Mutable Objects, Spatial Manipulation, And Performance Optimization

Mirabile Dictu, How Many Steps in this program? Developing digital, iterative, and algorithmic processes, sequences, and systems that generate high performing results, spaces, and/or objects

Alexander Maly

December 2010

Submitted towards the fulfillment of the requirements for the Doctor of Architecture degree

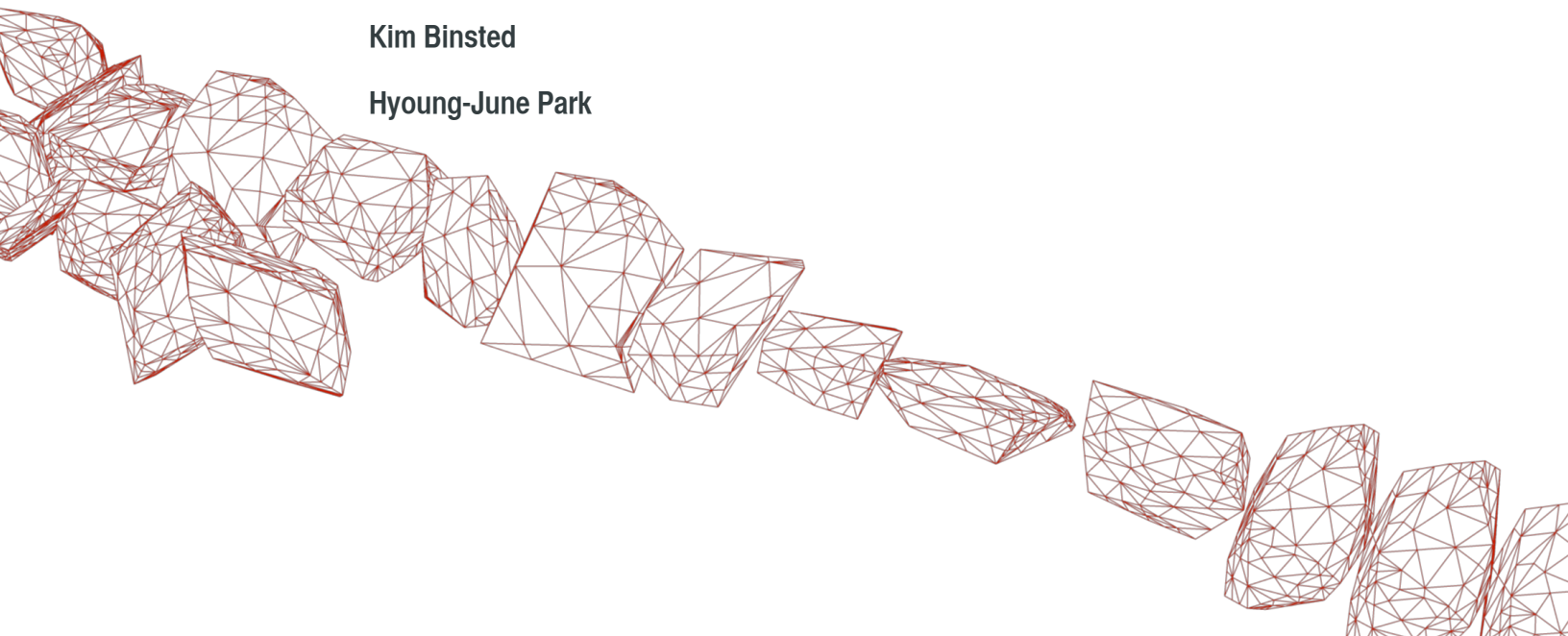
University Of Hawai'i
School Of Architecture

Doctorate Project Committee

Amy Anderson, Chairperson

Kim Binsted

Hyoung-June Park



Mutable Objects, Spatial Manipulation, And Performance Optimization

MIRABILE DICTU, HOW MANY STEPS IN THIS PROGRAM?
DEVELOPING DIGITAL, ITERATIVE, AND ALGORITHMIC PROCESSES, SEQUENCES, AND SYSTEMS THAT GENERATE HIGH PERFORMING RESULTS, SPACES, AND/OR OBJECTS

Alexander Maly

December 2010

WE CERTIFY THAT WE HAVE READ THIS DOCTORATE PROJECT AND THAT, IN OUR OPINION, IT IS SATISFACTORY IN SCOPE AND QUALITY IN PARTIAL FULFILLMENT FOR THE DEGREE OF DOCTOR OF ARCHITECTURE IN THE SCHOOL OF ARCHITECTURE, UNIVERSITY OF HAWAI'I AT MĀNOA.

Doctorate Project Committee

Amy Anderson, Chairperson

Kim Binsted

Hyoung-June Park

Acknowledgements


```
import sys
import time
import sem

def main(a=1):
	import extrasem
	extrasem.increment(2)

	print "thank you"

	while (1 or a in xrange(0, 1000)):

		print "thank you, merci beaucoup, muchas gracias, danke schon, mahalo!"

		for guide in [ "amy" , "hyoung-june" , "kim!" ]: print "i remain deeply appreciative for your advice, guidance, and encouragement, and support, " + guide

		for person in [adam, alex, andy, cage, dan, jamie, james, jolene, kat, kurt, laura, luan, misuk, sofro, syd]: print "thank you," + person + "for your all your advice, support, tolerance of stress behavior, suggestions of sanity-inducing diversions, couches to sleep on, food, and uncountable coffees..."

		for person in [mark, william, kirk, bundit, luke, and_the_rest_of_the_eight_crew]: print "thank you" + person + "for tolerating my strange schedule, and for encouraging the sometime slow process, and for laughing about the whole thing..."

		for person in [ang]: for num in xrange(0,99): print "thank you... thank you..."

		for person in family: print "surprise! i did it!" + "thank you for your unwavering encouragement..."

		main(a)

if( __name__== '__main__' ): main()
```

Table of Contents

Table of Figures

I. Abstract

Contemporary digital design techniques are powerful, but disjoint. There are myriad emerging ways of manipulating design components, and generating both functional forms and formal functions. With the combination of selective agglomeration, sequencing, and heuristics, it is possible to use these techniques to focus on optimizing performance criteria, and selecting for defined characteristics.

With these techniques, complex, performance oriented systems can emerge, with minimal input and high effectiveness and efficiency.

These processes depend on iterative loops for stability and directionality, and are the basis for optimization and refinement. They begin to approach cybernetic principles of self-organization and equilibrium. By rapidly looping this process, design 'attractors'–shared solution components–become visible and accessible.

In the past, we have been dedicated to selecting the contents of the design space. With these tools, we can now ask, what are the inputs to the design process, what is the continuum or spectrum of design inputs, and what are the selection criteria for the success of a design-aspect? These new questions allow for a greater coherence within a particular cognitive model for the designed and desired object.

There are ways of using optimization criteria that enable design freedom within these boundaries, while enforcing constraints and maintaining consistency for selected processes and product aspects. The identification and codification of new rules for the process support both flexibility and the potential for cognitive restructuring of the process and sequences of design.

II. New Potential

The capabilities of digital design systems allow for a procedural definition of space and a fundamentally different perspective on the design process. Sequenced and/or scriptable digital modeling interfaces enable algorithmic decision making, calculated analytical

metrics, just-in-time simulation of components' systems/parameters, and biological emulation such as evolutionary algorithms, iterated function systems, and artificial life simulations. I want to explore this expanded space, and to merge a scientific/logico-rational view of design with the fundamental subjectivity that inheres in the design process.

There is a cluster of topics emerging around digital design: algorithmic methods, performance evaluation, digital spatial analysis, simulation methods, and simulation of experience. Some of these topics are becoming important in theory and practice today; in the fields of interaction design, physical computing, environmental simulation, fluid dynamics analysis, building information modeling, advanced computer aided design (CAD), and smart geometry. The interaction of these topics stimulates the direction of this research.

There is disparate development in the theory and practice in many of these areas, but synthesis is still wanting. Each subject area has powerful procedures and mature technologies, but interaction between subject areas is sparse. The unique power of digital procedures can be brought to bear on important architectural problems. For example, there are many existing methods for free morphological design, for high performance buildings, for beautiful/elegant designs, and for delightful or rewarding spaces. But they frequently resist convergence. This research both furthers these separate methods, and identifies and experiments with ways of merging the powerful aspects of each area with each other's benefits.

Finally, the various methods and components can be engaged in a rubric that is implicitly or explicitly goal oriented. The nature of the design process relies on multiply-defined and highly interpretive success heuristics.

By connecting aspects of the above topic areas, certain new potentialities of design and processes of generating spatial can be approached.

III. Directionality

The overarching goals of this process are:

to develop an understanding of current work in evolutionary/simulational design methods

to proceed further in developing digitally defined models/simulations of designed objects and/or spaces.

to create and document a set of steps or method of generating evolutionary/simulational design

to experiment with existing software, and produce new tools to assist these goals

These foundational tools enable a number of future research and design areas of investigation. Using these tools, we can try:

to generate a finely tuned design object from these methods and tools, or generate a collection of many finely and separately tuned design objects

to apply evolutionary algorithms and methods to generate systems that perform significantly better than the status quo

to begin to simulate or digitally model spaces of human experience, and measure the relative contribution of various aspects of human experience

IV. Computational Opportunities

A set of benefits accrue to any computational processes. It is facile, within digital interfaces, to multiply objects, to sequence, to blend between the start and end of a sequence, and to precisely manipulate dimensions and variables. On the other hand, the available variety of options permits possible permutations that have the potential to overwhelm the task of manual choice.

In order to make sense of this set of unfolded possibilities, computational processes have to be again invoked. The attributes of any object can be checked, and simple conditional statements can select those objects that characterize a desired outcome.

As the selection criteria become more complicated, and the objects the criteria act upon become interlinked or multiple, the direct logical connection between attributes and actions become more tenuous and less directly causal. To an external observer, the behavior of the system becomes illogical to the point of mystery. At this point, intentional manipulation of inputs loses the thread of direct consequence, and becomes filtered or

convoluted. The system is then a 'black box', and the only way of evaluating the results is via measurement or analysis of said result.

Thus, the set of computational possibilities fall into three major categories: the input, the procedure, and the output, the generator, the generation, and the generated, the processed, the process and the product. The input, the source, is typically a reduced or compressed set of information, or fundamental geometry. The process has almost infinite range, from merely repeating the input to completely reconstructing, to deconstructing or destroying it. The outflow, the finished product, follows, but can be returned and reformed into an input.

This is a major departure from typical design processes. In essence, this conceptualization of creativity–the generation of design–exposes the interface between the designer and the world (or worlds). Creative individuals lay claim to mental autonomy, as the realm in which mind, experience, design constraints, and the spark of the individual's muse combine. This autonomy is made transparent and ephemeral when the design action occurs subsequent to the designers instructions.

These benefits have been explored heavily in the last few decades. The computational advantage has pervaded the processes of cinema, music production, architecture, urban planning, structural analysis, population modeling, biomolecular analysis, and even such craft-intensive practices as furniture design. It has also generated entire fields: interactive media, digital animation, 3d modeling and rendering, and within the design fields, rapid prototyping.

There are two modes of computational advantage. The direct influence of new digital tools is in facilitating the manipulation of complex phenomena such as multiple or parallel components, the ability to rapidly processing these phenomena, or the ease of editing.

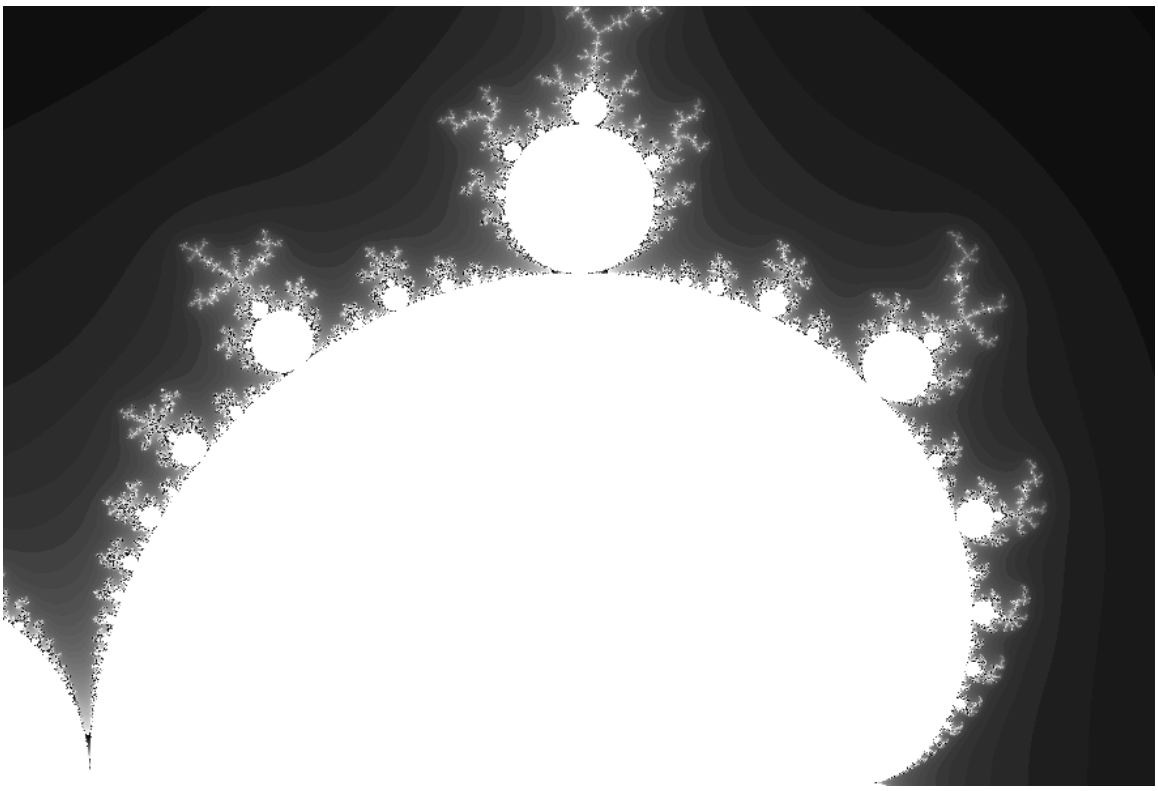


Figure 1. A visualization of a portion of the Mandelbrot set, generated by the author with code adapted from an example in Processing[1] by Daniel Shiffman.[2]

The second mode integrates the computational capabilities within the process itself; in this mode, the work could not exist without the computation. When visualized, these are sometimes termed 'procedural art' or 'computational art; the processes involved frequently use thousands of iterations of a simple subprocess (e.g. visualization of the Mandelbrot set) or the multiplication of simple modules with varying parameters.

[1] Processing.org, "Basics \ Processing 1.0", accessed November 14, 2010. http://www.processing.org/about/.

[2] "The Nature of Code at daniel shiffman", accessed November 14, 2010. http://www.shiffman.net/teaching/nature/.

Figure 2. A portion of J. Tarbell's *Sand.Dollar*, which exemplifies computational designs that are based on large quantities of repetitions.[3]

V. Components of Spatial Description

The spatial foundation of every object can be described using a set of points in three dimensions. There are a few common methods for dealing with spatialized points (three dimensional, within an x,y,z coordinate system). If the procedural input is a set of points, certain methods can be used to generate surfaces or volumes. In the software package Rhino, for example, this operation is called "pointset reconstruction."[4]

Conceptually, the simplest of these is the spanning circle. This consists, in 2 dimensions, of picking a center vertex and drawing a circle of a radius such that all points (or a maximized number of points) lie within the circle. This is also known as the "minimal

[3] J. Tarbell, Sand.Dollar, 2004. "Complexification | Gallery of Computation", accessed November 14, 2010. http://www.complexification.net/gallery/machines/sandDollar/.

[4] "labs:pointsetreconstruction · McNeel Wiki," last modified November 8, 2010. http://wiki.mcneel.com/labs/pointsetreconstruction.

enclosing circle" or "bomb problem."[5] In 3 dimensions, a sphere is drawn around all points: the "minimally enclosing ball" problem.[6] Generalized point coverage algorithms exist for more complex surfaces or objects. Qhull is a widely used implementation of convex hull coverage, which is the "smallest convex polygon (or polyhedron) containing points in [target set] S"[7]. Choosing the minimal coverage of the target as a convex hull is the "first pre-processing step for many, if not most, geometric algorithms."[8]

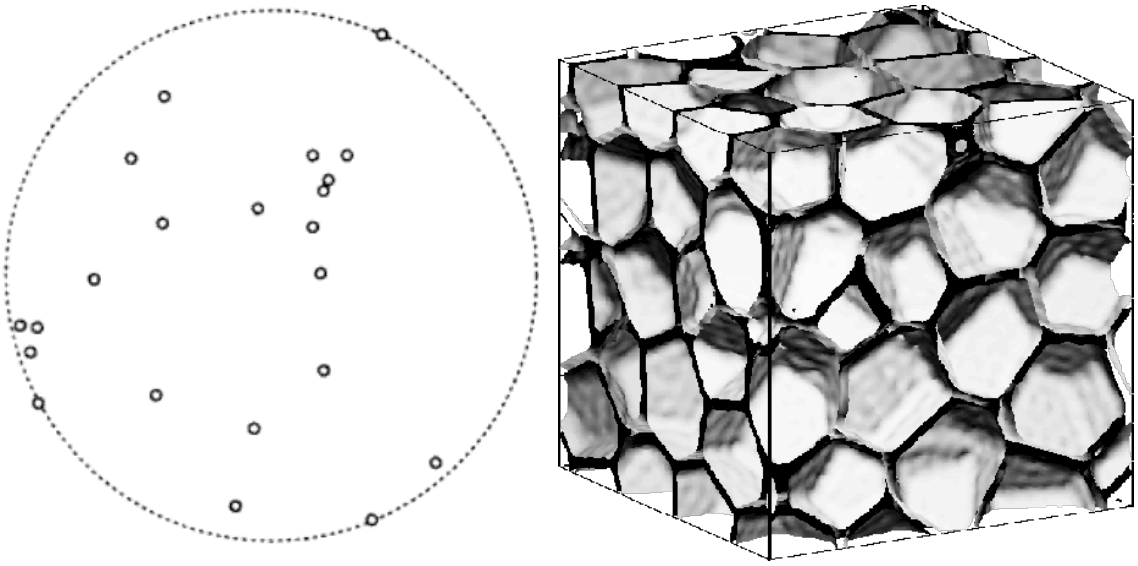


Figure 3: Left, the minimal enclosing sphere of a point cloud.[9] Right, the Voronoi surface in three dimensions.[10]

If the goal is to construct a connection map within the set of points, a Delaunay triangulation, and the corresponding "dual" graph, the Voronoi surface (also known as the Dirichlet Tesselation), shows the network of points. These geometric structures, which

---

[5] Weisstein, Eric W. "Minimal Enclosing Circle." From MathWorld--A Wolfram Web Resource. accessed November 14, 2010. http://mathworld.wolfram.com/MinimalEnclosingCircle.html

[6] Goodman, Jacob E., János Pach, and Emo Welzl. *Combinatorial and computational geometry.* Cambridge University Press, 2005, 293.

[7] Skiena, Steven S. *The Algorithm Design Manual*. 2nd ed. Springer, 2008, 568.

[8] *Ibid.*

[9] Goodman, Pach, and Welzl, 293.

[10] "Elastic moduli of model random three-dimensional closed-cell cellular solids - Voronoi tessellations", Last visited November 14, 2010. http://ciks.cbt.nist.gov/garbocz/closedcell/node5.html.

have edges equidistant from each point, are analogs to the boundaries formed by collections of soap bubbles.[11] These structures provide a way to investigate the density of a point cloud, as well as its structure.

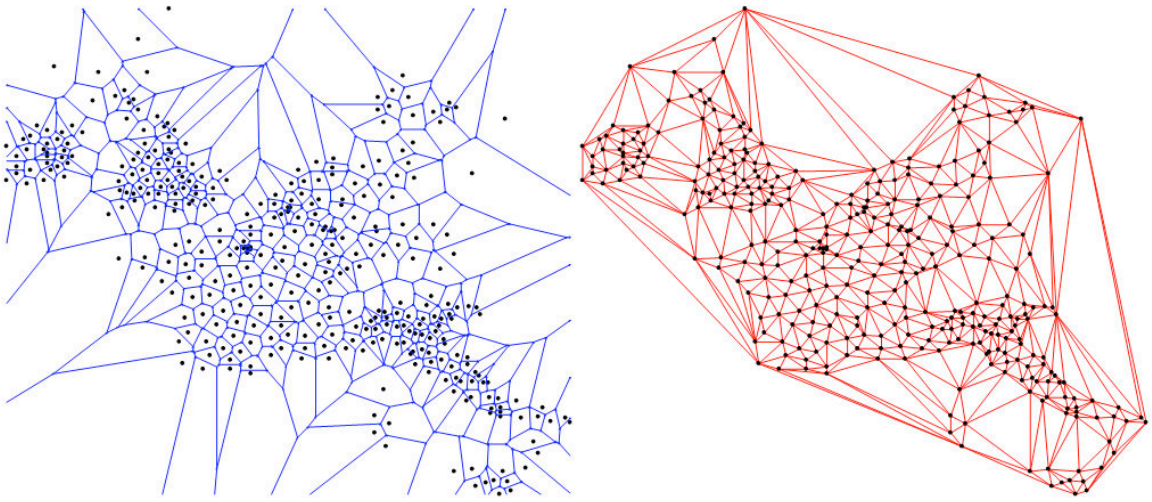


Figure 4: The Dirichlet tesselation of a point set.[12]

Another productive method for operating on a set of points is via division. Breaking the point set into its component subsets, by recursively dividing the point space, generates a tree data structure. This is an efficient way to identify spatial zones with equal number of points. The octree reduction of a point set (quadtree in two dimensions,) provides a way to understand the zonal density of space; it takes into account the entire population of points and not just the points per unit volume (density).

[11] http://wiki.mcneel.com/labs/pointsetreconstruction

[12] "Delaunay/Dirichlet Tessellation", Last accessed November 14, 2010. http://www.passagesoftware.net/webhelp/Delaunay_Dirichlet_Tessellation.htm.

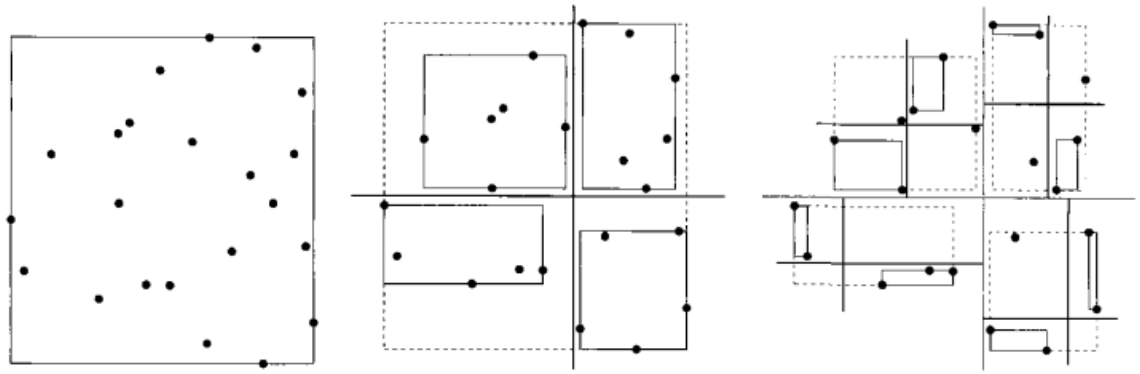


Figure 5: Three steps in the quadtree division of point sets in two dimensions.[13]

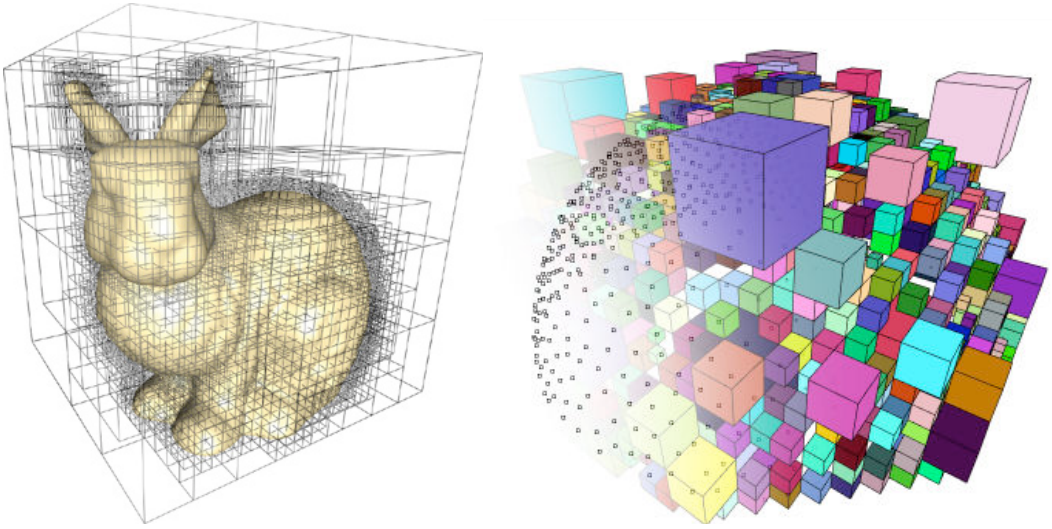


Figure 6: Left, octree description of a 3D model.[14] Right, octree of a point set on a sphere.[15]

[13] Calvo, Jorge Alberto. *Physical and numerical models in knot theory: including applications to the life sciences.* World Scientific, 2005, 328.

[14] "GPU Gems - Chapter 37. Octree Textures on the GPU", Last accessed November 14, 2010. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter37.html.

[15] http://wiki.mcneel.com/labs/pointsetreconstruction

VI. Types of Spatial Change

The result of these point operations generates a shape in two or three dimensions. The relatively simple set of operations that can be performed on a spatial figure are the ingredients for the variety exhibited by the variety in objects' morphology.

The object of the transformation is the first transformation parameter. The coordinate system can be transformed as well as the object, causing the perception of rotation, or scaling. This has the same result as when the object is itself transformed. Linear transformations are the simplest to calculate; they preserve straight lines but not parameters such as area, and do not include translations.[16] Linear transformations also include shearing, reflection, and orthographic projection: they "stretch", but do not "warp".[17] Affine transformations are linear transformations that include translation, thus allowing a greater range in the flexibility of the transformation.

VII. Challenges (i) : Abstracting Manipulation

Contradictions / Paradoxi

There is an internal tension, or irregularity, evaluating how to approach these manipulations. Visualization focuses on the communicative aspect of the spatial object. Manipulation (typically) focuses on the topological or topometric aspect, and ignores the visual. Historically, certain visual techniques have been prevalent (e.g. hair, textures, bump mapping, displacement.) These, however, have topological consequences, and break the object model; the operations become the essential characteristic of the object, rather than a parameter attached to the object. Some software allows for such morphological change to be 'baked' in to the object collection, but this still only partially solves the boundary problem. A solution might be a digital model that will accept modeling of the world in continuous yet separable layers. In this situation, the ground would always overlap into the domain of the figure, and the figure with the viewer. The

---

[16] Dunn, Fletcher, and Ian Parberry. *3D math primer for graphics and game development*. Jones & Bartlett Learning, 2002, 92.

[17] *Ibid.*

participating elements contain both textural info and larger scale info. That is, layers of scale are integral to the object, and have direct consequences in terms of volume, area, heat transfer, sensory impact, or other performance functions. In essence, each transformation would modify the underlying geometric system as a whole, and would depend on sequence; it would be a set of stacking modifiers.

Actions: Tooling

Under the general category of 'tooling,' Benjamin Aranda and Chris Lasch try to place their experimental new digital techniques into a conceptual framework of actions; they define the new capabilities of digital, scriptable geometry as verbs. Going further than the "clearly stated steps" of the algorithm, they extend the metaphor to recipes and "packaging of logic", and a "solvent to liquify" "the predilections of the architect and the inherent properties of the geometries" [18] This action-packed ideation positions the actual moves of the scripted logic as a kind of constructive procedure, and tends to weaken the typical emphasis on produced form or final product. However, it also tends to focus on the ephemeral and indefinite; it almost ignores the main thrust of the work: the scripts and algorithms that generate the dynamic geometry.

spiraling	produces a shape unlike any other because it is seldom experienced as geometry, but rather as energy.
packing	produces stability through adjacency
weaving	produces strength by combining two weak systems in a reciprocal pattern
blending	is a fundamental technique in the act of negotiation
cracking	following the rule of self similarity, cracking gives a sense of the larger whole
flocking	finds order through entropy
tiling	assembles a patterned tectonic

Figure 7: Tooling actions[19]

[18] Aranda, Benjamin, and Chris Lasch. *Pamphlet Architecture 27: Tooling*. 1st ed. (Princeton Architectural Press, 2005), 9.

[19] *Ibid*, 10, 22, 32, 40, 52, 62, 74.

Actions: Mutation

Francois de la Roche calls names a process somewhat differently. Giovanni Corbellini, referring to Heidegger's concept, mentions "paths that go off in the woods... as a metaphor for human existence, speaks of a continuous mutation."[20] The idea of continuous mutation is a potent model for how to imagine the act of transformation.

VIII.Ways Of Manipulating Spatial Entities

Acted Upon; Where the Participant Lives

When starting with a geometric construct and extrapolating into reality (bricks and mortar or physical systems,) it is critical to determine the region/zone/space in which the algorithm operates. What components are the foundational elements? What components cannot be manipulated? If perception or awareness is a plastic design element, the location of the design participant, or sensing agent, within the conceptual field is paramount.

In realities such as industrial design (the target of which is an object,) the participant handles or otherwise engages the focus of the activity; he or she handles a tool, or operates a vehicle, or plays with a toy.

In rooms or individual spaces, the user inhabits, or dwells, within the space. He or she enters or exits the space, occupies the space, and senses the space.

In larger sets of spaces, buildings, that depend upon infrastructure for their character, the participant circulates between spaces, and engages with sets of spaces. He or she traverses from one space to another.

Finally, in large scale exteriorized infrastructure, urban spaces, the individual agent travels through and navigates the spaces. Equally as important, the individual navigates the spaces between spaces, the negative space of the city, the exteriorized circulation space.

---

[20] Giovanni Corbellini, *Bioreboot: the architecture of R&Sie(n)* (Princeton Architectural Press, 2009), 37.

In this way, the chiasmus between and among pieces of the world always consists of a piece of a transitional space. That is, an object participates in a room, a room participates in a building, a building composes a city. While experimentation can be isolated to one scale, it always has implications throughout the taxonomy of spaces. However, since each scale can be considered distinctly, each scale can be approached with different computational tools.

Acting; Who Moves?

Within a particular scale and spatial zone, the components of a world respond variably to a transformation. In a biological situation, those that occupy a particular niche respond relative to the effect of the niche, individual species respond relative to the exposure of the species to the transformation, and individual organisms respond to their specific conditions. In fact, the populations of individuals often create or influence evolutionary important aspects of their niche, as well as responding to their "unique ecological and social setting."[21]

In human situations, psychological impacts affect behavior and dwelling as heavily as physical or social impacts. Injections or importations of non-human biology often change complex behavioral epiphenomena such as health or crime.[22]

IX. Algorithm

Any sequence of steps involved in generating or changing a digital object can be considered a kind of algorithm. Simple algorithms include repetition, multiplication, and evenly distributing objects (such as the 'Duplicate', 'Repeat', and 'Align' functions in many common software packages). Each step of a process can be as simple as adding a value to a (x,y) coordinate. As simple procedures are aggregated and interlinked, however, more complicated behavior can emerge. As rules and procedures become more complex, many

---

[21] Jablonka, Eva, and Marion J. Lamb. *Evolution in four dimensions: genetic, epigenetic, behavioral, and symbolic variation in the history of life*. MIT Press, 2005, 228.

[22] Hynes, H. Patricia, and Russ Lopez. *Urban health: readings in the social, built, and physical environments of U.S. cities*. Jones & Bartlett Learning, 2009, 270.

systems appear more biological. It only takes a few rules for this behavior to emerge. The artificial life experiments of the mid-20th century cyberneticists proved that very simply defined systems can exhibit complex behavior; e.g Conway's game of life, Lindenmeyer systems, or chaotic objects such as Lorenz' attractors.

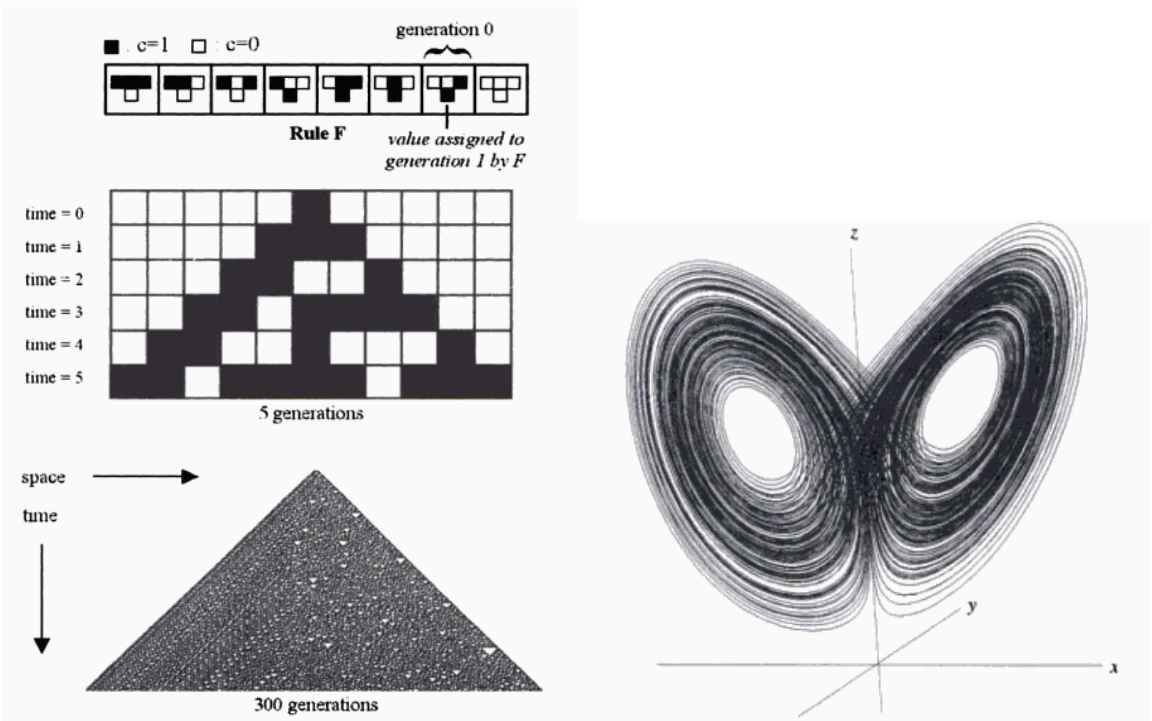


Figure 8: Left, a simple cellular automata system in time.[23] Right, a visualization of the Lorenz Attractor.[24]

The power of algorithms lie with their organization, selection, and analytical capabilities.

[23] Ilachinski, Andrew. *Cellular automata: a discrete universe*. World Scientific, 2001, 10.

[24] Peitgen, Heinz-Otto, Hartmut Jürgens, and Dietmar Saupe. *Chaos and fractals: new frontiers of science*. Springer, 2004, 648.

algorithmic capacity

sorting
arraying
analysis
generation
  spreading
  arranging
  trees/hierarchy
  subdividing

Figure 9: The power of algorithms.

Even thought processes can be differentiated into the category of more or less algorithmic. For example, Schwank identified the distinction as between 'functional' and 'predicative' thought patterns in young math students.[25]

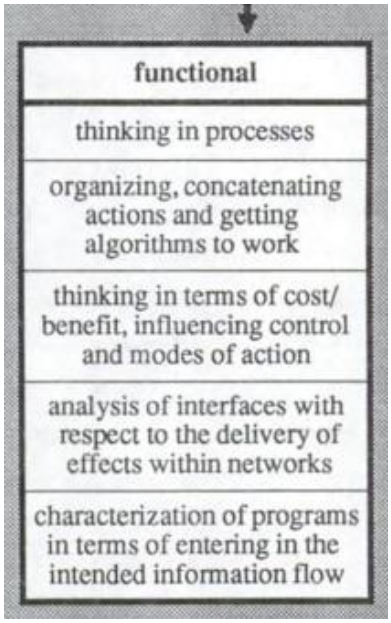


Figure 10: Algorithmic or functional thinking is both a cause and a consequence of certain pedagogical practices.[26]

The most far-reaching consequences of algorithmic methods and organizational structure occur when a target is subjected to algorithmic optimization techniques. These methods

[25] Schwank, Inge. "Cognitive Structures and Cognitive Strategies in Algorithmic Thinking," in Dettori, Giuliana, and North Atlantic Treaty Organization. Scientific Affairs Division. *Cognitive models and intelligent environments for learning programming.* Springer, 1993, 250.

[26] *Ibid.*

solve for maxima or minima in the efficiency of a system, and offer significant reward for restructuring a system. In addition, to balance multiple goals in a complex problem, multi-objective optimization can be used. This explores the state space of a problem along multiple dimensional axes.

Iteration

The loop of output becoming input, repeated over and over, is the iterative process. Gradual changes, shifting slightly from step to step, accrete into major leaps in the development of a system.

One important characteristic of an iterative system is the rate at which it changes. If it is dynamically stable, small changes tend to not effect the macro systemic properties. This case typically depends on negative feedback, where a change is counteracted by the effects of the change, and usually converges to a equilibrium. Unstable systems, conversely, accelerate away from the system's initial conditions, and exemplify positive feedback. These conditions rapidly diverge, and typically continue until they reach a fundamental limit of the system, such as food availability in biological populations.

The subset of iterative algorithms that can productively be brought into a generative geometric scheme tend to be near equilibrium but not stable. That is, systems that oscillate only semi-predictably, or that cycle between multiple equilibrium conditions.

One very popular iterative method is the fractal-generating iterated function system (IFS). An IFS typically takes a very simple substitution, repeats it recursively, and results in a theoretically infinitely detailed spatial object.

