

Branching Boogaloo: Botanical Adventures in Multi-Mediated Morphologies

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

Seeking to satisfy the
Computer Science Major
and the
Experimental Humanities Concentration



Diana Ruggiero
May 2016

Dedication

To Mom and Dad

(and Christina too I guess)

Acknowledgements

First, thank you to my family. I love you!

Second, thank you to:

My project advisor:

Keith O'Hara—Your straight talk and knowledge of useful resources and approaches was invaluable.

Faculty in Computer Science:

Sven Anderson and Becky Thomas

Faculty in Experimental Humanities:

Maria Cecire, Ben Coonley, Heidi Knoblauch, Gregory Moynahan, Olga Touloumi

Faculty in Other Awesome Areas of Knowledge:

Amir Barghi, Daniel Berthold, Rob Cioffi, Matt Deady, Amii Legendre, Tatyana Myoko von Pritwitz und Gaffron, David Shein, Ruth Zisman, Marina van Zuylen

Third, thank you to my friends. You are the kindest, smartest, most supportive, understanding and loving people in the universe and I am so so lucky to have you all in my life.

Fourth, thank you to my computers for being the absolute best toys and tools a girl could possibly want.

And finally, thank you to all the plants I took leaves off of. Good thing they grow back!

Table of Contents

Dedication	iii
Acknowledgements	iv
Abstract	1
Project Introduction	2
Part I: Looking At Leaves	6
Leaf Shape: Ancient Observations and Modern Terminology.....	8
Leaf Function: The Puzzle of Photosynthesis	20
Leaf Development: Hormones and Metaphysics	31
Patterns of Venation: Vascular Branching and the Unity of Phenomena.....	38
Intersection	51
Past Work in Formalized Morphology.....	52
Part II: FormaLeaf	80
The Language of L-systems: Grammars of Growth	81
Method: Approach, Algorithms, and Tools Used	94
Results and Discussion	140
Future Work	155
Project Conclusion	165
Bibliography	167
Appendix: Processing Code	171

In spring when, tired of restraining themselves, no longer able to hold back, they emit a flood, a vomit of green, they think they're breaking into a polyphonic canticle, bursting out of themselves, reaching out to, embracing, all of nature; in fact they're merely producing thousands of copies of the same note, the same word, the same leaf.

