoom': [str(x) for x in np.arange(0.001, 0.5,
                                                     0.001)],
        'palette': [[ '#ff00ff', '#ff0000', '#00ff00'], ...],
    ...
}
```

```
flatten('techniques') = flow-field:edgy:#ff00ff #ff0000
#00ff00:0.025
```

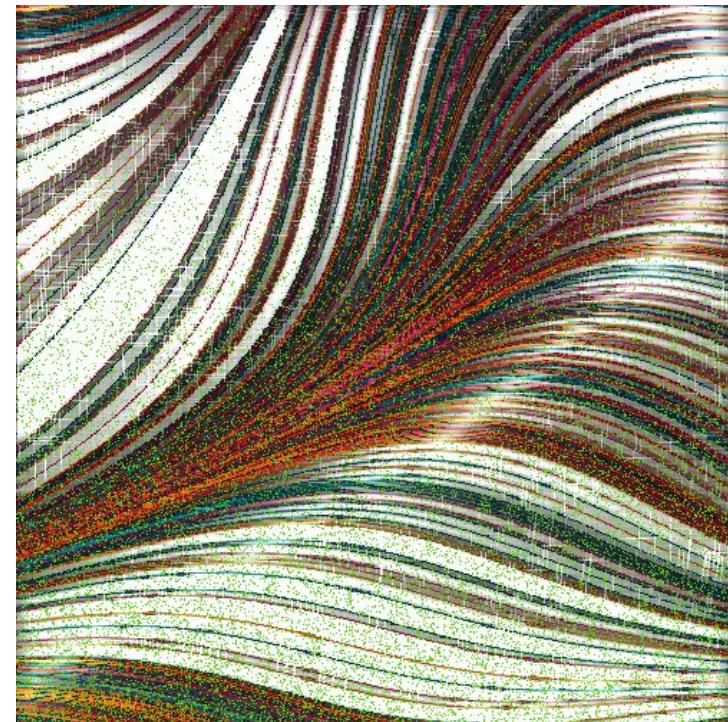
Resulting in something like this:

```
flow-field-2:65DEF1 A8DCD1 F06C9B  
F96900 F17F29:curvy:474:2,
```

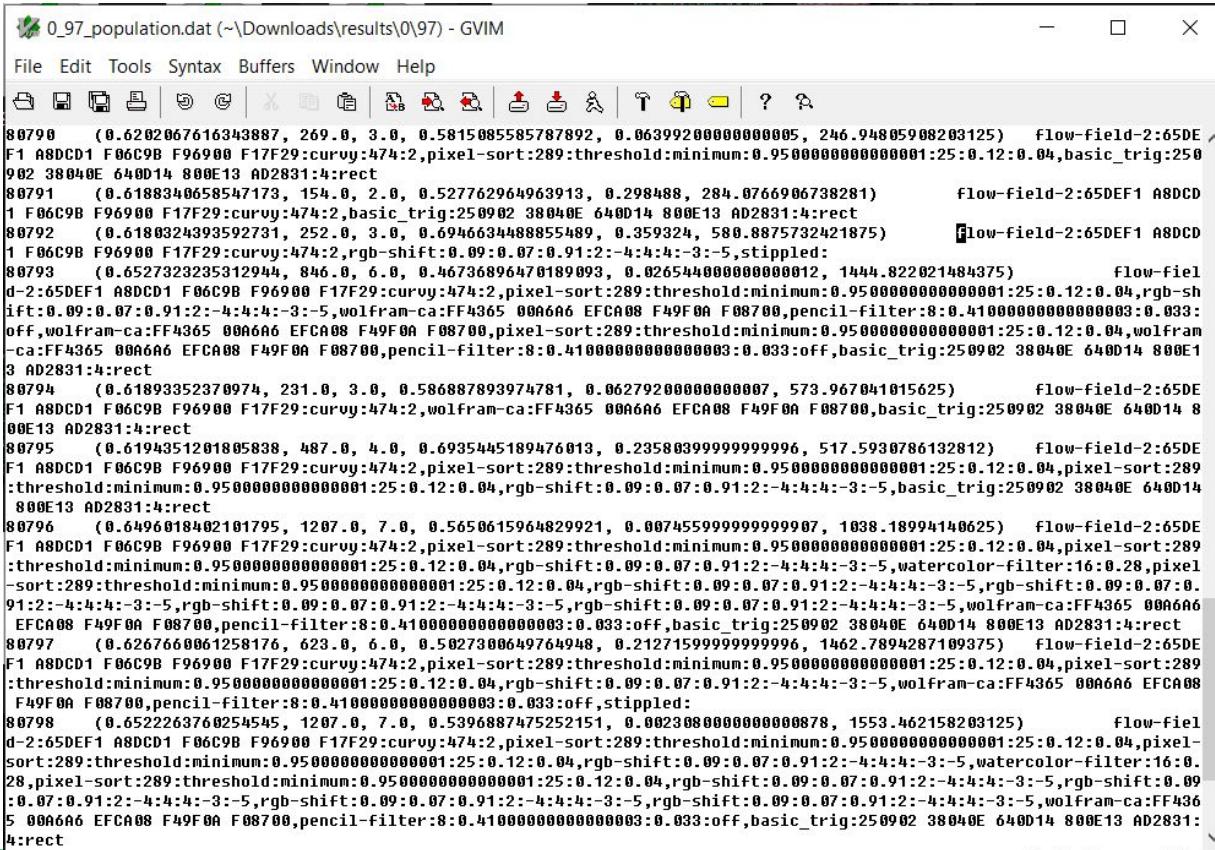
```
pixel-sort:289:threshold:minimum:0.9500  
0000000001:25:0.12:0.04,
```

```
pencil-filter:8:0.4100000000000003:0.0  
33:off,
```

```
stippled:
```



(Looks like this behind the scenes)



The screenshot shows a GVIM window displaying a file named "0_97_population.dat". The file contains numerous lines of text, each representing a flow-field element. The elements are defined by their position (x, y), size (width, height), color (red, green, blue), and various parameters such as pixel-sort, threshold, minimum values, and trig types. The text is in a monospaced font and is too long to be fully copied here.

```
0_97_population.dat (~\Downloads\results\0\97) - GVIM
File Edit Tools Syntax Buffers Window Help
File Edit Tools Syntax Buffers Window Help
80798 (0.6202067616343887, 269.0, 3.0, 0.5815085585787892, 0.06399200000000005, 246.94805908203125) flow-field-2:65DE ^
F1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,basic_trig:250
902 38040E 640D14 800E13 AD2831:4:rect
80791 (0.6188340658547173, 154.0, 2.0, 0.527762964963913, 0.298488, 284.0766906738281) flow-field-2:65DEF1 A8DCD
1 F06C9B F96900 F17F29:curvy:474:2,basic_trig:250902 38040E 640D14 800E13 AD2831:4:rect