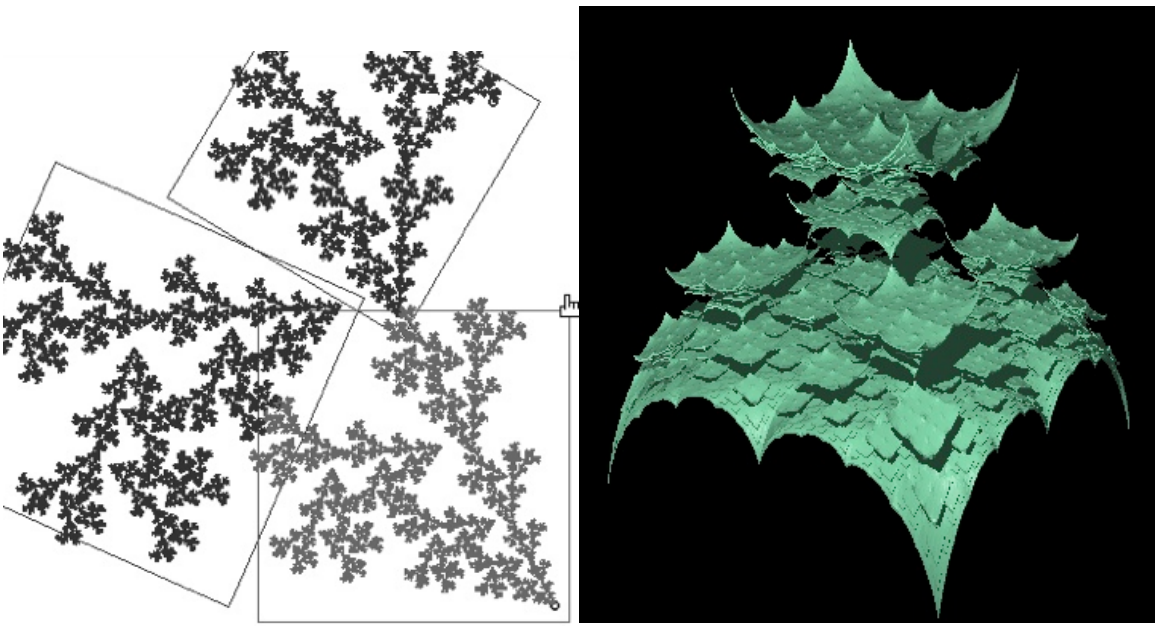


Figure 11: Left, an iterated function system (IFS) in two dimensions.[27] Right, an IFS in three dimensions. [28]

Because the boundary measurements of these objects are asymptotic , they are said to have fractional dimension; they have properties which are not compatible with a strictly lower-dimensional space. The two-dimensional IFS has an indefinite edge length, and thus the surface has dimension greater than two, while the three-dimensional object has a similarly difficult to measure surface area.

Generation

The taxonomy of algorithms includes a type that results in higher order structure; it generates internal relationships between components.

In the generative menagerie, there are a few predominant methods that emit complex results from simple rules. There are cellular automata (wolfram, CA, game of life), Lindenmayer systems, a specific subset of write-rewrite systems, iterated function systems (IFS), fractal equations, chaotic generators, fluid flows and turbulence amenable to Navier-Stokes equations, and there are basic mathematics such as polynomial and

---

[27] "Nonlinear Iterated Function Systems", accessed November 14, 2010, http://www.cg.tuwien.ac.at/research/vis/dynsys/nifs/.

[28] *Ibid.*

trigonometric functions. Each underlying algorithm is suited to a particular type of deployment.

Beyond generative procedures, there are composite interacting strategies. Flocking models have been developed, to simulate how groups of individual animals relate in space. Craig Reynolds wrote an early version of this system, which is still popular, entitled "boids,"[29] which is based on three simple rules for navigation for each 'boid': stay within a certain distance from your neighbor, stay away a certain distance from your neighbor, and navigate for direction and speed using the average direction and speed of the entire flock[30]. Other research has been done into crowd simulations, especially for modeling the capability of a space to successfully allow occupants to exit in the case of fire or emergency. Game designers and movie producers also have used crowd simulations, to develop realistic imagery of how large numbers of independent human agents act physically.

Generators are not necessarily simple constructors. The characteristic of generation can be brought out of nearly any process. Randomness, or truly stochastic processes, are often used to seed a function, or to initiate a system in a stability landscape.

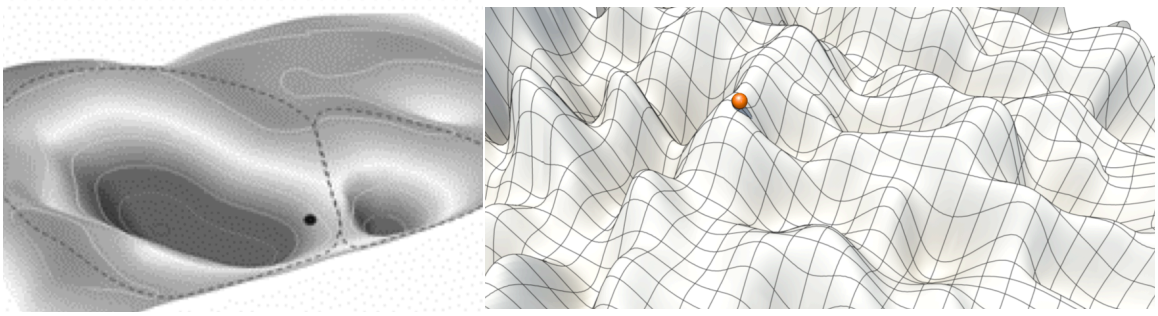


Figure 12: Stability topography as systemic state space. Left, low dimensional state space.[31] Right, high dimensional state space (by author).

---

[29] Reynolds, Craig. "Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)" http://www.red3d.com/cwr/boids/

[30] Grimm, Volker, and Steven F. Railsback. *Individual-based Modeling and Ecology*. Princeton University Press, 2005, 127.

[31] Walker, B., C. S. Holling, S. R. Carpenter, and A. Kinzig. 2004. "Resilience, adaptability and transformability in social–ecological systems." Ecology and Society 9(2): 5. http://www.ecologyandsociety.org/vol9/iss2/art5

X. Simulation

Simulated systems occupy a more specific niche in the realm of artificial realities. Most simulations exist to test a particular parameter; daylight, thermal transfer, stress-strain distributions, among others. At their core, simulations are mathematical models of certain 'real' phenomena, reduced in complexity to satisfy either the requirements for computing numeric solutions, or to avoid having to deal with interacting separate components (air and sun, for example).

XI. Direction of Development

The difference in simulation is often the difference between top-down or bottom-up analyses. Using a matrix of particles or a field full of agents, following the rules of automata (however intelligently,) results in a bottom up, decentralized situation. The analyses that are relevant to this situation do not usually allow for easily calculable models.

After the generation of a population or landscape of elements, each piece can be analyzed separately, and explicitly separate the interaction of individual pieces with its adjacent pieces. In structural mechanics, these involve the finite element model (FEM) or finite element analysis (FEA) method.[32] However, within generative programming, it is also possible to approach the analysis with a top-down method. That is, though a system may be generated atomistically, the evaluation criteria of systemic parameters can be holistically defined.

XII. Jumping Complexity Scale

While some optimization strategies have been rigorously explored, the potential for optimization to be applied to higher order problems remains significant. Some problems are numerically or procedurally so computationally intense as to be out of reach for most investigators. However, many higher level problems, especially in architecture or 3-

---

[32] Leondes, Cornelius T. *Computer-Aided Design, Engineering, and Manufacturing: Systems Techniques and Applications, Volume III, Operational Methods in Computer-Aided Design*. CRC Press, 2000.

dimensional structure, can be reduced to more geometrically primitive and approachable formulations. A townscape can be represented by a plane of grid squares, a building by a floor plan, or an intelligent agent by a dot.

XIII.Modeling System Structure

Components of a working system:

Agent or representation of agent
Grid or analytical space (e.g. finite element mesh)
Output space (e.g. density, distance, composite metric)
Evaluation method for output (e.g. units/area)
Iterator mechanism (e.g. timestep, genetic generation)

Figure 13: System components.

What participates in the scope of simulation?

human scale: objects to hold, spaces to inhabit, cities to explore
solid materials: cut away (subtractive), voxel manipulations
units: subdivide and break down and crackle and crumble
objects: stretch and skew and twirl and twist
modular accumulations: things that mutate & repeat & are located

how?
in groups, by relative locations, in grids
in particles, that cloud and swarm and flock and flow
along paths, that bundle and weave and tumble and braid

Figure 14: System participants.

XIV.Mimetics

There is a variety of sources from which to derive either simulational models or direct developmental or mechanical inspiration for morphological change. The idea of biomimicry directly refers to this flow of 'ideas' from nature into human culture, as used in mechanical or engineering systems. Performance tuning may also gain from looking to biological systems for inspiration.

The ecological concepts of centralized and decentralized control structures apply. A bottom up approach involves Conway's "Life" game, in which rules and initial conditions apply with only the most basic of intrinsic structures.

The top-down approach takes existing buildings or urban chunks, and varies them according to more complicated rules. An example of this could include instantiating a variety of building codes, or changing such things as height limits or setback distances.

XV. Memetics

Many of the ideas involved in this project are relatively popular within certain boundaries (genetic algorithms for generating interesting form, daylight simulations for ecological building). The hybridization of the ideas, perhaps consistent with Dawkins' meme hypothesis[33], supports a more flexible approach. Each algorithmic environment requires a certain characterization and constraint of inputs.

XVI. Optimization Types

Linear methods of optimization are suited for a certain class of problems. Nonlinear techniques tolerate a wider range. Stochastic optimization is often more effective at problems with multiple local maximums, where other methods can get stuck in local extrema.

Genetic algorithms are one type of stochastic optimization technique, best used for problems with ambiguous locations or number of global minima and maxima.

XVII. Evolution

The process of evolution has been encoded into computationally-compatible form. In the simplest case, a digital genetic sequence (usually illustrated as a chunk of binary information such as 001001010101011110) represents a description of a larger scale phenomenon, such as geometry, behavioral dynamics, or constraints. Then, a limited percentage of that genotype is randomized: the 'mutation'. The resulting alternative seed is used to generate an output (the phenotype) and the output is evaluated based on a analytical metric. Then, the output becomes the new input, and the process repeats. This is a 'genetic algorithm'.

---

[33] Dawkins, Richard. *The selfish gene*. Oxford University Press, 2006.

Unlike in biological evolution, digital evolutionary algorithms tend to ignore the developmental stage. Without development, "all you have is a big DNA or RNA molecule".[34] Integrating the more complicated evolutionary principles, such as hybridization and crossover, can provide more realism and a different evolutionary dynamic.

XVIII.Measurement

Fitness Function

The dynamics of the fitness function are central to the evolutionary process. The fitness function fundamentally determines the constraints and forces acting on the algorithmic evolution. This requires measurement criteria for success to be encoded as a scoring system.

The system can be measured directly via data from the system, or based on a composite or invented success criteria.

Performance Metrics

Several aspects of an individual object, building, or city can be easily identified as good candidates evaluating individual success. Thermal performance could be measured to minimize heat flux across a surface, or could be measured as the thermal stability of a space. Energy use per unit floor area is a measure of energy efficiency. Similarly with other services, such as lighting performance, or fresh air supply, a parameter averaged over the floor area gives performance efficiency for a building.

Within the building or urban context, the circulation paths of the inhabitants offer a way to measure the success of the system. Using the terms derived by analysis methods such as those used by "space syntax" researchers, paths can be scored by topological metrics

---

[34] Kumar, Sanjeev, and Peter J. Bentley. "Biologically Inspired Evolutionary Development." In *Evolvable Systems: From Biology to Hardware*, 99-106, 2003.

such as "centrality", distance metrics like "travel time", behavioral metrics like pedestrian flow rates, or network hierarchical metrics such as primary or secondary connections.[35,36]

Performance Targets

Measurements are not always explicit or exposed, for the performance targets towards which cities or buildings could be directed. Alternative measurements of urban performance could include a measure for acoustic isolation (or acoustic absorbance) towards the goal of minimizing traffic noise, the frequency of urban residents' watching the street, as Jane Jacobs described[37] as a measure of community security, the availability of daylight from the public sidewalk for light exposure, the width or ease of circulation paths for ease of pedestrian use, the frequency of curb cuts for low friction traffic, or many other parameters.

Individual buildings might be evaluated by existing architectural parameters like overall internal circulation path length, ease of exiting, path travel distance for disabled users. Some parameters are encoded in measurement systems such as LEED, where daylighting and thermal comfort control are scored for internal uses.[38] Other parameters are only accessible via proxy or reporting from users: overall happiness with a space, a sense of harmony or balance, a sense of sacredness, or even more ephemeral characters all present challenges for encoding into a scoring function. For example, human delight is a potential organizing rubric for a holistic evaluation. Alternatively, sociologically defined evaluations could be used.

On a larger scale, the entire system can be scored for city function. Infrastructure efficiency, sewer linear dimension per inhabitant, power production per unit area, energy

---

[35] Vis, B. N. *Built Environments, Constructed Societies*. Leiden:Sidestone Press, 2010, 49.

[36] C. Mulders-Kusumo, "Spatial Configuration of the Area Around Delft Central Station," in Bilsen, A. Van, F. van der Hoeven, and Jürgen Rosemann. *Urban transformations and sustainability: progress of research issues in urbanism 2005*. IOS Press, 2006, 67-71.

[37] Jacobs, Jane. *The Death And Life Of Great American Cities*. Rev. Ed., New York City: Vintage Books, 1992, 35.

[38] Cottrell, Michelle. *Guide to the Leed Green Associate (Ga) Exam*. John Wiley and Sons, 2010, 129.

use per square block, or compliance with building codes are all measurements on the city-wide scale.

In addition, metrics could be synthesized from the point of view of an inhabitant, but for arbitrary or non-rational purposes. A inhabitant might view the city as a video game, and prefer or give fitness benefit for complex or strategic urban geographies, that supply situational benefit (however, a petty criminal might happen to use a similar scoring system.)

Finally, the system could be scored along temporal grounds. Whether the system supplies its product on a just-in-time basis, how often it is late, or the average wait time could all factor into a total score. The classic example would be the transit network, but other temporally dependent services would also be evaluatable: perishable supplies such as ice distributors, or time-critical services such as messaging or other communication channels.

XIX.Analogs for Spatial Generation: Biology

Biological analogs can provide a holistical evaluation of performance in space. The elements that force biological evolution are many and varied, but are usually integrated into the metric of reproductive success.

D'Arcy Thompson's ideas[39] began the development of morphological descriptions of organisms using mathematical formulas. He cataloged and measured the growth patterns of entities like seashells, horns, tusks, and other skeletal features. These numeric descriptions are useful foundations for describing dynamic morphologies.

Descriptions of biological parameters can be used for a number of biological performance criteria. Evaluation of each of these are optimizable separately, and as composite metrics.

---

[39] Thompson, D'Arcy Wentworth. *On Growth and Form: A New Edition*. rev. ed., Courier Dover Publications, 1992, new ed. Cambridge: University Press, 1942.

Biologically or ecologically significant parameters
body size
surface/volume
thermal mass
energy balance/calorie uptake vs expenditure
limiting nutrient

Figure 15: Ways of measuring biological organisms

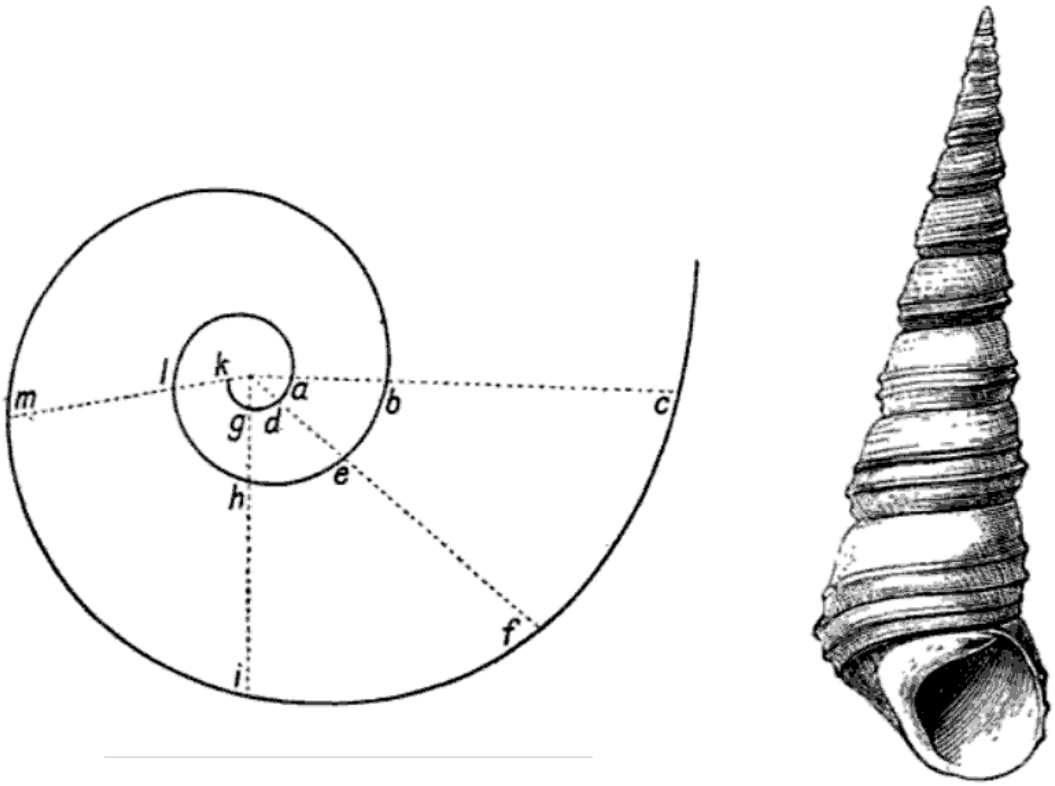


Figure 16. D'Arcy Thompson's investigations into the geometry of natural form, in this case a *Turritella duplicata*[40].

XX. Analogs for Spatial Analysis: Urban Systems

The city, also known as the urban fabric, provides a natural test bed for algorithmic determination than built form. The long time scale of change, in conjunction with the relatively strict code and developmental requirements, are inherently suited to algorithmic representation. John Frazer did this using artificial life models[41], in order to eventually

[40] Thompson, *On Growth and Form,* 771.

[41] Frazer, John. *An Evolutionary Architecture*. London: Architectural Association, 1995, 56.

develop a evolved structural system.[42] Future development investigated emergent rules. Others generate urban forms as cinematic backdrops using on-demand scripts to generate faux cities.[43,44,45] The functional representation of cities has an almost entirely different analytical foundation, and has been used heavily for such end goals as water modeling, pollution dispersion, air flow, or traffic system analysis.

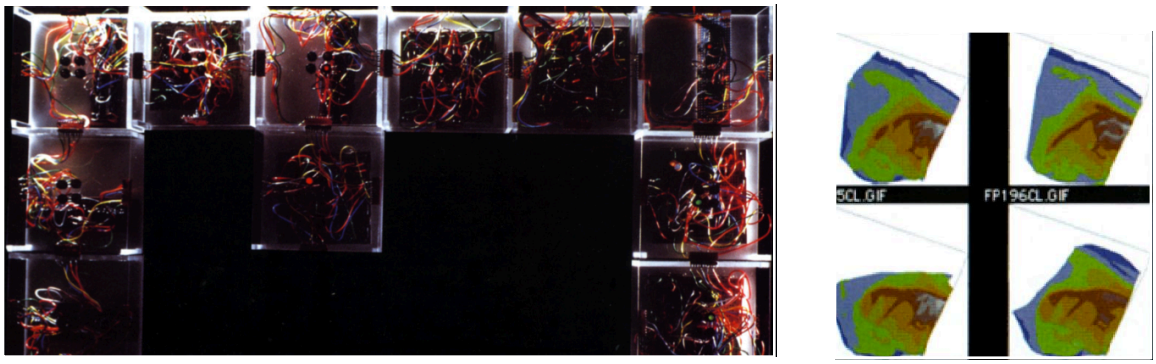


Figure 17: Left, Frazer's self-replicating cellular automata.[46] Right, elevation model in time.[47]

[42] *Ibid*, 70.

[43] "Custom MEL Tools", Accessed November 17, 2010. http://richsunproductions.info/vsfx705/modelingsuite.html.

[44] "City Gen 2 on the horizon | Wasabi 3D - Blog", Accessed November 17, 2010. http://www.wasabi3d.com/blog/index.php/2010/04/28/city-gen-2/.

[45] "SCG Screenshots", Accessed November 17, 2010. http://arnaud.ile.nc/sce/screenshots.php.

[46] Frazer, *An Evolutionary Architecture,* 56.

[47] Raper, Jonathan. *Multidimensional geographic information science*. London: Taylor and Francis, 2000, 239.

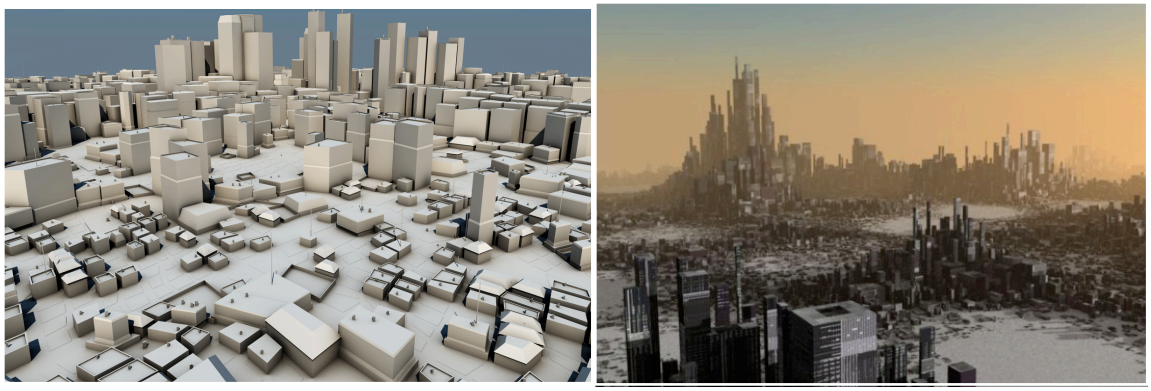


Figure 18: City generator script results. Citygen,[48] left, and SCG,[49] right.

XXI.Reality (The Non-Analogical)

The nature of this experimentation is continually being refined. Typical discourses refer to the real object, versus the computer, digital, or three-dimensional model. However, when the model takes direct input from the materialized entity, and deforms it into a new materialized entity, the 'real' component is less clearcut. Does the reality of the system lie in the interacting components? Or do the pieces each reside in their own reality? Various simulations have become complex enough to coherently model a world, whether or not the simulated world approaches the material world. Alternative environments all have interesting internally consistent rule sets, and an instantiation of a developmental process in each world could produce interesting differentiation.

There is another aspect of continuum from modeled to material reality: the process of building each reality. The new building process, while virtual at first, is in fact a building method of a different kind. In comparative terms, each building element is both continually subject to deformation, and constantly accessible. If it is encumbered by earlier placed elements, it is not restricted by gravity or load-paths, but instead by complexity or interconnectedness.

---

[48] http://www.wasabi3d.com/blog/index.php/2010/04/28/city-gen-2/

[49] http://arnaud.ile.nc/sce/screenshots.php.

XXII.Technology Adoption Sequence

The common understanding of the process of adoption of technology follows a exponential growth curve, approaching a limit of saturation. In this model, the new tools or materials are enmeshed with the old ways along an accellerating timeline.[50]

The adoption of the digital tools for spatial design inverts this process in a few specific ways. First, as usual, the digital and algorithmic tools are applied to prove their capabilities–the proof of concept. The current phase of development, however, is a retrenchment. These tools are now turned towards traditional forms and methods. That is, sculptural plasticity, masonry stacking, pin & joinery, and moment frames are all reimagined under the digital regime. The pressing question becomes, what is the intrinsic capability of the new tools? Are we uncomfortable with the new potential, and so move back to the known knowns? To paraphrase Louis Kahn's famous quote[51], "What does a vertex set want to be?"

XXIII.Software, Mind, and Cognitive Flow

There are a few resonant components in the range of software with computational spatial or modeling capability. Within each of these solutions, market forces and tuned processes specify exactly what kinds of operations are desired. Briefly, the products have a variety of characteristics.

However, the taxonomy of software has differentiated along certain lines. Rhino, for example, facilitates path operations (loft, sweep, rails). Sketchup works with normal (perpendicular) vectors, allows triangular surface folding. Each software constrains the possibility space in certain ways.

---

[50] Rogers, Everett M. *Diffusion of innovations*, 4th ed. New York: Simon and Schuster, 1995, 23.

[51] Wiseman, Carter. *Louis I. Kahn: beyond time and style : a life in architecture*. New York: W. W. Norton & Company, 2007.

constraint	mechanism
Direction of movement	(axis locking, coordinate systems)
Ease of manipulation	(location, rotation, scale)
Geometric foundational components	(points, faces, geometric primitives)
Layering and hierarchy	(transparency, multiple assignment)
Embedded time structure	(timeline, implicit sequencing)
Morphing and warping	(reversable transforms, parametric transforms)
Scriptability	(scripting language, IDE, script editor, partial exec.)
Experimentation ease	(undo buffer, shortcuts, transform history & reodering)
Industrial uptake	(commercial viability, size of community)
Model complexity	(primitives, accretions)

Figure 19: Table of modeling software parameters

Csikszentmihalyi has said, that "[a musician] needs just a piece of paper, where he can put down little marks...he can imagine sounds that have not existed before,"[52] that enables the creation of new reality, and in that situation it is as if the author doesn't exist.[53] Software is a way of manipulating impermanent scratches on paper. Scratches on screen, and scratches in data, also allow imagination of new realities.

The state of flow is a contiguous question to these concerns. Does the software allow the user into a flow state? Is there support or affordances that give the user a reduced resistance through frequent actions? If an designer is trying to create a new space that allows motion, or interaction, or a new spatial experience that has never existed before, does the software facilitate those challenges?

---

[52] ted.com, "Mihaly Csikszentmihalyi on Flow," FLV, Accessed November 18, 2010, http://www.ted.com/talks/mihaly_csikszentmihalyi_on_flow.html, 2004.

[53] *Ibid.*

category	name	function	file format
modeling	Autodesk Maya 2008 (with MEL scripting environment)	suited to generating 3- and 4-dimensional geometry, animations, and renderings	ma, mb
	Rhino 3D (with rhinoscript scripting environment)	see above, with focus on freeform shapes and transformations	3dm
	Blender (with python scripting environment)	see above, with focus on animation	blend
	Cinema 4D	animation and geometric modeling, with focus on animation	c4d
	Google Sketchup	rapid geometric modeling, with focus on architecture and design	skp
simulation	Autodesk Ecotect	environmental simulation, daylighting, thermal performance	dxf, eco, IFC
	NIST FDS	fluid dynamics, ventilation, smoke dynamics	fds, smv
	OpenCFD Openfoam	fluid dynamics, heat transfer, stress concentrations	openfoam
	Activestatics	structural demonstration	java
	RevitStructure, STAAD	structural analysis	rvt

Figure 20: Non-exhaustive selection of modeling and simulation software.

XXIV.Sandbox Coding Environments

The development of design-related code, or algorithmically dense design content, depends on a bipartite process. The actual code development, the 'programming' is subject to strict interpretation, and must be syntactically correct in order to run. That is, the program must be coherent to the programming environment. At the same time, the coder/designer/artist develops a design concept or design intent that consists of a variety of semi-related inputs, and may vary significantly with respect to the necessities of the software development. To synchronize these disparate processes, translation layers have been created to speed the connection between design and code; to allow high-level, abstract, code to be quickly converted to more powerful language environments. Alternatively, these translation layers allow simplified code to directly run.

There are been several programming environments developed to lower the overhead involved in creating software. Some of the most interesting environments have come from clusters of development around graphic programming or graphic data analysis. Specifically, the work of Ben Fry and Casey Reas at MIT's Aesthetics and Computation Group (ACG), who have developed an open source graphical environment for interaction and visualization of a variety of data sources: processing.[54] Within the

[54] Processing.org, "Basics \ Processing 1.0," accessed November 14, 2010, http://www.processing.org/about/.

Processing environment, the underlying code is exposed but the programming work necessary for generating patterns is minimized. This allows more direct experimentation with algorithmic and visual phenomena.

For audio/visual media experimentation and rapid development, the package vvvv is a popular environment[55]. Matlab and mathematica are both powerful analysis environments, but have steep learning curves. Maya is well suited to complicated modeling and animation, but has limited analytical capabilities. However, however, Maya encapsulates a python developing environment, which enables extending the analytics capabilities of the package.[56]

Finally, the pymel package[57] allows deep integration with standard python libraries. In conjunction with the pyevolve package,[58] this enables relatively simple integration of existing evolutionary algorithm content and settings within the spatial environment.

XXV.Speculative Experimentation

What is the potential of scripted generating procedures?

Just as systemic behavior can be characterized as bottom up or top down, we can do the same with visualizable analytics. One type of top down analysis would involve the use of complex or 'gestalt' focused tools, such as lighting or texturing, that would result in a whole scene being treated as a unified element. This is in opposition to the unit elements of small scale FEM or FEA modeling, which are usually based on individual pixels, vertices, or adjacencies.

---

[55] vvvv group, "vvvv: a multipurpose toolkit," accessed November 14, 2010, http://vvvv.org/propaganda.

[56] Autodesk, "Autodesk - Developer Center - Autodesk Maya", accessed November 13, 2010, http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=9469002.

[57] "pymel - Project Hosting on Google Code", accessed November 13, 2010, http://code.google.com/p/pymel/.

[58] "Pyevolve 0.6rc1 released ! | Pyevolve", accessed November 13, 2010, http://pyevolve.sourceforge.net/wordpress/?p=1164.

Connecting some of these processes together have great potential. Each speculative procedure is targeted at manipulating space to facilitate a certain functional or phenomenal result. The hybridization is flexible; however, while some operations are can be combined 'a la carte,' others have sequential dependencies.

Potential 1: Distance Functions

1. walls as segmentation of the space.
2. individual humans as occupying the spaces.
3. integrate distance of agent to eligible wall surface, averaged over 360˚ .
4. visualize by vertical bar at that point/grid location.

Potential 2: Voxel Density

1. typology of change/transformation/object creation
2. repetition + or subtraction -
3. voxel addition (random or directional/directed)
4. voxel subtraction (thinning of density)

Potential 3: Accretion/Dissolution

1. accumulation of moss/mold/hair attachment
2. erosion and grooves wearing away

Potential 4: Hybrid

1. circuit diagrams of procedural chaining
2. create voxels of density x
3. repeat y times with transition z (with turb?) @ each
4. apply eroder A
5. remove density B
6. corrode with filter C

Potential 5: Growth

1. initiate seed object (e.g. brick)
2. object follows growth pattern
3. growth parameters varied
4. growth constraints varied (e.g. gravity, wind, load)
5. [optional] object traverses context, accumulating input effects

Potential 6: Perception

1. take a cluster of objects
2. duplicate and move a spot light along a path (a proxy model of human perception)
3. the results of an agent with certain perceptual behavioral characteristics occupying a digital model; it is a pseudo human experiencer.
4. the model is then the experience of the experiencer over time, encoded by light.

XXVI.Challenges(ii) Indirect Extrapolation

In a sense, we can model the process of perception in a single frame, as a radiating source (for example, of light). Then, if we animate, then we see the procedural advance of perception in stepwise time.

We can change the model physical makeup based on its reaction as it interacts with a stimulus. For example, light can have a corroding effect. or wind can cause accretion.

In this kind of simulation, the rational or physical behavior of the components are not the driving force.

Physical simulations are of course important, and give the shape of the object / focus, but there is also metaphysical visualization; the visualization or communication of experience. The pure communication of experience may be more fully possible via a-rational or anti-rational models. The physical act is one that can be transmitted through our optical sensorium. Although it is not "experienced" per se, it is translated and applied. In the moment of viewing, we are not re-experiencing, but we are re-viewing an existing singular experience, with an eye toward the unfolding multiplicity of experience-corridors or perceptual avenues in the future. In viewing a modeled experience, we are creating an architecture of experience, or perhaps a meta-structure of meta-physics.

In what way could one communicate Steven Holl's use of light? How could you transmit the sense of geometric purity of Boulee, or the monumental banality of a typical strip mall or commercial office building?

What kind of analysis yields the depth of the suffering of the human spirit? What kind of process or procedure can renovate the soul?

XXVII.Design Phase

A. Description

The implementation of the cluster of topics discussed in this thesis, was shaped and directed towards demonstrating and exploring the conceptual foundations of the method, while ensuring the capability of a complete iterative procedure in script. Computational challenges and unfolding complexity constrained the level of realism or real applications of the iterative selective procedure, but the resulting process is a foundational step toward computational design optimization development.

The procedure is the generating sequence for the geometric and spatial work that follows. The individual subdivisions of the procedure include the generation of geometry (a unit object), the manipulation or 'forcing' of that geometry, an evaluation or 'fitness function' of the individual object, and a resulting set of iterations, interim objects, that culminate in an outcome object.

The variation of parameters that define the procedure include basics such as scale, location, and grid definition, but also proxy calculations of object complexity and pseudo-ecological parameters.

Within the overarching procedure, the sequence of operations is also determinate of the procedural outcome. Within each loop of a particular iteration, the component operations are executed as an ordered set. The manipulations are switched on independently within a certain sequence, and are themselves iterated as specified.

In addition, the choice of optimization algorithm is not an explicit variable, but nonetheless has intrinsically fundamental results. In this exploration, the optimization algorithm is held fixed. It is the primary critical piece of overall algorithm.

B. Components

The range of consequences of the system depends on the choice of the individual component. The geometric mesh face is the basic unit with which three-dimensional entities are represented within a surface modeling environment, and is the 'polygon' used

atomistically in rendering metrics to evaluate hardware performance. The individual plane, subdivided, is the primary surface unit. The cube and sphere is volumetrically the simplest form.

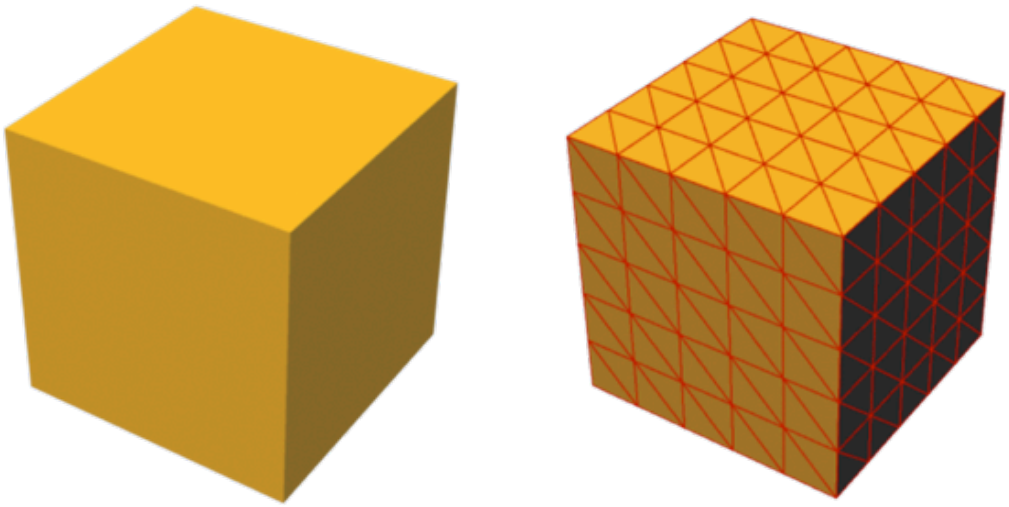


Figure 21: Cuboid component option. Subdivisions of the cube are visible in wireframe. Algorithms act on vertices and individual faces within the cube.

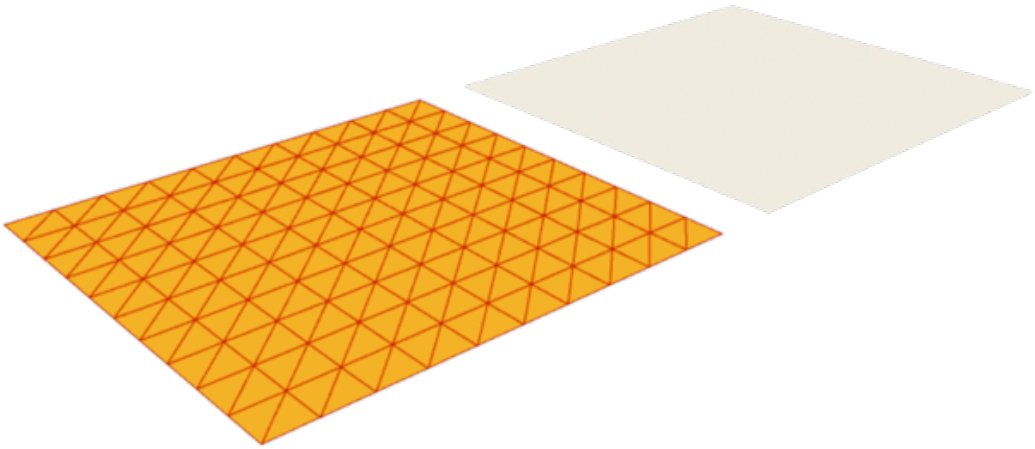


Figure 22: Planar component option. Planar subdivisions are similar to cube.

These spatial 'primitives' are the simplest to define geometrically. A bottom-up shape definition is the alternative procedural option. The qhull algorithm and software package facilitates taking a random cloud of three-dimensional points, and forming the convex hull around those points. This hull is a better approximation of real spatial objects, by reacting to simple primitives rather than a strictly defined plan or angular precision. The architecturally real world respects the vertical and horizontal planar limits, so by constraining the input points to the surface of a ideal cube, the convex hull becomes a precursor of an architectural object. It also deviates from the strictly orthogonal, to avoid the constraints implicit in staying normal to gravity forces. This deviation also results in the additional potential for a nuanced relationship between face normal vectors and incoming light on a particular face.