-Francis Ponge, *Flora and Fauna*

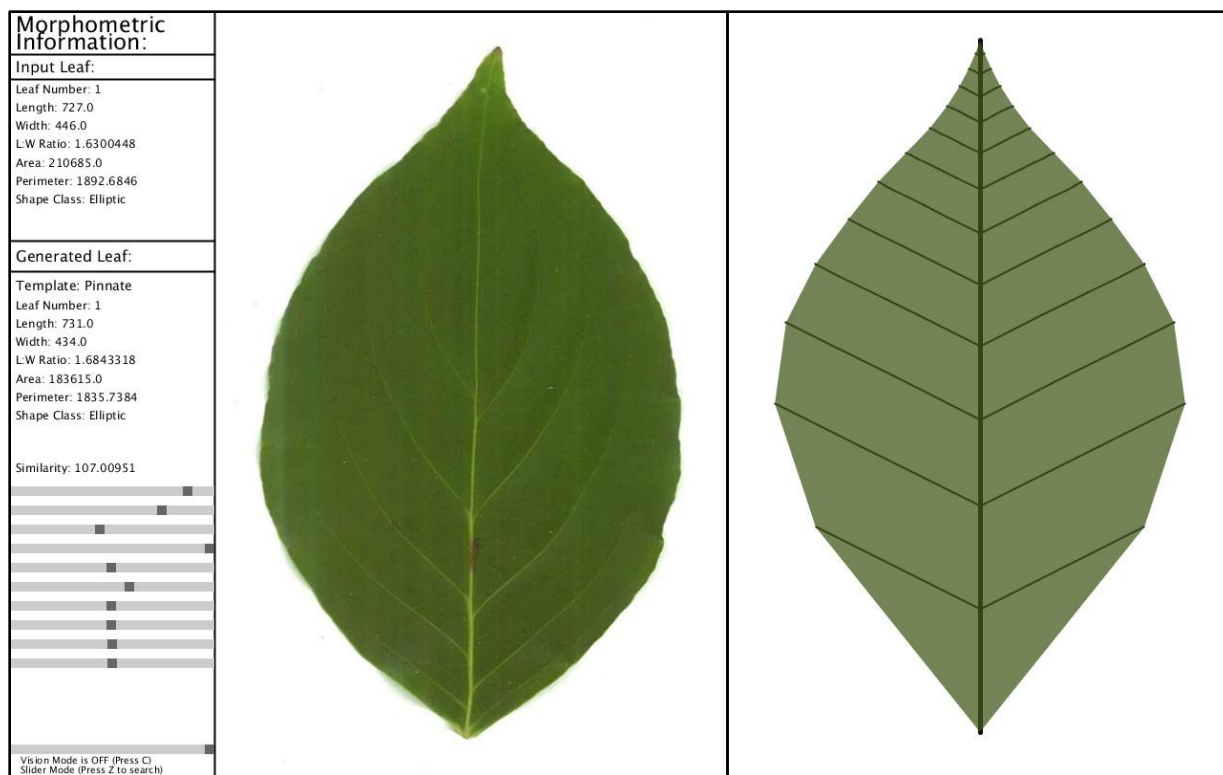
Abstract

FormaLeaf is a software interface for exploring leaf morphology using parallel string rewriting grammars called L-systems. Scanned images of dicotyledonous angiosperm leaves removed from plants around Bard's campus are displayed on the left and analyzed using the computer vision library OpenCV. Morphometrical information and terminological labels are reported in a side-panel. "Slider mode" allows the user to control the structural template and growth parameters of the generated L-system leaf displayed on the right. "Vision mode" shows the input and generated leaves as the computer 'sees' them. "Search mode" attempts to automatically produce a formally defined graphical representation of the input by evaluating the visual similarity of a generated pool of candidate leaves. The system seeks to derive a possible internal structural configuration for venation based purely off a visual analysis of external shape. The iterations of the generated L-system leaves when viewed in succession appear as a hypothetical development sequence. FormaLeaf was written in Processing.

Project Introduction

Motivations and Summary

This is a project about leaf morphology. In Part I, leaves are examined from a historico-scientific perspective in an attempt to understand them contextually. Next, the Intersection presents a historical overview of formalized morphology. Part II then first gives a description of a type of formal grammar called L-systems and then goes into detail about my own efforts towards building an interactive system which tries its best to automatically arrive at a plausible L-system representation of a leaf's internal venation structure by looking only at its outline shape. The result of these efforts has been a piece of software I've named FormaLeaf. It was written in Processing, a Java-based language which makes building interactive graphical systems streamlined and fun. FormaLeaf can hence be run on any computer with Processing (and the two required libraries) installed.



The software's name plays on its ability to both allow the user to explore leaf form in real-time by manipulating various template leaves using parameter sliders ('Form a' Leaf) while also being a nod to what underlies the computer-generated graphic on the right-hand side—a formal language representation of leaf form ('Formal' Leaf). As the ultimate goal was to obtain automated representation, in addition to the interactive Slider Mode the program also has a Search Mode. Here computer vision techniques analyze both the input leaf and a pool of continuously generated candidate leaves in an attempt to find an L-system leaf with a structural and dimensional configuration which matches the input. Because of the complexity of this problem, this hardly ever results in a leaf that actually resembles the one on the left. I consider it a success, however, if the generated leaf's schematic "template" (and hence its venation and lobation structure) matches that of the input leaf. While this automated search was the original aim of the project, it ended up as one of the least developed/functional pieces of the whole thing.

Why leaves? Put simply, a leaf makes a good microcosm. It's an interesting piece of the universe which happens to express many general properties—if you have to narrow your focus, may as well pick something that smells universal. Reflected in the variable finalities of its form, the function of its metabolism, and the process of its growth and branching is one possible image of the whole organic universe. The deeper our understanding of a leaf, the deeper our understanding of everything! Leaves are life in an expression not of the 'animal', and hence looking at specifics of plant form and functioning provides an interesting perspective of our own place. As an organ of a larger organism, a leaf is both part and whole—this makes it convenient to fit inside a scanner.