80792 (0.6180324393592731, 252.0, 3.0, 0.6946634488855489, 0.359324, 580.8875732421875) flow-field-2:65DEF1 A8DCD
1 F06C9B F96900 F17F29:curvy:474:2,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,stippled:
80793 (0.6527323235312944, 846.0, 6.0, 0.4673689647018909, 0.0265440000000000012, 1444.822021484375) flow-fiel
d-2:65DEF1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-sh
ift:0.09:0.07:0.91:2:-4:4:4:-3:-5,wolfram-ca:FF4365 00A6A6 EFCA08 F49F0A F08700,pencil-filter:8:0.4100000000000003:0.033:
off,wolfram-ca:FF4365 00A6A6 EFCA08 F49F0A F08700,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,wolfram
-ca:FF4365 00A6A6 EFCA08 F49F0A F08700,pencil-filter:8:0.4100000000000003:0.033:off,basic_trig:250902 38040E 640D14 800E1
3 AD2831:4:rect
80794 (0.61893352370974, 231.0, 3.0, 0.586887893974781, 0.06279200000000007, 573.967041015625) flow-field-2:65DE
F1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,wolfram-ca:FF4365 00A6A6 EFCA08 F49F0A F08700,basic_trig:250902 38040E 640D14 8
00E13 AD2831:4:rect
80795 (0.6194351201805838, 487.0, 4.0, 0.69354545189476013, 0.23580399999999996, 517.5930786132812) flow-field-2:65DE
F1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,pixel-sort:289
:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,basic_trig:250902 38040E 640D14
800E13 AD2831:4:rect
80796 (0.6496018402101795, 1207.0, 7.0, 0.5650615964829921, 0.00745599999999997, 1038.18994140625) flow-field-2:65DE
F1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,pixel-sort:289
:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,watercolor-filter:16:0.28,pixel
-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,rgb-shift:0.09:0.07:0.
91:2:-4:4:4:-3:-5,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,wolfram-ca:FF4365 00A6A6
EFCA08 F49F0A F08700,pencil-filter:8:0.4100000000000003:0.033:off,basic_trig:250902 38040E 640D14 800E13 AD2831:4:rect
80797 (0.6267660061258176, 623.0, 6.0, 0.5027300649764948, 0.21271599999999996, 1462.7894287109375) flow-field-2:65DE
F1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,pixel-sort:289
:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,wolfram-ca:FF4365 00A6A6 EFCA08
F49F0A F08700,pencil-filter:8:0.4100000000000003:0.033:off,wolfram-ca:FF4365 00A6A6 EFCA08 F49F0A F08700,pencil-filter:8:0.
4100000000000003:0.033:off,basic_trig:250902 38040E 640D14 800E13 AD2831:4:rect
80798 (0.6522263760254545, 1207.0, 7.0, 0.539688747525151, 0.0023080000000000878, 1553.462158203125) flow-fiel
d-2:65DEF1 A8DCD1 F06C9B F96900 F17F29:curvy:474:2,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,pixel-
sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,watercolor-filter:16:0.
28,pixel-sort:289:threshold:minimum:0.9500000000000001:25:0.12:0.04,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,rgb-shift:0.09:
0.07:0.91:2:-4:4:4:-3:-5,rgb-shift:0.09:0.07:0.91:2:-4:4:4:-3:-5,wolfram-ca:FF436
5 00A6A6 EFCA08 F49F0A F08700,pencil-filter:8:0.4100000000000003:0.033:off,basic_trig:250902 38040E 640D14 800E13 AD2831:
4:rect
```

The **real** problem - fitness

How do we turn human preference into something measurable?

How do you objectively measure an artistic output?

- Can we turn to machine learning?
 - Bias in datasets?
- Can we turn to LLMs?
 - Copyright issues?
- Can we just ask a bunch of humans?
 - Different preferences?

Fitness functions (or, how we currently check "goodness")

$ff_{min(genome)}$	Minimize the number of duplicate genes across individuals.
$ff_{max(techniques)}$	Maximize the diversity of included techniques in a genome.
$ff_{max(RMS)}$	Maximize the pixel difference between individual image objects using an RMS difference metric.
$ff_{max(Chebyshev)}$	Maximize the pixel difference between individual image objects using a Chebyshev difference metric.
$ff_{max(negative\ space)}$	Minimize the difference between a target “negative space” and the actual “negative space” within an image (70% was the target percentage used during evaluation).

Table 1: *GenerativeGI* fitness functions.

Now the question that is just **burning** through all your collective minds...

What happens if these fitness functions fight with each other?

- Traditionally these would be **weighted**
- Or, a multi-objective evolutionary algorithm used...
 - But that would take even longer to run

$ff_{min}(genome)$
$ff_{max}(techniques)$
$ff_{max}(RMS)$
$ff_{max}(Chebyshev)$
$ff_{max}(negative\ space)$

Lexicase selection

(or, what Prof. Moore has been researching a lot of)

Standard genetic algorithms use one or few fitness functions (and weight them)

- Difficult to focus on multiple fitnesses!

Lexicase - **many-objective** search

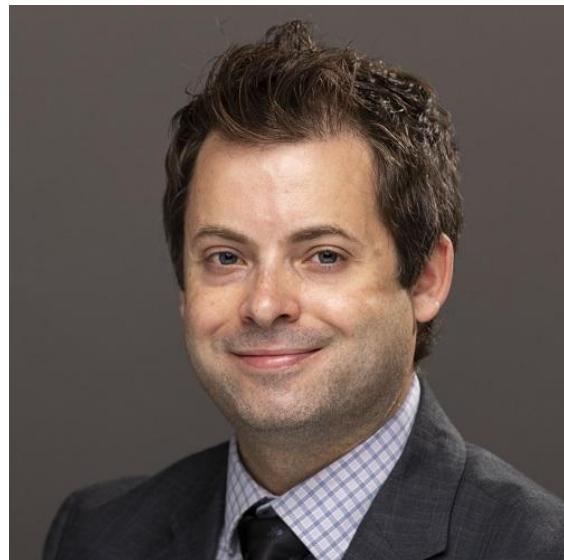
- Randomly shuffle fitness functions and evaluate all
- If one individual is significantly better than another individual then it is **selected** for procreation

Our goal: incorporate as many fitness functions as necessary to get to "real art"

What are we "looking for?"

For this, a glitch art aesthetic

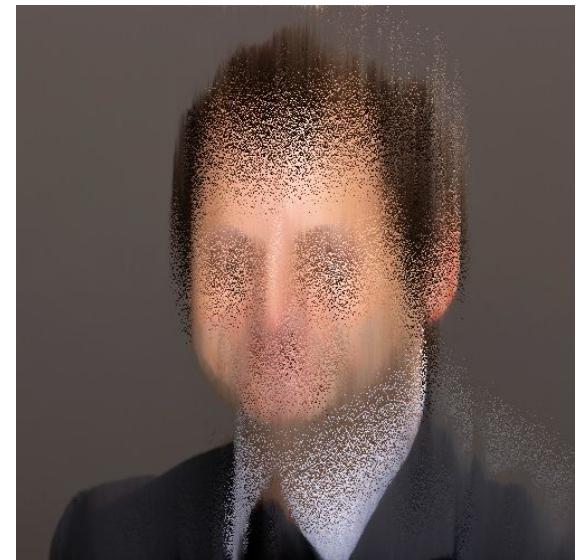
0 ticks



72000 ticks



120000 ticks



<https://happyCoding.io/examples/p5js/images/pixel-sorter>

Outcomes?

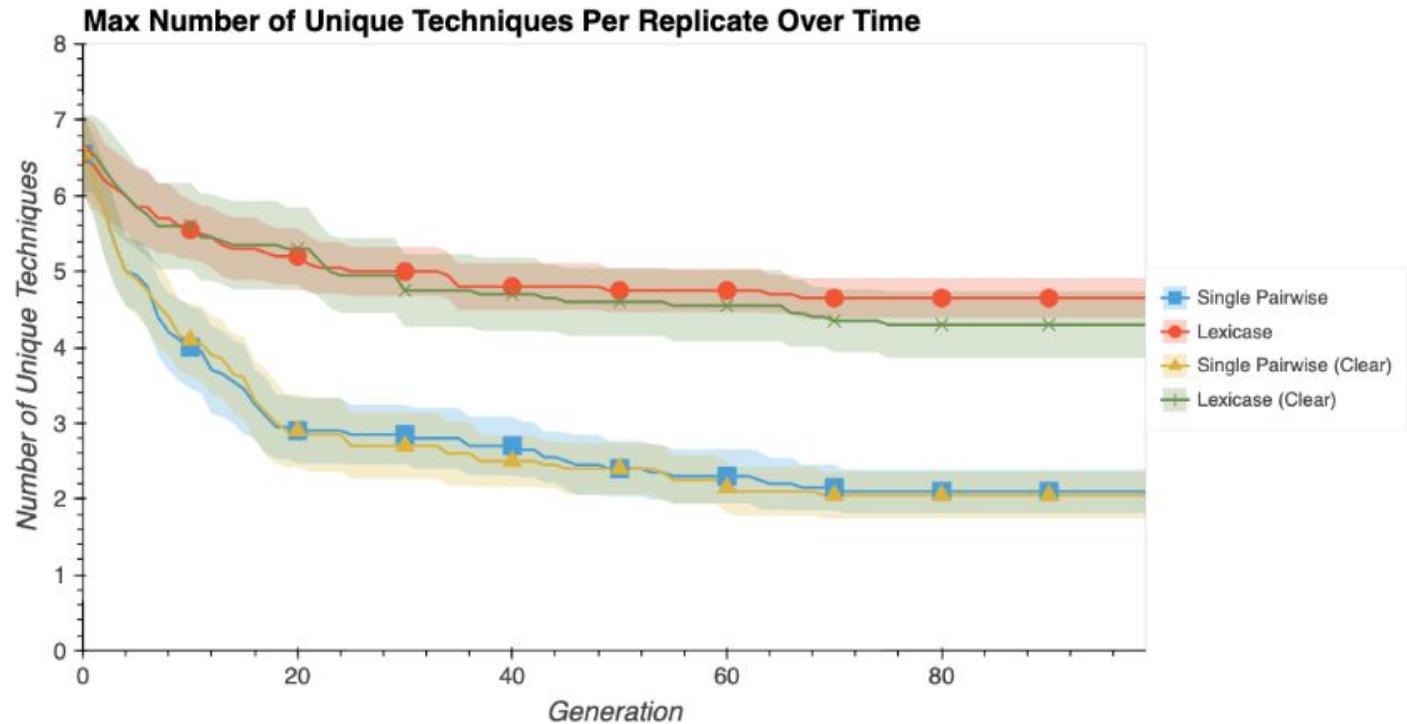


Fig. 16: Population maximum of the number of unique techniques in an individual's genome over time across replicates. Shaded areas represent 95% confidence intervals.

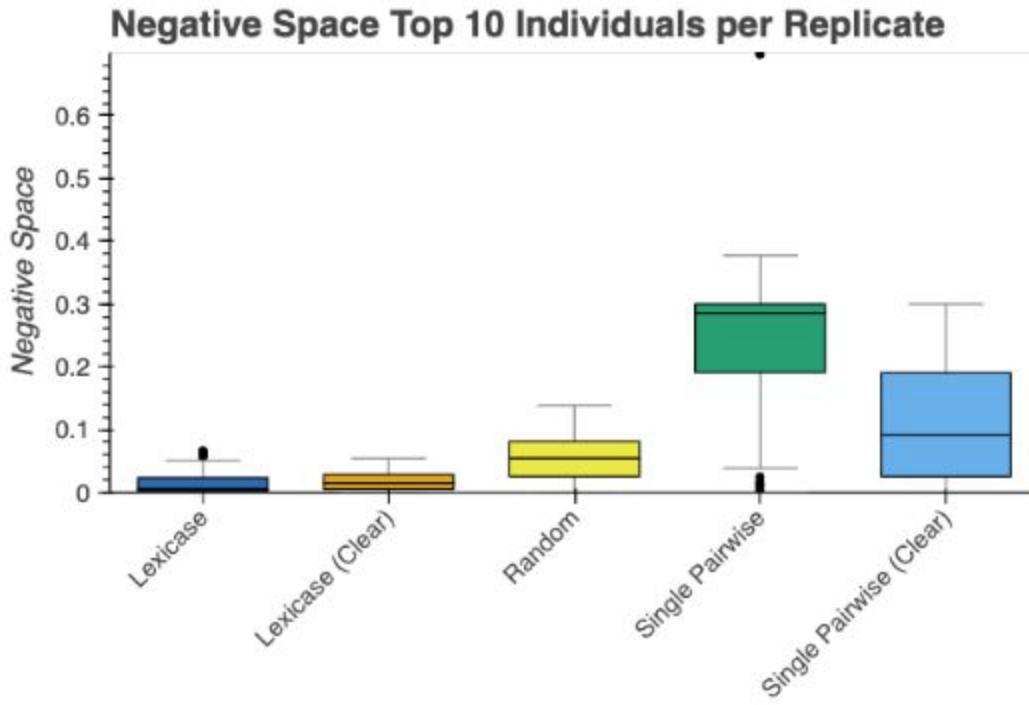
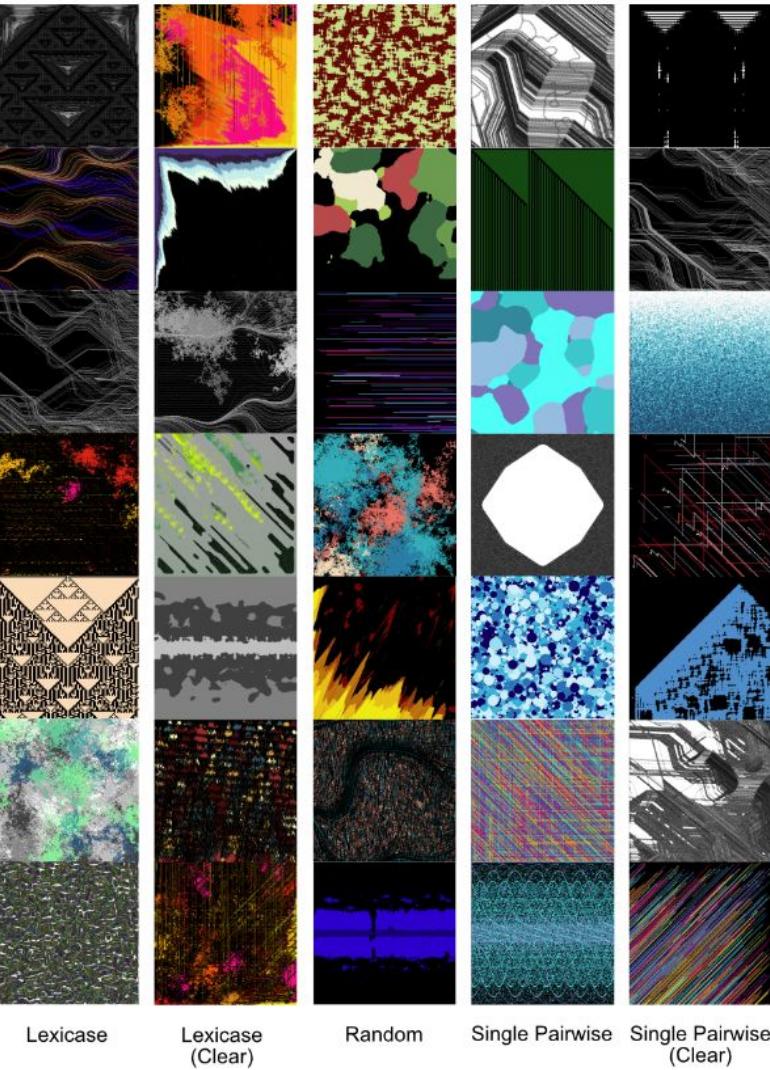
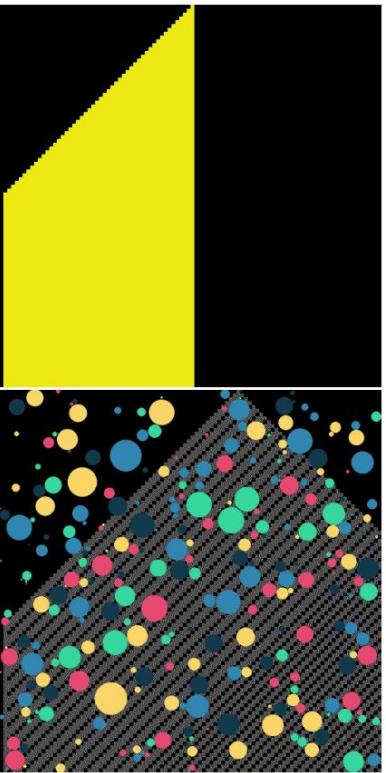
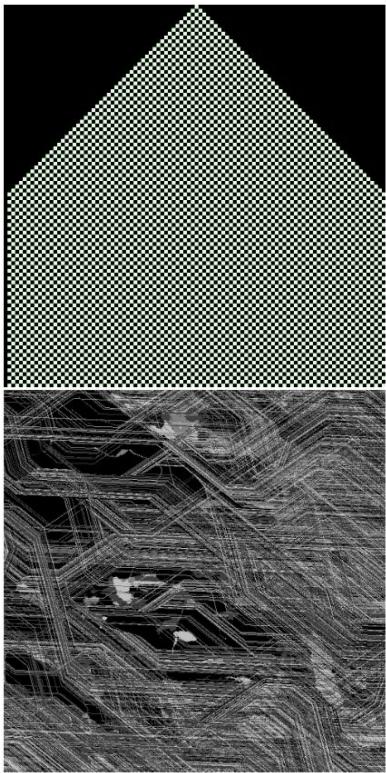


Fig. 14: Negative Space score for the top 10 individuals based on their negative space score per replicate across treatments. Lower scores indicate more successful individuals.



And, if you wish to try it out

<https://github.com/GI2023-GenerativeGI/GI2023/tree/ASE-GI-Extension>

Next steps (or, work in progress)

Two active projects

- 1) Incorporate a classifier to determine if our outputs are close to "real art"
- 2) Live display where users can interact with the evolutionary process

AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT



<https://efredericks.github.io/qvsu-cis367/demos/voronoi-sin.html>

Computing Seminar on March 14th: Voronoi Diagrams

Dividing Space Closest to Specific Points

Speakers: L Dettling and Dan Dietsche

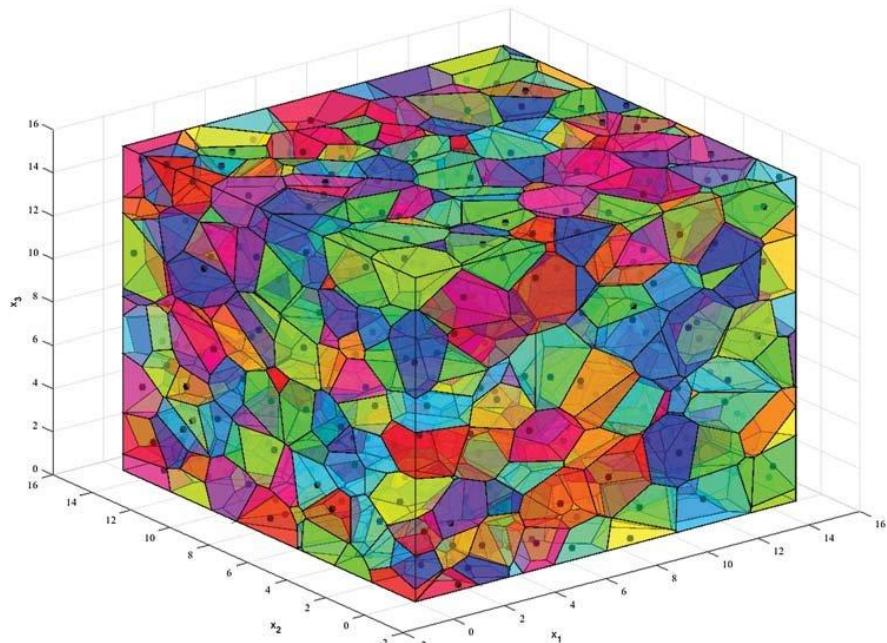


Fig 1. 3-D Voronoi Diagram

https://www.researchgate.net/figure/Initial-3D-Voronoi-diagram_fig2_351028108

Given a space and a set of n points, one divides the space such that all the space is grouped by which point is closest.

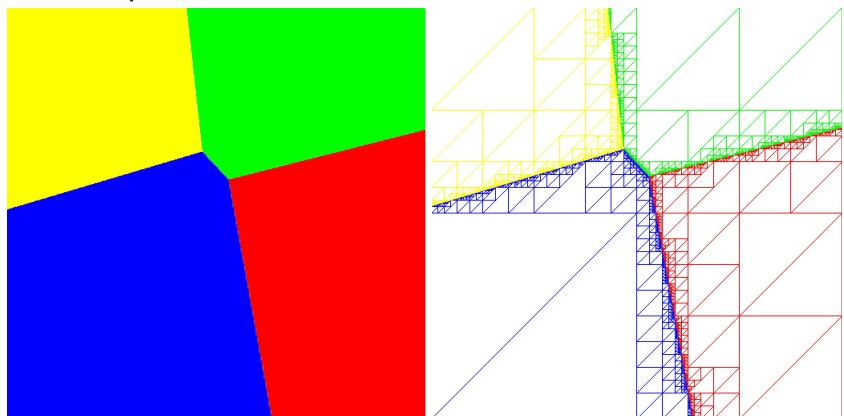


Fig 2. 2-D Voronoi Diagram with 4 seeds.