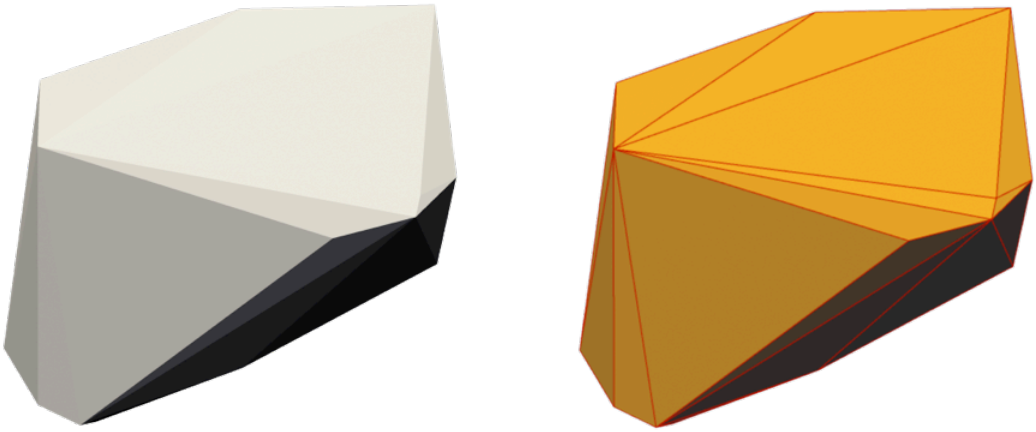


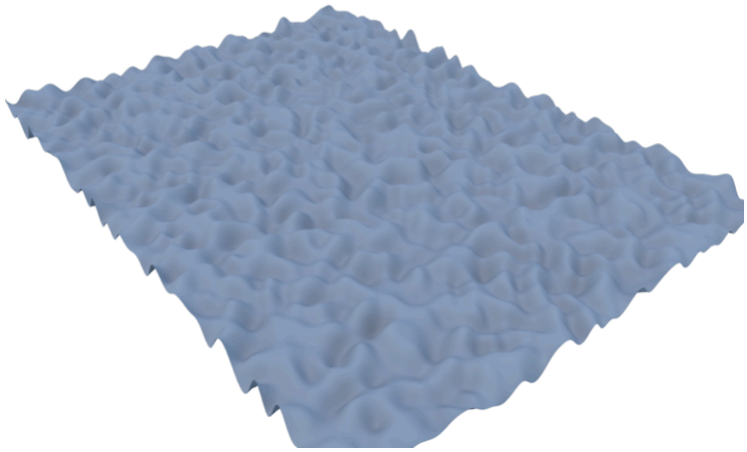
Figure 23: Convex hull component option. Hulls are formed of unitary faces, which are not subdivided.

Algorithm Description	
Component	**regular polyhedron**
Parameters	**scale, spacing, material, origin, random range**
Operations	**generate, move**
Iterations	**10 x 20**
Population	**200**
Generations	**1**
Selection	**none**

Figure 24: Stepwise shape creation.

C. Deformation / Transformation / Operation

The deformation process is subsequent to the foundational unit. Several deformations perform as functional analogs for physical processes. The basic deformation for the plane is the manipulation of control vertices along the normal vector to the plane; in local space, along the z axis from the plane face.



Algorithm Description	
Component	**vertices of the subdivided plane**
Parameters	**vertical deformation of each vertex**
Operations	**move along z axis**
Iterations	**1**
Population	**400**
Generations	**1**
Selection	**none**

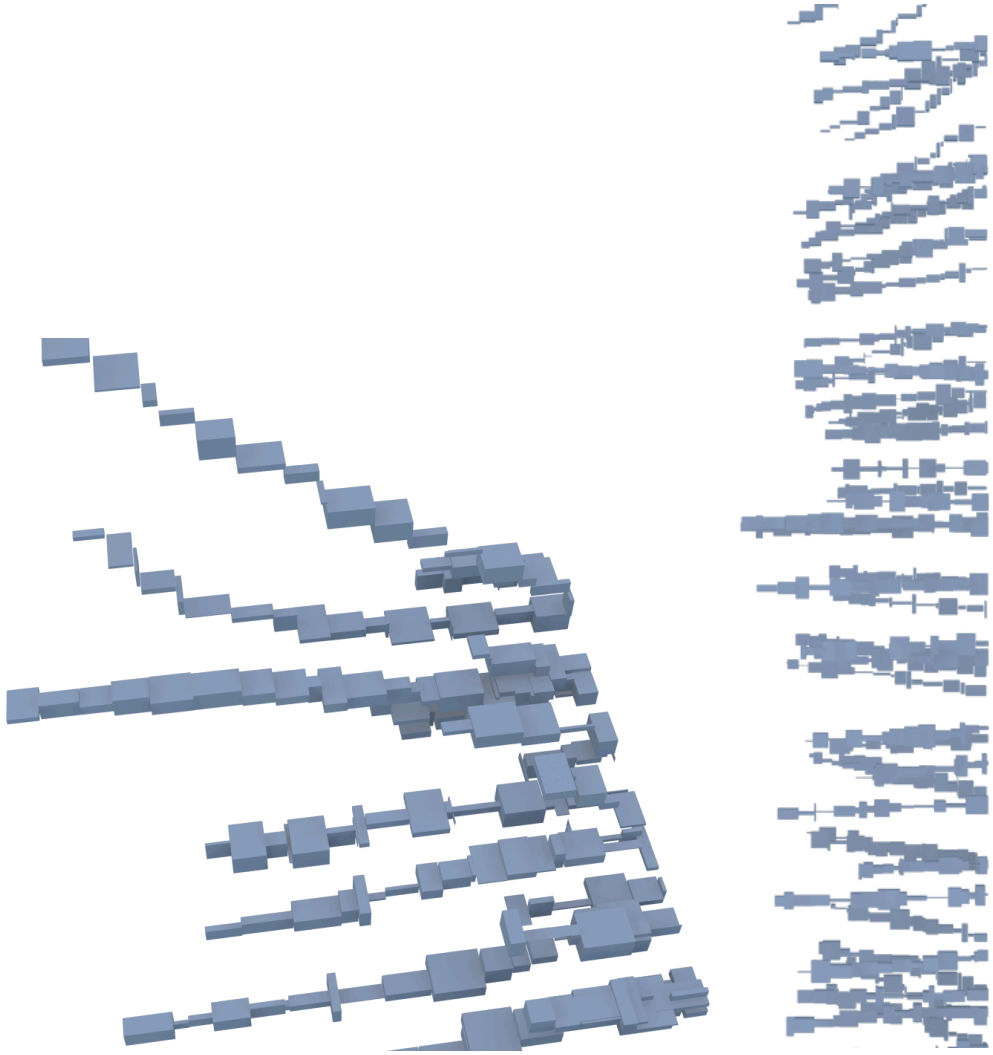
Figure 25: Planar deformation.

The deformation of the plane can be performed proportionally to an axis location. This is a linear proxy, that is roughly equivalent to a falloff value; as the center transformation moves further away from the targeted shape, the transformation intensity decreases. Most physical processes, including acoustical energy transmittance, thermal conductance, earthquake intensity, or electrical charge in a medium, exhibit a falloff behavior, e.g. "with a simple nondirective source, the sound intensity will fall off as the distance from the source is increased."[59]

---

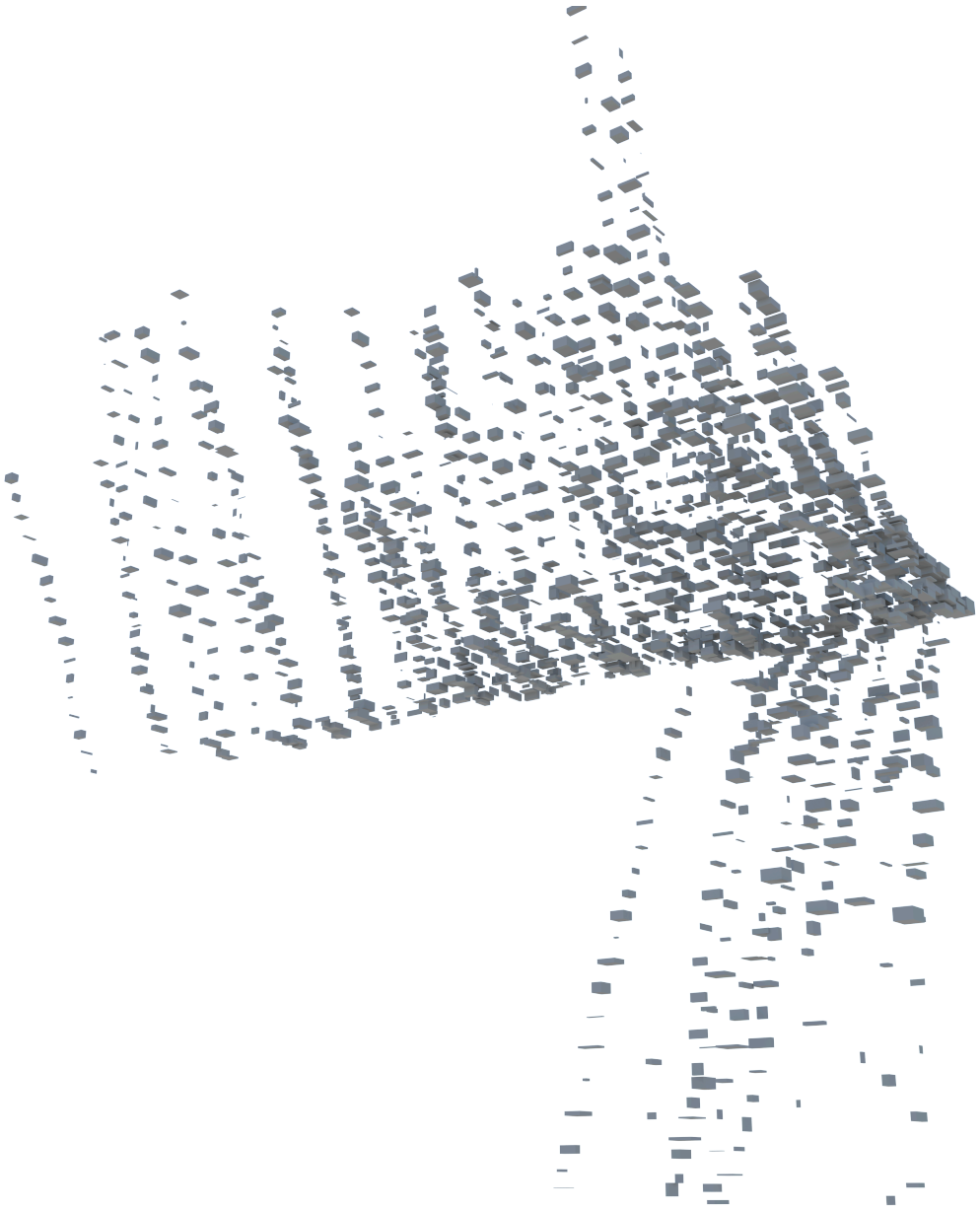
[59] William J. Cavanaugh, Gregory C. Tocci, and Joseph A. Wilkes, *Architectural Acoustics: Principles and Practice* (John Wiley and Sons, 2009), 11.

Parametric variations include the dimension of individual units, the boundary marginal zone, and the number of individual objects per line of objects.



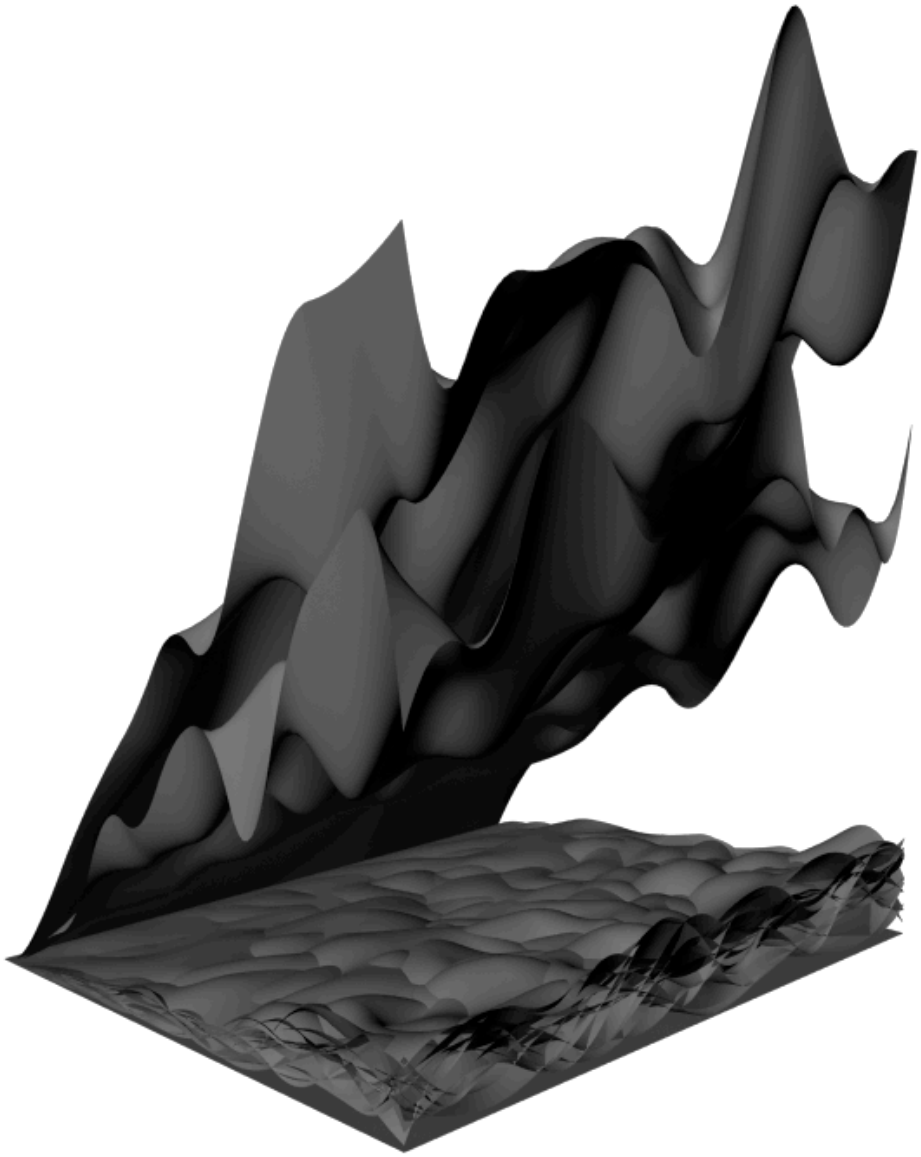
Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along z axis**
Operations	**generate, move to boundary of previous**
Iterations	**20 x 50**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

Figure 26: The distribution of individual elements via each unit's dimension in sequence.



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along z axis, gap size to next individual**
Operations	**generate, move to boundary, add gap dimension**
Iterations	**20 x 50**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

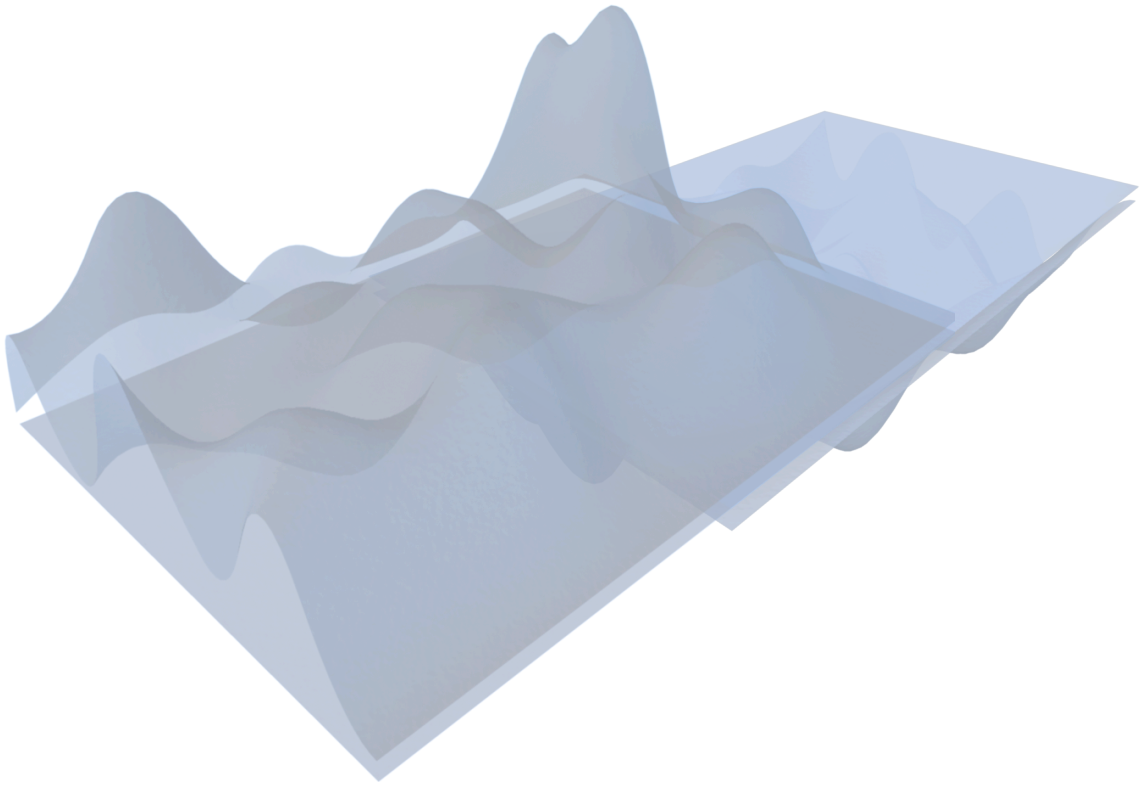
Figure 27: The distribution of individual elements via each unit's dimension in sequence.



Algorithm Description	
Component	**vertices of subdivided plane**
Parameters	**vertex along z axis, distance from origin**
Operations	**move vertex along z scaled to distance from origin**
Iterations	**12 x 400**
Population	**4800**
Generations	**1**
Selection	**none**

Figure 28: Vertices transform via falloff function.

The planar transformation takes the form of an overhead plane, or a landscape topographic surface. The topology can be multivalent: physical, psychogeographic, thermal, acoustic, or ecological.



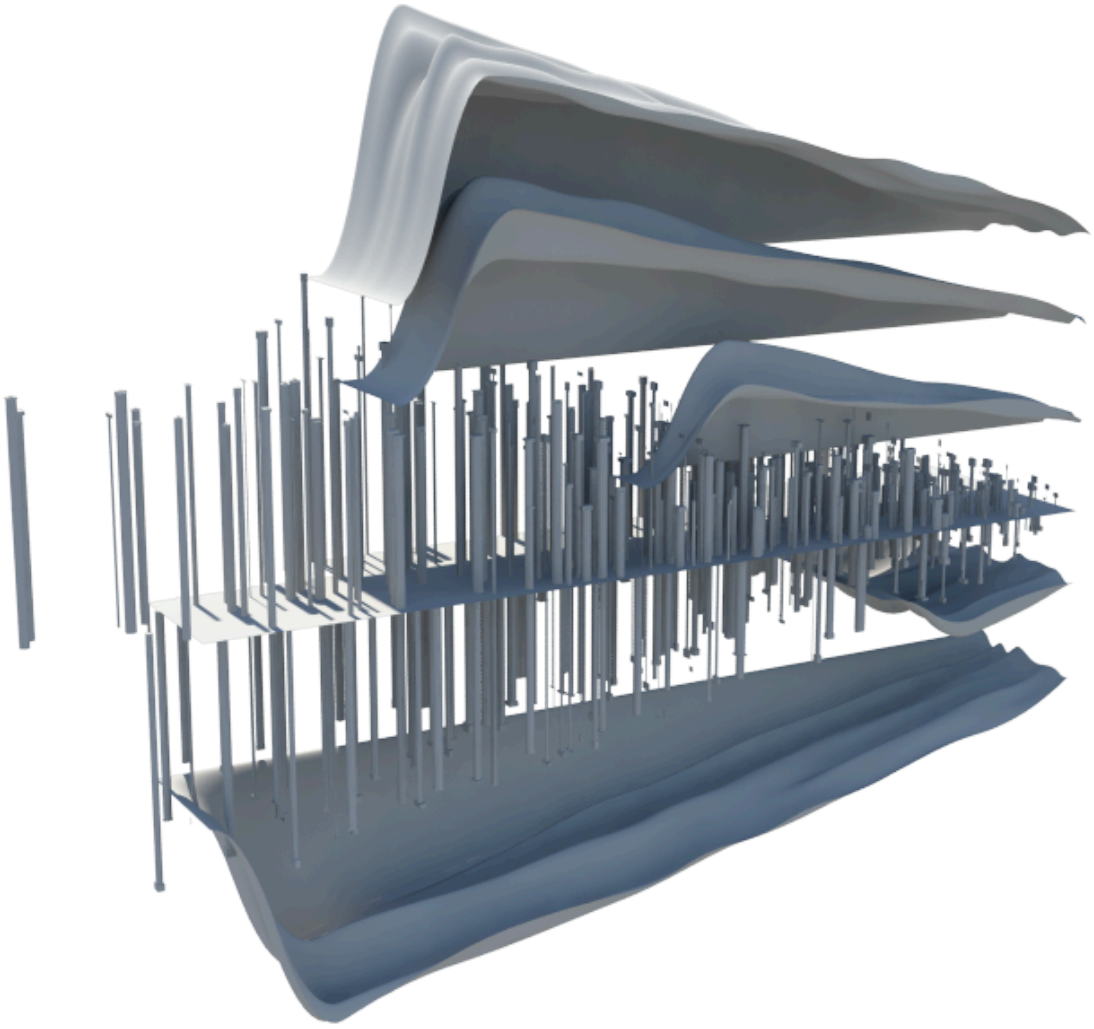
Algorithm Description	
Component	**vertices of subdivided plane**
Parameters	**vertex along z axis**
Operations	**move vertex along z scaled to distance from origin**
Iterations	**12 x 12**
Population	**144 x 2**
Generations	**1**
Selection	**none**

Figure 29: Planar roles, superior and inferior.

D. Sequencing

Individual units, that respond and interact on an individual level with a whole entity, are useful proxies. The simple system of vertical supports and planar membranous 'roof' elements results in a complex interacting condition. The individual verticals act as structural supports and embedded individuals, while the membranes spread over the

population of individuals provide nonstructural functions: shade, spatial definition, and enclosure. In addition to the direct effects, the dense clustering of elements bounds the circulation space and defines an implicit expression of surface.



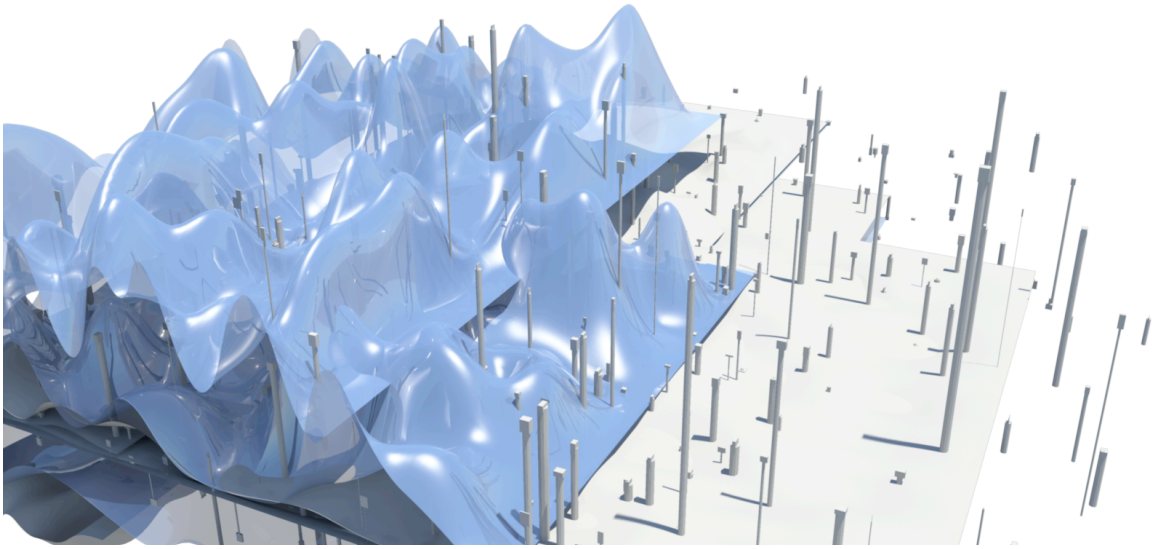
Algorithm Description	
Component	**vertices of subdivided plane, cylinder dimensions, cuboid dimensions**
Parameters	**vertex along z axis, location and scale of cylinder, location and scale of cuboid**
Operations	**generate, move object, move vertices**
Iterations	**100 x 5**
Population	**500**
Generations	**1**
Selection	**none**

Figure 30: Surface and vertical arrangement.

Algorithm Description	
Component	**vertices of subdivided plane, cylinder dimensions, cuboid dimensions**
Parameters	**vertex along z axis, location and scale of cylinder, location and scale of cuboid**
Operations	**generate, move object, move vertices**
Iterations	**100 x 6**
Population	**600**
Generations	**1**
Selection	**none**

Figure 31: A dense field of columnar elements.

The field is then expanded and differentiated into sparse freestanding columns (individuals) and columns engaged with the surface element.



Algorithm Description	
Component	**subdivided planes, cylinders, cuboids**
Parameters	**location in space, dimension along each x & y axes, distribution in space, intensity of z move**
Operations	**generate, move objects, scale objects, move vertices**
Iterations	**6 x 100**
Population	**600**
Generations	**1**
Selection	**none**

Figure 32: Two landscape types: sparse and engaged.

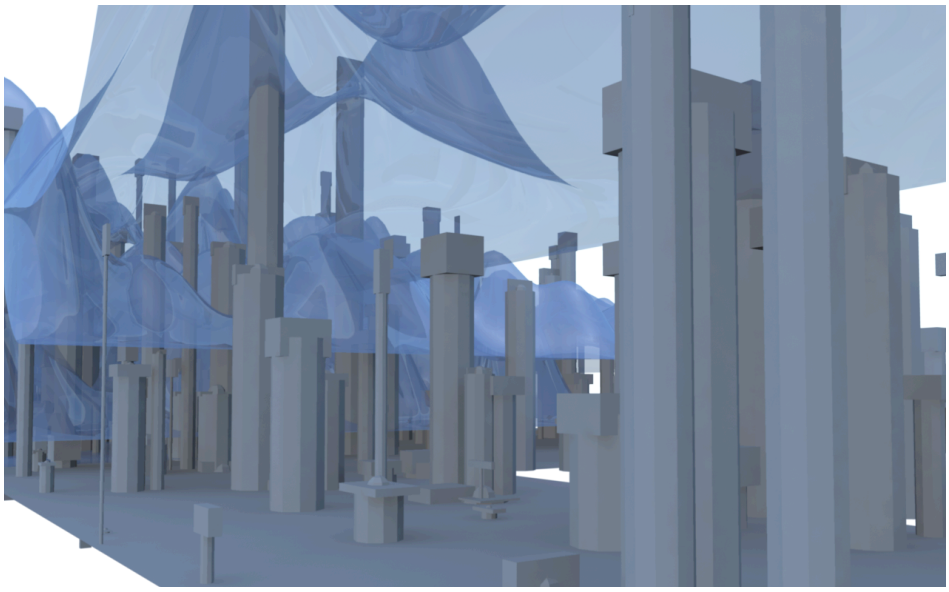
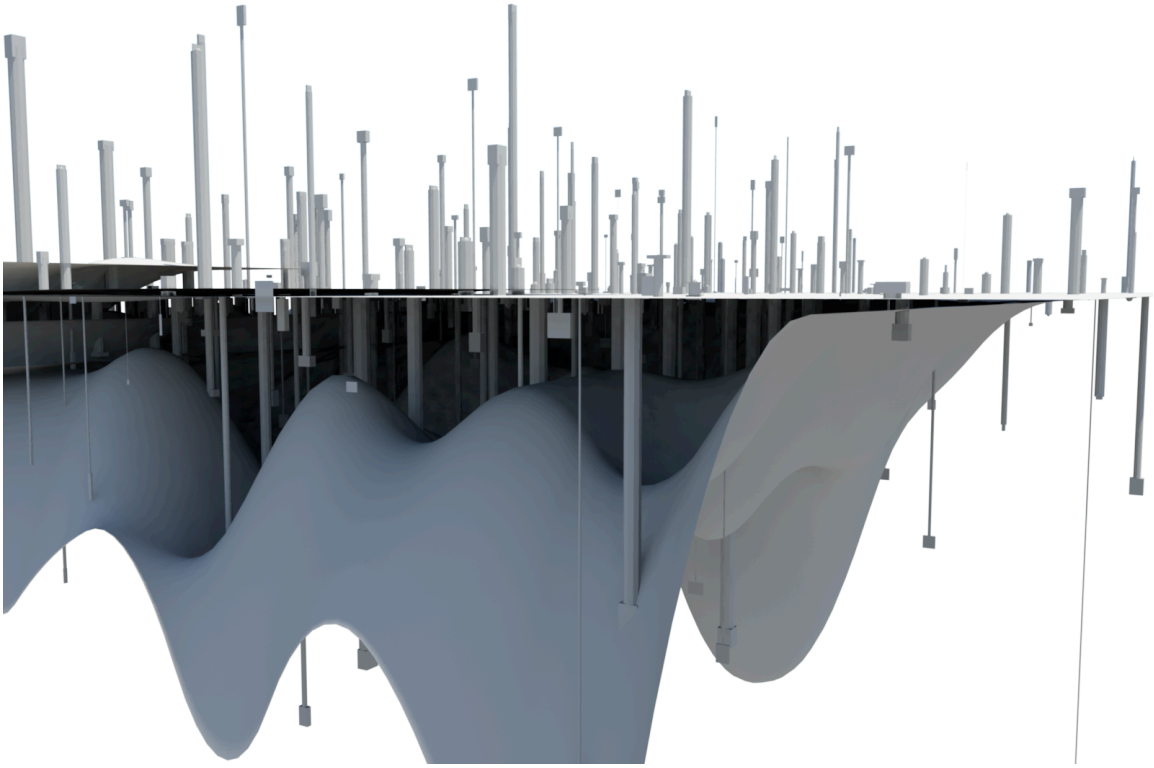


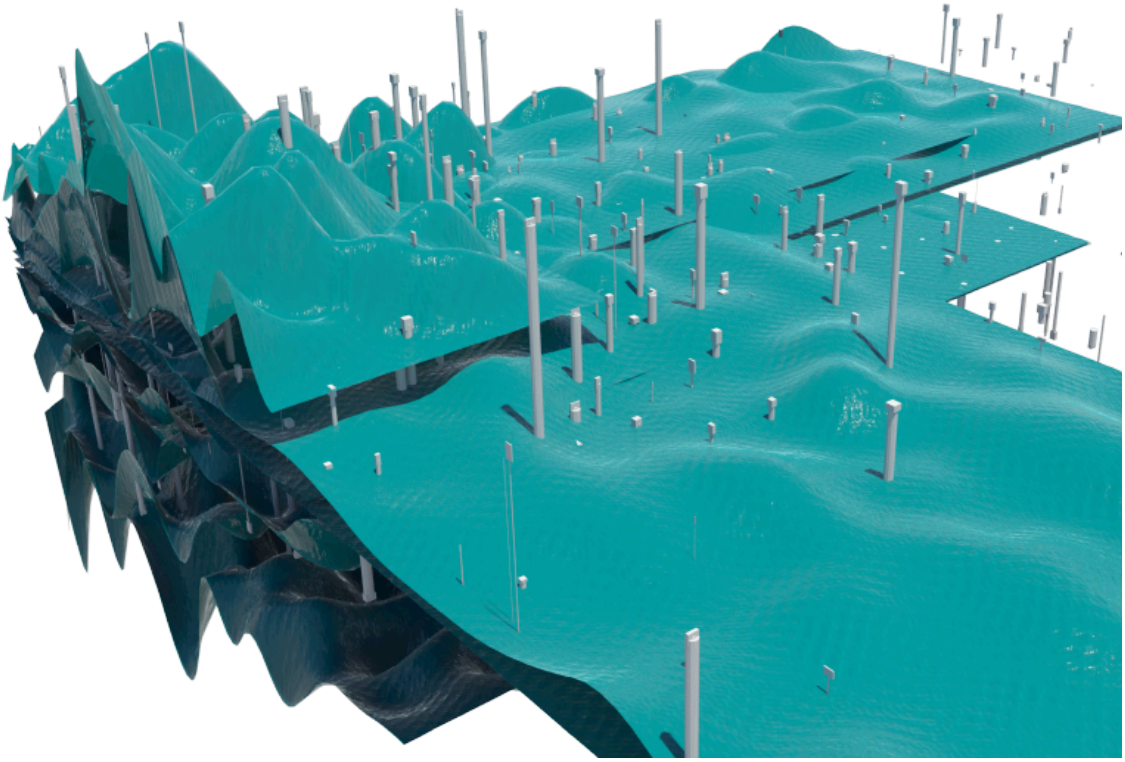
Figure 33: Previous figure, alternate view. Engaged landscape from the interior.



Algorithm Description	
Component	**subdivided planes, cylinders, cuboids**
Parameters	**location in space, dimension along each x & y axes, distribution in space, intensity of z move**
Operations	**generate, move objects, scale objects, move vertices**
Iterations	**6 x 100**
Population	**600**
Generations	**1**
Selection	**none**

Figure 34: View through the horizontal plane.

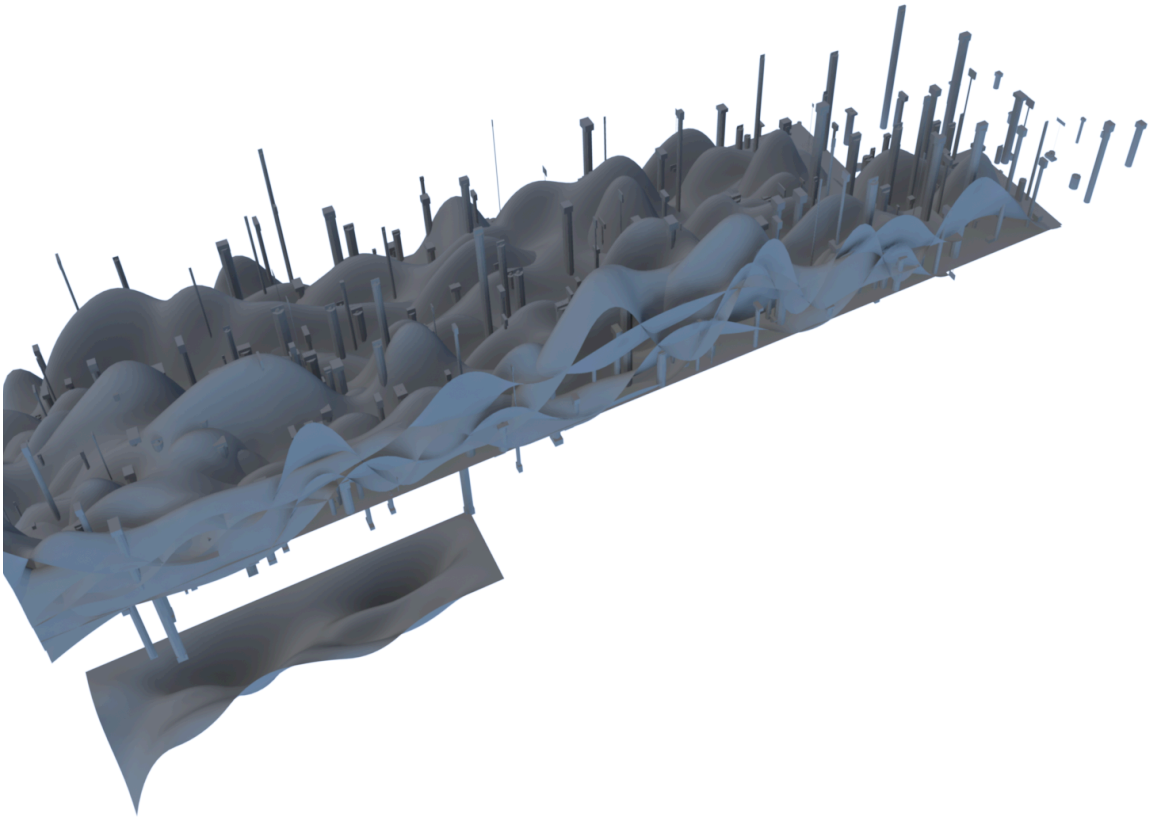
The repeated overlaying of the membranous element builds up a multiple landscape, and implies a multiple reading of these objects. Each membrane-scape might refer to a unique parameterization of the individual unit spaces, or might connect the individual parameters into a global representation of their interacting system.



Algorithm Description	
Component	**subdivided planes, cylinders, cuboids**
Parameters	**location in space, dimension along each x & y axes, distribution in space, intensity of z move**
Operations	**generate, move objects, scale objects, move vertices**
Iterations	**10 x 100**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

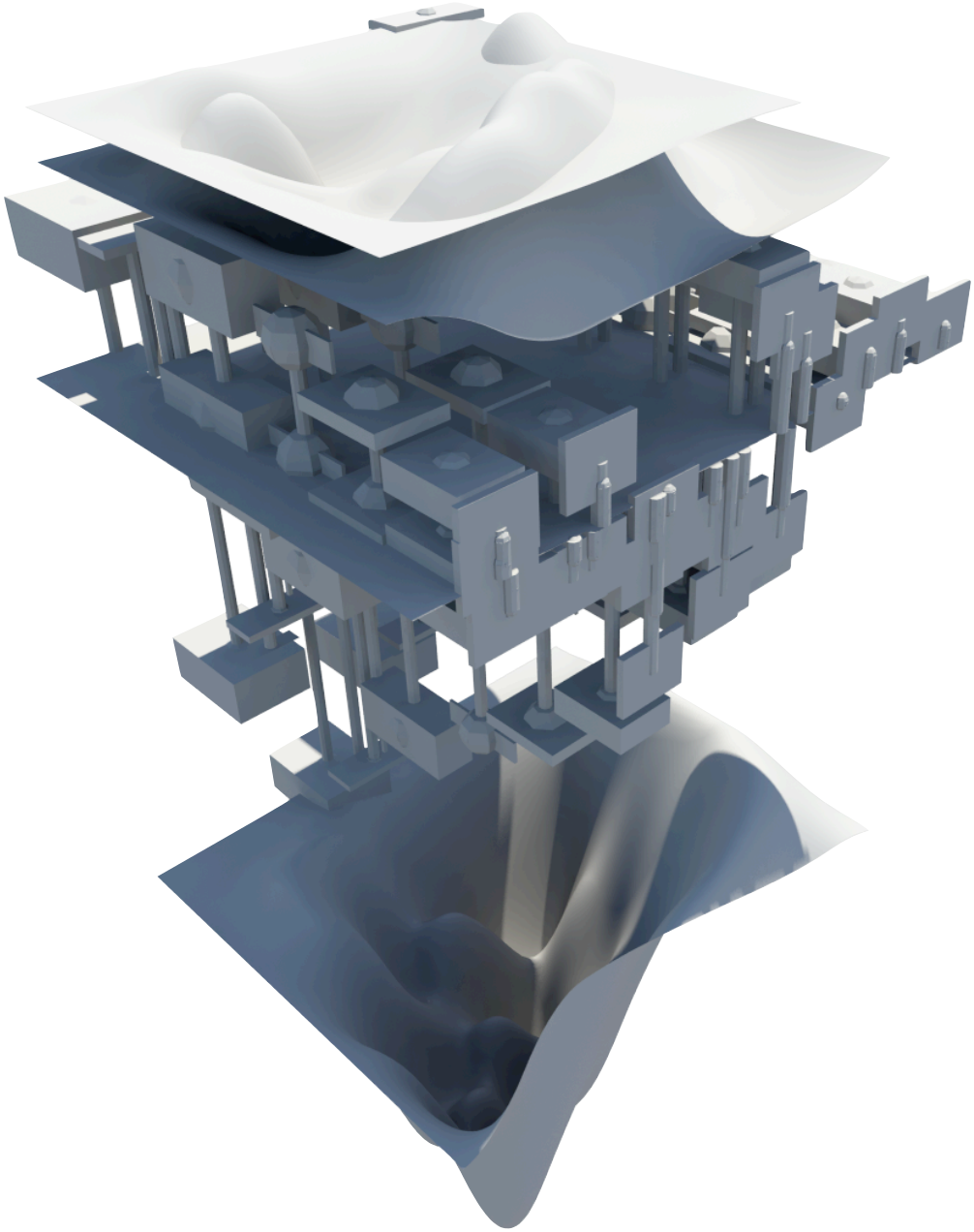
Figure 35: Overlay of multiple column/surface systems.