The unique significance of the leaf as a subject of inquiry is echoed by a number of writers. Biologist and author Steven Vogel writes of the leaf as a “biological everyman, an ordinary and ubiquitous living thing that provides the subject for an exploration of our immediate physical world” and selects it as the representative “protagonist” of his book about the biomechanics of life.¹ Numerous plant morphologists (among them Caspar Wolff and Johann Wolfgang von Goethe) conjectured that every plant organ is just a modified leaf by virtue of its apparent universality. Agnes Arber, author of the remarkable 1950 *Natural Philosophy of Plant Form* even states that “the flowering plant[...]offers innumerable ‘microcosmic’ aspects.”² For this project, I select the leaf.

Less pertinent but still of note is the ecological importance of leaves. This is not to suggest a project like this has any actual ecological application. If it does, I don’t know about it. Regardless, as the means by which most of the solar energy from the sun is introduced into the ecosystem, leaves are especially integral to the existence of life on Earth as we know it and thus (I believe) worth everyone’s time to think about. Writes Vogel:

What’s minimally needed to generate order are three items: (1) a source of energy and (2) a sink for energy, with the latter at a lower potential (cooler or lower down) than the source, and (3) some coupling system to draw on this energy flow.

For our earth, the sun provides the source, and the sink is outer space or, in immediate terms, the cold sky. What’s the coupling system? One system exceeds in importance by some vast factor all others put together. It’s photosynthesis, as done by green plants, algae, and some kinds of bacteria. Without photosynthesis (or some substitute), nothing like the present kind of complex, highly ordered life could exist. Leaves are *really, really* important.³

And of course, leaves are just as important as they are beautiful.

¹ Vogel, 2.

² Arber, 1.

³ Vogel, 16.

Some notes on sources: Any images scanned from books or obtained from online sources have been cited. Unless otherwise stated, the included photographs and images are my own. Quotations and information are cited in footnotes⁴ and there is a bibliography at the end of the project.

I titled this project “Branching Boogaloo: Botanical Adventures in Multi-Mediated Morphologies” in part due to my interest in media theory. My hope is that by engaging with a topic through a number of different disciplinary mediums a more complete understanding may be possible—if not more complete, then at least multi-faceted. This is partly what “Experimental Humanities” means to me. It also means allowing myself to take experimental risks, so if chunks of this project are rough around the edges or seem all over the place it’s because I’m out of my depth or otherwise ran out of time to make a section cohesive. But it doesn’t hurt to try—and in any case, I learned a lot.

Above all else, I hope you enjoy what you choose to read or see of my project!

⁴ Regarding footnotes, for all of Part I and for the Intersection footnotes with anything other than pure citational information are printed in green.

Part I: Looking At Leaves

Botanical History and Biological Architecture

Looking at Leaves

Botanical History and Biological Architecture

Considering the importance of leaves, it's remarkable how frequently they are overlooked. In the Northeast, outside of autumn peak and the subsequent clean-up leaves tend to blend in to the background of human activity. With the intent of first arriving at a general understanding of leaves, scientific information has been interleaved with facts and primary sources pertaining to some particularly interesting historical developments of the human understanding of leaves. The hope is that technical information which might otherwise appear boring or dry to some readers becomes enlivened within a human context. Modern research is also cited when appropriate and points relevant to aspects of the larger project at hand are described. We address in turn: leaf shape, leaf function, leaf development, and patterns of venation.

Leaf Shape: Ancient Observations and Modern Terminology

This section concerns the shapes of leaves and leaf parts and the names they've been given. Leaf terminology has developed in large part to aid in species identification.

The designated successor and junior colleague of Aristotle, Theophrastus is often referred to as the first dedicated botanist¹ or else as the “Father of Botany.” His study *Enquiry into Plants* describes with precision many aspects of plants, including the idiosyncrasies of certain species, seasonal behaviors, and the variable forms taken by different plant parts and organs. Section X of Book I is dedicated to leaves, where Theophrastus writes:

Leaves differ also in their shapes; some are round, as those of pear, some rather oblong, as those of the apple[...]²

Theophrastus has here observed the variable form of the *lamina*—the leaf blade. His comments extend further:

Again there are various other differences between leaves; some trees are broad-leaved, as vine fig and plane, some narrow-leaved, as olive pomegranate myrtle.³

Any human with uncompromised eyesight confronted by a leaf notices instantly its basic geometry, as shape is an exceedingly obvious visual property. It speaks to the clarity of shape as a property that two thousand-year-old descriptors remain intelligible and translatable. The words used by Theophrastus to describe different leaf shapes rebound through later texts and are echoed in modern terminology. The most salient point of these

¹ The Intersection begins with a brief reconsideration of the tendency to name Theophrastus as the origin-point of all botanical science—for this section, though, he gets his due. Also worth mentioning is that a number of pre-Socratic philosophers are credited by later sources as having theorized about plants (for example: Empedocles on plant sex, Anaxagoras on gas exchange) though it does not appear that they carried out any extended study.

² Theophrastus, I.X.5, 73.

³ Theophrastus, I.X.4, 71.

passages is the sheer variety of differences of shape—Theophrastus hardly attempts an exhaustive catalogue but instead simply points to a handful of examples.

In an effort to manage this variety, botanical texts demonstrate a move over time from heuristic names based off of resemblances towards quantitatively measured, well-defined labels. Heuristic labels based off shape resemblances are poetic but tend towards a potentially unwieldy nomenclature with lots of very specific terms for different forms. For example, in his 1751 publication *Philosophia Botanica*, the enormously influential taxonomist Carl Linnaeus lists 62 separate names for different leaf blade shapes.

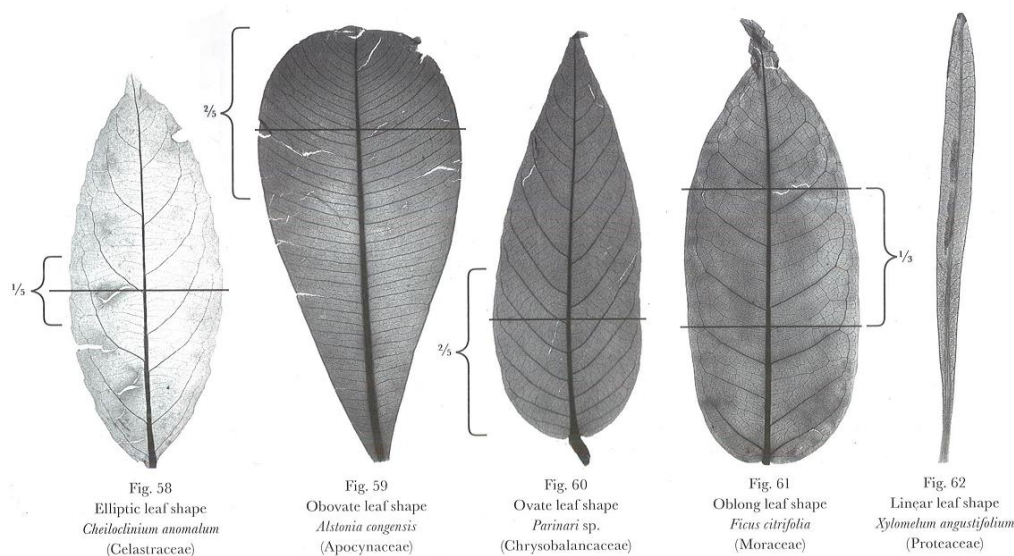


Figure 1: Left—Leaf shapes in Linnaeus' *Philosophia Botanica* (Tab. II)
Right—Linnaeus' list of Latin shape terms translated by Hugh Rose.

Older terminology (such as that presented by Linneaus above) is organized in such a way that leaf shape is presented as a whole, with the entire pictured form given its own name. However, there is little to suggest that these labels are meant to be mutually exclusive. These weren't efforts at *classifying* leaf shapes as much as they were just establishing collections of words used to talk about them. Later terminology becomes more precise in its organization concerning the parts of the leaf shape being named. