



PORTFOLIO

2020

2024

Dan Kenerson

Front:
Lorenz strange attractor,
front-perspective view

Lorenz Equation:
 $dx = (a \cdot (y - x)) \cdot dt$
 $dy = (x \cdot (b - z) - y) \cdot dt$
 $dz = (x \cdot y - c \cdot z) \cdot dt$





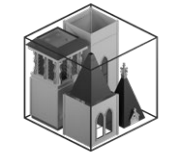
On the Creative Process:

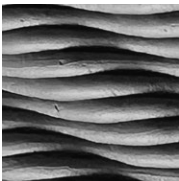

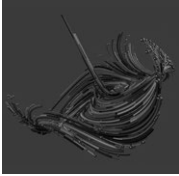
“Wisdom is not a product of schooling but of the lifelong attempt to acquire it.” – Albert Einstein

I move through life creating—whether through the lens of a camera, the stroke of a pen, or the clack of a keyboard. Lessons learned in the darkroom about the interplay of light and shadow find their way into pen-plotting techniques. Algorithms developed through programming breathe life into generative architecture. My creative pursuits are not only an outlet for expression but also a constant source of learning and inspiration to explore further.

As a lifelong student of design and technology, I strive to expand my understanding with every project and every medium I engage with. This portfolio highlights how my work connects across disciplines, showcasing pivotal moments where critical concepts were discovered and applied.

Contents.

	AutoVisualizer Integrating AI for Grasshopper.	01 P06
	Redefining Home Shaping homes through conversation.	02 P12
	Reimagining the Familiar Using algorithms for conceptual iteration.	03 P18

	Shaping Code Experiments in custom slicing.	04 P22
	Reframing Color Shooting with the trichrome process.	05 P26
	Drawing Chaos Generating complexity through strange attractors.	06 P30

AutoVisualizer

Integrating AI for Grasshopper.

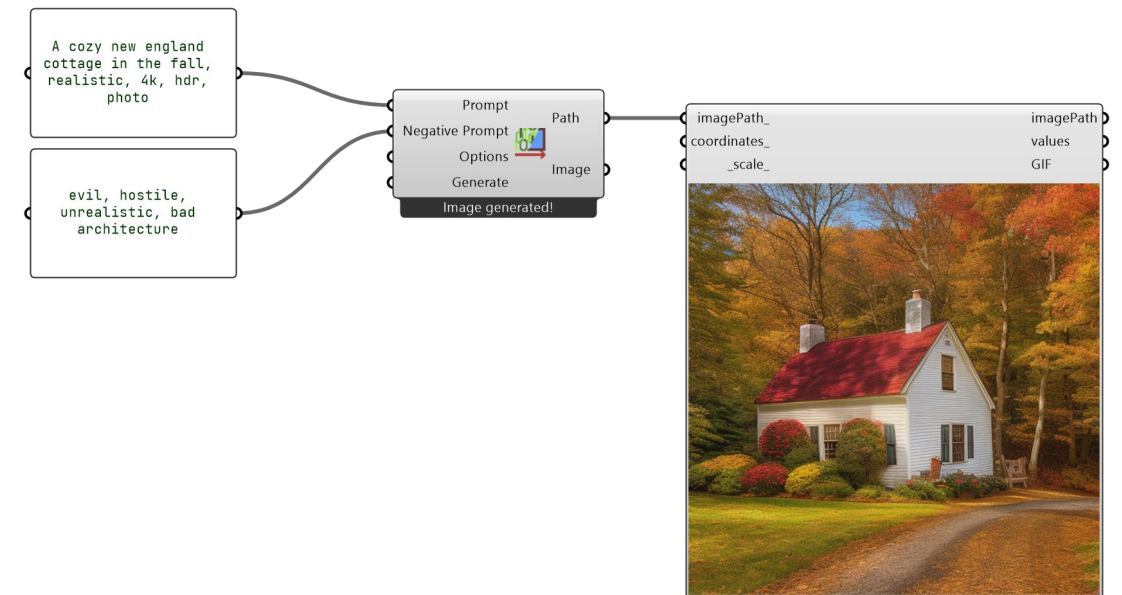
Year: 2023-2024

Tools: Grasshopper, Automatic1111 API, C#

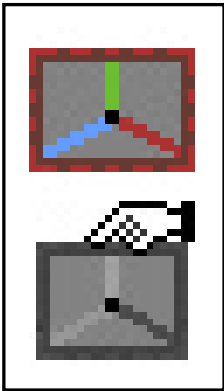
AutoVisualizer is a plugin I designed to bridge architectural design and generative AI, offering seamless integration with the Automatic1111 Stable Diffusion API. This tool allows users to generate captivating AI-driven images directly within Grasshopper, providing both ease of use and fine-grained control over the image output.

With AutoVisualizer, designers can streamline their workflow, moving effortlessly between algorithmic modeling and visualization. The plugin gathers user input, formats it for the API, and processes the results in real-time. The generated images can then be customized, manipulated, or saved for immediate use, all within the same environment.

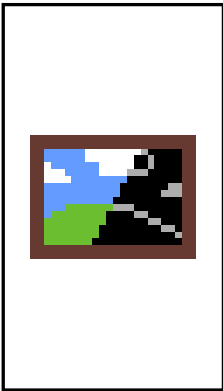
AutoVisualizer not only simplifies the image generation process but also serves as a stepping stone for exploring the potential of AI-driven creativity in architectural design.



Current Plugin Components.



Capture View Components



ControlNet Components

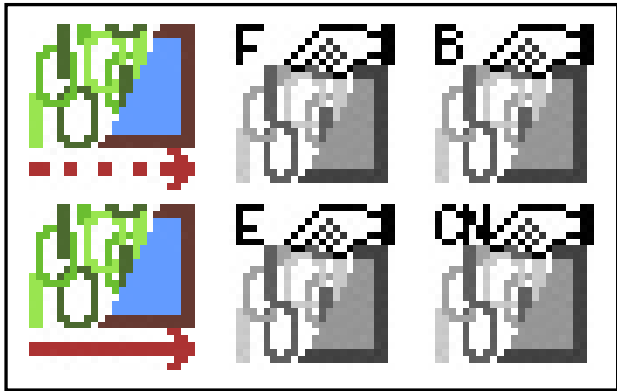


Image Generation Components

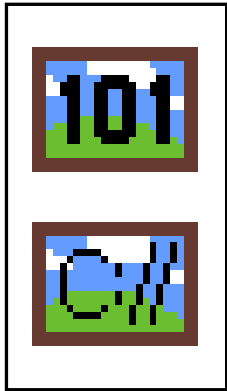


Image Viewing Components

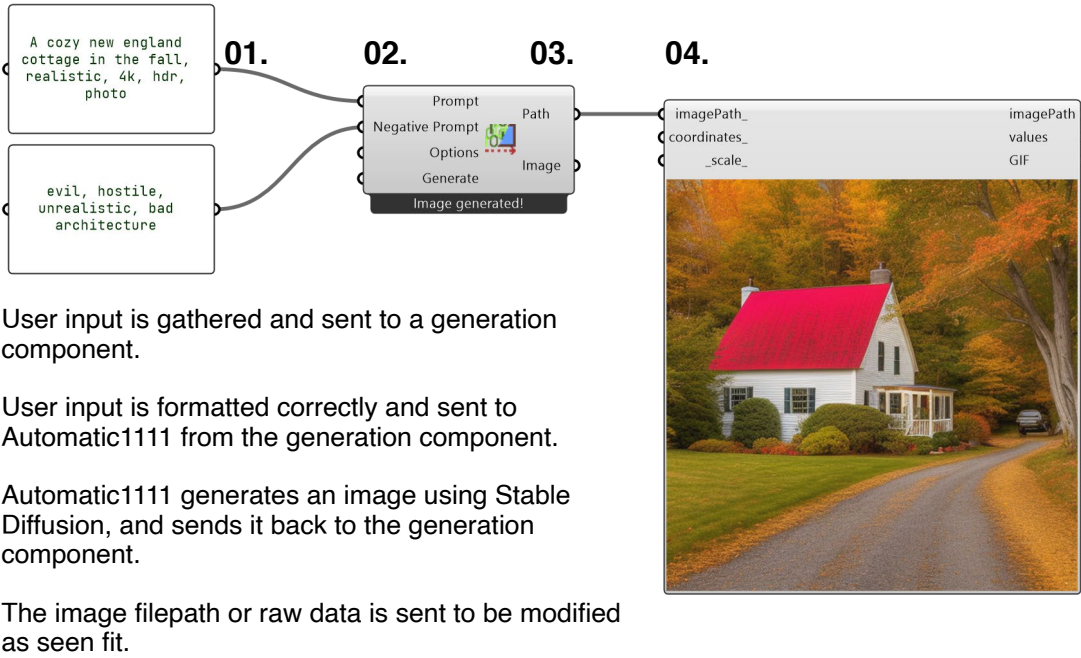
How It Works.

01. User input is gathered and sent to a generation component.

02. User input is formatted correctly and sent to Automatic1111 from the generation component.

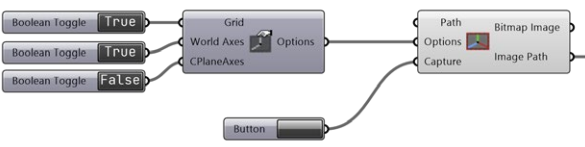
03. Automatic1111 generates an image using Stable Diffusion, and sends it back to the generation component.

04. The image filepath or raw data is sent to be modified as seen fit.



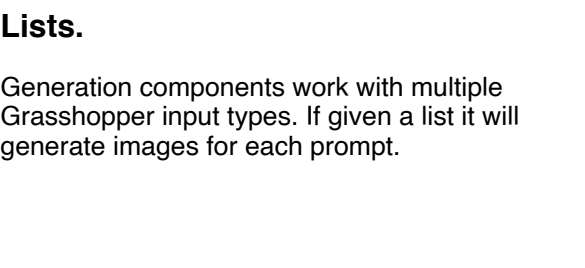
The diagram illustrates a four-step process: 1. User input (prompts like 'A cozy new england cottage in the fall, realistic, 4k, hdr, photo' and 'evil, hostile, unrealistic, bad architecture') is gathered. 2. The input is formatted and sent to the 'Automatic1111' generation component. 3. The component generates an image using Stable Diffusion. 4. The resulting image (a house with a red roof) is sent to a modification component that outputs imagePath, coordinates, and scale values.

Plugin Usage Examples.



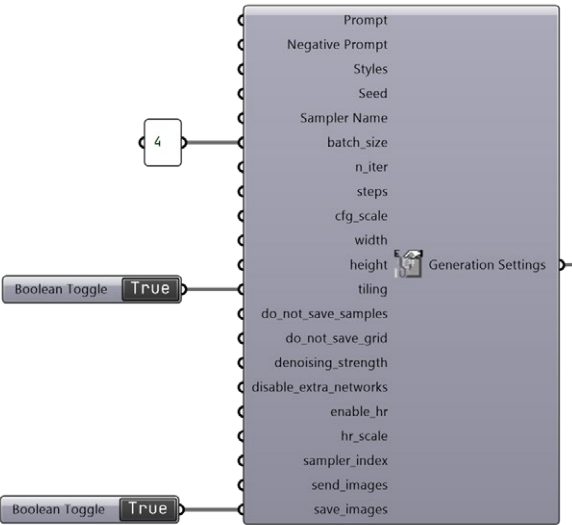
Screen Capture.

No more exporting images to use in AI generation! Capture view components make selecting your screen a breeze.



Lists.

Generation components work with multiple Grasshopper input types. If given a list it will generate images for each prompt.



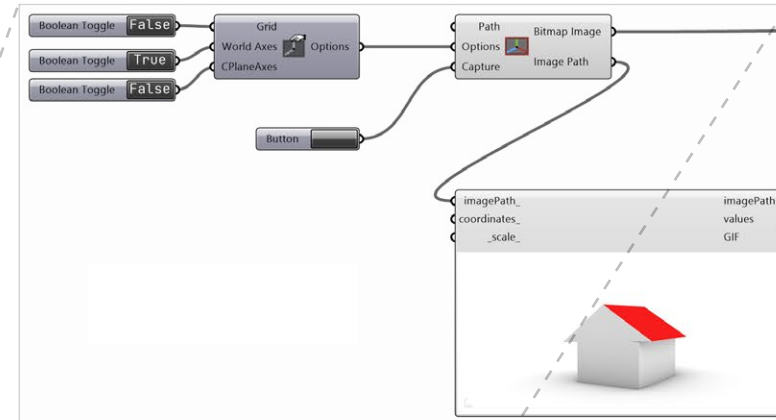
Options.

Four options components allow varying levels of customization to the generation settings.

This one, with a batch size of four, provides us with 4 images to pick between. It also makes the resulting images tileable and has Automatic1111 save them.

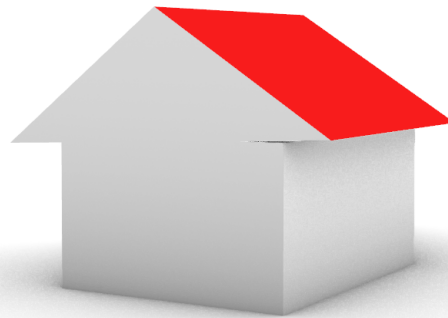
Putting It All Together.

In this example a simple model in Rhino can be visualized with only a few nodes.



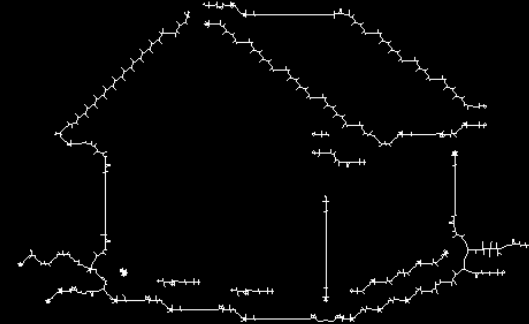
Capture View

The viewport is captured by the component, with settings for hiding the viewport grid, world axes, or CPlane axes.



ControlNet

ControlNet interprets the captured image to create an influence map for the output. Here, canny edge detection was used.



GenerateASync

Final output image, combining the text prompts and the edges defined with ControlNet.



02

Redefining Home

Shaping homes through conversation.

Year: 2022-2023

Team: Lucas Kamal, Talia Mamayek, Max Wojtas

Tools: Revit, Illustrator

This project began with conversations. Our team engaged with public housing residents, college students, and local stakeholders to understand the evolving needs of modern families. These insights revealed a demand for housing that supports diverse living arrangements, fosters community, and feels like home.

The resulting design utilizes a point-block typology to maximize space efficiency while creating a warm, residential atmosphere. By incorporating flexible layouts and shared spaces, the project offers multi-generational and nontraditional families the opportunity to connect while maintaining privacy.

Rooted in user feedback, Redefining Home demonstrates how thoughtful, collaborative design can shape housing solutions that prioritize both individuality and community.

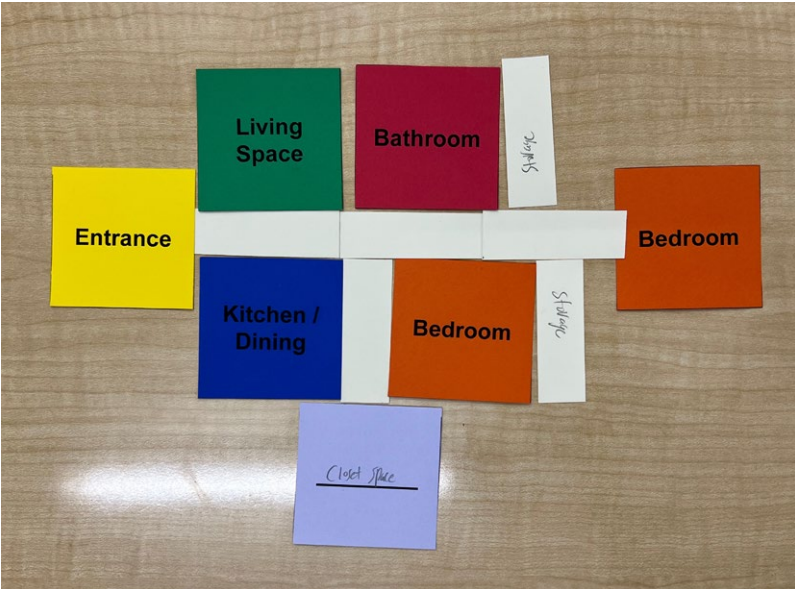


Render by Lucas Kamal

A Layout That Works.

Focus groups were held in order to gather data about what people like and dislike about their current living arrangements. They were then tasked with creating ideal apartment layouts for different living situations.

Time spent in each location, as well as the importance, sentiment and personal experience were all recorded during the focus groups as well.



Above: An example of one participant’s ideal two-bed, one-bath apartment

Data Analysis.

Popular Connections.

The most popular connections present in the participants’ layouts coincidentally align with a gradient of spaces from most public to private.

Waking Hours.

Where time is spent is important. These are the spaces with the highest average use per week among participants.

What’s Important.

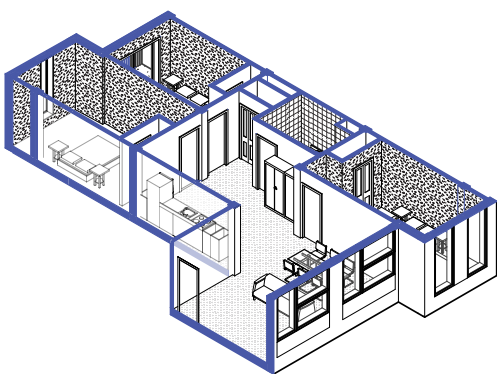
Some spaces, while used less, are valued higher. This is based on average personal perceptions, rating spaces on a scale of 1-10.



Creating Typologies.

After data analysis was completed, three apartment types were created. Our building has single-bed, three-bed, and five-bed apartments. The latter is designed to be for multigeneraitonal families or co-living.

These aimed to address wants and needs mentioned in the focus groups, like private access to laundry and workspaces seperate from the normal living space.



Three-Bed Apartment.

Five-Bed Apartment.

Single-Bed Apartment.



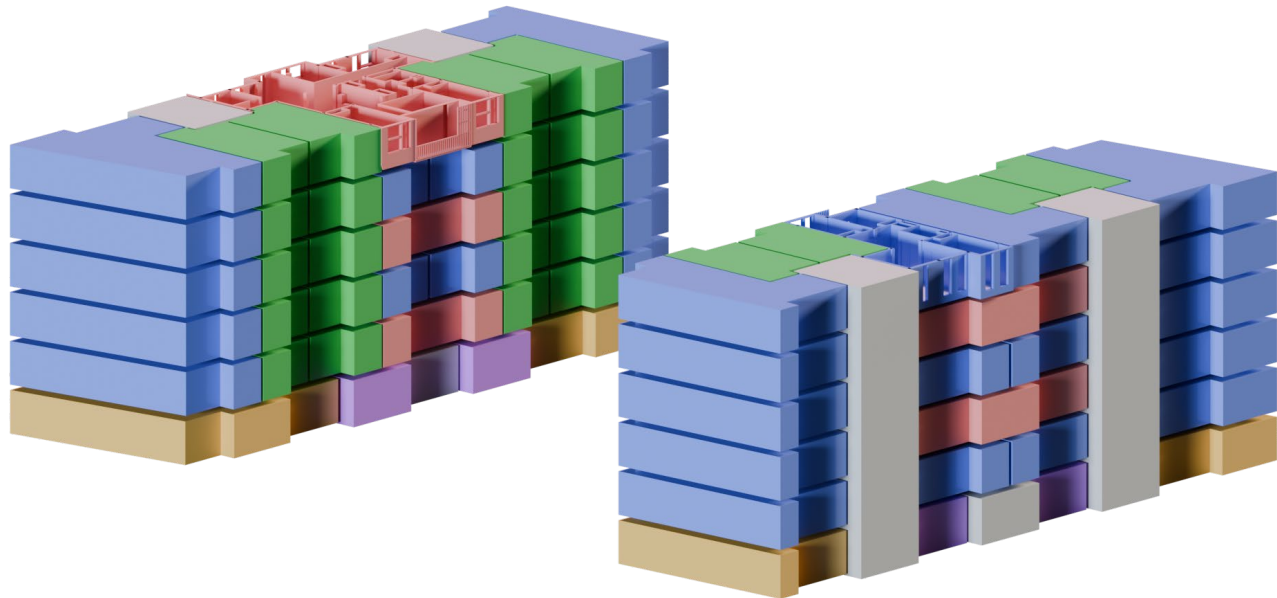
Point Access Block.

Integral to the design of our system is the use of point access block. This is a form of building access defined by a single stair servicing multiple floors of apartments. In our design, to conform with second egress laws in the U.S. the elevator is fire rated, and seperate from the exit stairs.

Typical building floorplan.



Building layout.

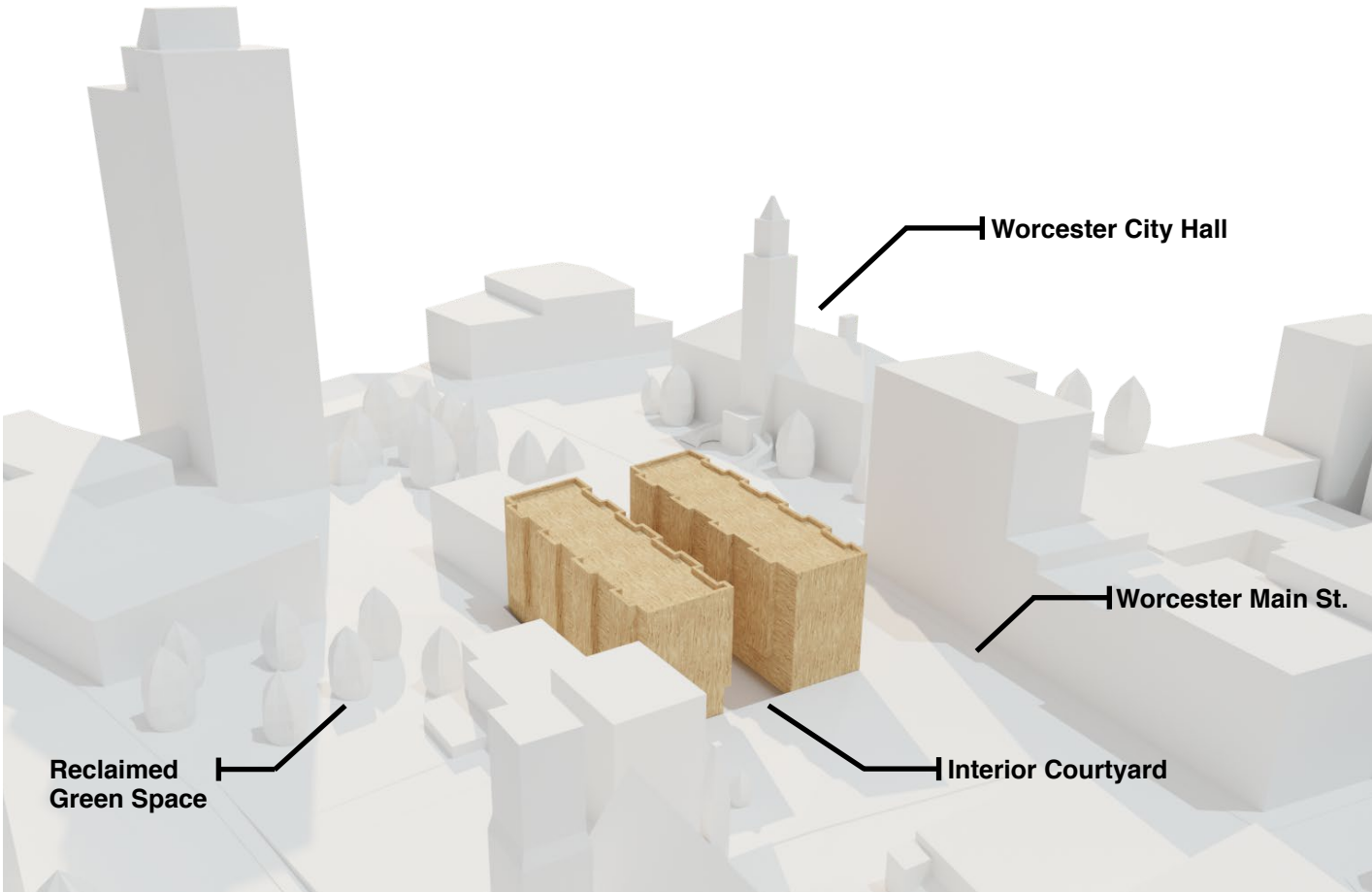


Key: Single-Bed, Three-Bed, Co-Living (5-Bed), Community Space, Retail

Scaling the Design.

Point access block significantly reduces the area used by circulation in the building, by not needing hallways that span the entire length. Instead, smaller, more private landings, shared by just four units create an urban “front porch.” This design choice not only fosters a sense of community, but also enables cross-ventilation in the units; both the 3-bed and 5-bed layouts span the entire floor-plate for optimal airflow and natural light.

Located in Worcester, MA, the proposed building leverages data-driven design to enhance urban living. Community and retail spaces on the first floor encourages the use of third spaces and neighborhood engagement. An interior courtyard and reclaimed green space further occupant well-being and sense of place.



03

Reimagining the Familiar

Using algorithms for conceptual iteration.

Year: 2024

Tools: Blender, Python, JavaScript, Three.js

Of the many different ways to achieve procedural generation wave function collapse has captivated me for a long time. The algorithm works by limiting possibilities within a space, determined by the spaces around it. Imagine sudoku: each cell is bound by rules based on its row, column and square. Uncertainty narrows as the grid fills until a single correct solution is evident.

I first experimented with this idea 7 years ago—a crude terrain generator in Python that filled tiles based on a ruleset.

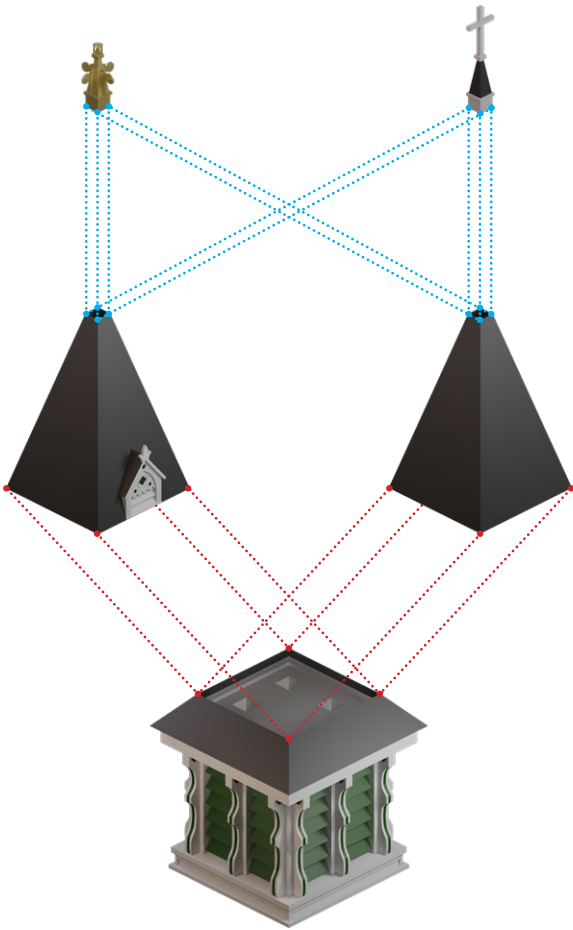
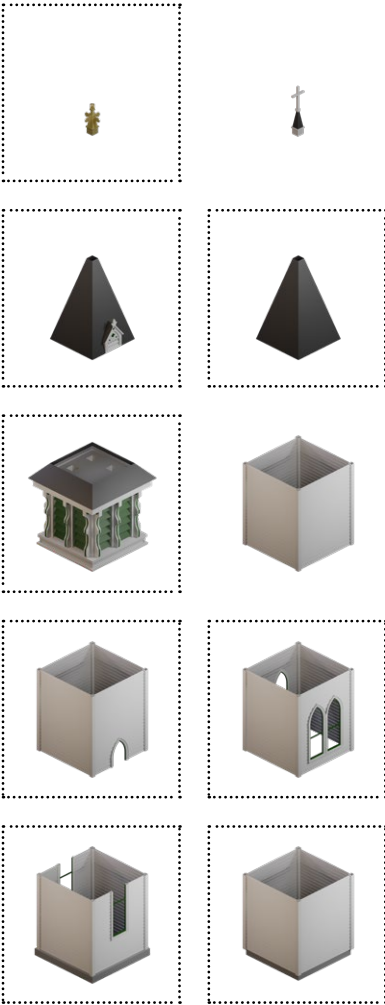
Today, I revisit this algorithm to explore its potential in generating conceptual forms procedurally.



The Tiles.

The tiles used in making these structures are displayed below. Each is sized to work in a 3D grid of cubes.

Note: For each cube shown, there are the 3 alternate rotations of 90° which are also used in the set.



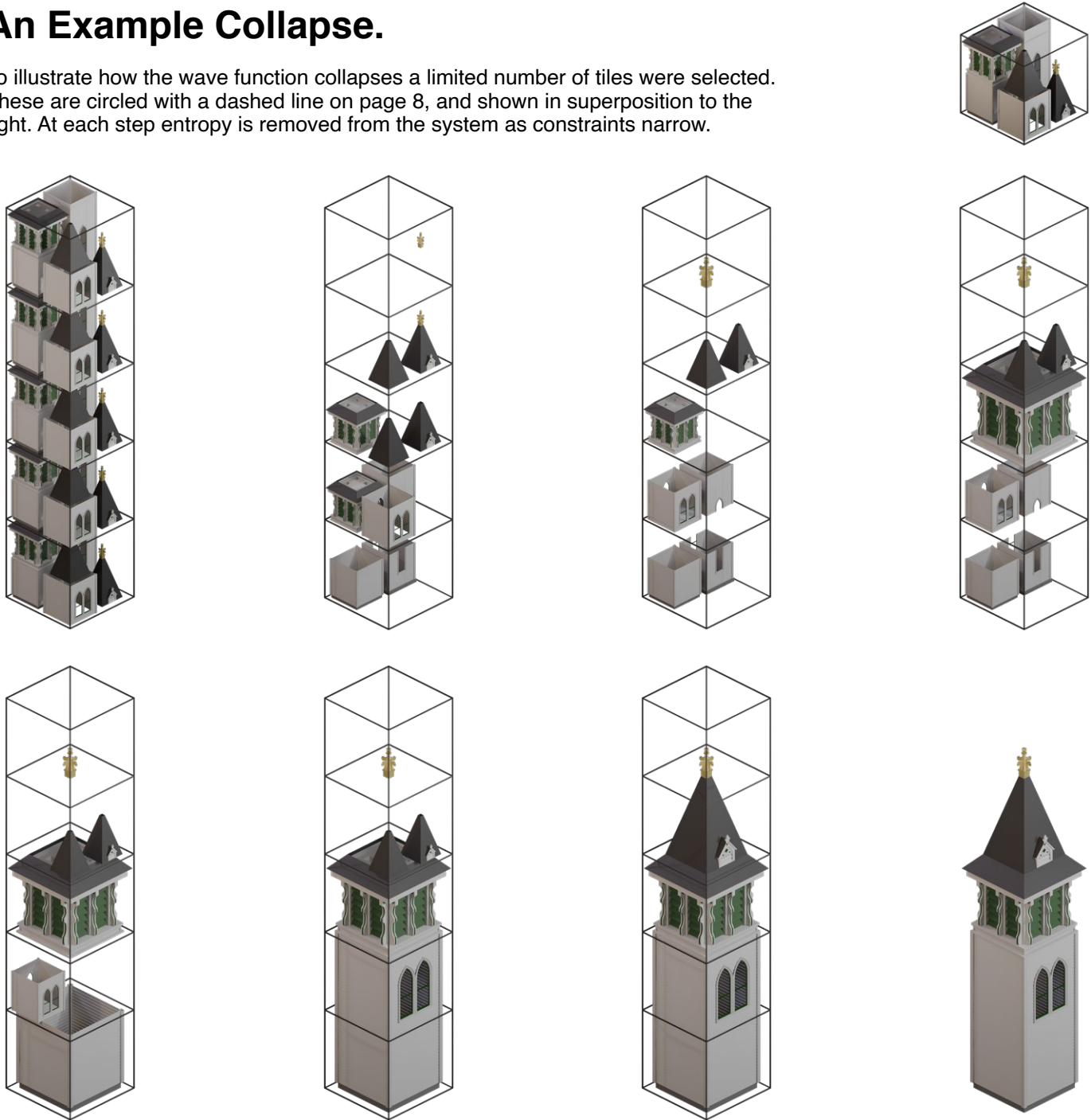
Connections.

Valid connections between pieces are determined by the vertices along a bounding face. If the vertices match then the cube is a valid match. Not all faces have rotationally symmetrical vertices, so connections can also depend on the rotation of both cubes.

Once a cube is selected for a space all invalid options are removed as neighbors, since the vertices will not match up.

An Example Collapse.

To illustrate how the wave function collapses a limited number of tiles were selected. These are circled with a dashed line on page 8, and shown in superposition to the right. At each step entropy is removed from the system as constraints narrow.



04

Shaping Code

Experiments in custom slicing.

Year: 2023-2024

Tools: Grasshopper

As both a makerspace enthusiast and experienced 3D printer user, I've long used GCode prepared from slicers like Cura. However, by creating custom slicing algorithms tailored to specific fabrication methods, I've been able to go beyond what traditional slicers can do.

This project showcases experiments in custom slicing, spanning diverse applications from clay 3D printing to the precision of pen plotting in 2.5D. Each algorithm is designed to optimize fabrication outputs, blending computational design with the craftsmanship of digital fabrication.

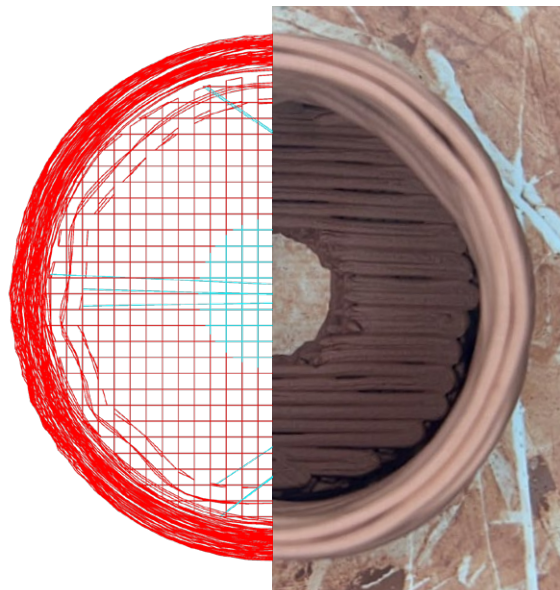
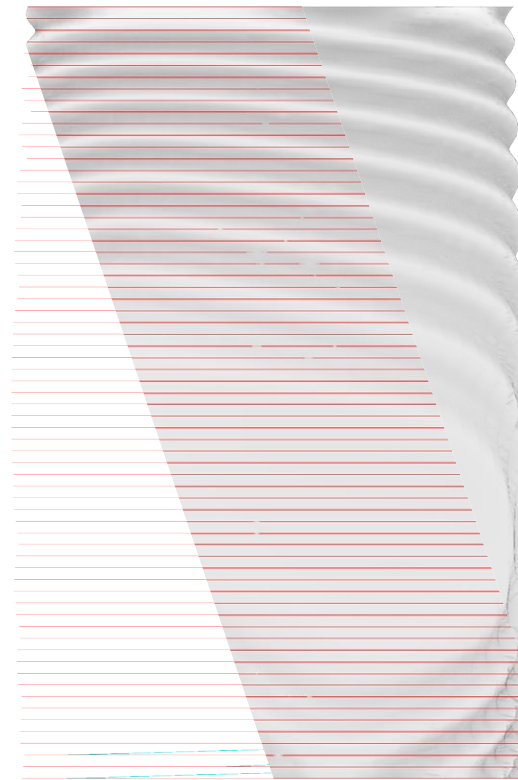
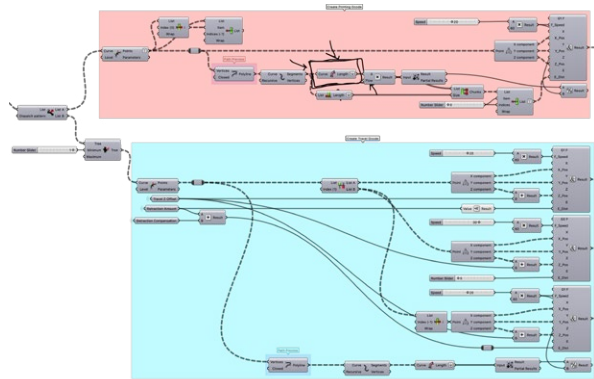
Shaping Code demonstrates how thoughtful scripting can transform digital ideas into tangible, meticulously crafted forms.



3D Printing.

Through multiple iterations of trial and error, and referencing the GCode command library, a working script was created to output GCode for a variety of 3D printers. Options for feed rates, wall thickness and whether to spiralize the print were all implemented.

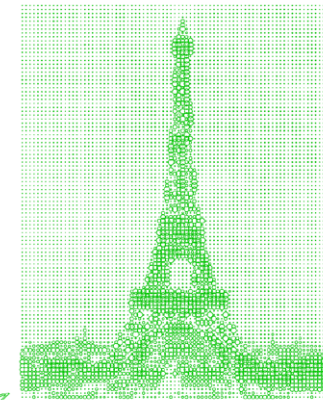
The main medium used for this was clay on a delta printer, but plastic was also experimented with.



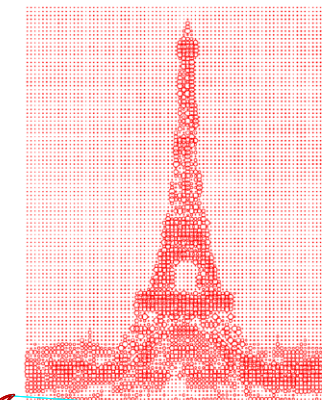
Pen Plotting.

Modifying the 3D printer GCode script for work on a 2D plane allows for pen plotting or brushwork.

Design.



Path Preview.

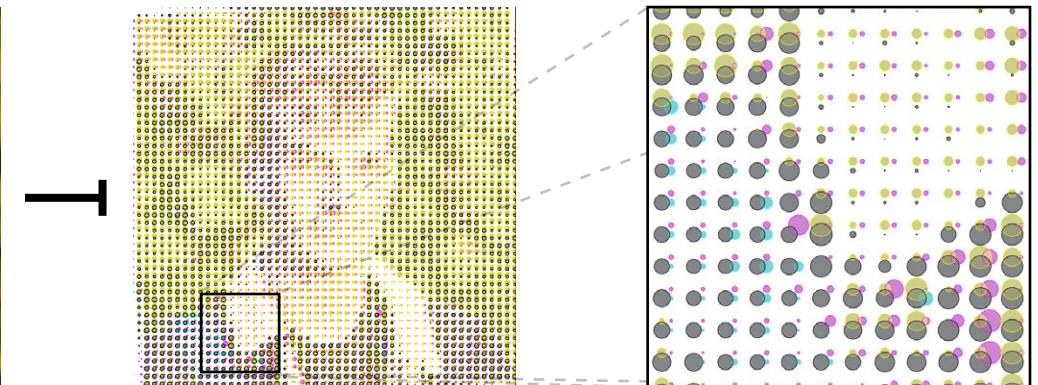
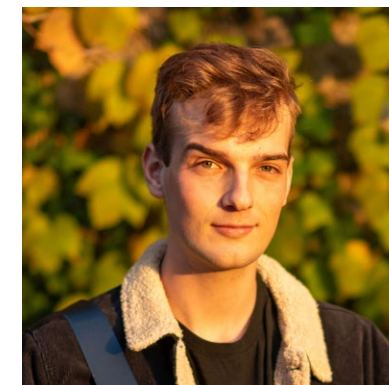
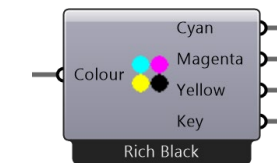


Plotted.



Split CMYK.

Grasshopper does not come with a node that splits a color into CMYK values, which are often used for printing. With my experience making AutoVisualizer I created a component to get these values. It allows for easier creative color plotting.



05

Reframing Color

Shooting with the trichrome process.

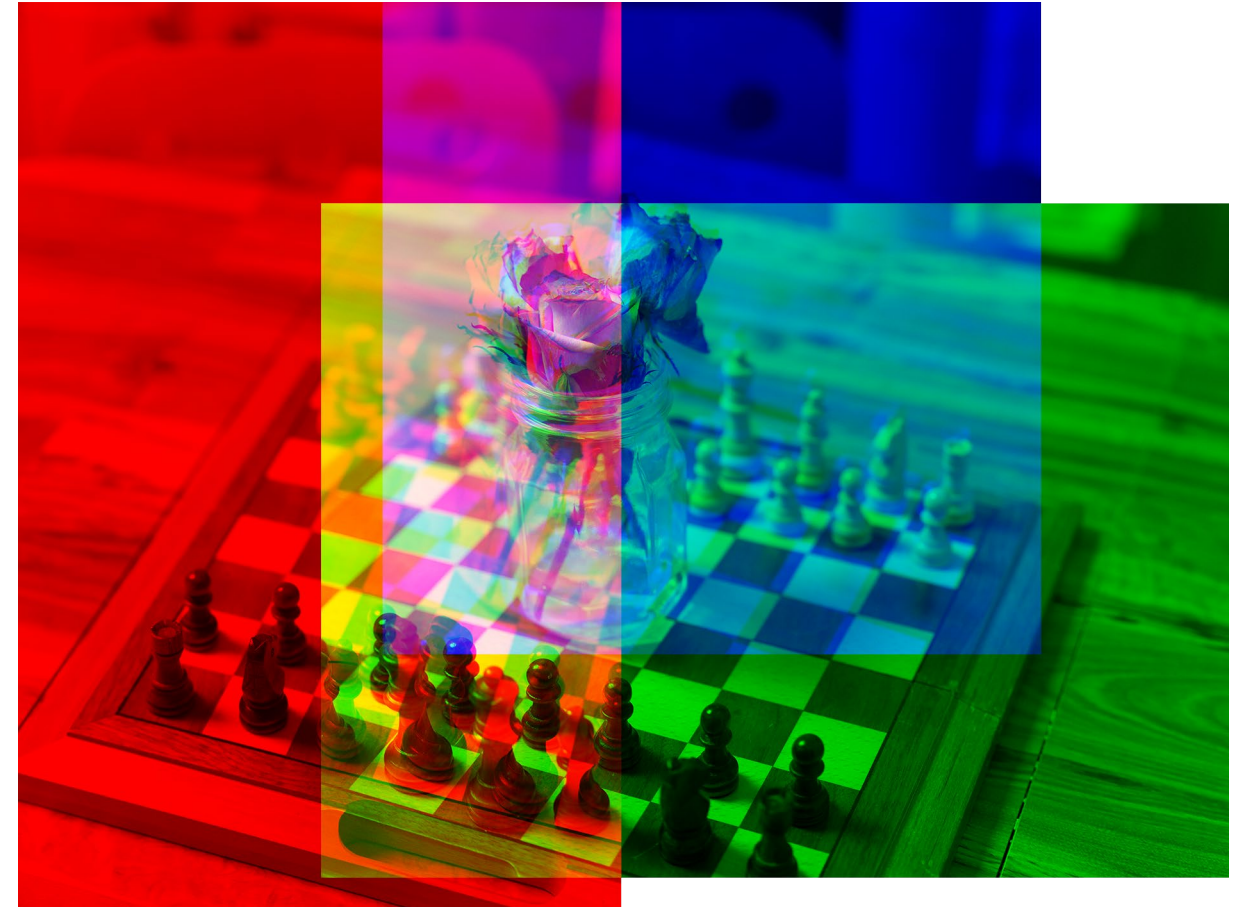
Year: 2023

Tools: Digital Camera, Trichrome Filters, Photoshop

The trichrome process, rooted in early photographic methods, involves capturing three separate images using red, green, and blue filters and then combining them to create a full-color photograph. By revisiting this historical technique, I explored the interplay of color, light, and composition in a uniquely modern context.

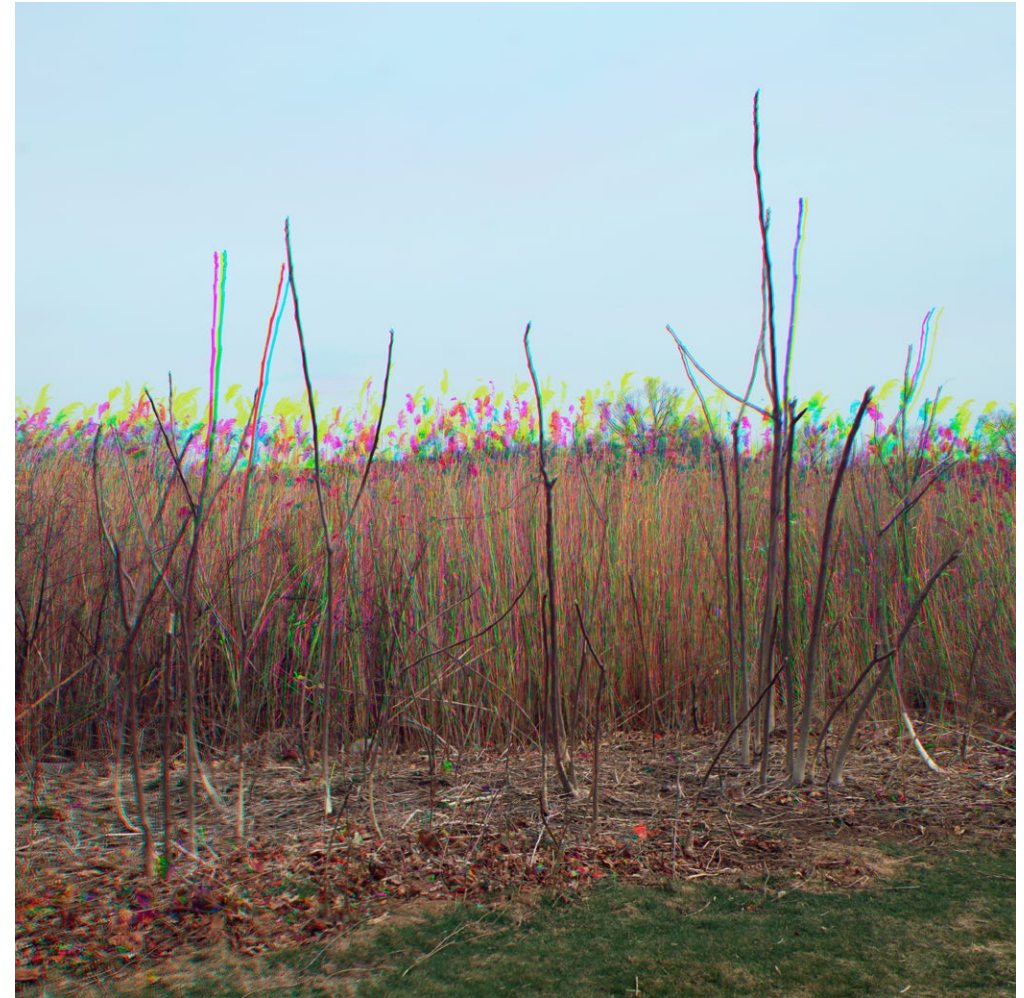
Using a digital camera and custom filters, I embraced the challenges of aligning and merging these filtered captures into a cohesive whole. Each image became an experiment in reconstructing color perception, where subtle shifts in alignment or exposure revealed vibrant, unexpected textures and patterns.

This project bridges analog traditions and digital precision, offering a contemporary take on a technique that reframes how we perceive and compose with color.





“Three Moves Ahead”



“Crowned by Breeze”

Drawing Chaos

Generating complexity through strange attractors.

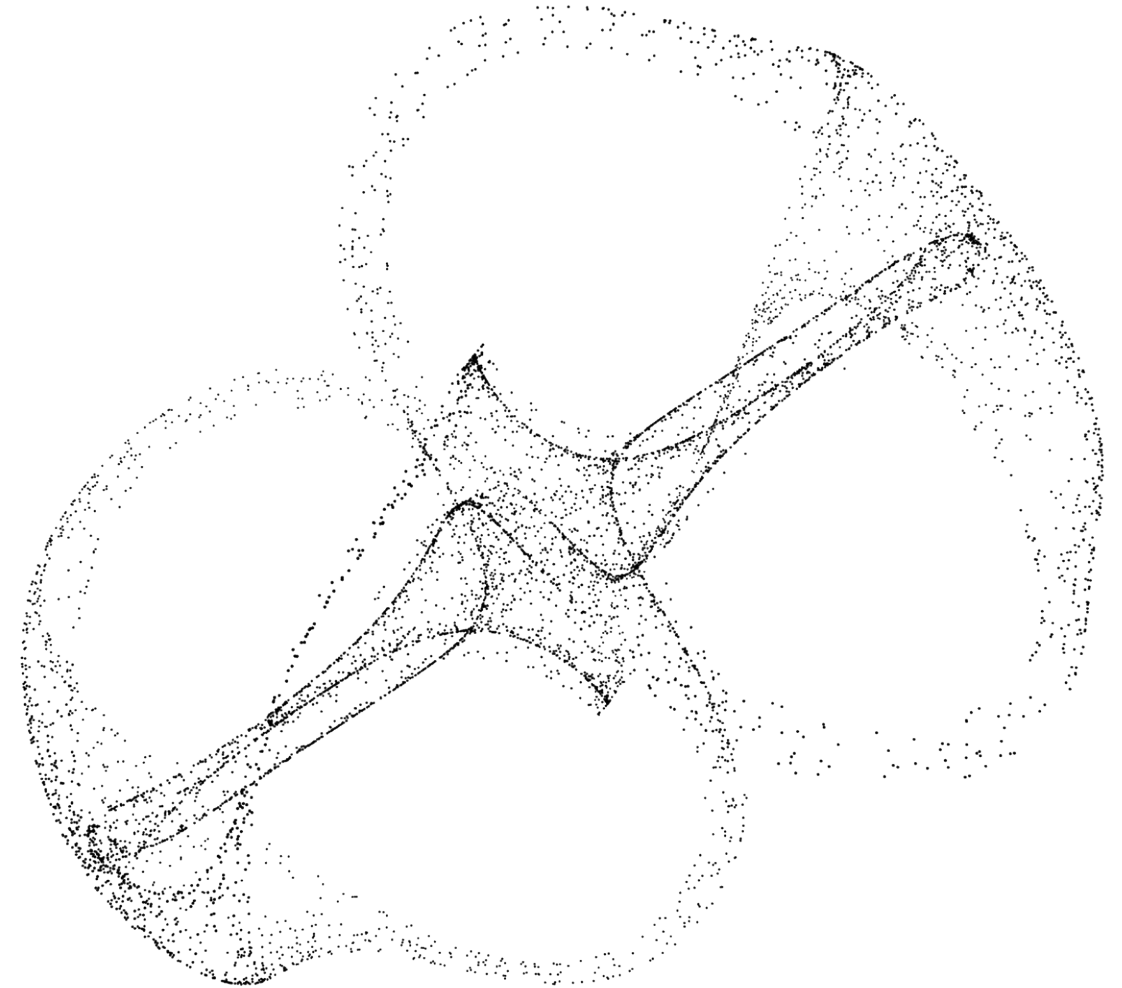
Year: 2023-2024

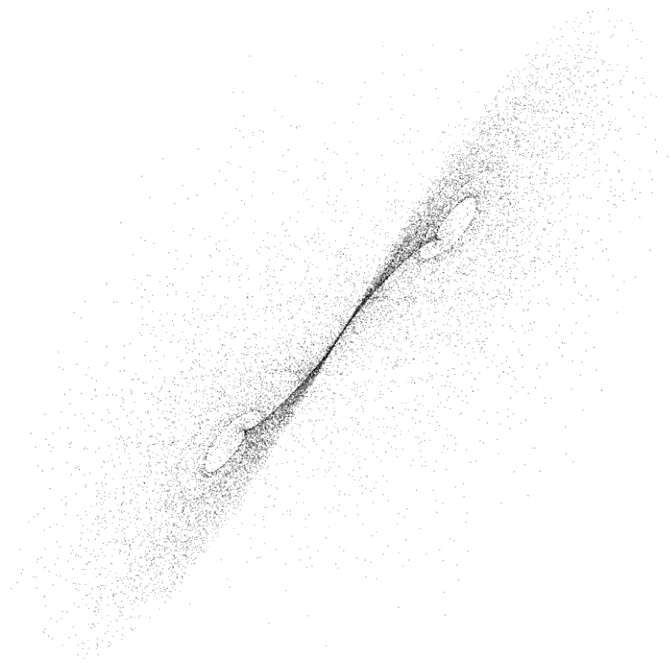
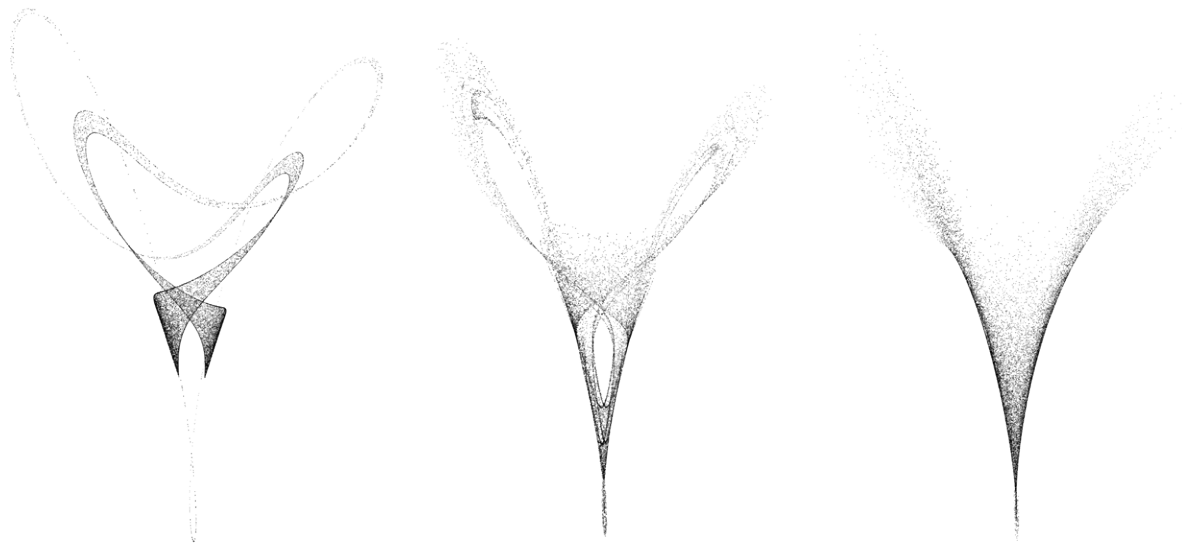
Tools: Blender

When I first encountered the term strange attractors in my senior year of college, I was intrigued by the challenge of visualizing these complex mathematical systems. At the time, I had been exploring Blender's simulation nodes for other projects, and it felt natural to adapt this tool to transform abstract equations into artistic representations.

Strange attractors are more than just visually compelling—they serve as a way to represent entropy and the inherent chaos within complex systems. Through iterative exploration, I developed techniques to display these equations in dynamic and engaging ways, blending science and art to reveal the beauty hidden within mathematical complexity.

This project celebrates the intersection of computation, art, and unpredictability, offering a glimpse into the ordered chaos that underpins strange attractors.





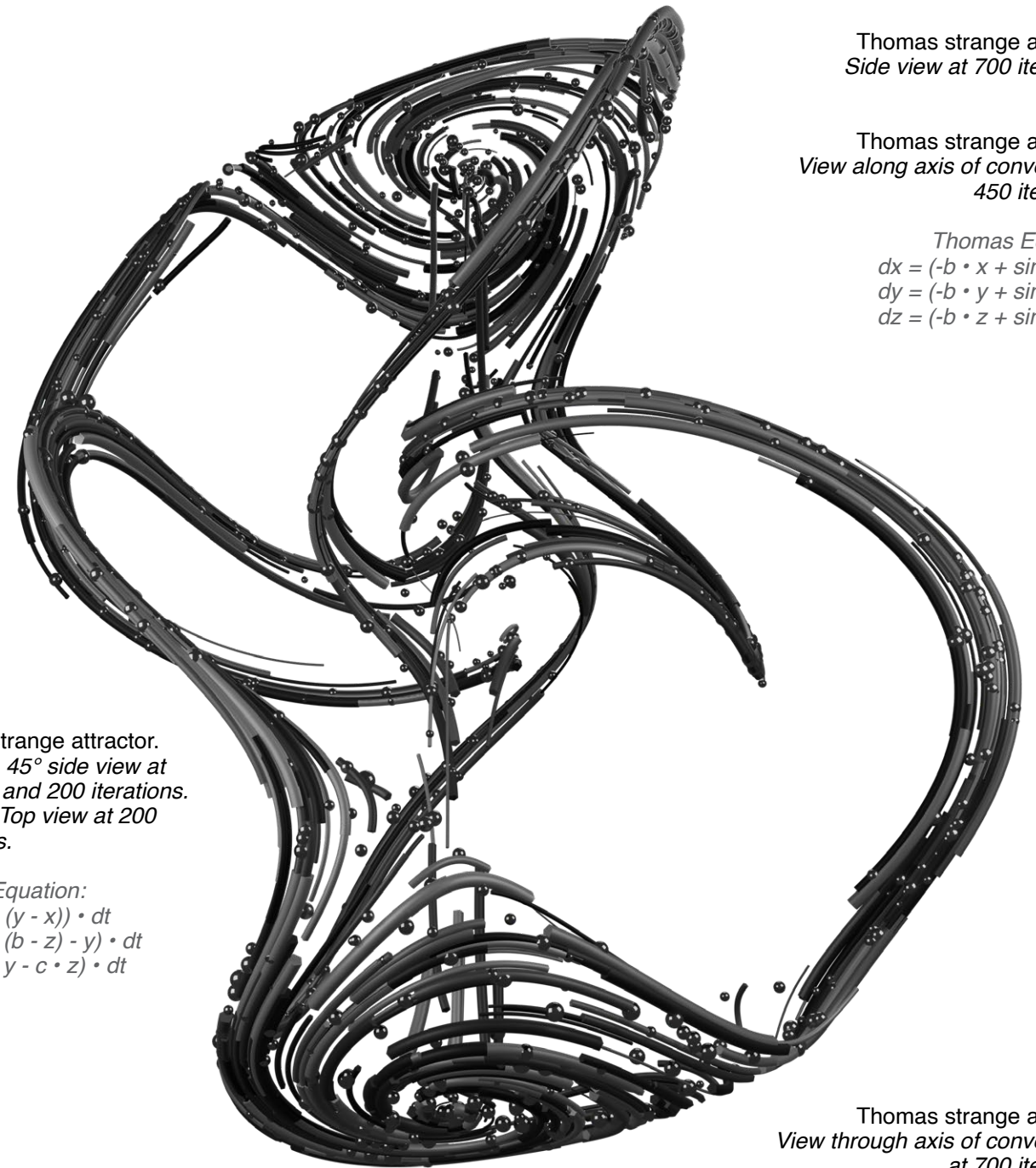
P34:
Lorenz strange attractor.
Top row: 45° side view at
50, 100, and 200 iterations.
Bottom: Top view at 200
iterations.

Lorenz Equation:

$$dx = (a \cdot (y - x)) \cdot dt$$

$$dy = (x \cdot (b - z) - y) \cdot dt$$

$$dz = (x \cdot y - c \cdot z) \cdot dt$$



P33:
Thomas strange attractor.
Side view at 700 iterations.

P35:
Thomas strange attractor.
View along axis of convergence
450 iterations.

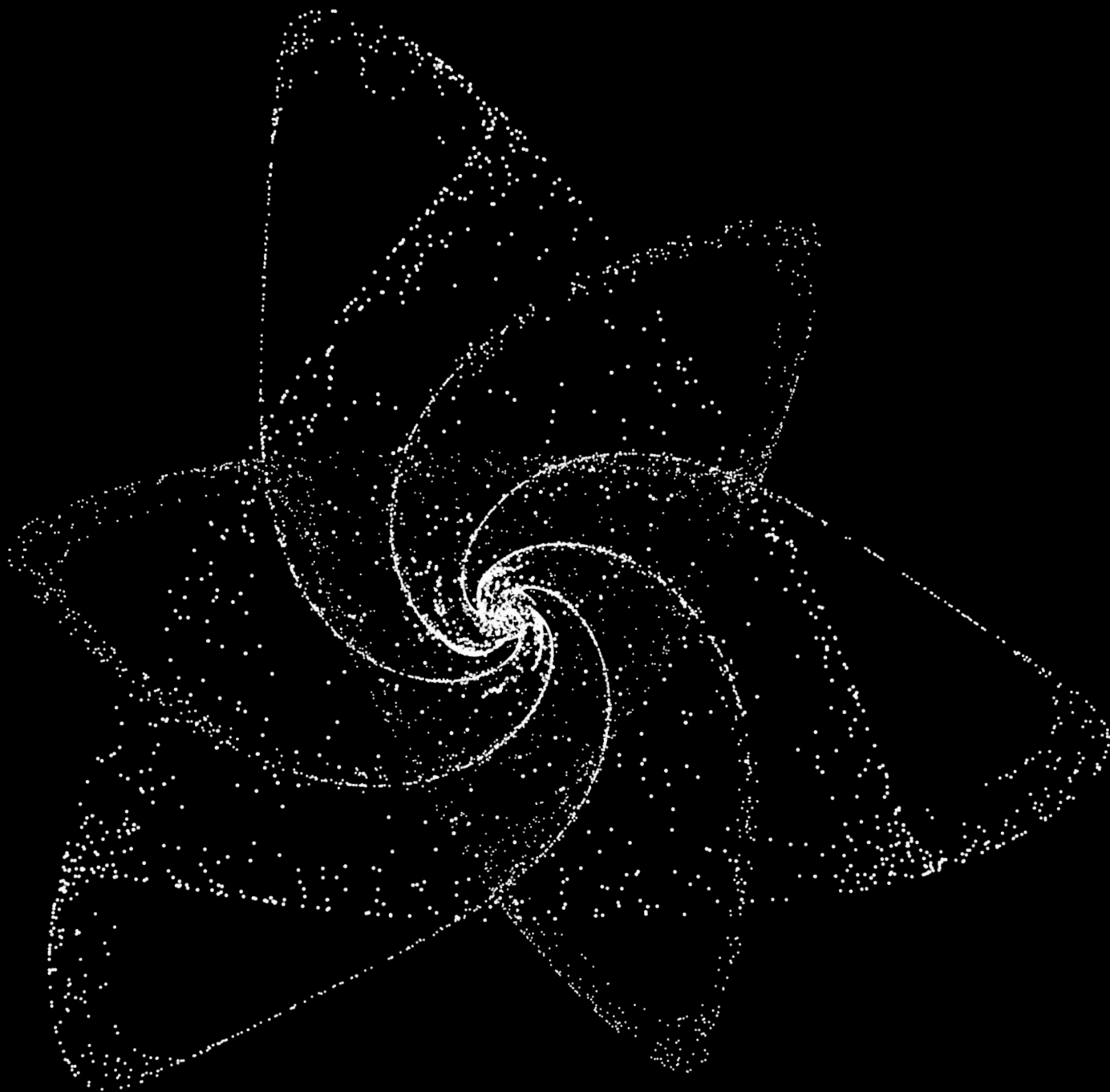
Thomas Equation:

$$dx = (-b \cdot x + \sin(y)) \cdot dt$$

$$dy = (-b \cdot y + \sin(z)) \cdot dt$$

$$dz = (-b \cdot z + \sin(x)) \cdot dt$$

P36:
Thomas strange attractor.
View through axis of convergence
at 700 iterations.



</PORTFOLIO>



Dan Kenerson