Membranous stacking also can indicate the repeated action of a similar system, for example the laying done of bone material as a protection against future injury. In the landscape, this accretion of surfaces might protect against a landscape violation, or might be a reaction to repeated excavations.



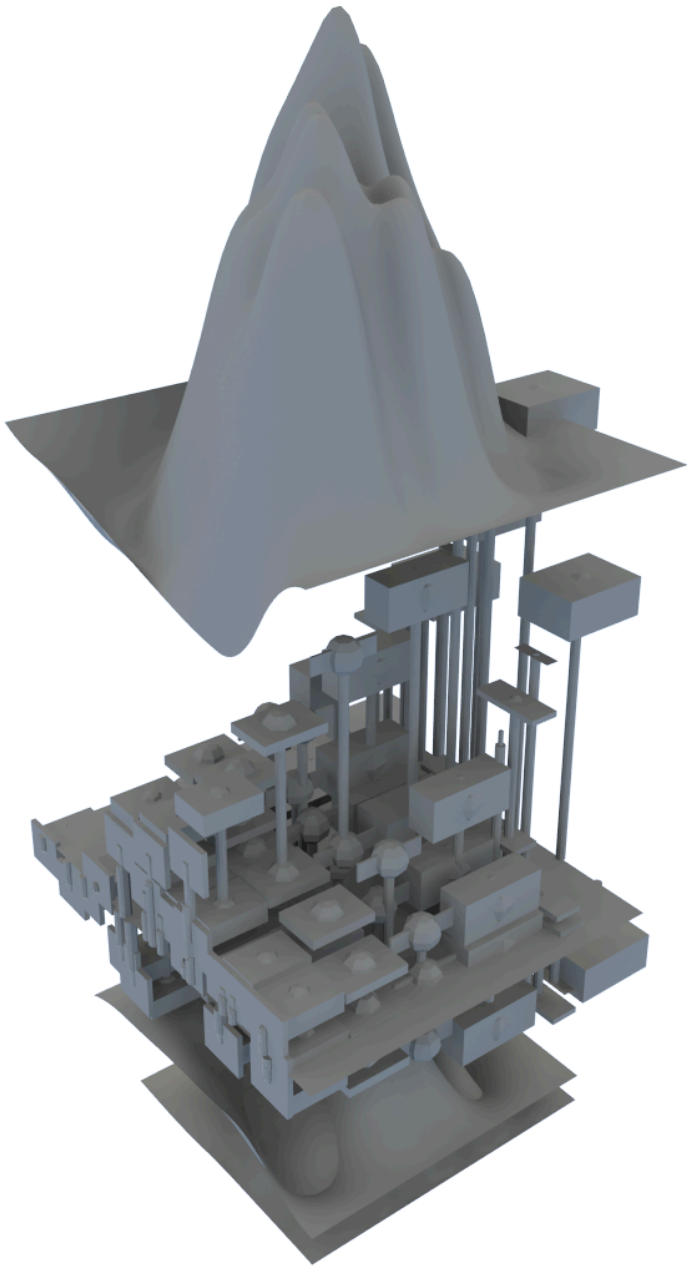
Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along z axis, alignment along x axis**
Operations	**generate, move objects, scale objects, move vertices**
Iterations	**20 x 100**
Population	**2000**
Generations	**1**
Selection	**none**

Figure 36: The topology of surface and column depends on the proportion of generating processes along each axis.



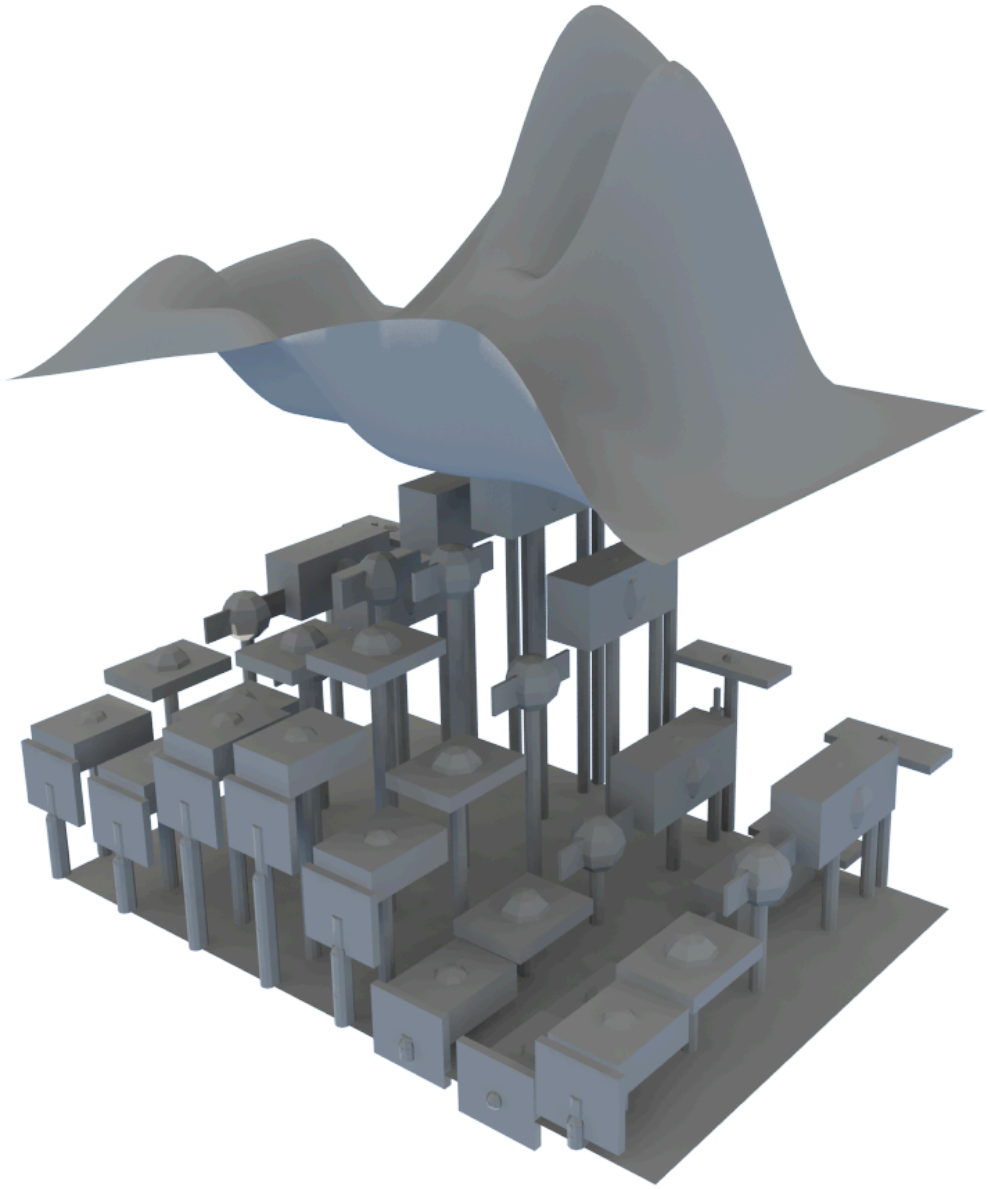
Algorithm Description	
Component	**cuboid, polyhedron, cylinder, subdivided plane**
Parameters	**dimension along each axis, step size along z axis, alignment along x axis**
Operations	**generate, move objects, scale objects, move vertices, smooth**
Iterations	**5 x 64**
Population	**320**
Generations	**1**
Selection	**none**

Figure 37: Surface, box, and column system version 1.



Algorithm Description	
Component	**cuboid, polyhedron, cylinder, subdivided plane**
Parameters	**dimension along each axis, step size along z axis, alignment along x axis**
Operations	**generate, move objects, scale objects, move vertices, smooth**
Iterations	**3 x 64**
Population	**192**
Generations	**1**
Selection	**none**

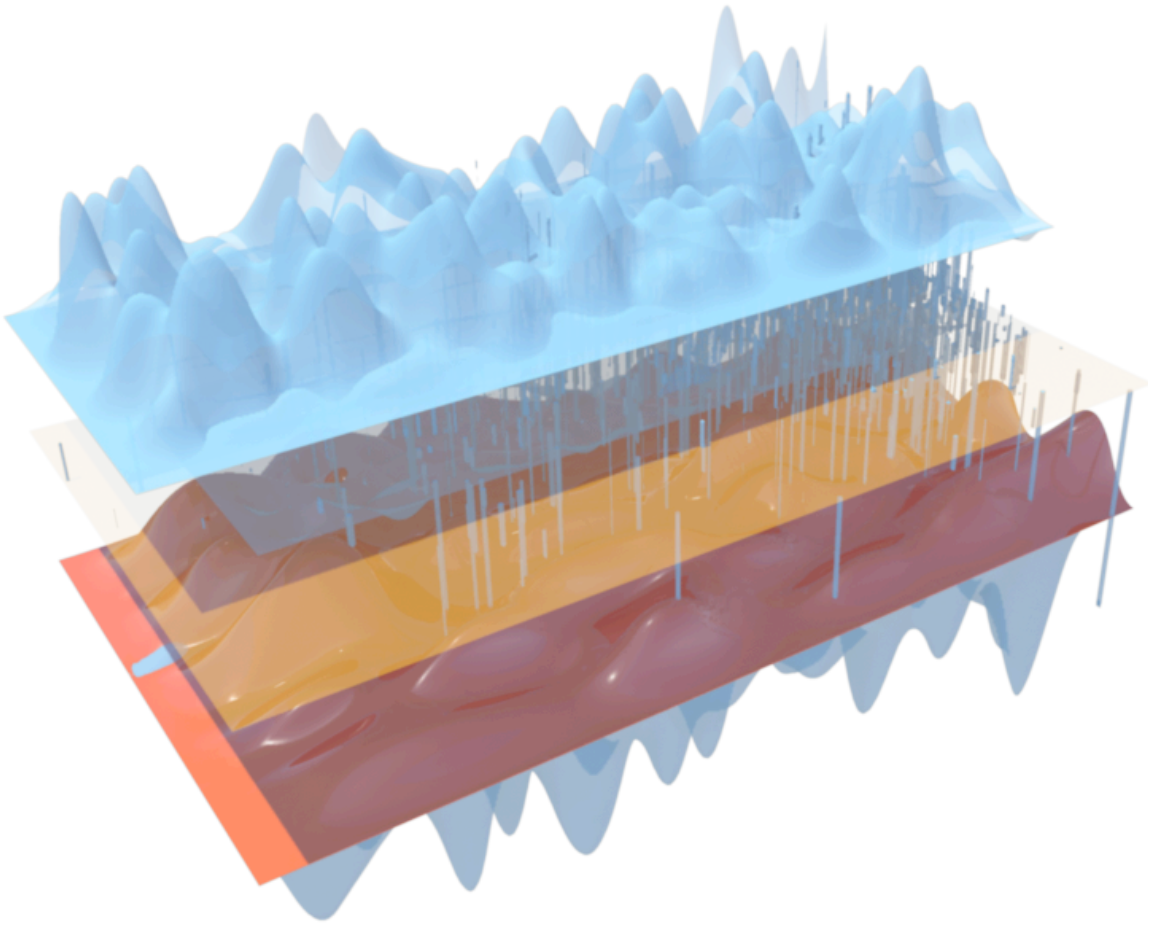
Figure 38: Surface, box, and column system 2.



Algorithm Description	
Component	**cuboid, polyhedron, cylinder, subdivided plane**
Parameters	**dimension along each axis, step size along z axis, alignment along x axis**
Operations	**generate, move objects, scale objects, move vertices, smooth**
Iterations	**2 x 64**
Population	**128**
Generations	**1**
Selection	**none**

Figure 39: Surface, box, and column system 3.

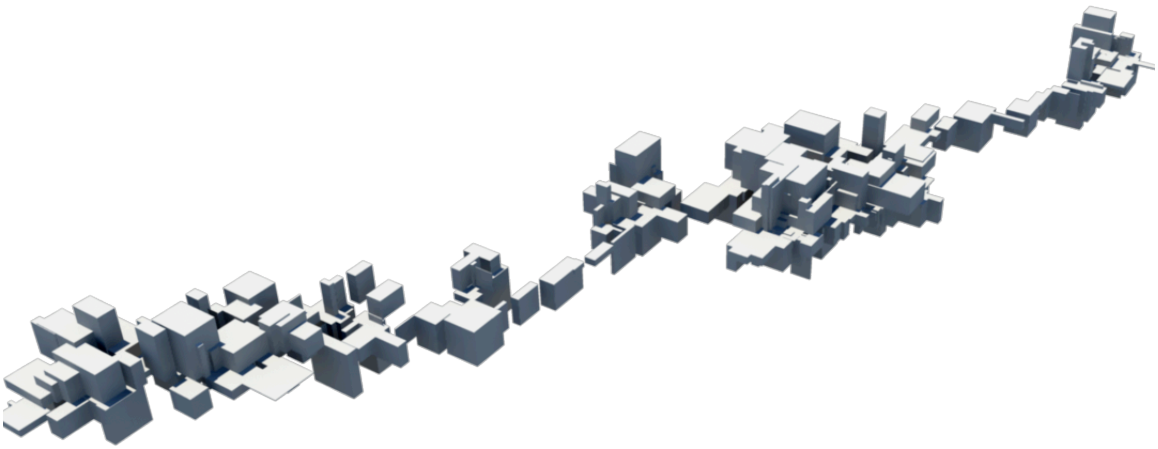
Separation of the membrane over the population of component objects indicates a more analytical function (rather than strict representation.) The analytical membrane could indicate population, density, economic indicators, or electromagnetic signalling.



Algorithm Description	
Component	**cuboid, polyhedron, cylinder, subdivided plane**
Parameters	**dimension along each axis, step size along z axis, alignment along x axis**
Operations	**generate, move objects, scale objects, move vertices, smooth**
Iterations	**5 x 400**
Population	**2000**
Generations	**1**
Selection	**none**

Figure 40: Surface, box, and column system delaminated and exposed.

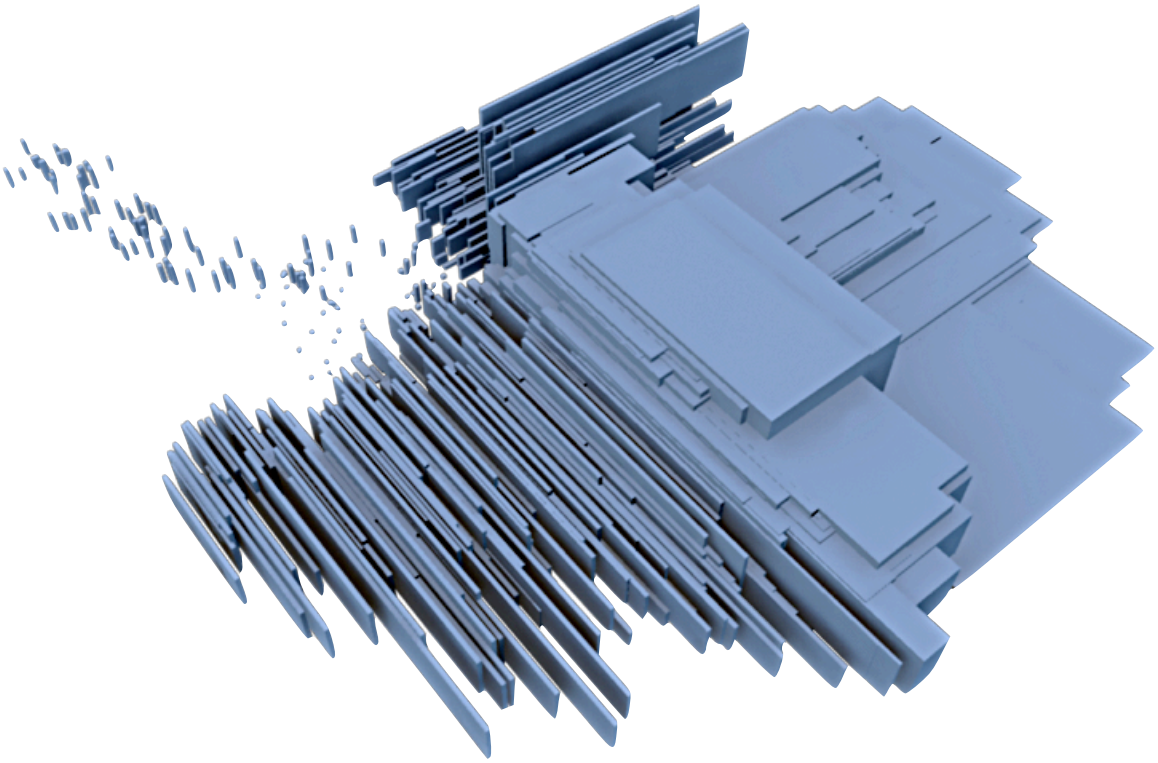
E. Action and Selection



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along each axis**
Operations	**generate, move to boundary of previous**
Iterations	**200**
Population	**200**
Generations	**1**
Selection	**none**

Figure 41: Sequence of cube generation.

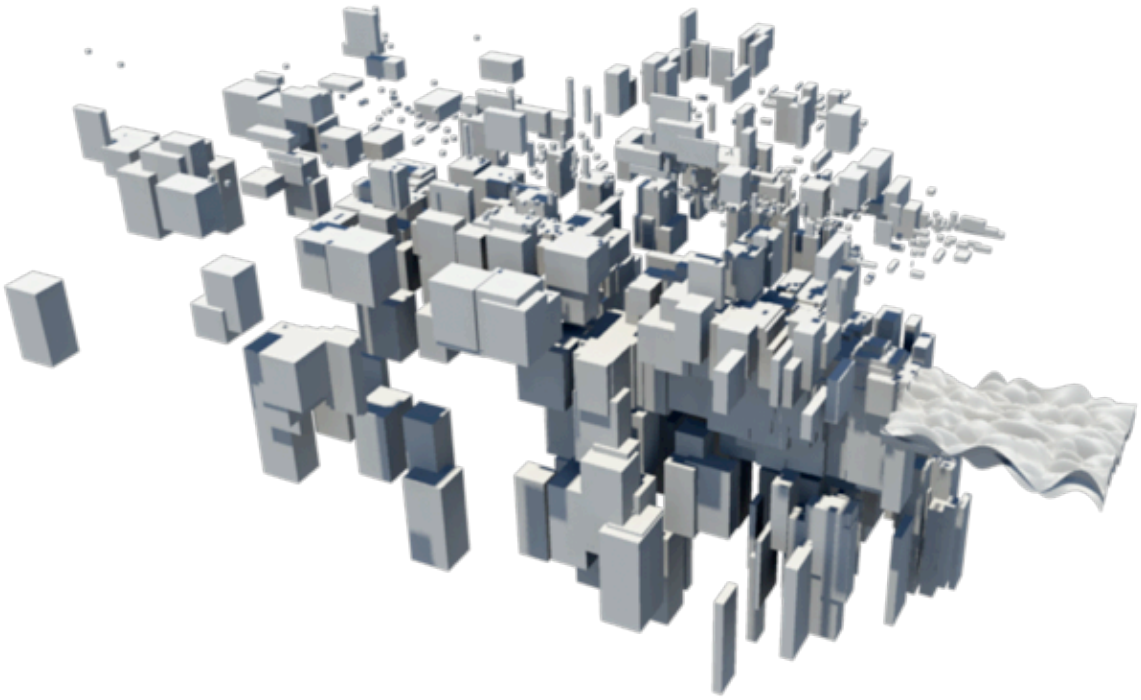
The distribution operation has a different end state, when used to assemble a population of objects contiguously. The population can be stacked or aligned to a key object, or the gaps between individuals can be eliminated, leading to a direct face-to-face stacking in the direction of least occupancy.



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along each axis**
Operations	**generate, sort**
Iterations	**400**
Population	**400**
Generations	**1**
Selection	**stacking along x and y axes by axial dimension**

Figure 42: Dimensional sorting of cubes.

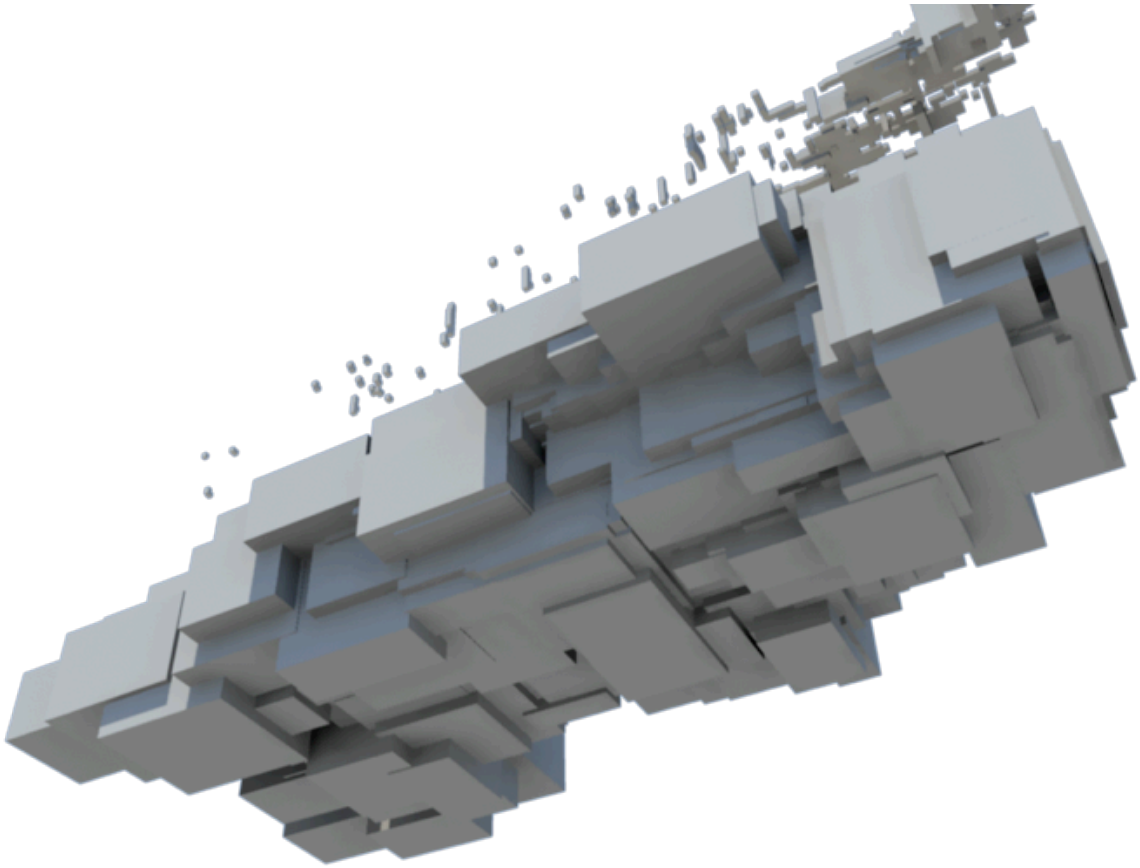
Alternatively, the stacking can be done parametrically, with the characteristics of each object determining in what order and in what direction the object is sorted. Extracting these basic aspects is used for migrating objects out of a population. The extract, in the above case, is a grouping or sub-population that satisfies certain parametric criteria.



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along each axis**
Operations	**generate, sort**
Iterations	**400**
Population	**400**
Generations	**1**
Selection	**stacking along each axis by axial dimension**

Figure 43: Dimensional layer and separation of cubes by dimension.

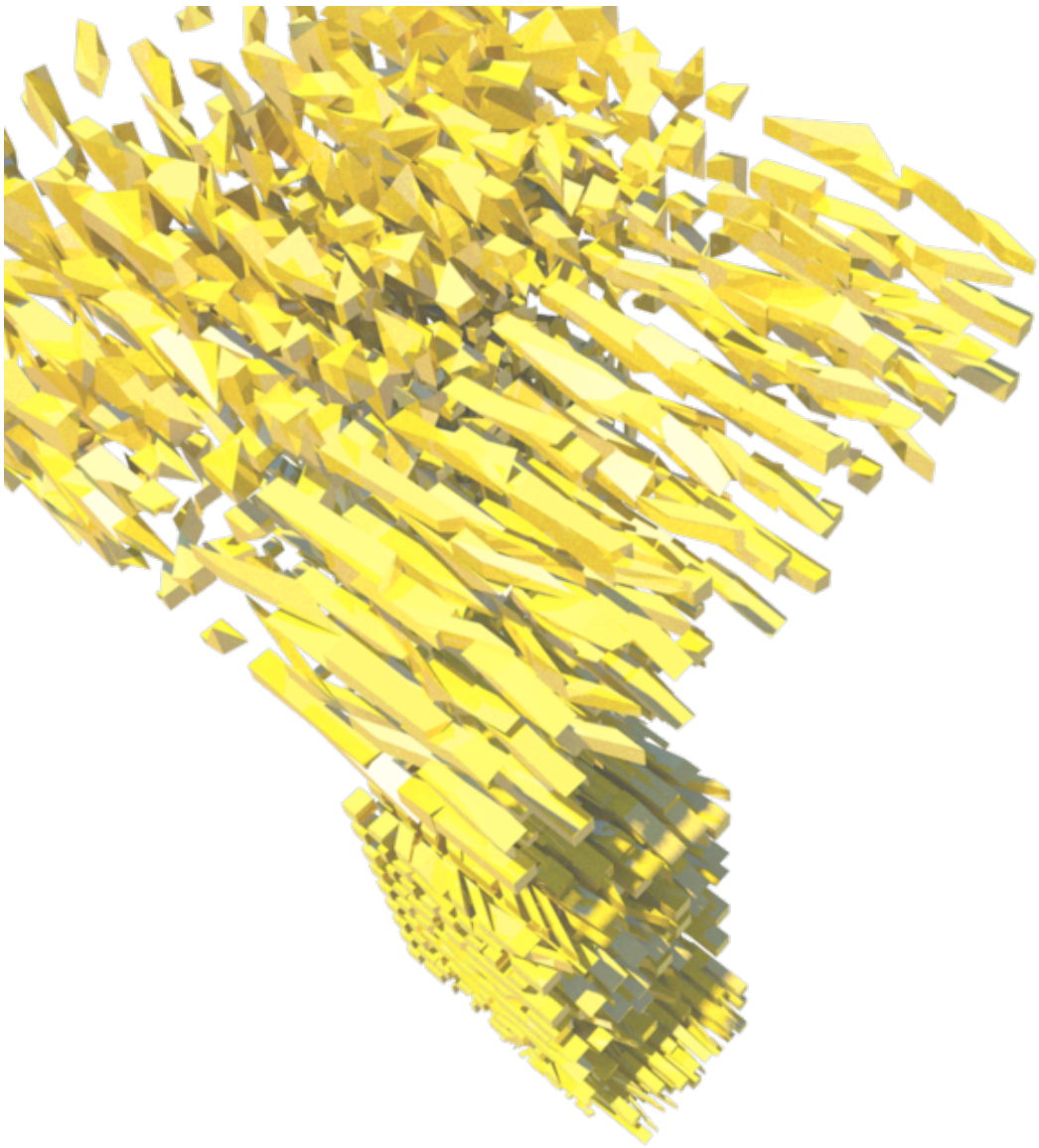
The stacking and sorting in three dimensions has an analog in density relationships, for example the size of stones in flowing water. Differentiation into classes (of total volume, here) is a way of intrinsically classifying a population. Classification can be used for functional grouping in order to identify subsequent operations.



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along each axis**
Operations	**generate, sort**
Iterations	**400**
Population	**400**
Generations	**1**
Selection	**separation along z axis by axial dimension**

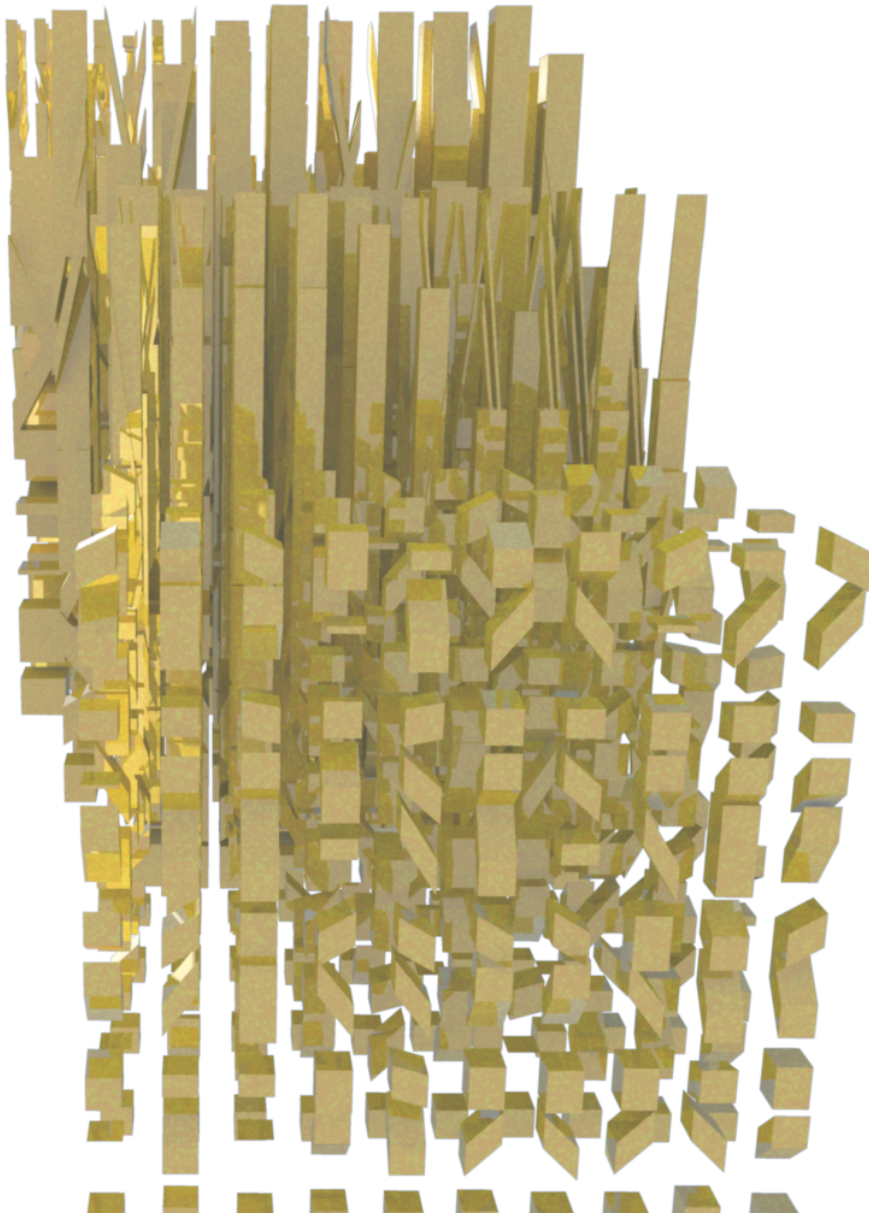
Figure 44: Dimensional layer and separation of cubes by dimension.

Operations are also performed on relatively uniform populations, in which the operation is paramaterized by its position in space. Below, the operation of shearing a cubic object acts as a proxy for the penetration of light into the population space; the isolated individuals at the bottom are well shaded and relatively untouched, well the exposed faces are drastically transformed.



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along x,y, & z axis, shear intensity along long axis of cube**
Operations	**generate, arrange in grid, shear, move to z grid location**
Iterations	**225 x 20**
Population	**4500**
Generations	**1**
Selection	**none**

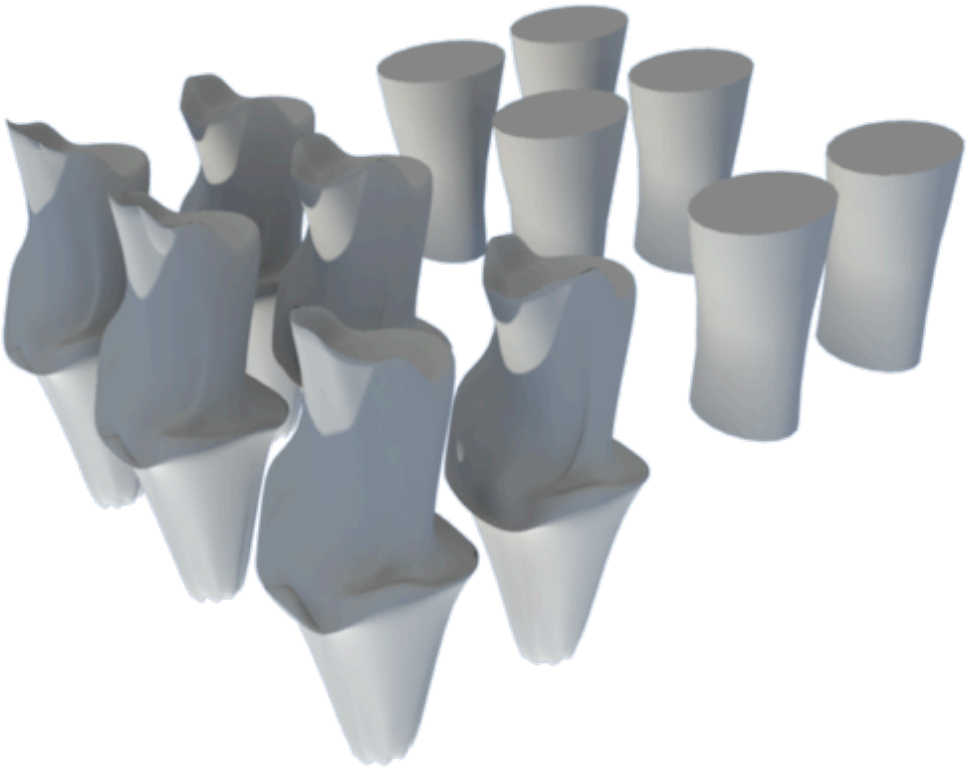
Figure 45: Dimensional sorting of variably transformed cubes.



Algorithm Description	
Component	**cuboid**
Parameters	**dimension along each axis, step size along x,y, & z axis, shear intensity along long axis of cube**
Operations	**generate, arrange in grid, shear, move to z grid location**
Iterations	**225 x 20**
Population	**4500**
Generations	**1**
Selection	**none**

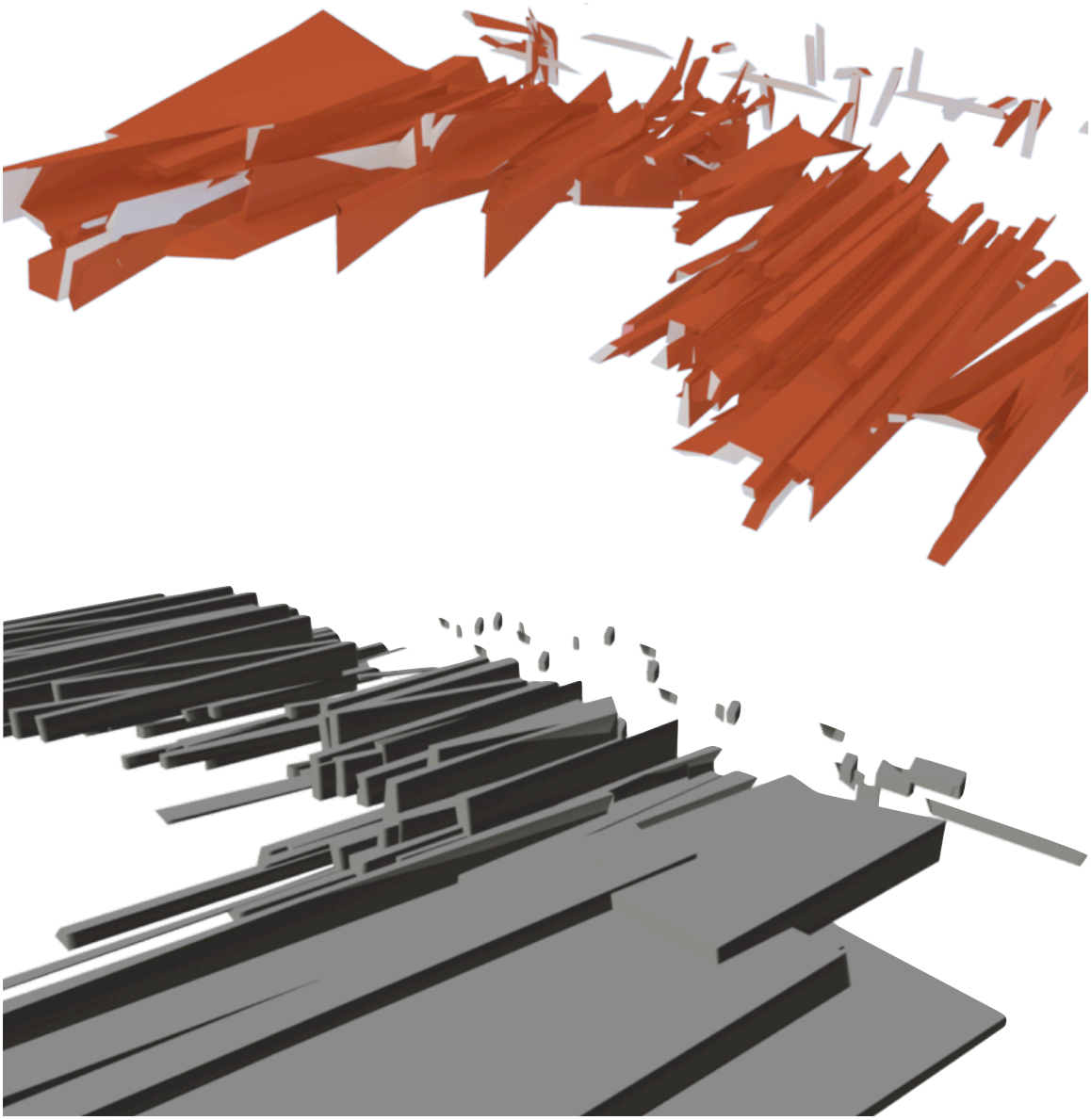
Figure 46: Transformation of cubes occur as a contingent factor of sequence in stack, as well as location within layer.

Transformations based on physical processes are more involved than relatively orthogonal vertex moves. Light, specifically, is a intensely relevant phenomenon for architectural objects and spaces. In the following procedures, the light input refers to the available light on the face of an object. This light is used to deform the faces and vertices of an object, such that an objects faces and vertices grow proportionally to their light input. In literal terms, this is a photo-synthetic procedure: light generates enlarged form.



Algorithm Description	
Component	**subdivided cylinder**
Parameters	**light intensity**
Operations	**generate, move vertex in z per incoming light**
Iterations	**2 x 6**
Population	**12**
Generations	**1**
Selection	**none**

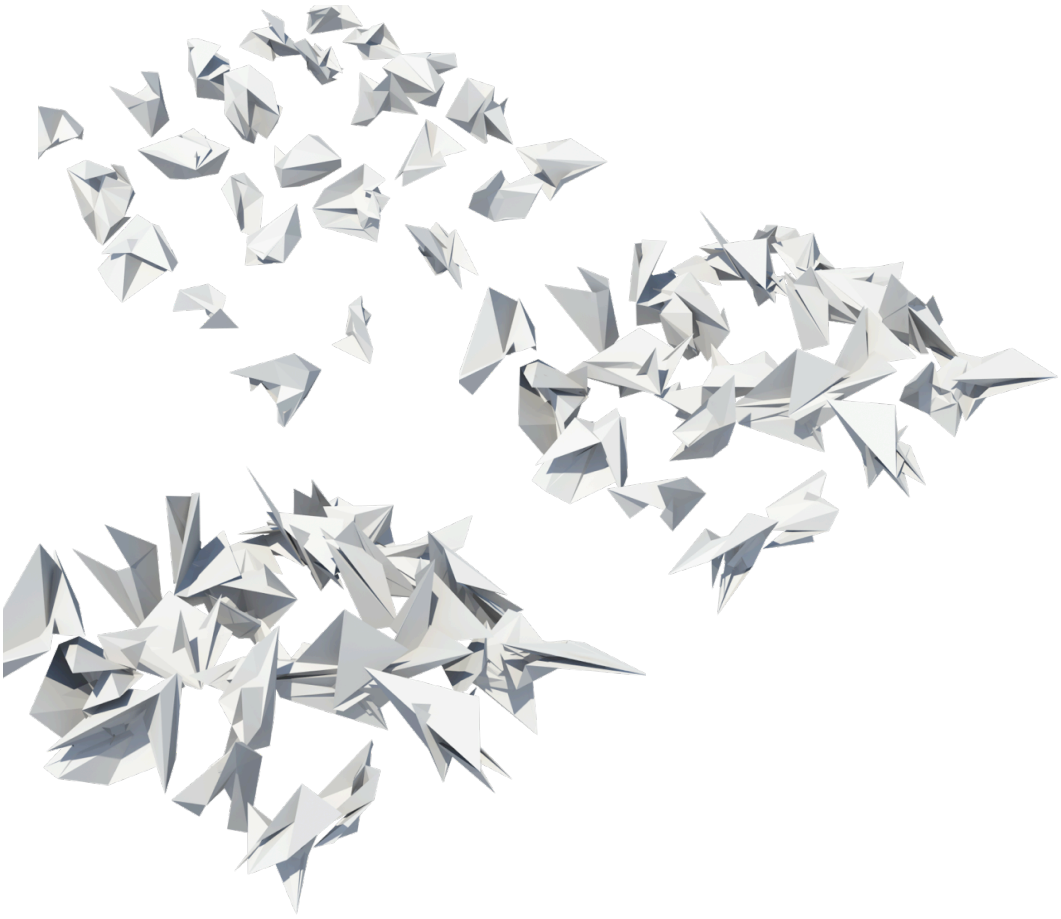
Figure 47: Enformation of simple cylinders under light (front 6 objects are enformed, back 6 are geometrically transformed.)



Algorithm Description	
Component	**cuboid**
Parameters	**light intensity, object scale along each axis**
Operations	**generate, move to boundary of adjacent, deform via incoming light**
Iterations	**50 x 2**
Population	**100**
Generations	**1**
Selection	**separated by overall edge length**

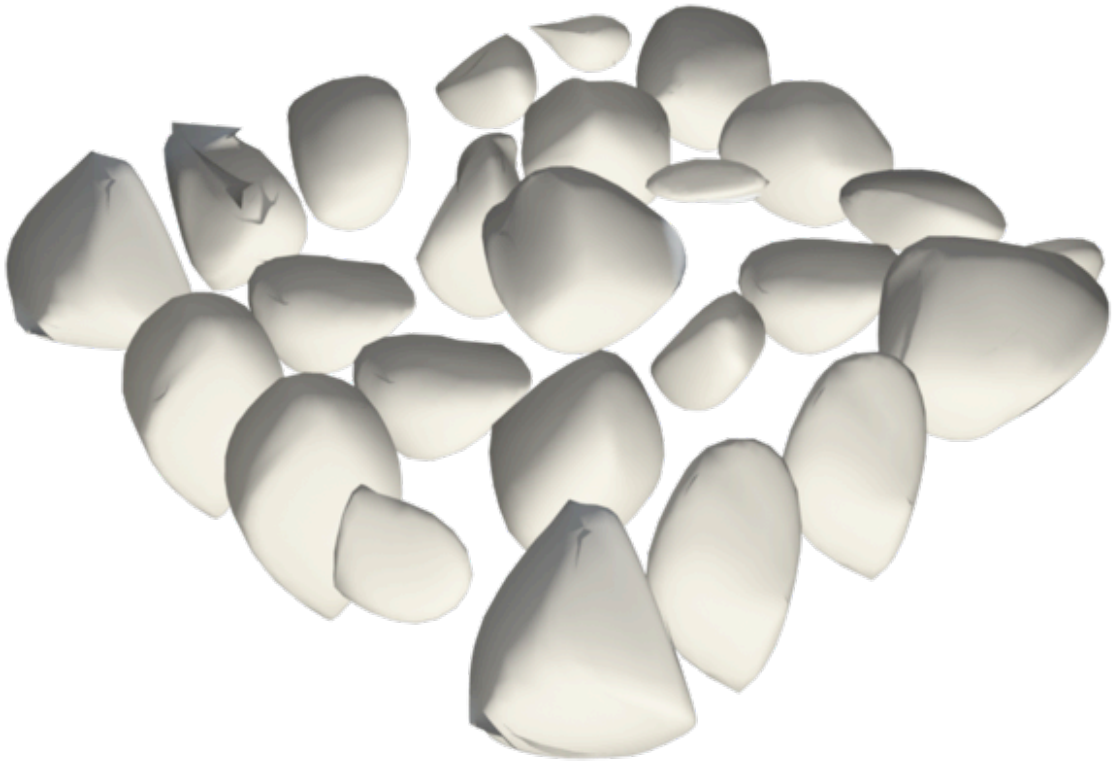
Figure 48: Enformation of basic cubes under the influence of light.

Light deformation on a collection of objects begins to position an object within a 'light field.' That is, the intensity of the light is dependent on the entire population, not merely on the incident angle for a particular unit. The self-shading of an individual then interacts with the self and other-shading of its neighbors, which can have nondeterministic results. In combination with underlying object properties of a modeling environment, the light phenomena on vertices is not always explicit.



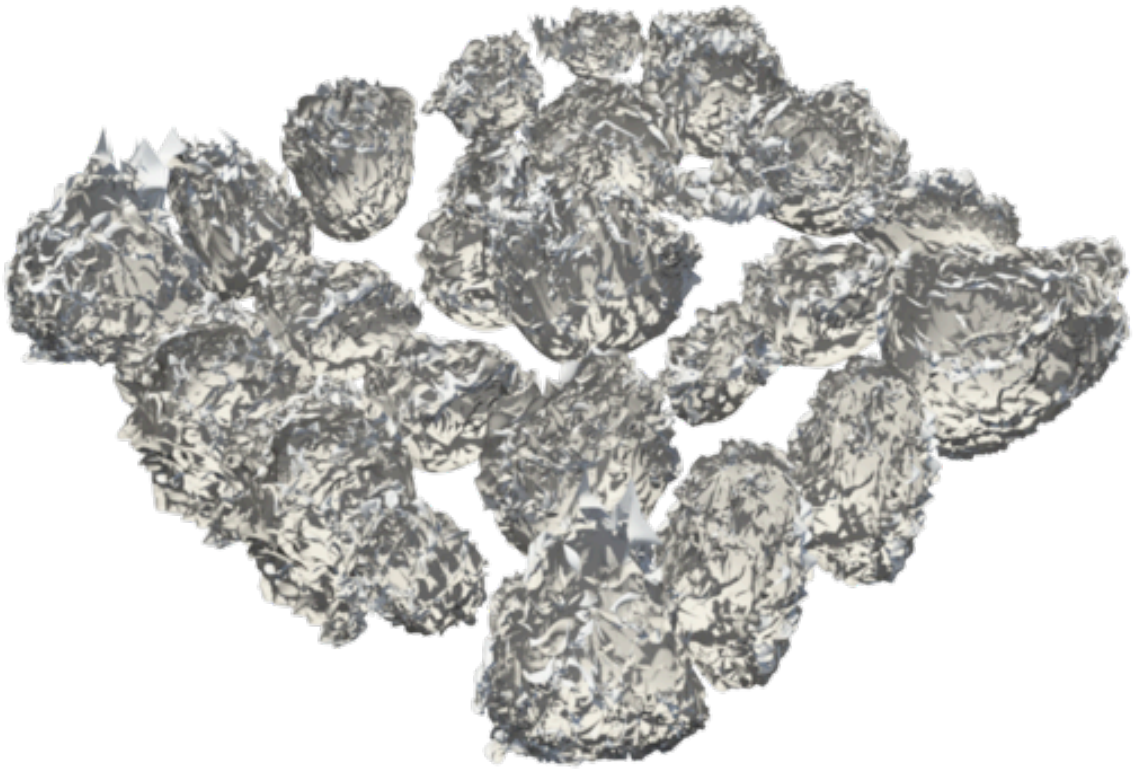
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud, light intensity**
Operations	**generate, deform vertices via incoming light**
Iterations	**25**
Population	**25**
Generations	**3**
Selection	**none**

Figure 49: Light influence on convex hulls, (top left to bottom left, clockwise, iteration set 1,2, and 3.)



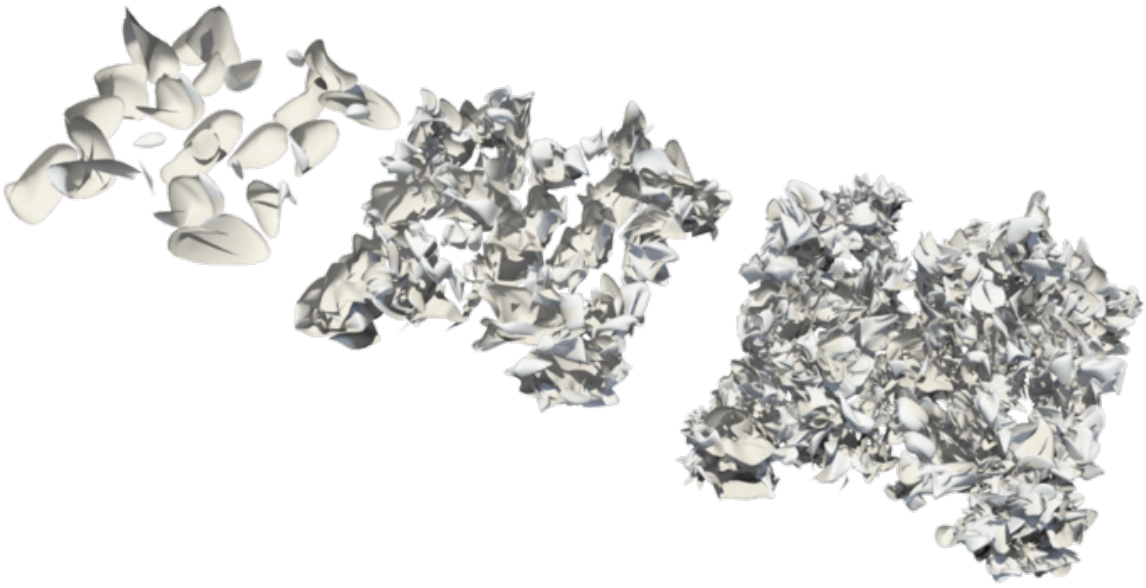
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud, smoothing intensity**
Operations	**generate, smooth**
Iterations	**25**
Population	**25**
Generations	**1**
Selection	**none**

Figure 50: Convex hulls under the influence of smoothing.



Algorithm Description	
Component	**convex hull**
Parameters	**point cloud, smoothing intensity, light intensity**
Operations	**generate, smooth, deform with light, smooth**
Iterations	**3**
Population	**25**
Generations	**1**
Selection	**none**

Figure 51: Convex hulls under the influence of smoothing and light.

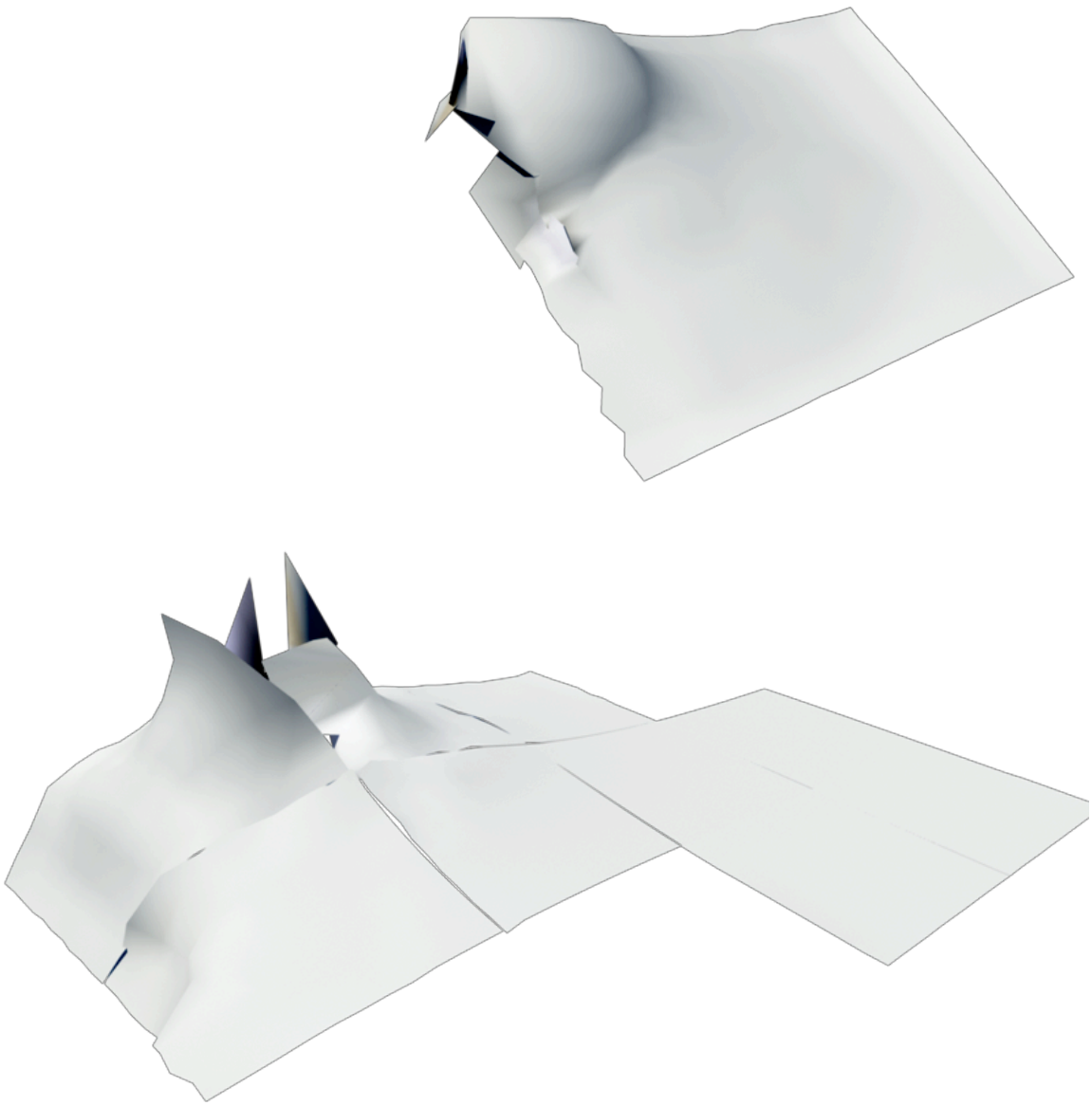


Algorithm Description	
Component	**convex hull**
Parameters	**point cloud, smoothing intensity**
Operations	**generate, deform under light, smooth**
Iterations	**from left to right, 1, 2, 3**
Population	**25**
Generations	**3**
Selection	**none**

Figure 52: Convex hulls, 3 iterations of smoothing and light influence.

Light distribution also has fundamental iterational properties. The deformation due to a light system is directly dependent on the configuration of the population of objects prior to the incident light arriving. Thus, multiple iterations of incident light reshaping an object is very different than a single iteration of high intensity light. Both of these are significantly different than an extremely faint light manipulation iterated very many times.

Light sources are critical for the dynamics of light-based manipulation operations. A point source, seen below, has local influence and widely varying directionality.

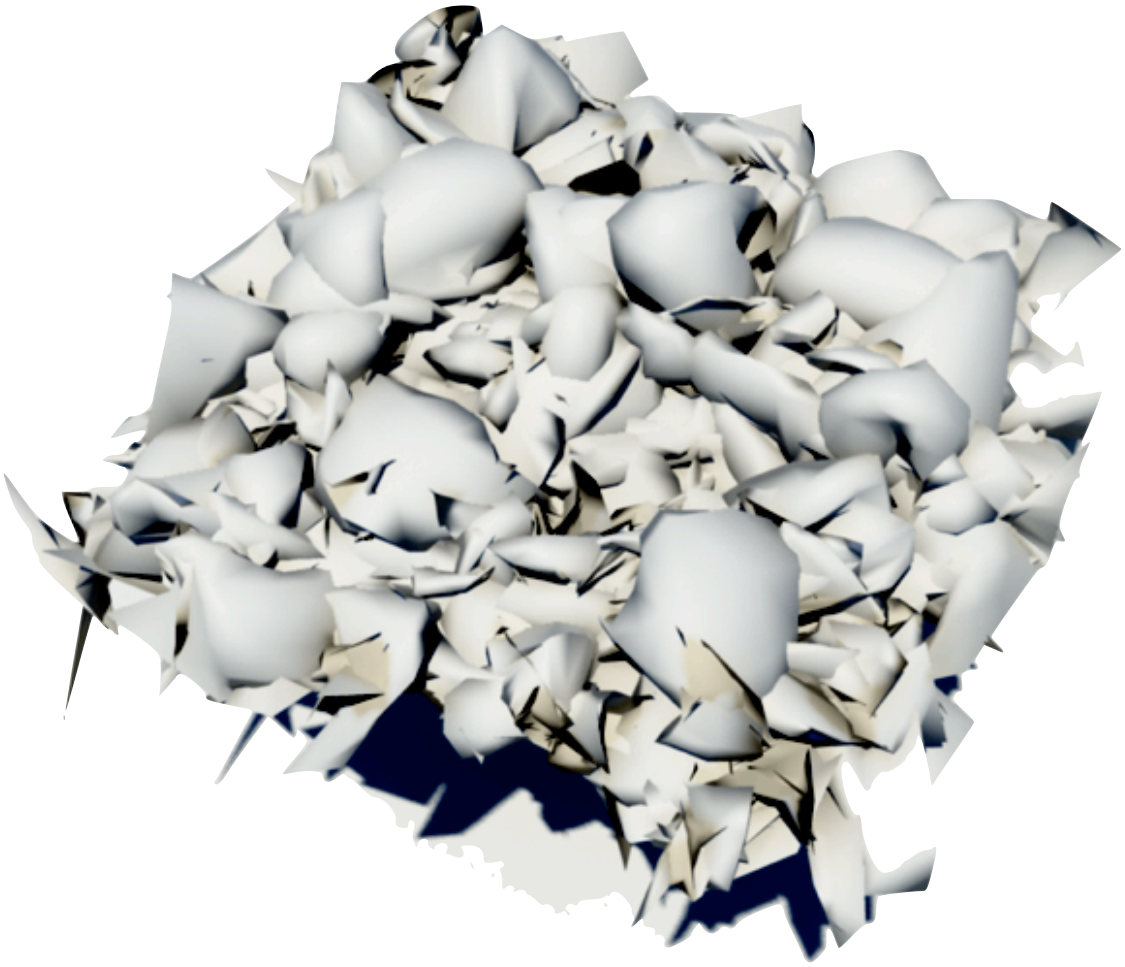


Algorithm Description	
Component	**subdivided planes**
Parameters	**light intensity, light source location**
Operations	**generate, tile, deform along light vector**
Iterations	**1**
Population	**6**
Generations	**1**
Selection	**none**

Figure 53: Set of planar deformations at boundary, due to light source forcing.

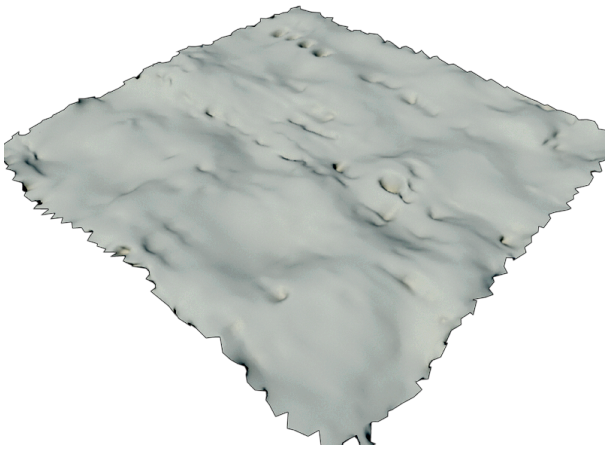
Algorithm Description	
Component	**subdivided planes**
Parameters	**light intensity, light source location**
Operations	**generate, tile, deform along light vector, smooth, deform along light vector**
Iterations	**3**
Population	**6**
Generations	**1**
Selection	**none**

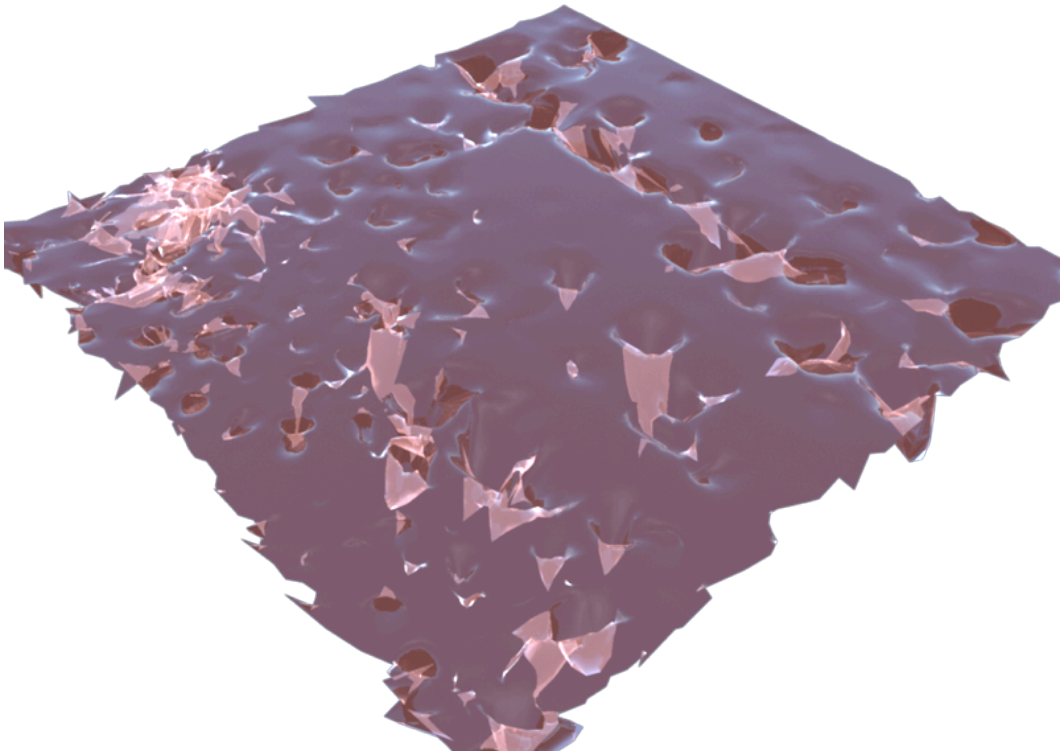
Figure 54: Planar modules under multiple iterations of spotlight.



Algorithm Description	
Component	**subdivided planes**
Parameters	**light intensity, light source location**
Operations	**generate, tile, deform along light vector, smooth, deform along light vector**
Iterations	**5**
Population	**6**
Generations	**1**
Selection	**none**

Figure 55: Planar modules under multiple iterations of smoothing, unidirectional light, and spotlight.



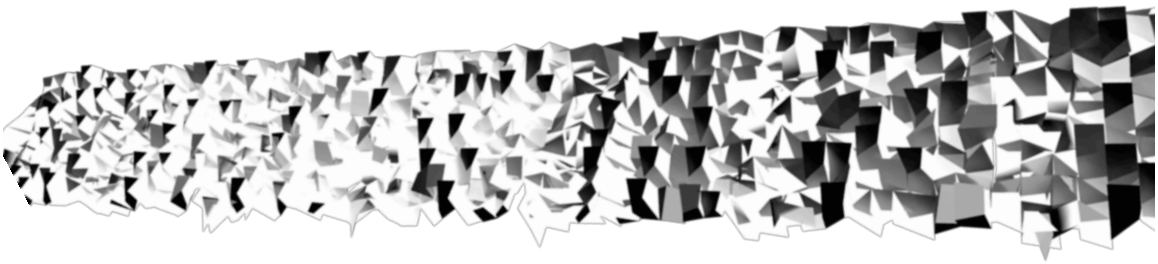


Algorithm Description	
Component	**subdivided planes**
Parameters	**light intensity, light source location**
Operations	**generate, deform at low intensity, repeat**
Iterations	**100**
Population	**1**
Generations	**1**
Selection	**none**

Figure 56: Planar unit under high numbers of directional light iterations.

Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity**
Operations	**generate, deform, repeat**
Iterations	**1, 10, 100**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

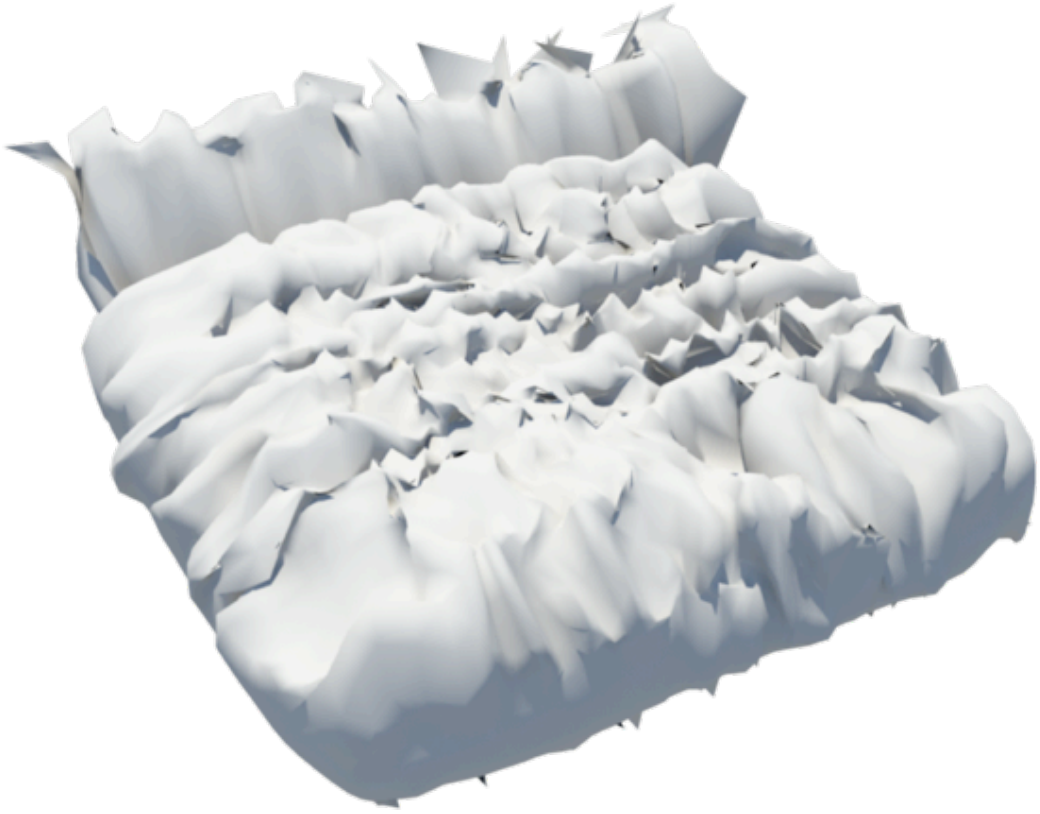
Figure 57: From left to right, iteration 1, 10, and 100 of a gridded plane under unidirectional light influence.



Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate, deform under light**
Iterations	**1**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

Figure 58: Planar grid under sequence of face dislocation, light influence.

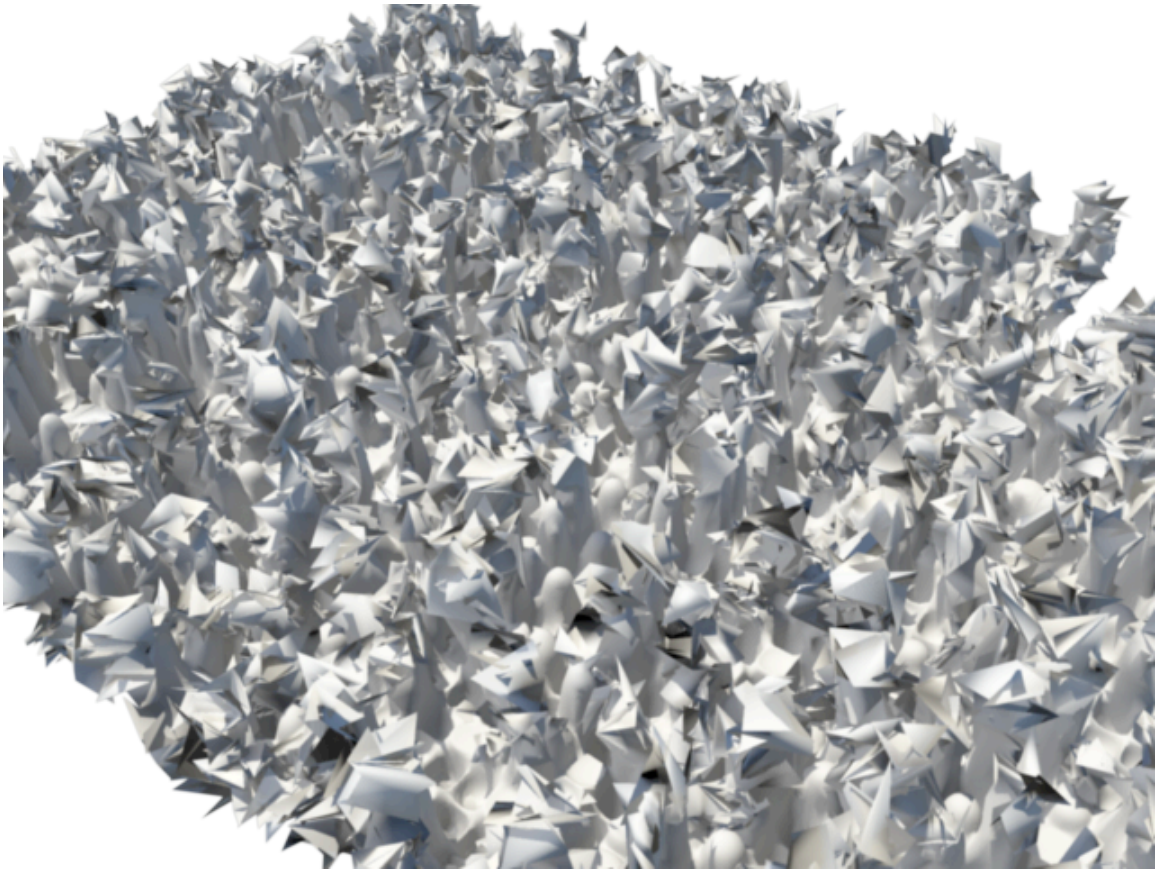
Sun-analog light sources, as shown immediately above, have very strong directionality and effect the entire population of objects relatively similarly.



Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate, smooth, deform under light, smooth, deform, smooth, deform**
Iterations	**1**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

Figure 59: Planar grid under sequence of smoothing, light influence, smoothing.

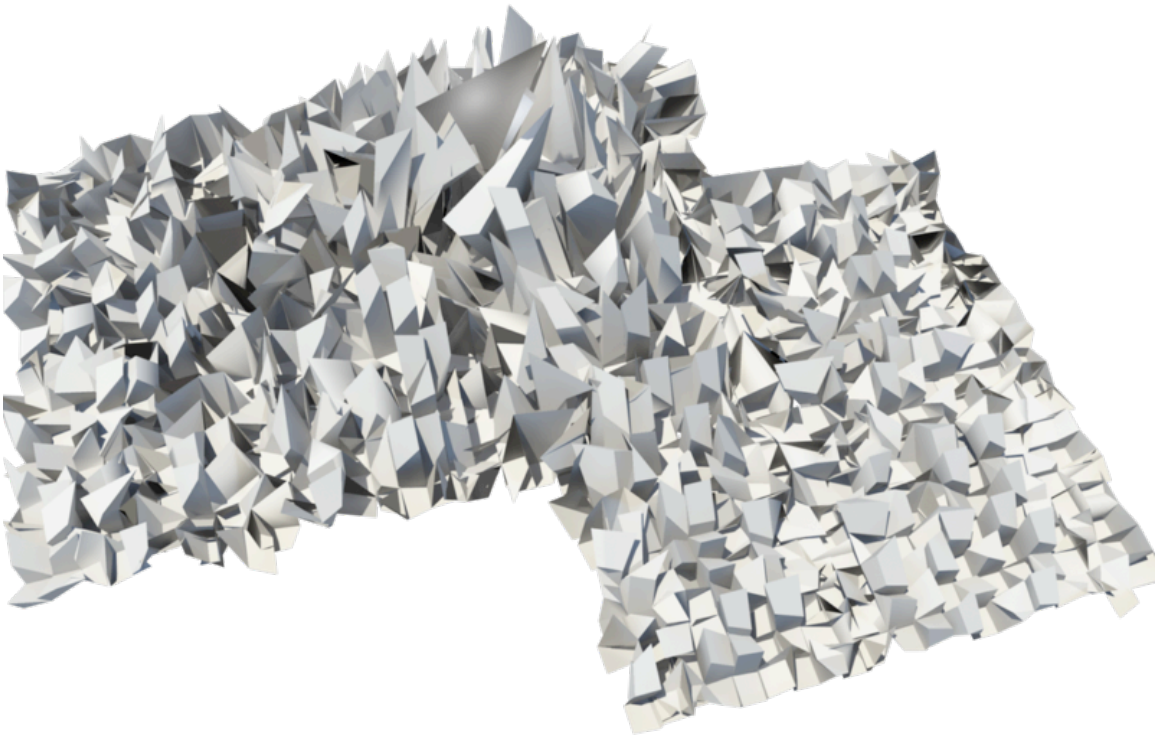
Characteristics internal to the manipulated objects qualitatively determine the manipulation. Subdivisions and quantity of individual components (vertices or faces) can have a dramatic effect.



Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate, smooth, deform under light, smooth, deform, smooth, deform**
Iterations	**10**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

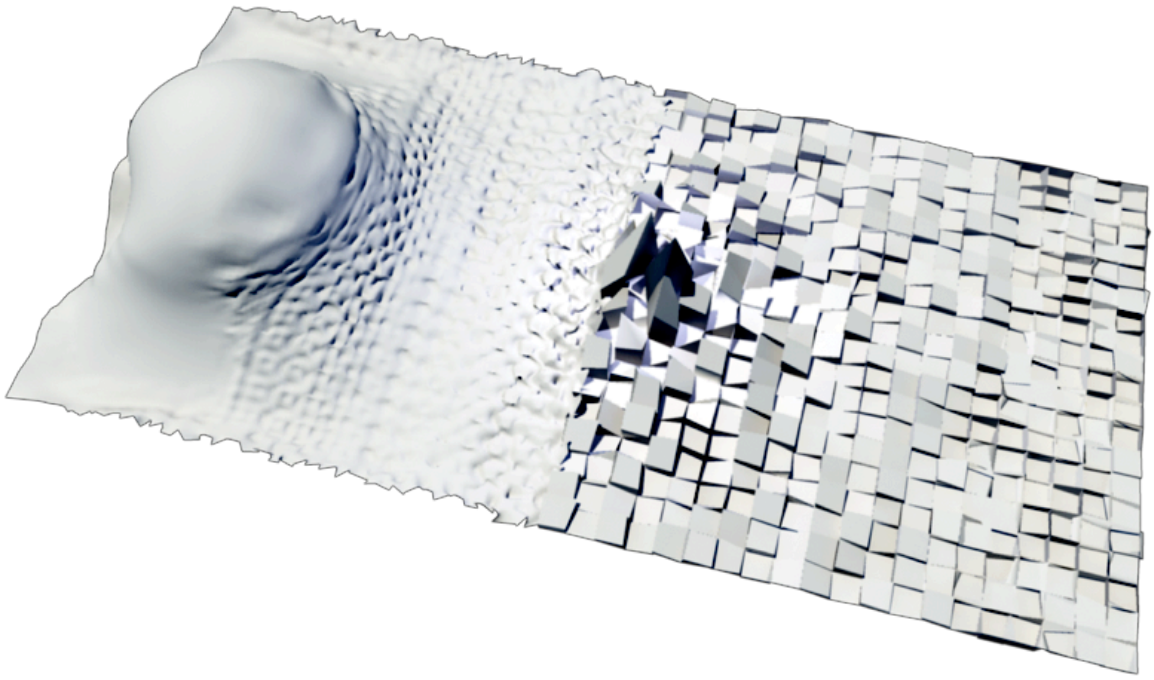
Figure 60: Planar grid under sequence of smoothing, multiple iterations of light influence and smoothing.

High numbers of iterations can also have counter-intuitive results. The variation in the precision of calculation builds up through iterations, and can have a randomizing effect. However, a small number of iterations does not guarantee simple results.



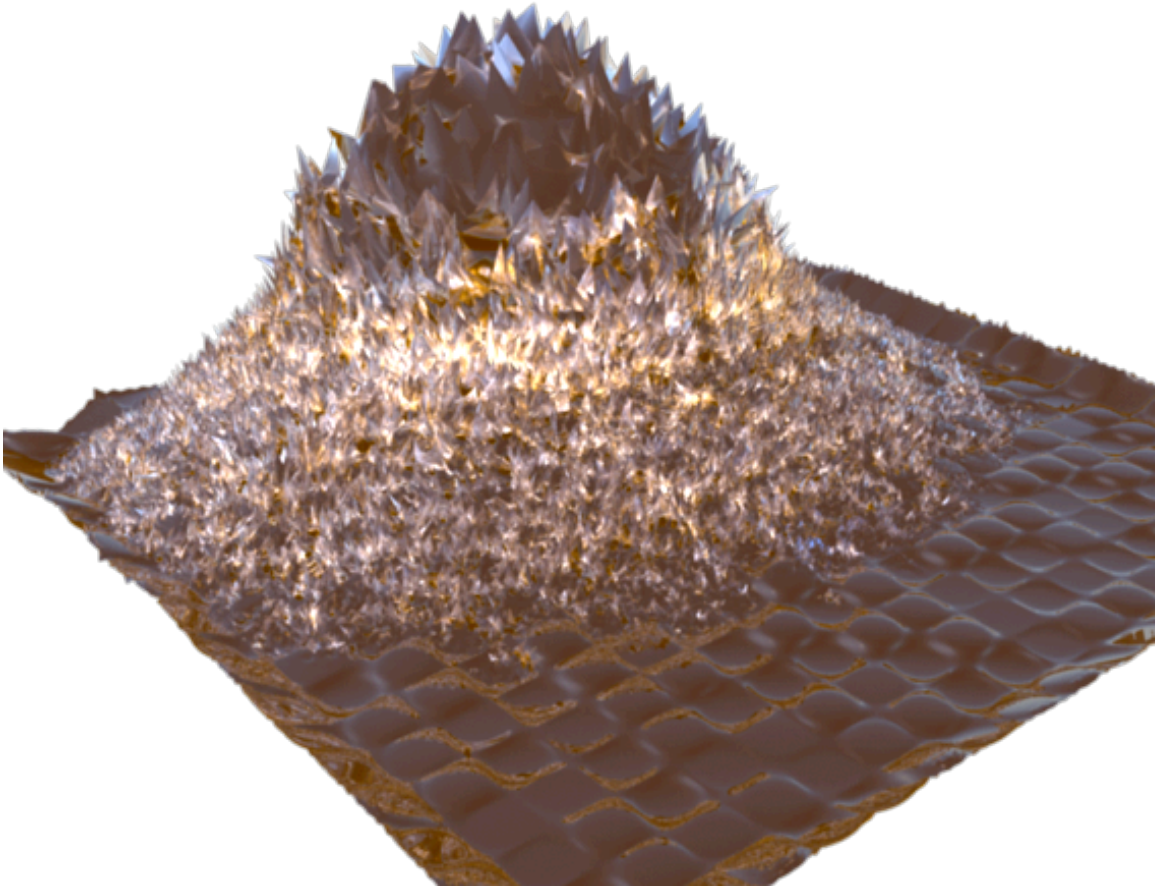
Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate, deform under light**
Iterations	**3**
Population	**800**
Generations	**1**
Selection	**none**

Figure 61: Planar grid under sequence of face dislocation and light influence.



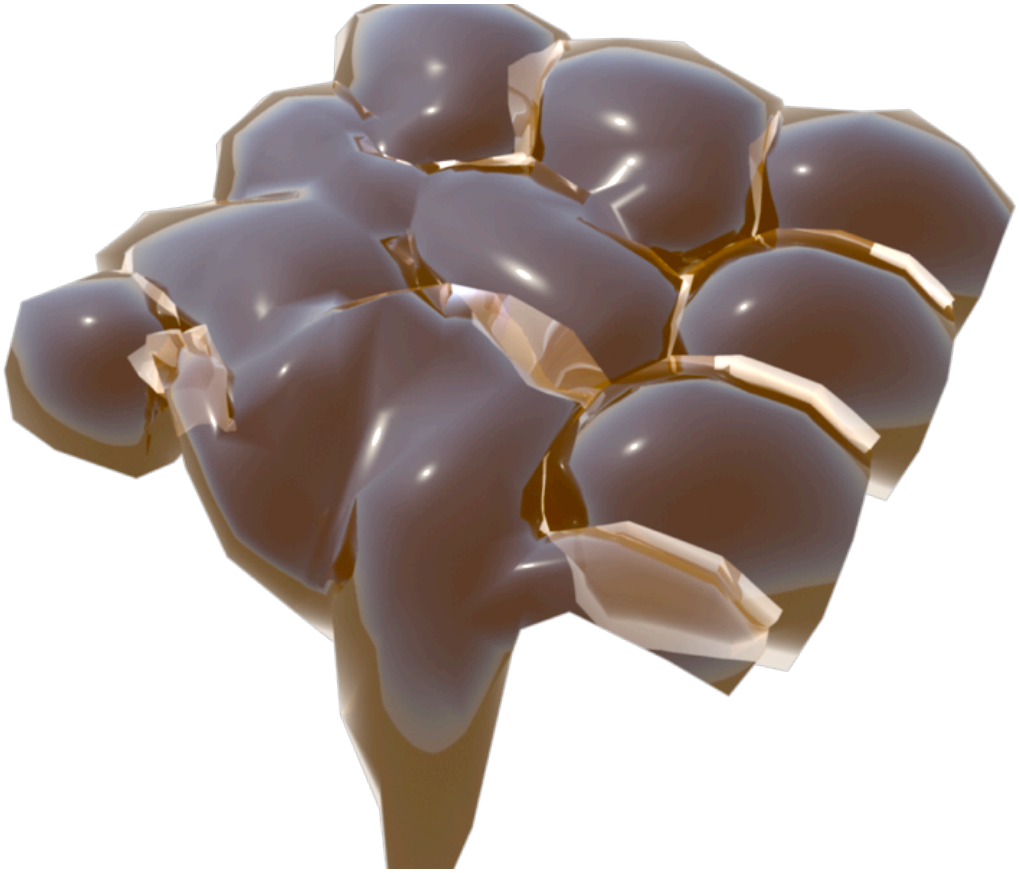
Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate, deform under light, smooth (left plane only), deform (left plane only)**
Iterations	**1**
Population	**400**
Generations	**1**
Selection	**none**

Figure 62: Comparison planar grid under sequence of point light influence vs face dislocation and subsequent point light manipulation.



Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate, smooth, deform under light**
Iterations	**1**
Population	**400**
Generations	**10**
Selection	**none**

Figure 63: Planar grid under sequence of face dislocation, smoothing, face dislocation, and light influence.

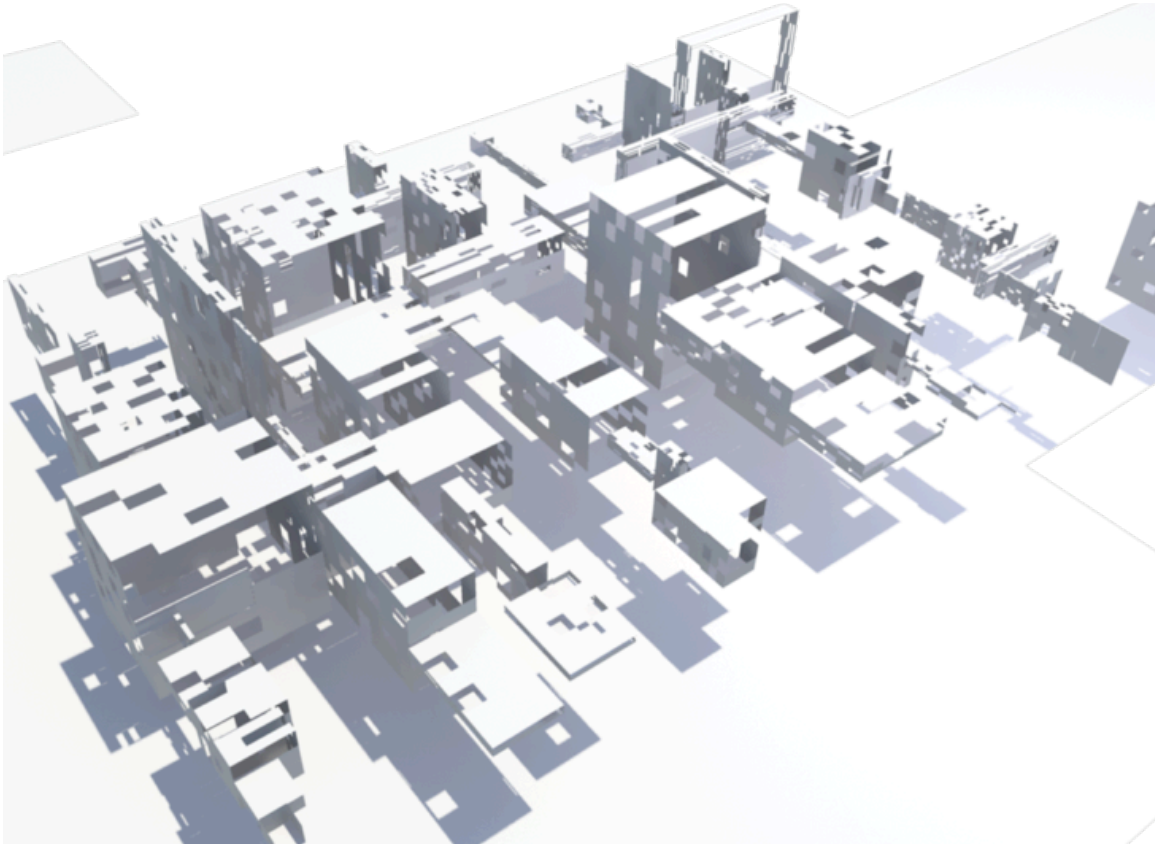


Algorithm Description	
Component	**subdivided plane**
Parameters	**light intensity, light source location**
Operations	**generate shape, generate lights, deform under lights**
Iterations	**1**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

Figure 64: Planar grid under sequence of multiple point light influences.

Algorithm Description	
Component	**subdivided plane**
Parameters	**removal ratio**
Operations	**generate, selectively delete**
Iterations	**1**
Population	**ca. 1000**
Generations	**1**
Selection	**none**

Figure 65: Planar grid under sequence face selection and elimination.



Algorithm Description	
Component	**subdivided cuboid**
Parameters	**removal ratio**
Operations	**generate, selectively delete**
Iterations	**1**
Population	**ca. 10000**
Generations	**1**
Selection	**none**

Figure 66: Cubic grid under sequence face selection and elimination.

F. Stochastic Optimization/GA

Given the inputs of multiple transformation operations, responding to driving forces of light, expansion along face normals, and subdivision of surfaces, the final step was to optimize the population of objects towards a goal. The genetic algorithm structure enables a balancing of arbitrary goal functions, and allowed for both specific direction towards desired parameters, and the capacity for identifying unexpected successful parameters.

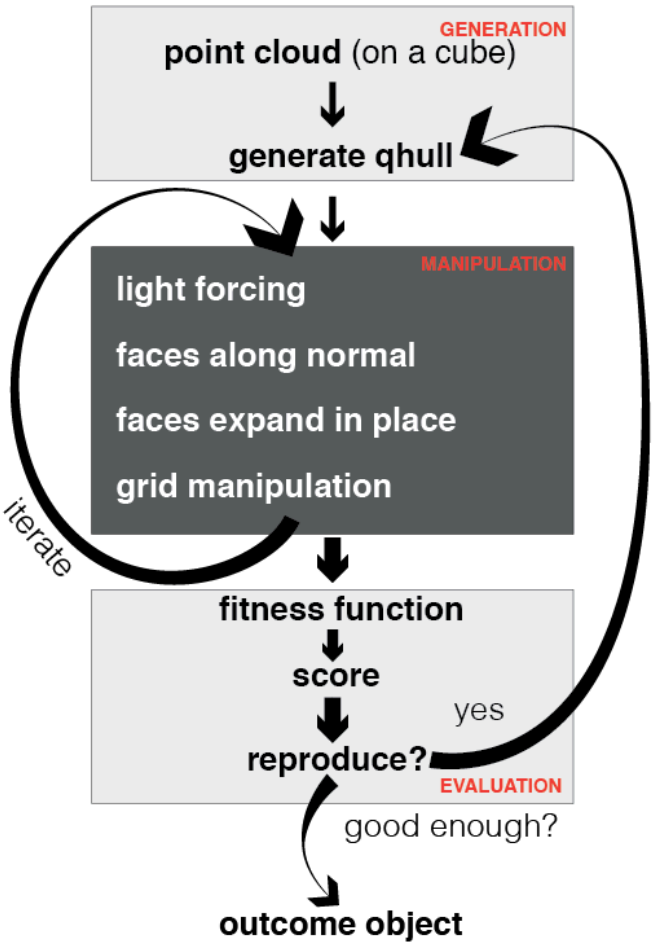


Figure 67: General procedural structure of the optimization sequence.

The parameters for the pyevolve package were held near the defaults, at the values shown below.

Genetic Algorithm Parameters	
mutator type	**swap**
mutation rate	**2%**
crossover	**one-point**
crossover rate	**80%**
selection type	**rank**

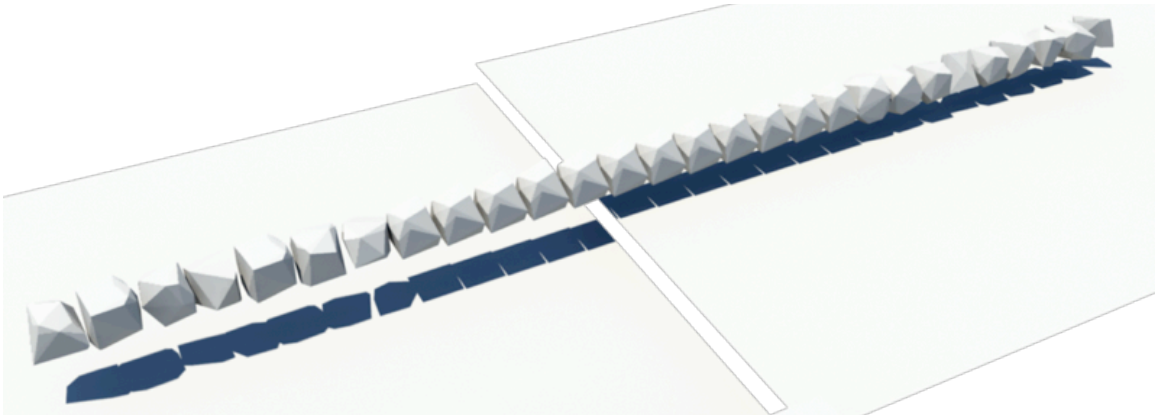
Figure 68: Genetic algorithm configuration parameters

G. Genotype / Phenotype

The process of optimization included a bipart structure temporally. The objects were generated from part of the genotype, and then operated on–manipulated–based on separate genetic data. These operations generated the phenotype: the resultant individual object. This phenotype was then evaluated for the fitness function score.

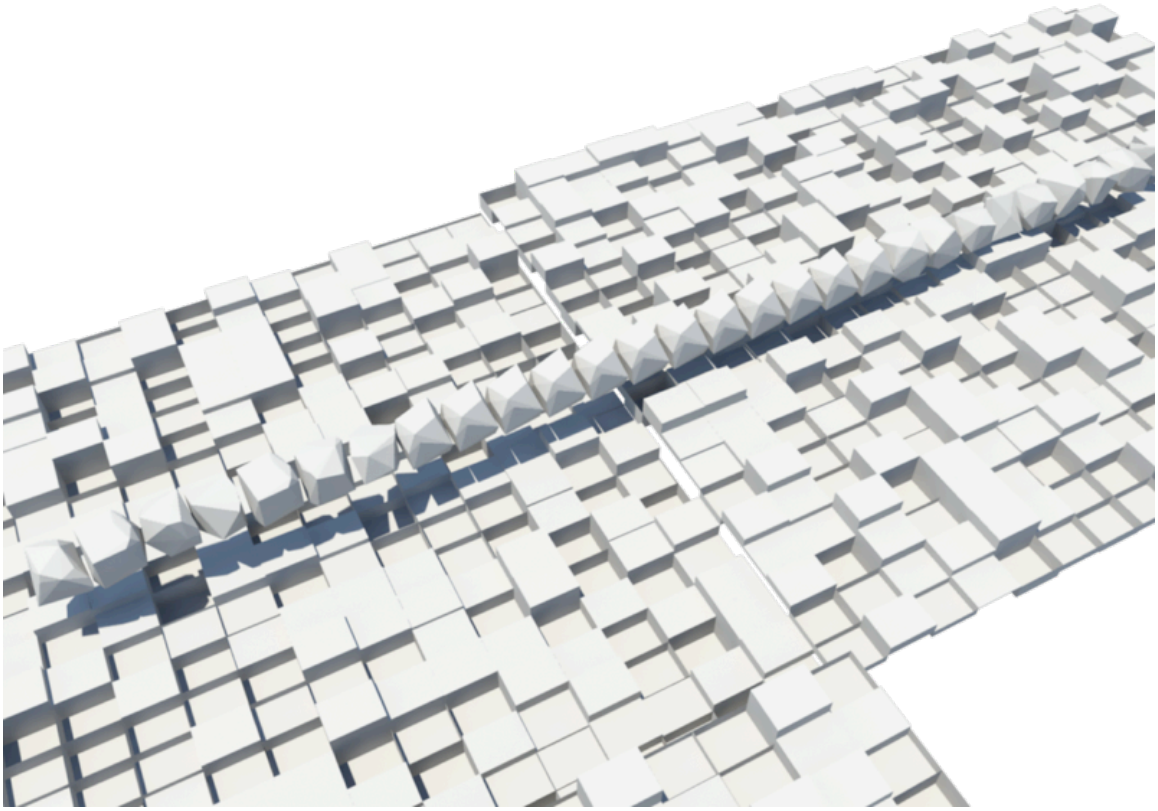
H. Fitness Functions

The initial fitness function is for high complexity. This is evaluated via a proxy calculation, using the vertex count of individuals divided by the edge length of the individual. When the transformation is limited to a scaling transform, the results are as anticipated. However, as more transformations are enabled, the individuals converge on a degenerate portion of the state space, where volume is minimized and shape and vertex relationships are irrelevant.



Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate**
Iterations	**1**
Population	**28**
Generations	**1**
Selection	**none**

Figure 69: Convex hull sequence with scale transformations.

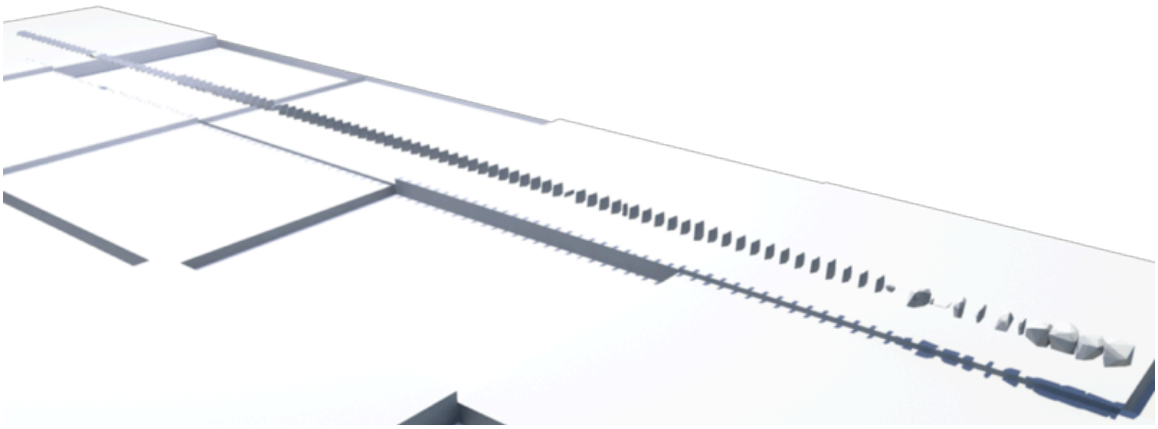


Algorithm Description	
Component	**convex hull, subdivided plane**
Parameters	**point cloud locations, extrusion range**
Operations	**generate**
Iterations	**1**
Population	**28**
Generations	**1**
Selection	**none**

Figure 70: Convex hull sequence with scale transformations and grid face displacement.

Algorithm Description	
Component	**convex hull, subdivided plane**
Parameters	**point cloud locations**
Operations	**generate, deform plane under light**
Iterations	**1**
Population	**28**
Generations	**1**
Selection	**none**

Figure 71: Convex hull sequence with scale transformations, grid face displacement, and grid light deformations.

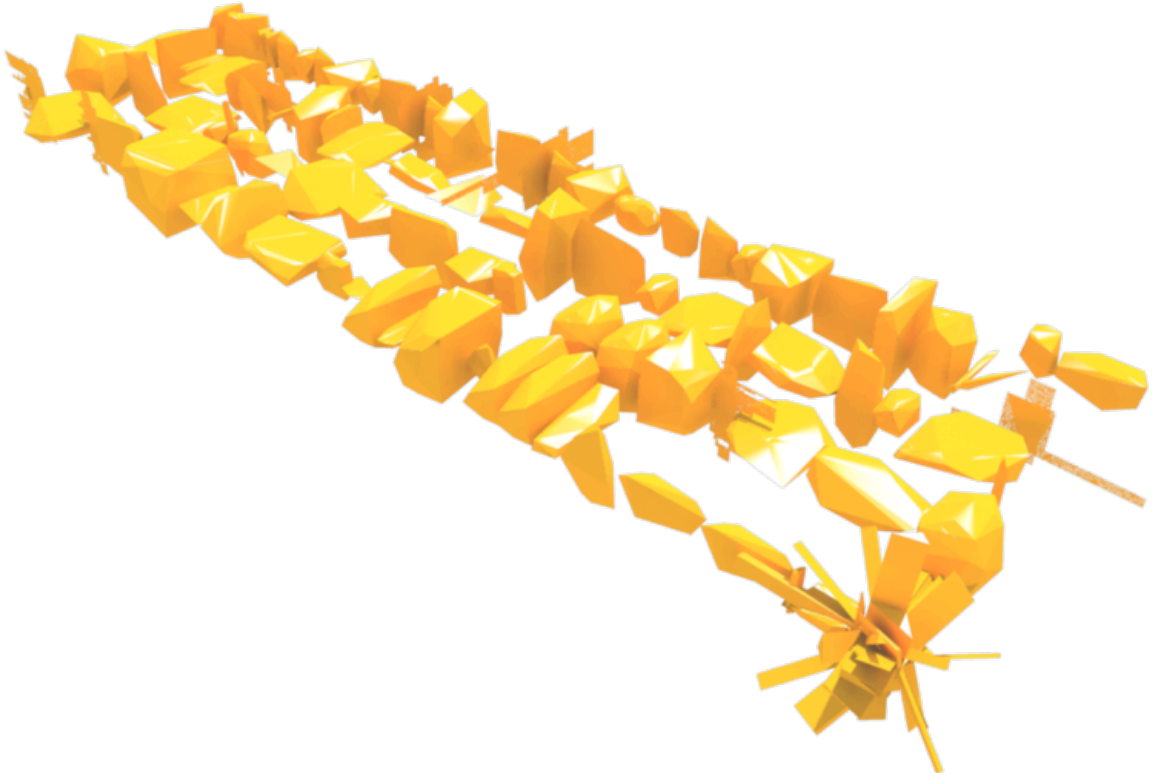


Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate**
Iterations	**1**
Population	**100**
Generations	**1**
Selection	**none**

Figure 72: Convex hull sequence with rapid degeneration due to malformed fitness function.

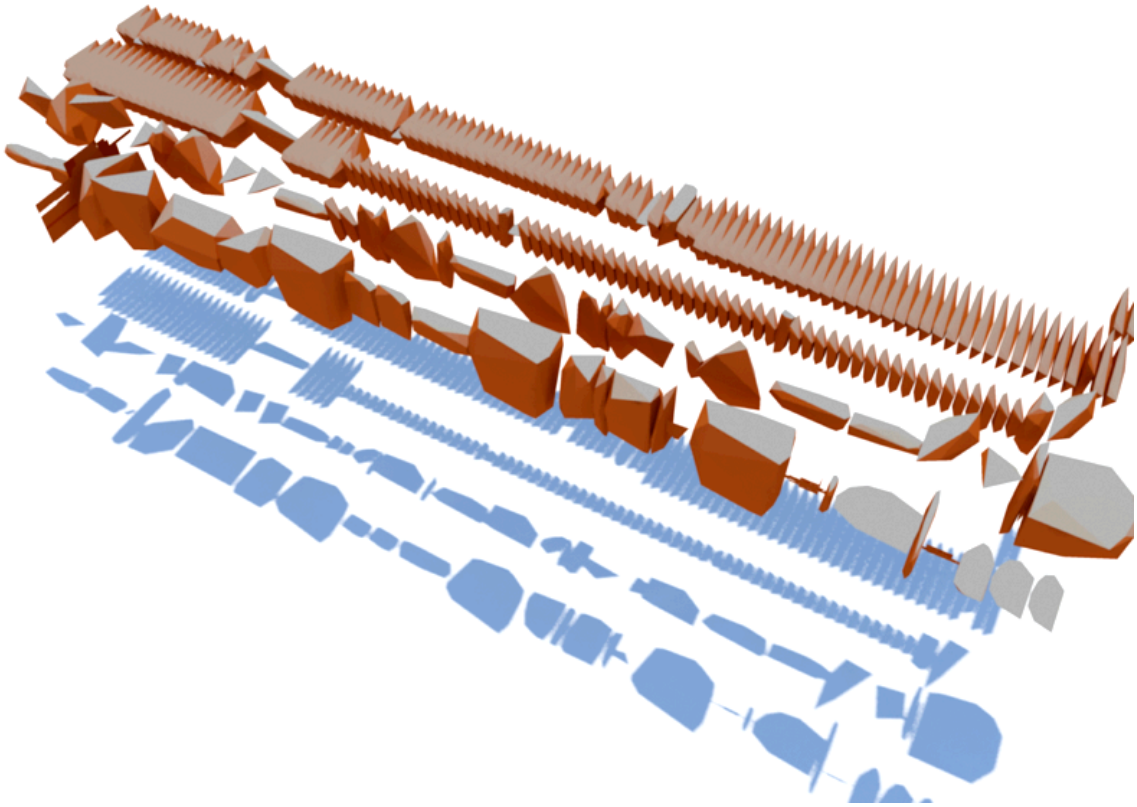
I. Complexity

The proxy evaluation criteria used was the ratio of vertex quantity to edge length total. This provides an estimate of an object's geometric complexity. Maximizing this value encourages a shape with a more complicated morphology, while minimizing the "proxy complexity" metric encourages ease of construction or material efficiency.



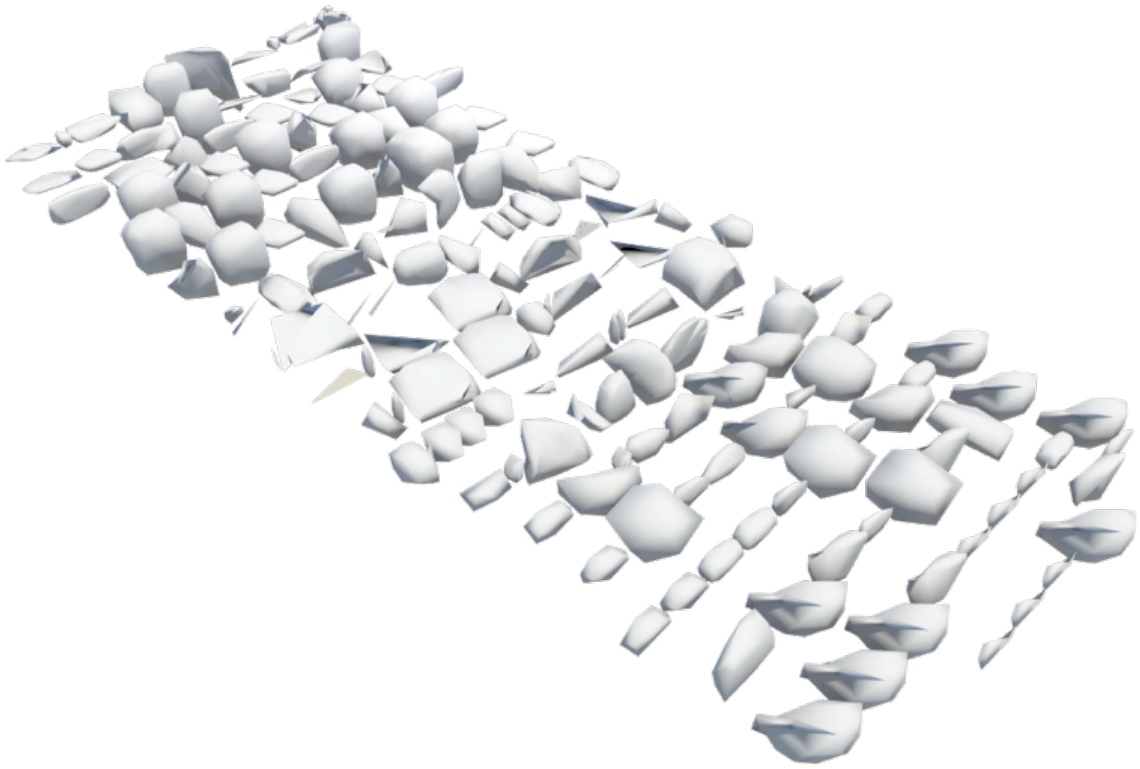
Algorithm Description	
Component	**convex hull**
Parameters	**dimension along each axis, step size along z axis**
Operations	**generate, move to boundary of previous**
Iterations	**20 x 50**
Population	**20**
Generations	**4**
Selection	**ratio of vertices to total edge length**

Figure 73: Convex hull population selected for vertex complexity proxy maximization.



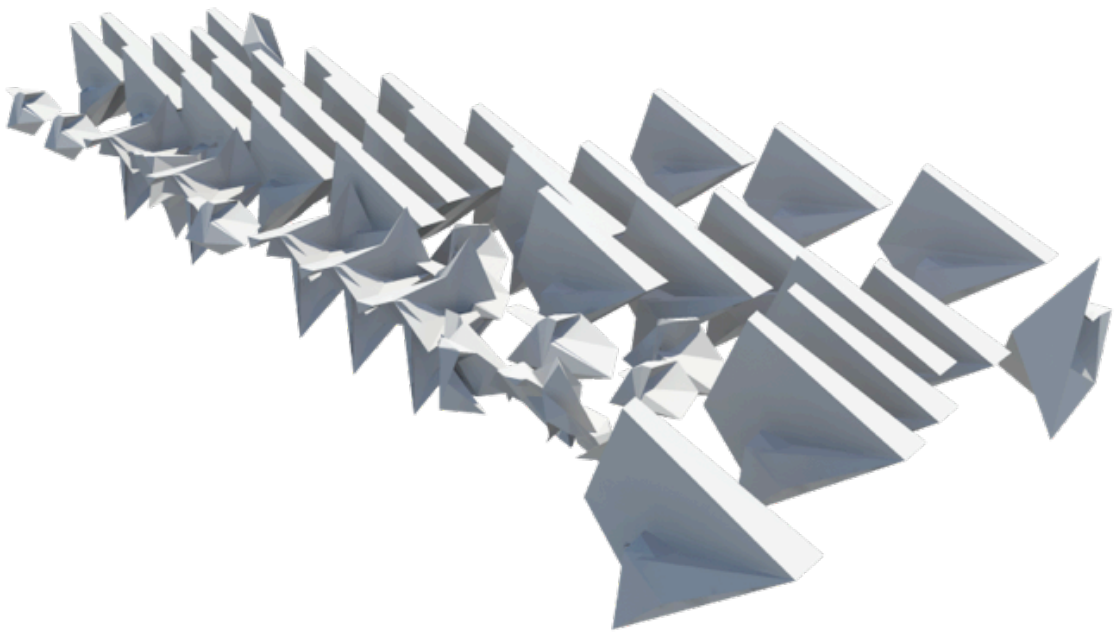
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate**
Iterations	**4 x 100**
Population	**100**
Generations	**4**
Selection	**ratio of vertices to total edge length**

Figure 74: Convex hull population selected for vertex complexity proxy maximization.



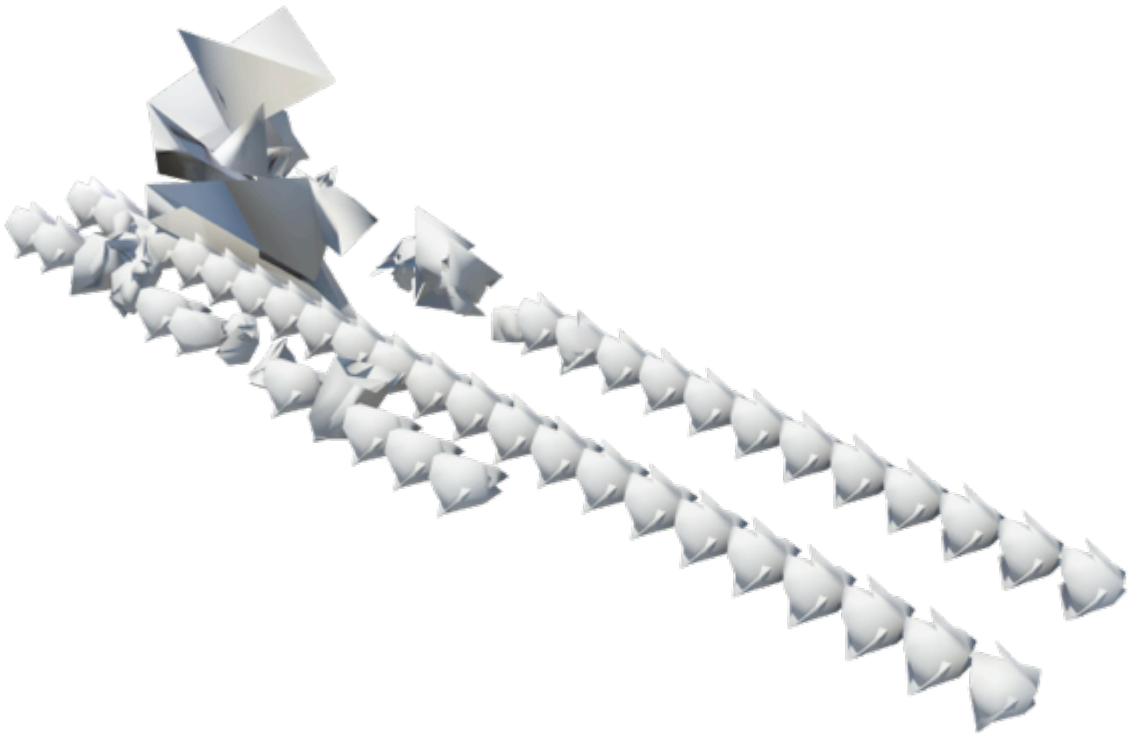
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, smooth**
Iterations	**10 x 20**
Population	**10**
Generations	**20**
Selection	**ratio of vertices to total edge length**

Figure 75: Convex hull population selected for vertex complexity proxy maximization, with interim smoothing transformation.



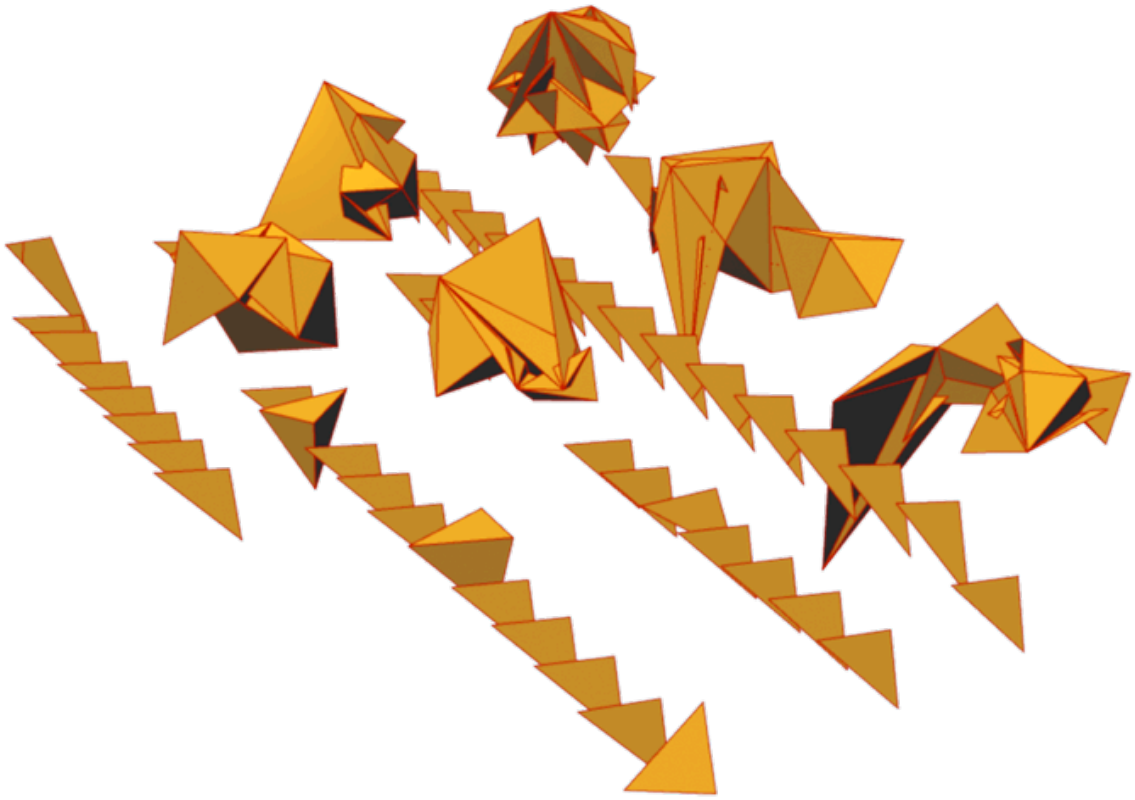
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, deform under light**
Iterations	**10 x 20**
Population	**10**
Generations	**20**
Selection	**ratio of vertices to total edge length**

Figure 76: Convex hull population selected for vertex complexity proxy maximization, with strong light deformation.



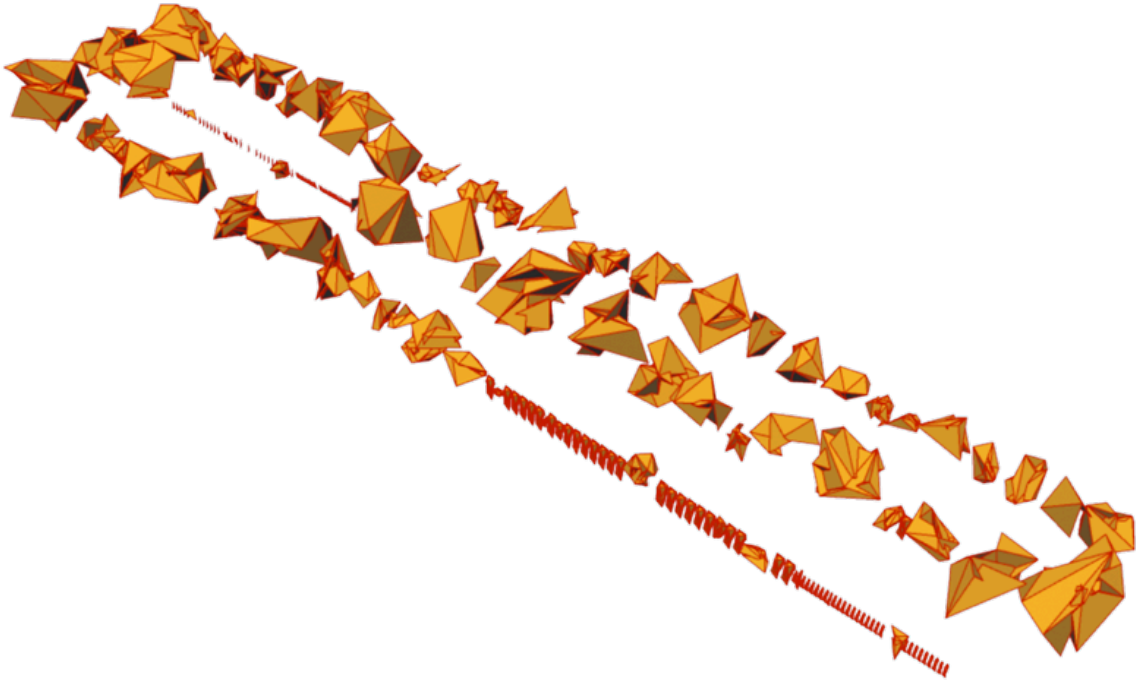
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, deform under light**
Iterations	**10 x 20**
Population	**10**
Generations	**20**
Selection	**ratio of vertices to total edge length**

Figure 77: Convex hull population selected for vertex complexity proxy maximization, with smoothing and light deformation.



Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, deform under light**
Iterations	**10 x 20**
Population	**10**
Generations	**20**
Selection	**ratio of vertices to total edge length**

Figure 78: Convex hull population selected for vertex complexity proxy maximization, with light deformation but without smoothing.



Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, deform under light**
Iterations	**4 x 100**
Population	**100**
Generations	**4**
Selection	**ratio of vertices to total edge length**

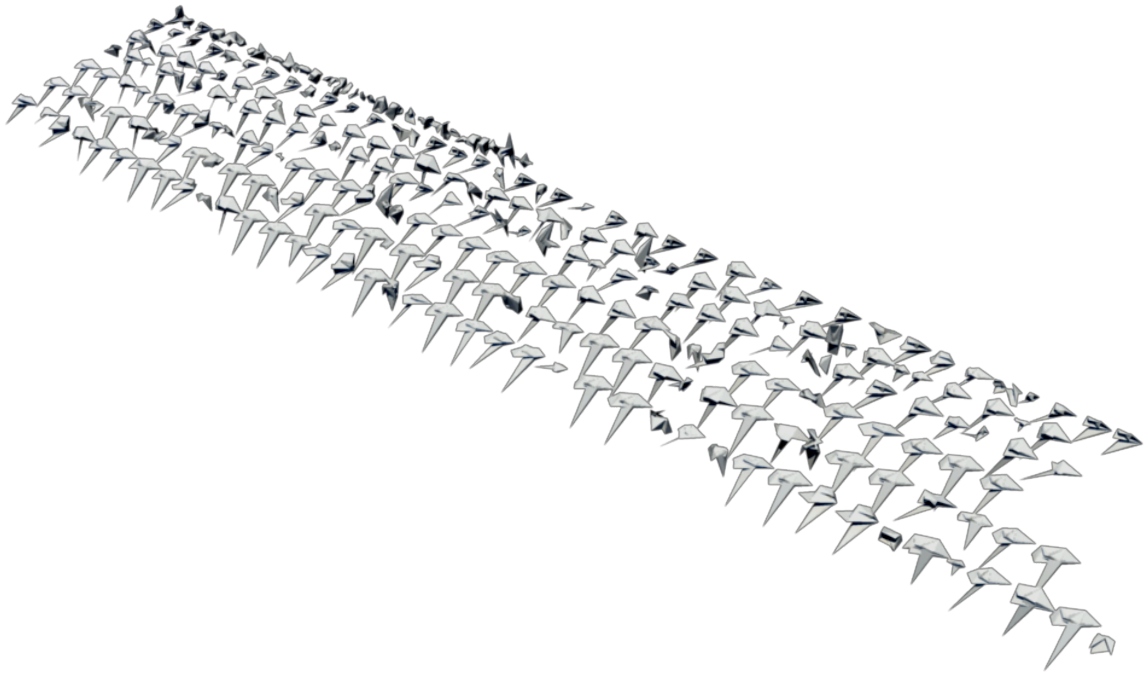
Figure 79: Convex hull population selected for vertex complexity proxy maximization.

J. S/V/C: Surface to volume and complexity

Optimization of vertex density can be either for maxima or minimization.

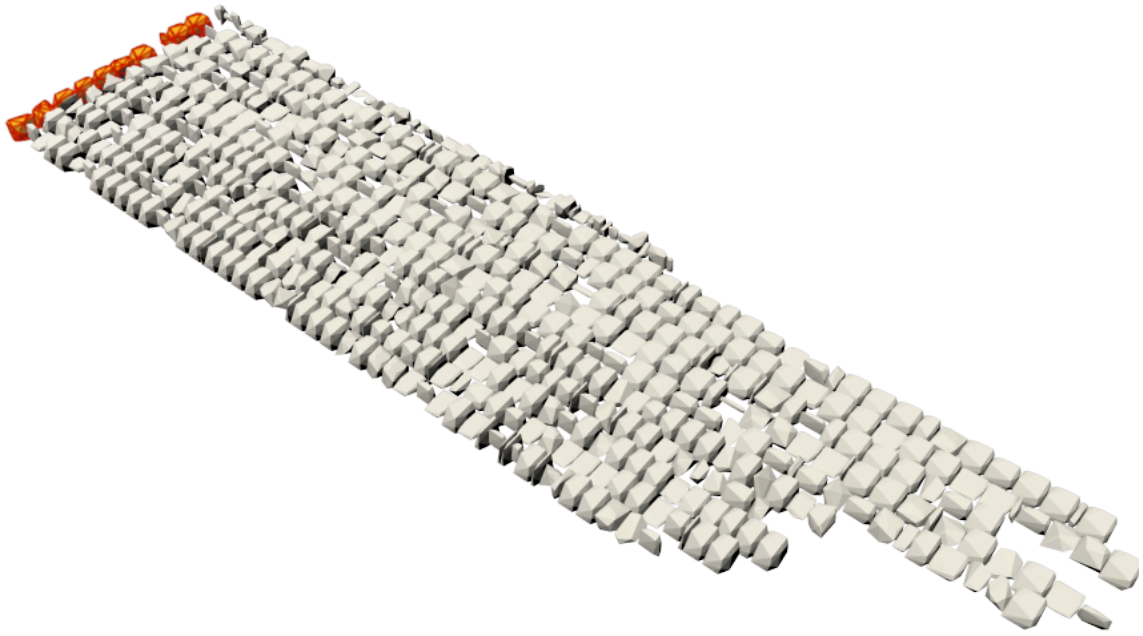
By combining the proxy for vertex density with a normalization for volume, the population does not converge on degenerate cases. In addition, by increasing the population count, the population dynamics become more reliable.

The lighting input creates artifacts based on outlier points, that contribute disproportionately to the overall volume, and therefore increase the individual's fitness score.



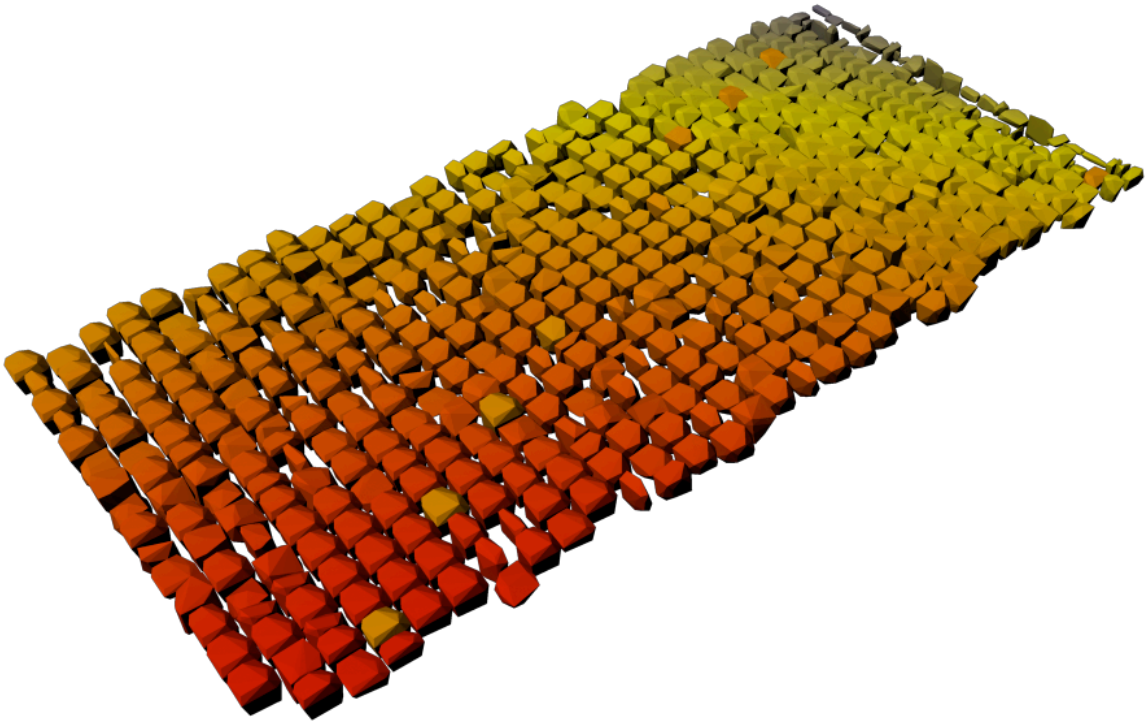
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, deform under light**
Iterations	**8 x 100**
Population	**100**
Generations	**8**
Selection	**ratio of vertices to total edge length to volume**

Figure 80: Convex hull population with volume normalization and light influence.



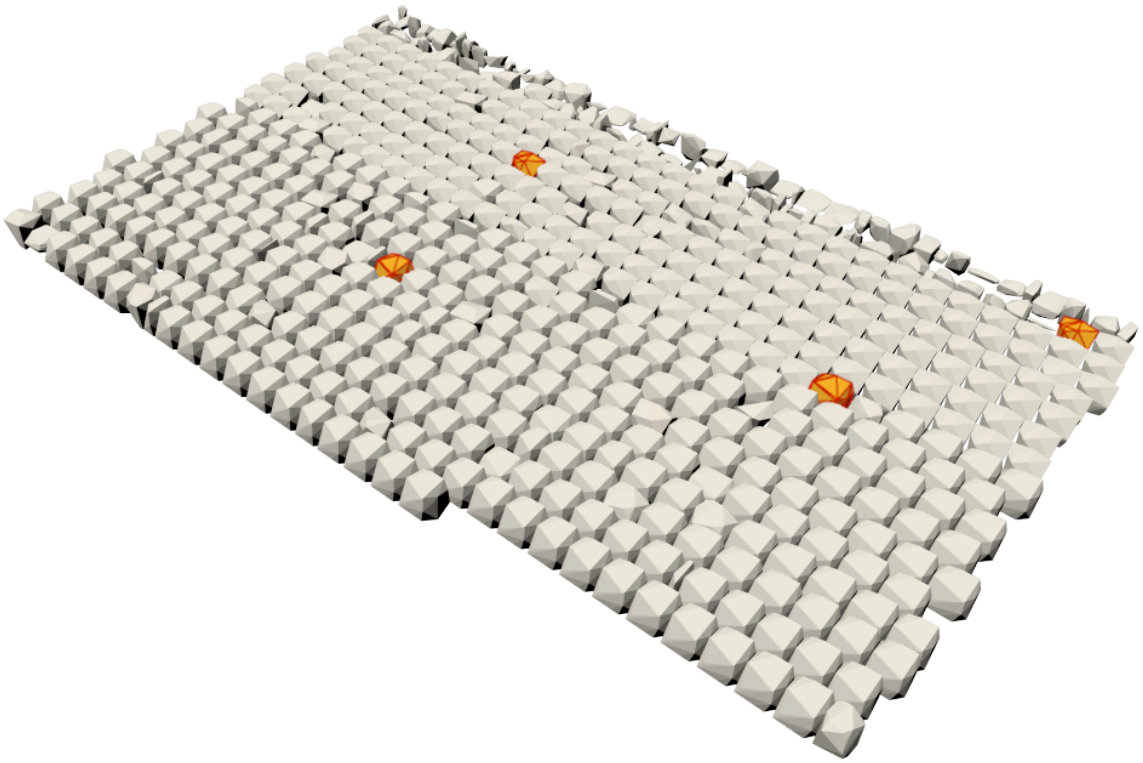
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate**
Iterations	**10 x 60**
Population	**60**
Generations	**10 x 60**
Selection	**ratio of vertices to total edge length to volume**

Figure 81: Convex hull population selected for vertex complexity with volume normalization and individual identification.



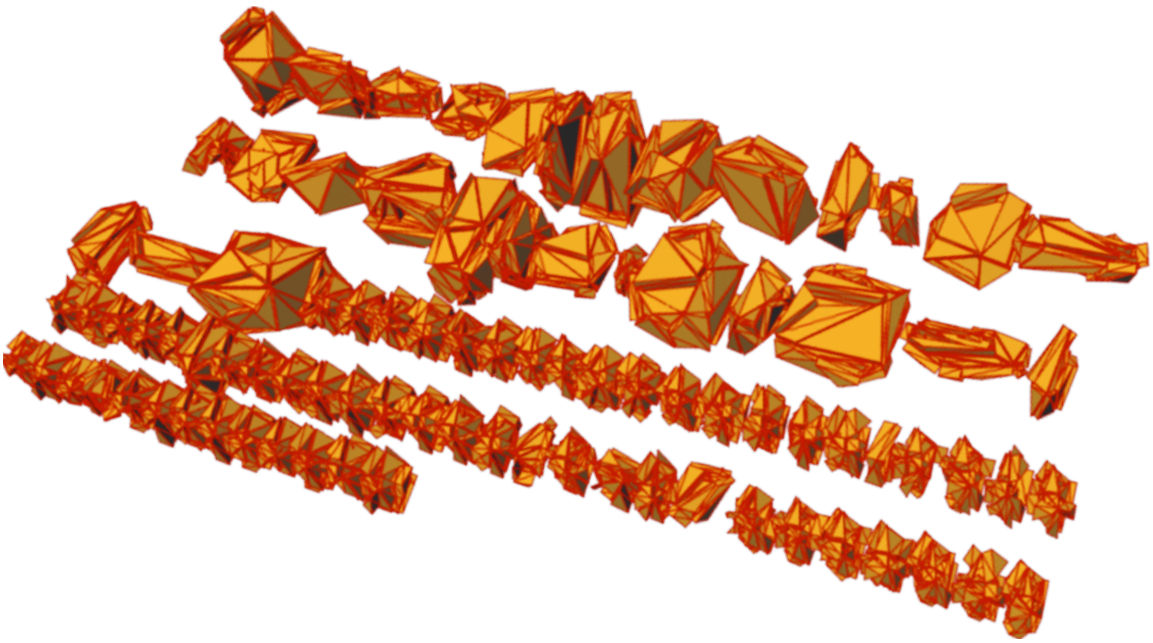
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate**
Iterations	**10 x 100**
Population	**100**
Generations	**10**
Selection	**ratio of vertices to total edge length to volume**

Figure 82: Convex hull population selected for vertex complexity with volume normalization. Red indicates more recent population.



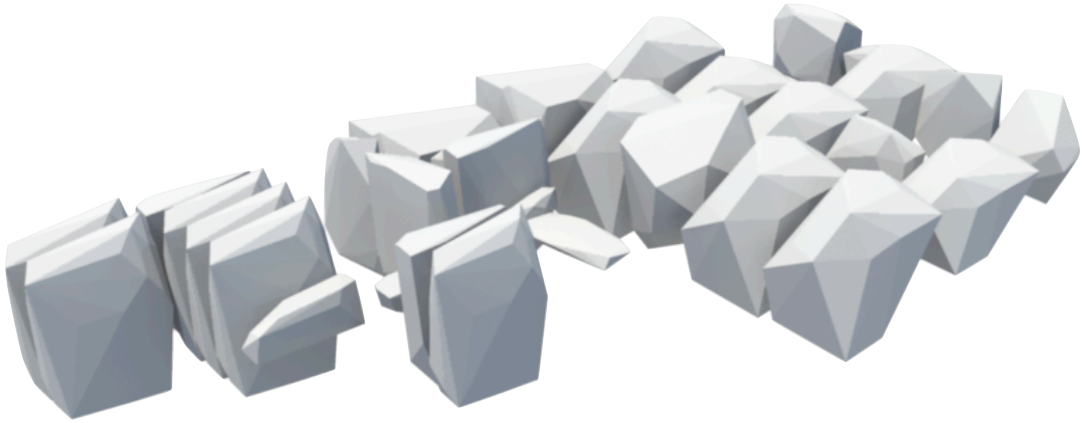
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate**
Iterations	**4 x 100**
Population	**100**
Generations	**4**
Selection	**ratio of vertices to total edge length to volume**

Figure 83: Convex hull population selected for vertex complexity with volume normalization.



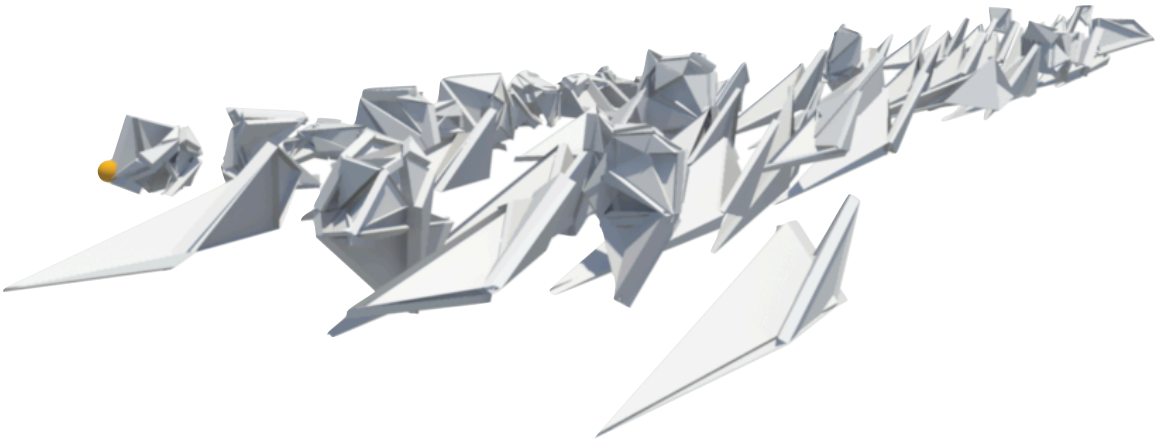
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations, deformation intensity**
Operations	**generate, extrude faces**
Iterations	**30 x 2**
Population	**30**
Generations	**2**
Selection	**ratio of vertices to total edge length to volume**

Figure 84: Convex hull population selected for complexity with volume normalization and extrusion of faces.



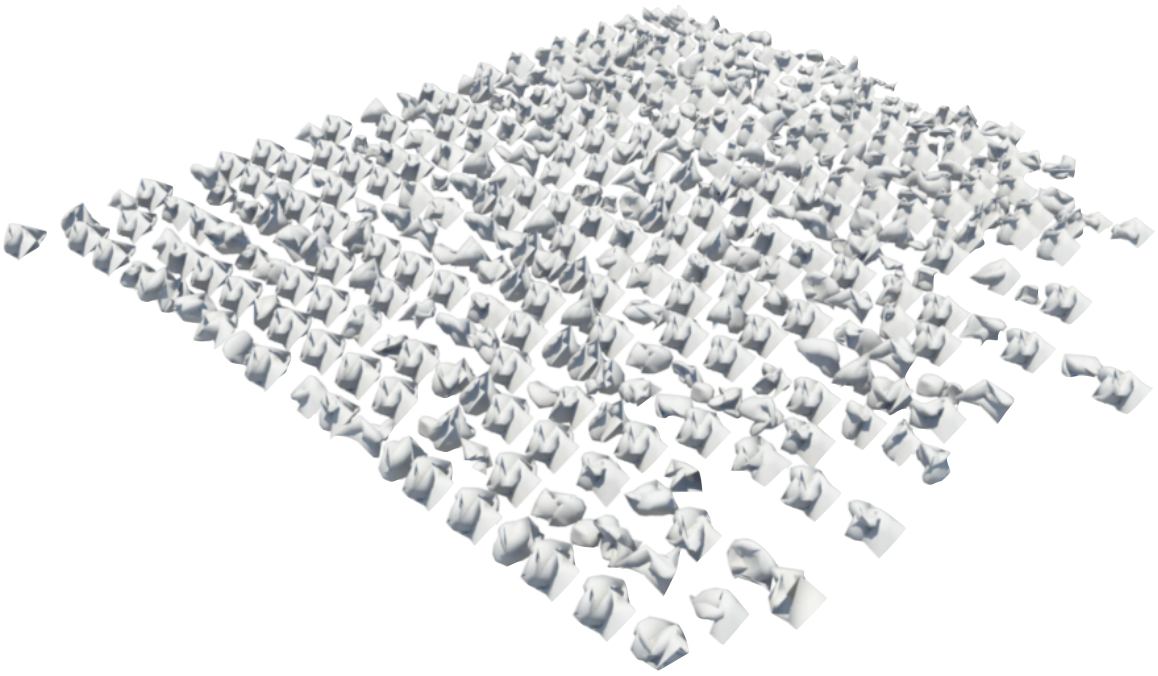
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations, deformation intensity**
Operations	**generate, scale**
Iterations	**5 x 2**
Population	**2**
Generations	**5**
Selection	**ratio of vertices to total edge length to volume**

Figure 85: Convex hull population selected for vertex complexity with volume normalization.



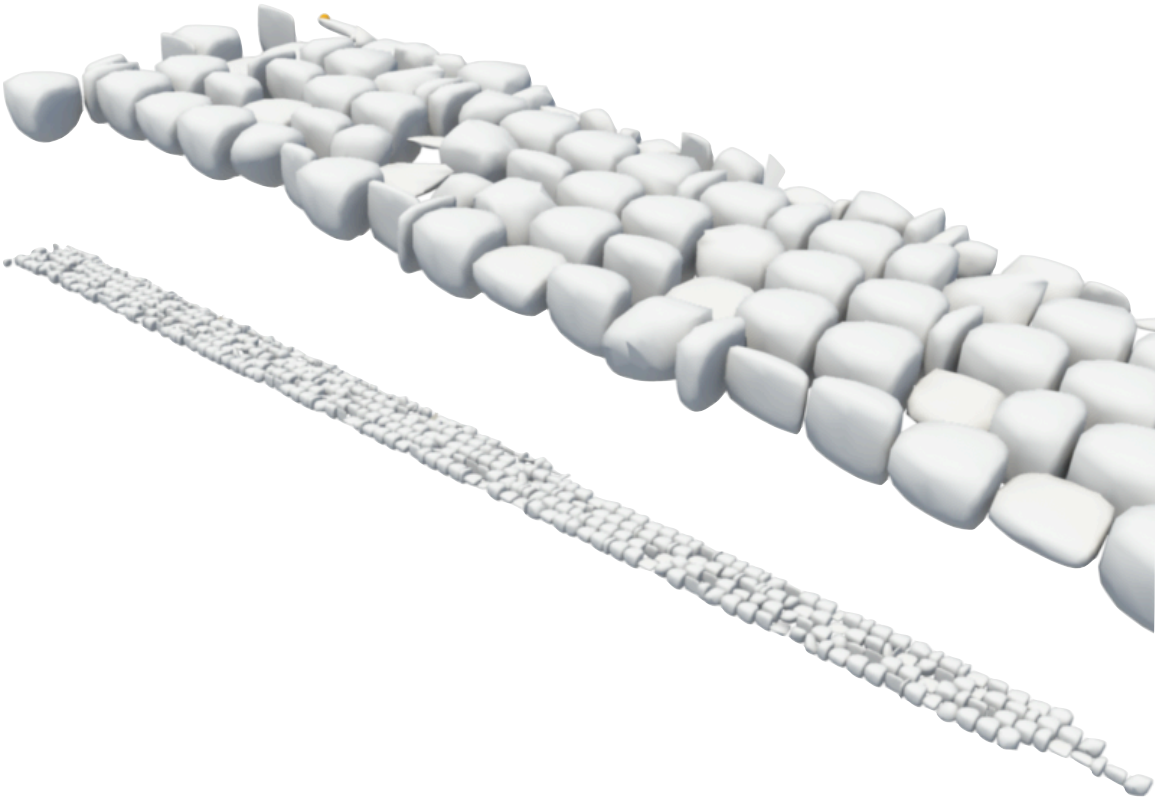
Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations, deformation intensity**
Operations	**generate, scale, extrude faces along normals**
Iterations	**6 x 3**
Population	**18**
Generations	**3**
Selection	**ratio of vertices to total edge length to volume**

Figure 86: Convex hull population selected for vertex complexity with volume normalization and extrusion of faces along face normal.



Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations, deformation intensity**
Operations	**generate, deform under light**
Iterations	**20 x 20**
Population	**20**
Generations	**20**
Selection	**ratio of vertices to total edge length to volume**

Figure 87: Convex hull population selected for vertex complexity with volume normalization and extrusion of faces along face normal.



Algorithm Description	
Component	**convex hull**
Parameters	**point cloud locations**
Operations	**generate, smooth**
Iterations	**4 x 100**
Population	**100**
Generations	**4**
Selection	**ratio of vertices to total edge length to volume**

Figure 88: Convex hull population selected for vertex complexity with volume normalization and extrusion of faces along face normal. Optimized object at far upper left.

XXVIII.Optimization Conclusion

The optimization successfully operated on individual objects for maximum or minimum proxy evaluation of complexity, normalized against a proxy evaluation for volume. While artifacts of the modeling system caused fitness benefits for a number of counter-intuitive object configurations, the overall goal seeking nature of the genetic algorithm was clearly operational. These optimizations, executed under varying morphological and analogical influences, extend the clear potential for the use of algorithmic methods on goal-directed design processes.

XXIX.Potential Future Structure / Future Work

This work scratches the surface of what is possible using the latest generation of digital tools. The permutational expansion of each of these procedural tools results in a large set of potential new methods: from optimization of fluid dynamics to fractal expansion of traffic patterns.

Specifically, there are several areas, with potentially interesting consequences, that bear immediate further exploration within the scope of this project: the source of geometry/ object, the source of spatial configuration, the type of embedded simulation, the kind of selectivity within a population of objects, the translation of proxies and creation of proxy evaluation criteria, the optimization criteria, and the methods of optimization.

Geometric Generation

This investigation focused on generating basic geometries and then operating on those geometries. Further sources of geometry include algorithmically defined elements such as chaotic structures, fractal patterns (L-systems, Sierpinski gaskets, or the like) or regular mathematically defined geometries (including a variety of polyhedron or hyperbolas). On the other end of the geometric spectrum are specific, evolved forms, based on biological structures. Individual organisms, adapted to a particular niche, could be digitized and evaluated along niche-similar functional lines within an algorithmic structure. Although the selection criteria will not (and cannot possibly) be identical to the organisms niche and environmental context, the generalization of the niche forces can be used to create geometry-defining effects on individual objects (or buildings) within a analogical functional niche.

Renovation vs. Demolition

Existing structures, buildings, or other human-designed objects can also be digitally captured and evaluated by both innovative and classical criteria. The practical use of evaluating renovations would be immense, as well as providing an spatial and geometric cost benefit analysis. These analyses would allow for an innovative method of post-occupancy analysis as well, scoring a space or building based on objective metrics.

Finally, although not without problems, this kind of analysis can suggest a measure of which spaces should be preserved, and which should be redeveloped or re-imagined. There would not always be an incentive to always wipe the slate clean, however. Existing structures could be mutated or evolved, rather than either demolished or repaired. Small changes within a complex system can have outsized responses.

One area of great potential encompasses the method for defining high-performance criteria. By extracting selection criteria for performance from existing buildings which are accepted as high-performance, a data resource could be generated. These performance envelopes of known-good buildings could be invaluable for generating future high performing entities.

Redefinition of Performance

These performance criteria, both for known-good and imaginary systems, must include an expansive definition of performance: one that includes psychological performance, psychology of space, and ecological performance in the biological community. These non-standard evaluative methods have enormous potential for selecting holistic performance. Structural efficiency is relatively easy to analyze, and therefore easy to apply. Ecological interdependence is less accessible, but significantly more impactful for the operational and contextual consequences of building in a certain way or certain place.

Algorithm

The genetic algorithm is relatively effective for balancing multiple evaluation criteria and finding an optimum. However, there is a cluster of both stochastic and more linear strategies that can be applied to similar problems. Simulated annealing, random search, Newtonian methods, hill-climbing, and other algorithms all have different approaches to complex problems. A hybrid problem-solving method, or a method in which many optimizations are tried on the same problem, may result in a more robust solution set.

Within each of these algorithms, the parameters can be rigorously permuted and evaluated for success. Within the genetic algorithm, for example, the number of generations, mutation rate, and relative strength of the components of the fitness function

can all vary, with great effect on the algorithms efficacy. Future applications of these algorithm clusters take as input the problem definition in combination with the range of algorithm parameters. These algorithms are parametricizable just as is the geometry on which they work.

The selection criteria, within a genetic algorithm specifically defined as a fitness function, is one of the key components in any optimization loop. This thesis focused on a fitness function that acted as a simple proxy for a shape's complexity. However, including humans in the loop is a powerful way of including cultural, historical, and deep neurological knowledge about a target domain. Either within human populations (focus groups) or individually (expert guidance,) future optimizations would benefit from using a human component.

Human / Ecological Roles

In addition, other biological evaluators could be conceived as participants in the process. Several species have been proved able to communicate symbolically. Even symbolism would not be strictly necessary, as an optimization loop could be defined that included ecological signalling or translated biological signalling into the algorithm. A simple example: the algorithm uses a biological sunflower, and by optically tracking its bloom angle, encodes the optimal path of light for a geographically and thermally similar system.

Non-biological systems could also be encoded for input into the optimization loop. Existing roof angles, pedestrian flows, space utilization, or other parameters could all be digitized and translated into selection criteria.

Finally, this thesis focussed solely on solo objects. There are two ways of interacting with a field of objects, or a population of individuals. The selection and fitness evaluation of an individual can be analyzed in reference to that individual in an idealized field of generic individuals. This is a pseudo-population type of analysis, in which the individual is still the evaluation focus, but the success of the individual depends on its preset neighbors. Alternatively, the entire population can be the focus for the optimization. In this type of algorithm, the genetic data is orders of magnitude larger, with all individuals defined on a

single genome object. Then, the fitness function is a function on the whole population. The result is a population of populations, in which the highest-scoring population is the 'best individual' (the best individual population.) Logically, this process can be continued to a population of population populations, but the number of components rises exponentially and requires a concomitant increase in computational resources.

XXX.Summary

The overarching organizing function for my thesis was the construction of a software prototype. This prototype generates a system of spatial elements, which is the result of an iterated procedural algorithm. This system allows for measurement or analysis, and can be optimized or 'tuned' to satisfy specified performance criteria.

The prototype system exhibits some parameter-critical behavior; certain stable optima are much too simple to be useful, and many characteristics have to be constrained manually or pseudo-manually to direct the process towards a coherent end-state goal. In general, however, the stochastic optimization of these spatial systems shows the potential for directed, goal oriented, guided-autonomic design elements. Specifically, the design of the algorithm, in this case the genetic fitness function, becomes the object of the design process. The procedural structure carries the generating potential for the subsequent design result. With the further development of more sophisticated and precise optimization criteria (e.g. fitness functions,) true computational design-assistance is achievable.

Appendix A: Bibliography

"City Gen2 on the horizon | Wasabi 3D – Blog", Accessed November 17, 2010. http://www.wasabi3d.com/blog/index.php/2010/04/28/city-gen-2/.

"Custom MEL Tools", Accessed November 17, 2010. http://richsunproductions.info/vsfx705/modelingsuite.html.

"Delaunay/Dirichlet Tessellation", Last accessed November 14, 2010. http://www.passagesoftware.net/webhelp/Delaunay_Dirichlet_Tessellation.htm.

"Elastic moduli of model random three-dimensional closed-cell cellular solids – Voronoi tessellations", Last visited November 14, 2010. http://ciks.cbt.nist.gov/garbocz/closedcell/node5.html.

"GPU Gems – Chapter 37. Octree Textures on the GPU", Last accessed November 14, 2010. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter37.html.

"labs:pointsetreconstruction · McNeel Wiki," last modified November 8, 2010. http://wiki.mcneel.com/labs/pointsetreconstruction.

"Nonlinear Iterated Function Systems", accessed November 14, 2010, http://www.cg.tuwien.ac.at/research/vis/dynsys/nifs/.

"Pyevolve 0.6rc1 released ! | Pyevolve", accessed November 13, 2010, http://pyevolve.sourceforge.net/wordpress/?p=1164.

"pymel – Project Hosting on Google Code", accessed November 13, 2010, http://code.google.com/p/pymel/.

"SCG Screenshots", Accessed November 17, 2010. http://arnaud.ile.nc/sce/screenshots.php.

"The Nature of Code at daniel shiffman", accessed November 14, 2010. http://www.shiffman.net/teaching/nature/.

Aranda, Benjamin, and Chris Lasch. *Pamphlet Architecture 27: Tooling.* 1st ed. (Princeton Architectural Press, 2005).

Autodesk, "Autodesk – Developer Center – Autodesk Maya", accessed November 13, 2010, http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=9469002.

C. Mulders-Kusumo, "Spatial Configuration of the Area Around Delft Central Station," in Bilsen, A.

Van, F. van der Hoeven, and Jürgen Rosemann. *Urban transformations and sustainability: progress of research issues in urbanism 2005*. IOS Press, 2006, 67–71.

Calvo, Jorge Alberto. *Physical and numerical models in knot theory: including applications to the life sciences.* World Scientific, 2005.

Cottrell, Michelle. *Guide to the Leed Green Associate (Ga) Exam*. John Wiley and Sons, 2010.

Dawkins, Richard. *The selfish gene.* Oxford University Press, 2006.

Dunn, Fletcher, and Ian Parberry. *3D math primer for graphics and game development.* Jones & Bartlett Learning, 2002.

Frazer, John. *An evolutionary architecture.* London: Architectural Association, 1995.

Giovanni Corbellini. *Bioreboot: the architecture of R&Sie(n)* (Princeton Architectural Press, 2009).

Goodman, Jacob E., János Pach, and Emo Welzl. *Combinatorial and computational geometry.* Cambridge University Press, 2005.

Grimm, Volker, and Steven F. Railsback. *Individual-based Modeling and Ecology.* Princeton University Press, 2005.

Hynes, H. Patricia, and Russ Lopez. *Urban health: readings in the social, built, and physical environments of U.S. cities.* Jones & Bartlett Learning, 2009.

Ilachinski, Andrew. *Cellular automata: a discrete universe.* World Scientific, 2001.

Jablonka, Eva, and Marion J. Lamb. *Evolution in four dimensions: genetic, epigenetic, behavioral, and symbolic variation in the history of life.* MIT Press, 2005.

Jacobs, Jane. *The death and life of great American cities.* rev. ed., New York City: Vintage Books, 1992.

Kumar, Sanjeev, and Peter J. Bentley. "Biologically Inspired Evolutionary Development." In *Evolvable Systems: From Biology to Hardware,* 99–106, 2003.

Leondes, Cornelius T. *Computer-Aided Design, Engineering, and Manufacturing: Systems Techniques and Applications, Volume III, Operational Methods in Computer-Aided Design.* CRC Press, 2000.

Peitgen, Heinz-Otto, Hartmut Jürgens, and Dietmar Saupe. *Chaos and fractals: new frontiers of science.* Springer, 2004.

Processing.org, "Basics \ Processing 1.0", accessed November 14, 2010. http://www.processing.org/about/.

Raper, Jonathan. *Multidimensional geographic information science.* London: Taylor and Francis, 2000.

Reynolds, Craig. "Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)" http://www.red3d.com/cwr/boids/

Rogers, Everett M. *Diffusion of innovations,* 4th ed. New York: Simon and Schuster, 1995.

Schwank, Inge. "Cognitive Structures and Cognitive Strategies in Algorithmic Thinking," in Dettori, Giuliana, and North Atlantic Treaty Organization, Scientific Affairs Division. *Cognitive models and intelligent environments for learning programming.* Springer, 1993.

Skiena, Steven S. *The Algorithm Design Manual.* 2nd ed. Springer, 2008.

Tarbell,J. Sand.Dollar, 2004. "Complexification | Gallery of Computation", accessed November 14, 2010. http://www.complexification.net/gallery/machines/sandDollar/.

ted.com, "Mihaly Csikszentmihalyi on Flow," FLV, Accessed November 18, 2010, http://www.ted.com/talks/mihaly_csikszentmihalyi_on_flow.html, 2004.

Thompson, D'Arcy Wentworth. *On Growth and Form: A New Edition.* rev. ed., Courier Dover Publications, 1992, new ed. Cambridge: University Press, 1942.

Vis, B. N. *Built Environments, Constructed Societies.* Leiden:Sidestone Press, 2010.

vvvv group, "vvvv: a multipurpose toolkit," accessed November 14, 2010, http://vvvv.org/propaganda.

Walker, B., C. S. Holling, S. R. Carpenter, and A. Kinzig. 2004. “Resilience, adaptability and transformability in social–ecological systems.” Ecology and Society 9(2): 5. http://www.ecologyandsociety.org/vol9/iss2/art5

Weisstein, Eric W. "Minimal Enclosing Circle." From MathWorld—A Wolfram Web Resource. accessed November 14, 2010. http://mathworld.wolfram.com/MinimalEnclosingCircle.html

William J. Cavanaugh, Gregory C. Tocci, and Joseph A. Wilkes, *Architectural Acoustics: Principles and Practice* (John Wiley and Sons, 2009).

Wiseman, Carter. *Louis I. Kahn: beyond time and style : a life in architecture.* New York: W. W. Norton & Company, 2007.

Appendix B: Related Works

Artificial Life Models in Software. London: Springer-Verlag, 2005.

Advances in Evolutionary Computing for System Design. Studies in computational intelligence v. 66. Berlin: Springer, 2007.

Behavioral Mechanisms in Evolutionary Ecology. Chicago: University of Chicago Press, 1994.

Beistegui, Miguel de, and ebrary, Inc. *Thinking with Heidegger Displacements*. Studies in Continental thought. Bloomington: Indiana University Press, 2003.

Bentley, P.J., ed.: *Evolutionary Design by Computers.* Morgan Kaufmann (1999)

Bentley, P., Corne, D., eds.: *Creative Evolutionary Systems*. Morgan Kaufmann (2001)

Bermejo Tirado, Jesús. *La Arquitectura Sagrada Ibérica: Orígenes, Desarrollos Y Contextos*. BAR international series 1800. Oxford: Archaeopress, 2008.

Bioclimatic Housing: Innovative Designs for Warm Climates. London: Earthscan, 2008.

Biomaterials Fabrication and Processing Handbook. Boca Raton: CRC Press/Taylor & Francis, 2008.

Broughton, T., Coates, P., Jackson, H.: *Exploring 3d design worlds using lindenmayer systems and genetic programming*. In Bentley, P.J., ed.: *Evolutionary Design by Computers*. Morgan Kaufmann (1999)

Carruthers, Peter. *The Architecture of the Mind: Massive Modularity and the Flexibility of Thought*. Oxford: Clarendon Press, 2006.

Chaos and Complexity in the Arts and Architecture. New York: Nova Science Publishers, 2007.

Crowe, Norman. *Nature and the Idea of a Man-Made World: An Investigation into the Evolutionary Roots of Form and Order in the Built Environment*. Cambridge, Mass.: MIT Press, 1995.

Crowther, Richard L. *Ecologic Architecture*. Boston: Butterworth Architecture, 1992.

Davern, Jeanne M. *Architecture, 1970–1980: A Decade of Change*. New York: McGraw-Hill, 1980.

De Landa, Manuel. *A Thousand Years of Nonlinear History*. Swerve editions. New York: Zone Books, 1997.

Dercole, Fabio. *Analysis of Evolutionary Processes: The Adaptive Dynamics Approach and Its Applications*. Princeton series in theoretical and computational biology. Princeton: Princeton University Press, 2008.

Leach, Christopher et al. *Digital Tectonics*. Chichester: Wiley-Academy, 2004.

Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering. Berlin: Springer, 2004.

Evolutionary Algorithms in Engineering Applications. Berlin: Springer, 1997.

Evolutionary Computation. Bristol: Institute of Physics Publishing, 2000.

Evolutionary Computation: The Fossil Record. New York: IEEE Press, 1998.

Flake, Gary William. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. The MIT Press, 2000.

Frazer, J.: *An Evolutionary Architecture*. Architectural Association, London (1995)

Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies. New York: Springer Science+Business Media, 2006.

Hausladen, Gerhard. *ClimateSkin: Building-Skin Concepts That Can Do More with Less Energy*. Basel: Birkhäuser, 2008.

Helvey, T. C. *The Age of Information; an Interdisciplinary Survey of Cybernetics*. Englewood Cliffs, N.J: Educational Technology Publications, 1971.

Hemberg, M., O'Reilly, U.M.: *Integrating generative growth and evolutionary computation for form exploration*. (Submitted to Genetic Programming and Evolvable Machines)

Hight, Christopher. *Architectural Principles in the Age of Cybernetics*. New York: Routledge, 2008.

Hornby, G.S., Pollack, J.B.: *The advantages of generative grammatical encodings for physical design*. In: Congress on Evolutionary Computation. (2001)

Indefensible Space: The Architecture of the National Insecurity State. New York: Routledge, 2008.

International Botanical Congress. *Axioms and Principles of Plant Construction: Proceedings of a Symposium Held at the International Botanical Congress, Sydney, Australia, August 1981*. Hague: M. Nijhoff/Dr W. Junk Publishers, 1982.

International Conference on Evolvable Systems. *Evolvable Systems: From Biology to Hardware*. Berlin: Springer, 1997.

King, Ross. *Emancipating Space: Geography, Architecture, and Urban Design*. The Guilford Press, 1996.

Koza, J.R.: *Genetic programming: on the programming of computers by means of natural selection*. MIT Press (1992)

Krige, John. *Science, Revolution, and Discontinuity*. Atlantic Highlands, N.J: Humanities Press, 1980.

Kumar, S., Bentley, P.J., eds.: *On growth, form and computers*. Elsevier (2003)

Kumar, Sanjeev, and Peter J. Bentley. "Biologically Inspired Evolutionary Development." In Evolvable Systems: From Biology to Hardware, 99–106, 2003.

Latour, Bruno. *Politics of Nature: How to Bring the Sciences into Democracy*. Cambridge, Mass: Harvard University Press, 2004.

Leydecker, Sylvia. *Nano Materials in Architecture, Interior Architecture, and Design*. Basel: Birkhäuser, 2008.

Lim, C. J. *Devices: A Manual of Architectural + Spatial Machines*. Amsterdam: Elsevier, 2006.

Maher, M.L.: *A model of co-evolutionary design*. Eng. Comput. (Lond.) 16(3–4) (2000) 195–208

Mendler, Sandra, and Obata & Kassabaum Hellmuth. *The HOK Guidebook to Sustainable Design / Odell, William.* New York: Wiley, 2000.

Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin ; New York: Springer-Verlag, 1996.

Mikellides, Byron. *Architecture for People: Explorations in a New Humane Environment*. New York: Holt, Rinehart, and Winston, 1980.

Mitchell, M.: *An introduction to genetic algorithms.* MIT Press (1996)

Modeling Biology: Structures, Behavior, Evolution. The Vienna series in theoretical biology. Cambridge, Mass: MIT Press, 2007.

Murawski, K., Arciszewski, T., Jong, K.A.D.: *Evolutionary computation in structural design*. Eng. Comput. (Lond.) 16(3–4) (2000) 275–286

NanoBioTechnology: Bioinspired Devices and Materials of the Future. Totowa, N.J: Humana Press, 2008.

National Trust for Historic Preservation in the United States, Society of Architectural Historians., and American Institute of Architects. *Old & New Architecture :* Washington, D.C.: Preservation Press, National Trust for Historic Preservation, 1980.

NBIC Convergence. *The Coevolution of Human Potential and Converging Technologies*. Annals of the New York Academy of Sciences v. 1013. New York, N.Y: New York Academy of Sciences, 2004.

Norberg-Schulz, Christian. *The Concept of Dwelling: On the Way to Figurative Architecture*. Architectural documents;. [Milan] : Electa ; New York : Rizzoli, 1985.

O'Neill, M., Ryan, C.: *Grammatical Evolution – Evolving programs in an arbitrary language*. Kluwer Academic Publishers (2003)

Patterns 2: Design, Art and Architecture. Basel: Birkhäuser, 2008.

Pfammatter, Ulrich. *Building the Future: Building Technology and Cultural History from the Industrial Revolution Until Today*. Munich: Prestel, 2008.

Pinker, Steven. *The Stuff of Thought: Language as a Window into Human Nature*. New York: Viking, 2007.

Prusinkiewicz, P., Lindenmayer, A.: *The algorithmic beauty of plants*. Springer (1991)

Romero, J., Machado, P., Santos, A., Cardoso, A.: *On the development of critics in evolutionary computation artists*. In: EvoMusart workshop, 6th European Conference on Genetic Programming, Essex (2003)

Royal Society (Great Britain). *The Growth and Form of Modular Organisms*. Philosophical transactions of the Royal Society of London v. 313, no. 1159. London: The Royal Society, 1986.

Salkind, Neil J. *Statistics for People Who (think They) Hate Statistics*. Excel ed. Thousand Oaks: SAGE Publications, 2007.

Shepheard, Paul. *Artificial Love: A Story of Machines and Architecture*. Cambridge, Mass: MIT Press, 2003.

Shi, X.G., Gero, J.S.: *Design families and design individuals*. Eng. Comput. (Lond.) 16(3–4) (2000) 253–263

Stitt, Fred A. *Ecological Design Handbook: Sustainable Strategies for Architecture, Landscape Architecture, Interior Design, and Planning*. London: McGraw-Hill, 1999.

Sudjic, Deyan, and Future Systems (Firm). *Future Systems*. London: Phaidon, 2006.

The Virtual Dimension: Architecture, Representation, and Crash Culture. 1st ed. New York: Princeton Architectural Press, 1998.

Thompson, D'Arcy Wentworth. *On Growth and Form: A New Edition*. rev. ed., Courier Dover Publications, 1992, new ed. Cambridge: University Press, 1942.

Tsui, Eugene. *Evolutionary Architecture: Nature as a Basis for Design*. New York: John Wiley, 1999.

Watanabe, Makoto Sei. *Induction Design: A Method for Evolutionary Design*. The IT revolution in architecture. Basel: Birkhäuser, 2002.

von Neumann, J.: *The theory of self-reproducing automata*. University of Illinois Press (1966)

Appendix C: Selected Code

Curve Generation (MEL)

```
//string $c1=curveGen(30);
global proc curveGen(int $lenA){
	string $createCurve = "curve -d 3 ";
	string $myCurve;
	//generate random curves
	for($i=0;$i<$lenA;$i++){
		float $random = `rand 3`;
		$createCurve = ($createCurve + " -p " + " 0 " + $random + " " + $i);
		}
	//create curve object
	print $createCurve;
	eval($createCurve);
}
global proc curveMult(int $cReps, int $cLength){
	for($i=0;$i<$cReps;$i++){
		curveGen($cLength);
		move -r $i ($cLength/2) ($cLength%4);
	}
}
```

Cubic Array Generation, Boundary Only (MEL)

```
global proc threeGridBoundary(string $shapeCmd){
//specify number of items on each side of cube
int $sideNum=11;
	for($i=0;$i<$sideNum;$i++){
		for($j=0;$j<$sideNum;$j++){
			for($k=0;$k<$sideNum;$k++){
				if($i=1 || $j=1 || $k=1){
				eval($shapeCmd);
				move -r $i $j $k;
				}
			}
		}
	}
}
```

Cube Set (pymel)

```
#cubegen in time 100820-174927
from pymel.all import *
import random, time

#if existing, move out of way
existingCubes=ls("*Cube*", type="transform")
if len(existingCubes)>1: print([a.translateBy([0,-50,0]) for a in existingCubes])

#make a bunch of cubes (20 length side)
w1=20
h1=12
d1=14
count1=12
grid1=25
grid2=24
""" testing
a=polyCube()
setKeyframe(a)
a[0].translateBy([-2,3,-3])
setKeyframe(a, t=60)
select(a)
keyframe(a,query=1);
dir(a[0].getShape())"""

for i in range(0,count1):
    for j in range(0,count1):
        c=polyCube(w=w1,h=h1,d=d1)
        setKeyframe(c,t=30*i+30*j)

        c[0].translateBy([w1*i+grid1*i,random.uniform(-5,5),d1*j+grid2*j])
        setKeyframe(c,t=5*i*j)
        face=c[0].f[int(random.uniform(0,6))]
        facePoints=face.getPoints()
        for z in facePoints:
            z.x+=-10*i%72
            z.y+=2*(count1-i)*random.uniform(-2,2)
        face.setPoints(facePoints)
        setKeyframe(c,t=30*i+30*j+15)

#move same face of each cube randomly
"""
setF=ls("*.f["+str(int(random.uniform(0,6)))+"]")
len(setF)
for i in range(0,10,1):
    collP=setF[i].getPoints()
    for j in collP:
        j.x+=random.uniform(-10,10)
        j.y+=random.uniform(-10,10)
        j.z+=random.uniform(-10,10)
    setF[i].setPoints(collP)
    print "done"
    """
```

Cubic Array Generation (pymel)

```
#cubegen in time perpoint 100821-001934
from pymel.all import *
import random, time

#if existing, move out of way
existingCubes=ls("*Cube*", type="transform")
if len(existingCubes)>1:
    for a in existingCubes:
        a.translateBy([0,-50,0])
        keyframe(a.translateY,edit=True,relative=True,valueChange=-50)

#make a bunch of cubes (20 length side)
w1=4
h1=7
d1=6
count1=7
grid1=12
grid2=21
""" testing
a=polyCube()
setKeyframe(a)
a[0].translateBy([-2,3,-3])
setKeyframe(a, t=60)
select(a)
keyframe(a,query=1);
dir(a[0].getShape())"""

for i in range(0,count1):
    for j in range(0,count1):
        c=polyCube(w=w1,h=h1,d=d1)
        setKeyframe(c,t=10*i+10*j)

        c[0].translateBy([w1*i+grid1*i,random.uniform(-5,5),d1*j+grid2*j])
        setKeyframe(c,t=30*i+30*j)
        face=c[0].f[int(random.uniform(0,6))]
        setKeyframe(face,controlPoints=True,t=30*i+30*j)

        facePoints=face.getPoints()
        for z in facePoints:
            z.x+=-10*i%72
            z.y+=2*(count1-i)*random.uniform(-2,2)
        face.setPoints(facePoints)
        setKeyframe(face,controlPoints=True,t=30*i+30*j+60)

        setKeyframe(c,t=30*i+30*j+15)

#move same face of each cube randomly
"""
setF=ls("*.f["+str(int(random.uniform(0,6)))+"]")
len(setF)
for i in range(0,10,1):
    collP=setF[i].getPoints()
    for j in collP:
        j.x+=random.uniform(-10,10)
```

```
        j.y+=random.uniform(-10,10)
        j.z+=random.uniform(-10,10)
    setF[i].setPoints(collP)
    print "done"
    """
```

Move Each Object A Small Random Distance (MEL)

```
global proc jiggleMe(string $listOfThings[],float $magnitude){
      for ($individual in $listOfThings){
            if(nodeType($individual)=="transform"){
                  vector $ma_translateVector=`getAttr ($individual+".translate")`;
                  //print ($ma_translateVector);
                  $ma_translateVector += <<rand(-$magnitude,$magnitude),rand(-$magnitude,
$magnitude),rand(-$magnitude,$magnitude)>>;
                  setAttr ($individual + ".translate") ($ma_translateVector.x)
($ma_translateVector.y) ($ma_translateVector.z);
            }
            else { print (".");}
      }
}
```

Generate Ball And Stick Model

```
# ball-and-stick-cols 100221-1849.py
# import relevant libs
import maya.cmds as cmds
import random as rand
import math

# init var
cubeShift=[0,0,0]
cubePos=[0,0,0]
cubeDimF=[]
cubeDimFPrev=[0,0,0]

numLines=int(rand.uniform(5,12))
print numLines
numCubes=int(rand.uniform(5,17))
print numCubes
cubeDimArray=[]
cubeDim1d=[]
cubeDim2d=[]
# init 2d list for shape dimension values as floats
numLinesGroup=range(0,numLines)
numCubesGroup=range(0,numCubes)
tripleCoord=([x * 0.1 for x in range(0, 3)])
for a in numCubesGroup:
      cubeDim1d.append(tripleCoord)
for b in numLinesGroup:
      cubeDim2d.append(cubeDim1)
# print cubeDim2d


for itrA in numLinesGroup:
      lateral_shift=rand.uniform(15,20)
```

```
        cubePos[0]=0
        cubePos[1]=0
        cubePos[2]=0

        cubeDimFPrev[0]=0

        for itrB in numCubesGroup:
                cubeDim2d[itrA].append([rand.uniform(0,1)*5,rand.uniform(0,1)*5,rand.uniform(0,1)*5])
                cubeShift[0]=cubeDim2d[itrA][itrB][0]
                cubeShift[1]+=rand.uniform(-1,1)
                cubeShift[2]=0
                cubePos[0]+=cubeDimFPrev[0]/2+cubeDim2d[itrA][itrB][0]/2
                cubeDimFPrev[0]=cubeDim2d[itrA][itrB][0]
                cubePos[1]+=cubeShift[1]
                cubePos[2]=itrA*lateral_shift

# scale by dim 2d list, not scale by position in loop which would be *itrB
                obj1=cmds.polySphere(r=cubeDim2d[itrA][itrB][0]*.25,sx=7,sy=6)
                obj2=cmds.polyCube(width=cubeDim2d[itrA][itrB][0],height=cubeDim2d[itrA][itrB]
[1],depth=cubeDim2d[itrA][itrB][2])

                cmds.select(obj1, add=True)
                cmds.select(obj2, add=True)

                cmds.move(cubePos[0],cubePos[1],cubePos[2])

                obj3=cmds.polyCylinder(r=.5,h=cubePos[1],sx=10,sy=3,sz=6,n='stake'+str(itrA)+str(itrB))
                cmds.move(cubePos[0],cubePos[1]/2,cubePos[2])

                cmds.select(workingPlane+".cv["+divA+"]["+divB+"], r=True")
```

Generate Curved Surface

```
#controlled curve surface generator 100506-195627
# import relevant maya libs
from pymel.core import *
import random, time

#c1=curve_generate(30)
cs1=surface_generate(50,5,50,5,.51)
#loft(cs1)

def surface_generate(points,height,lines,width,ratio):
    curveset=[]
    for a in xrange(lines):
        select(cl=1)
        c_single=curve_generate(points,height,1/ratio)
        select(c_single)
        move([width*a*ratio,0,0],relative=1)
        curveset.append(c_single)
    curvesetsurface=loft(curveset)
    return curvesetsurface

def curve_generate(points,height,ratio):
    pointset=[]
```

```
    for a in xrange(0,points):
        pointset.append([0,random.uniform(0,height),a*ratio])#method 1 positive
    return curve(degree=3,p=pointset)
```

Generate Recursive Tree

```
#branching 3d 100509-211407
"""
adapted from
 * Recursive Tree
 * by Daniel Shiffman.
"""

# import relevant maya libs
from pymel.core import *
from pymel.util import *
import random, time

theta=0.0#initialize

def drawLines():
    current_vector=datatypes.Vector(0,0,0)#origin start
    theta=random.uniform(0,math.pi)#angle seed
    thetaZ=theta/5#change later, z angle
    length1=50
    #addition_vector=datatypes.Vector(length1*math.sin(theta),length1*math.cos(theta),length1*math.sin
(thetaZ))
    #next_vector=current_vector+addition_vector
    #curve(d=1,p=[current_vector, next_vector])
    #current_vector=next_vector
    branchMe(length1,current_vector,theta,thetaZ)
    #branchMe(length1,current_vector,-theta,-thetaZ)

def branchMe(length,curr,theta,thetaZ):
    length*=0.5
    print "branch"+str(length)
    theta=random.uniform(0,math.pi)#angle seed
    thetaZ=theta/10
    if length > 4:
        #print curr
        theta+=theta
        thetaZ+=thetaZ
        addition_vector=datatypes.Vector(length*math.sin(theta),length*math.cos(theta),length*math.sin
(thetaZ))
        next_vector=curr+addition_vector
        curve(d=1,p=[curr, next_vector])
        #print next_vector
        branchMe(length,next_vector,theta,thetaZ)
        branchMe(length,next_vector,-theta,-thetaZ)
        branchMe(length,next_vector,theta/2,thetaZ/2)
        branchMe(length,next_vector,-theta/2,-thetaZ/2)

def main():
    drawLines()

if __name__=='__main__':
```

```
start_time=time.clock()
result=main() #execute if not imported
print "completed"
```

Scatter Objects Sequentially

```
#chainscatter-minmax 100318-000804
# ga test start

# import relevant libs
from pymel.core import *
from pymel.core.animation import deformer
import random
import pyevolve
print pyevolve.__version__

# declare array variables
start_vector=[]
records_vector=[]
next_loc=[0.0,0.0,0.0] #zero out counter vector
dir_select=[-1.0,1.0,1.0,-1.0]

select(all=1)
#all_objects=selected()
all_objects_count=len(selected())
select(cl=1)

#init genetic info
bin_list=[(int(round(random.uniform(0,1)))) for gen_len in range(0,all_objects_count)]
#bin_list=[dir_select[int(round(random.uniform(0,3)))] for gen_len in range(0,len(all_objects))]
print bin_list
# define necessary functions
def shmork(selectedSet,geneticString):

	for idx1,itrBox in enumerate(selectedSet):
		dir_mag=[dir_select[g] for g in geneticString[idx1:idx1+3]]#select direction as list
		#print dir_index
		if(itrBox.getShape()):
			bb=itrBox.getShape().boundingBox()
		else:
			bb=itrBox[0].getBoundingBox()
		#print bb
		next_loc[0]+=bb.width()*dir_mag[0] #position x coord
		next_loc[1]=bb.height()*.5*dir_mag[1] #position on plane of origin
		next_loc[2]+=bb.depth()*.5*dir_mag[2] #position z coord
		#print(next_loc)
		itrBox.translate.set(next_loc)


def cubegen(cubenum):
	for a in xrange(0,cubenum,1):
		start_vector.append(random.uniform(1,10))
		tmpA=start_vector[a]
		obj1=polyCube(w=tmpA,h=tmpA,d=tmpA,sx=4,sy=4,sz=4,n="cuber")
		#dir(obj1[0].getShape().inputs()[0])
```

```
def specialBox(selectedSet):
	for pieceA in selectedSet:
		records_vector.append(pieceA.boundingBox().min())
		records_vector.append(pieceA.boundingBox().max())

	minsX=min([piece[0] for piece in records_vector])
	minsY=min([piece[1] for piece in records_vector])
	minsZ=min([piece[2] for piece in records_vector])

	maxsX=max([piece[0] for piece in records_vector])
	maxsY=max([piece[1] for piece in records_vector])
	maxsZ=max([piece[2] for piece in records_vector])

	#print(minsX,minsY,minsZ,maxsX,maxsY,maxsZ)
	#print(len(records_vector))
	return minsX,minsY,minsZ,maxsX,maxsY,maxsZ

# use functions on example sets
cubegen(10)
testCollection=ls("*cuber*",type='transform')
#select(testCollection)
#deformer( type="squash")
select(cl=1)
#for a1 in (ls("*cuber*",type='transform')):
#	if(random.uniform(0,2)<=2): # always, for now, use this to not activate some pieces
#		select(a1,add=1)

shmork(testCollection, bin_list) # spread objects around

bounds1=specialBox(testCollection) # eval the bounds of the spread
x_spread=bounds1[3]-bounds1[0]
y_spread=bounds1[4]-bounds1[1]
z_spread=bounds1[5]-bounds1[2]

score_spread=x_spread+y_spread+z_spread #fitness function!!
print bounds1
```

Quickhull Interface

```
from pymel.core import *
import random

"""
for a in ls('*Cube*',type='transform'):
    b=(a.rotate.get())
    a.rotate.set(b+45)
    a.getShape().vtx.translateBy([0,-1,-2])
"""


#f=newFile(f=1)

class rBoxqHull:
    name="rbqh"
    status=None
    def __init__(self):
        pass
        #self.obj1=self.createHull()

    def createHull(self):
        ssize=str(int(random.uniform(120,190))) #formatted for sh input
        #print ssize
        r2=util.shellOutput('/Users/xaei/Desktop/0-all/qhull-2010.1/rbox D3 '+ssize+' \
        |/Users/xaei/Desktop/0-all/qhull-2010.1/qhull o',convertNewlines=1)
        r2s=r2.split("\n") #separate into lines
        params=r2s[1].split() #get quantity of each element from qhull
        numPts=int(params[0]) #first element
        numFaces=int(params[1]) #first element
        pointSet=[sA.split() for sA in r2s[2:2+numPts]]#r2s into lines, starting at index 2
        facetSet=[sA.split() for sA in r2s[2+numPts:2+numPts+numFaces]]#r2s into lines, starting at
index 2
        #[str(no)+'a' for no in xrange(0,10)]
        #for point in xrange(0,numPts,1): print r2s[point].split()
        #for a in (xrange(0,10)): print a

        shapeTmp=[]
        for facet in facetSet:
            tmpPointIndices=[int(a) for a in facet[1:]]
            tmpPointSet=[[float(c) for c in pointSet[b]] for b in tmpPointIndices]
            shapeTmp.append(polyCreateFacet(p=tmpPointSet))
        shapeUnited=polyUnite(shapeTmp)
        return shapeUnited

for j in xrange(0,5):
    for i in xrange(0,5):
        a=rBoxqHull()
        hullA=a.createHull()
        move(hullA,[i*2,0,j*2])
    #move("polySurface2357",[10,20,30])
        #c=b[0].getShape().vtx
        #for d in c:
        #    d.translateBy([random.uniform(0,.11),random.uniform(0,.11),random.uniform(0,.11)])
        #selected()[0].vtx
```

Genetic Algorithm Controller

```
# evolve-total vol mut elit 101110-150956.py
# import relevant maya libs
from pymel.core import *
from pymel.core.datatypes import *
import random
import manipulations_am as ma

# GA imports
from pyevolve import G1DList
from pyevolve import GSimpleGA
from pyevolve import DBAdapters

try:
    del setupFlag
    newFile(f=1)
except NameError:
    setupFlag=0
    ma.setupFile()
    ma.customPointLight()
#del setupFlag #for testing

chk=ma.initCheckfile()
numGen=10
sizePop=50

def eval_func(passed_genome):
    if(not(util.path(chk).exists())):
        #print "check file disappeared... killing process"
        return 0

    #print ga.getCurrentGeneration()
    #print [a for a in passed_genome]
    print passed_genome[0:]

    #reset score
    score=0.0

    #generate
    target_object=generate_object(passed_genome[0:5],"Z")[0]

    #print "eval..."+str(target_object)
    #ma.smoothObject(target_object)

    #target_object=generate_object([50,22,43,22,44,15],"A")[0]
    #target_object.vtx
    #passed_genome=[int(random.uniform(0,100)) for a in xrange(0,30)]
    #manipulate
    #select("Zaa*") #select only this run?
    #polyGeoSampler
(target_object,dg=1,ids=1,cs=1,lo=0,fs=1,sf=0.2*100,su=1,cdo=0,sampleByFace=1)
    #ma.extrudeFaceObject(target_object)
    #ma.prelightDisplace(target_object, passed_genome[6:11])
    #ma.expandFaceAlongNormals([target_object],passed_genome[11]*.1,up=1)
```

```
    #polySmooth(target_object)
    #ma.prelightDisplace(target_object, passed_genome[6:11])
    #polyReduce(target_object,border=passed_genome[12]*.
01,keepBorder=0,percentage=passed_genome[13]*4)
    #ma.prelightLight(target_object, passed_genome[20:25])
    #ma.prelightDisplace(selected())
    #ma.prelightDisplace(target_object,[50,60,70,80,90,99])
    #ma.faces extrusion
    #ma.faces move out along normals
    #evaluate
    field_data=ma.evalField([target_object])
    print field_data
    #ma.proxyComplexity(ma.evalField([selected()[0]]))
    #type(selected())
    #print field_data
    proxyComplexityScore=ma.proxyComplexity(field_data)
    objBBox=target_object.boundingBox()
    proxyVolumeScore=objBBox.height()*objBBox.width()*objBBox.depth()
    score=proxyComplexityScore+0.01*proxyVolumeScore
    #score=1/(score+1)#minimize score
    #delete(selected())
    #move time forward, hide current set of objects
    print "score:",score,

    hide(target_object)
    return score # fitness quantification

def generate_object(args, prefix="A"):
      #points for qhull
      ssize=str(int(args[0])+8)#ensure volumetric object
      #scaleConstant=[2,2,2]
      multScalar=.1
      scaleConstant=(Vector(args[1:4])*multScalar)+(1,1,1)
      #print scaleConstant

      r2=util.shellOutput('/Users/xaei/Desktop/0-all/qhull-2010.1/rbox D3 '+ssize+' \
      |/Users/xaei/Desktop/0-all/qhull-2010.1/qhull o',convertNewlines=1)
      r2s=r2.split("\n") #separate into lines

      params=r2s[1].split() #get quantity of each element from qhull
      numPts=int(params[0]) #first element
      numFaces=int(params[1]) #first element
      pointSet=[sA.split() for sA in r2s[2:2+numPts]]#r2s into lines, starting at index 2
      facetSet=[sA.split() for sA in r2s[2+numPts:2+numPts+numFaces]]#r2s into lines, starting at index
2

      shapeTmp=[]

      for facet in facetSet:
            tmpPointIndices=[int(a) for a in facet[1:]]
            tmpPointSet=[[float(c) for c in pointSet[b]] for b in tmpPointIndices]
            shapeTmp.append(polyCreateFacet(p=tmpPointSet))
      #sUnitedShape=polyUnite(shapeTmp, name="abc")
      sUnitedShape=polyUnite(shapeTmp, name=prefix+"aa")
      #sUnitedVtx=polyMergeVertex(sUnitedShape)
      sUnitedEdge=polySewEdge(sUnitedShape[0].e)
```

```
        #selectMode(object=1)
        #sUnitedShape[0].setScale(scaleConstant)
        #use this instead:
        scale(sUnitedShape[0].vtx,scaleConstant,r=1,dph=1)
        #delete(sUnitedShape,constructionHistory=1)
        return sUnitedShape

def evo(field_size=1, numGenerations=2, sizePopulations=10):

        #genome size for each indiv=point locs (3) + locrotscale of object (9)
        genome_size=3*field_size+9+5#+50 as a buffer

        #ga module
        genome=[] #init genome object
        genome = G1DList.G1DList(genome_size)
        genome.evaluator.set(eval_func)
        ga = GSimpleGA.GSimpleGA(genome)
        ga.setGenerations(numGenerations)
        ga.setPopulationSize(sizePopulations)
        ga.setElitism(True)
        ga.setMutationRate(0.1)
        ga.stepCallback.set(step_callback)

        #print ga.getCurrentGeneration

        #ga.initialize()
        #ga.step()

        ga.evolve(freq_stats=50)
        return ga
        #ga.bestIndividual()q

def step_callback(gp_engine):
    #delete all deformers cache history
    #bakePartialHistory(all=1)
    #delete(all=1,ch=1)
    pass

def main():
    gaResult=evo(1, numGenerations=numGen, sizePopulations=sizePop)
    #delete(all=1,ch=1)
    ma.spreadObjects(ls("*aa*",type="transform"),sizePop*3)
    popset=[a for a in gaResult.getPopulation()]

    #hide(all=1)
    showHidden(ls("*aa*",type="transform"))
    #[popset[i]==popset[i+1] for i in xrange(0,len(popset)-1)]
    return gaResult,popset
if __name__=='__main__':
    gaTest=main() #execute if not imported
print gaTest[0].bestIndividual().genomeList
eval_func(gaTest[0].bestIndividual())
```

```
# manipulations 101108-014236.py
from pymel.core import *
import random
import math

"""archi morpho manipulation library """

def setupFile():
    importFile("/Users/xaei/Desktop/101027-223448 sun template.mb")
    #a=directionalLight(intensity=3)

def customPointLight(intensityValue=100,decay=100,color=(1,1,1),rotation=(0,0,0),location=(0,0,0)):
      a=pointLight(intensity=intensityValue, decayRate=1, rgb=(.8,.8,1))
      aP=a.getParent()
      aP.setRotation([23,45,66])
      aP.setTranslation([1,100,100])

def expandFaceAlongNormals(s=0,mag=1,up=0):
    """move faces out along normal vector"""
    for piece in s:
        for face in piece.faces:
            if(up==1):
                if face.getNormal('world').y > 0.0:
                    normVec=face.getNormal('world')
                    move(face,normVec*mag,r=1)
            else:
                normVec=face.getNormal('world')
                move(face,normVec*mag,r=1)

def prelightDisplace(chunk=0,args=[50,50,50,50,50]):
    #polyGeoSampler(chunk,dg=1,ids=1,cs=1,lo=1,fs=1,sf=.01*float(args
[1]),su=1,cdo=1,sampleByFace=1)
    polyGeoSampler(chunk,dg=1,ids=1,cs=1,lo=0,fs=1,sf=0.5*float(args
[1]),su=1,cdo=0,sampleByFace=1)

def prelightLight(chunk=0,args=[0,0,0,0,0]):
    polyGeoSampler(chunk,dg=0,ids=1,cs=1,lo=1,fs=1,sf=.01*float(args
[1]),su=1,cdo=1,sampleByFace=1)

def smoothObject(chunk=0,args=[1,1.0,1]):
      if(len(chunk)>2):
                for piece in chunk:
                        polySmooth(piece, dv=args[0], c=args[1], cch=args[2])
      else: polySmooth(chunk, dv=args[0], c=args[1], cch=args[2])

def extrudeFaceObject(chunk=0,zz=0):
    #passed in from faceField only
    #if(len(chunk)>1):
    #    print "error too many objects"
    if(type(chunk.getShape())!=nt.Mesh):
        print "no valid piece"
    else:
        for face in chunk.f:
                ra=[random.uniform(-a,a)*.8 for a in xrange(1,7)]
                if(random.uniform(0,10)>10):
                    #polySetToFaceNormal(face)
```

```
                    polyExtrudeFacet(face,ls=(ra[0:3]),lt=(ra[3:6]))
                    polyExtrudeFacet(face,ls=(ra[0:3]),lt=(ra[1:4]))
                else:
                    polyExtrudeFacet(face,ls=(1,1,1),lt=(0,0,random.uniform(-1,1)))

def moveFaceObject(chunk):
    if(type(chunk.getShape())==nt.Mesh):
        for face in chunk.f:
            move(face,(0,random.uniform(-1,1),0),relative=1)

def faceField(chunk=0, extrudeFlag=1):
    if(type(chunk)!=nt.Transform):
        for piece in chunk:
            print piece+" just got enfaced",
            if(extrudeFlag==1):
                extrudeFaceObject(piece)
            else:
                moveFaceObject(piece)

def evalObject(chunk=0):
    if(type(chunk.getShape())==(nt.Mesh or nt.Transform)):
        vertexCount=chunk.numVertices()
        edgeLength=sum([a.getLength() for a in chunk.e])
        return (vertexCount,edgeLength)

def evalField(chunk=0):
    chunkReturn=[]
    for piece in chunk:
        if(type(piece)==nt.Transform):
            chunkReturn.append(evalObject(piece))
    return [a for a in chunkReturn]

def proxyComplexity(inputs):
    #proxy for complexity; vertices per unit linear material
    #always pass evalField into this
    return sum([a[0]/a[1] for a in inputs])

def initCheckfile():
    #init file for this run
    tmpFileCheckName=date(format="YYMMDD"+"-"+"hhmmss"+"_tmpFile")
    tmpFilePath="/Users/xaei/Desktop/mayaProcessTmp/"+tmpFileCheckName
    print tmpFilePath+" is the interruptor"
    util.path(tmpFilePath).touch()
    return tmpFilePath
"""
#check for file existance
chk=initCheckfile()
while(util.path(chk).exists()):print 1
else: print "check file disappeared... killing process"
"""

def deleteSomeFaces(chunk=0):
    print "defacing something..."

    if(type(chunk.getShape())==nt.Mesh):
        faceNum=len(chunk.f)
```

```
        defaceRatio=.3
        randomFaces=[int(random.uniform(0,faceNum)) for a in xrange(0,faceNum*defaceRatio)]
        delete(chunk.f[randomFaces])
        #print "chunk",
        #for face in chunk.f:
            #print ".",
            #if(random.uniform(-1,1)>.5):
                #print "!",
                #delete(face)

#([deleteSomeFaces(a) for a in selected()])

#type(selected()[1])
#faceField(selected())
#never do this: faceObject(selected())
#setupFile()
#c=evalField(selected())
#never do this: evalObject(selected()[0])
#proxyComplexity(c)
#for a in xrange(0,10):prelightDisplace(selected())

def spreadObjects(chunk=0,wrapLength=35.0,sizePop=40):
    boxSum=0.0
    lineSum=0.0
    prevWidth=0.0
    prevDepth=0.0
    maxDepth=0.0
    #wrapLength=35.0
    prevLineCount=0.0
    lineCount=0.0
    depthSum=0.0
    for counter,tx in enumerate(chunk):
                if(type(tx)==nt.Transform):
                        #print tx,
                        ea=tx.getShape()
                        #print tx
                        #print tx.getShape()
                        box=ea.boundingBox()
                        #print tx.sz.get()
                        #tx.rotate.set([0,0,0])
                        #print box.width()*tx.sz.get()
                        thisWidth=box.width()*tx.sx.get()
                        thisDepth=box.depth()*tx.sz.get()
                        boxWidth=.5*thisWidth+.5*prevWidth
                        #totals box linear dimension
                        boxSum+=boxWidth
                        #resets at beginning of line
                        lineSum+=boxWidth
                        popCounter=math.floor(counter/sizePop)
                        #lineSum=boxSum%wrapLength
                        #line increment by dimension length
                        lineCount=math.floor(boxSum/wrapLength)
                        #print lineCount,
                        #or line increment by population length
                        lineCount=math.floor(counter/sizePop)
                        #print lineCount
```

```
                            #print lineSum,depthSum,"xy coord",
                            #if(lineSum<thisDepth): maxDepth=0
                            prevWidth=thisWidth
                            #print maxDepth,thisDepth
                            if(lineCount!=prevLineCount):
                                depthSum+=maxDepth
                                prevDepth=maxDepth
                                maxDepth=thisDepth
                                prevLineCount=lineCount
                                lineSum=0

                            #print "max depth=",maxDepth, "sum depth=", depthSum
                            #interesting!! stairstep pattern
                            #move(tx,[maxDepth*(boxSum/wrapLength),0,lineSum])
                            if(thisDepth>maxDepth):maxDepth=thisDepth
                            target=[lineSum,0,depthSum+maxDepth*.5]
                            #print "target",target
                            move(tx,target)
    print boxSum,maxDepth,lineCount,depthSum

#cubegen
#for a in xrange(0,12): polyCube(sx=a,sy=a,sz=a,d=a,w=a,h=a)
#spreadObjects(selected())
```

This page intentionally left almost entirely but not quite blank

fin.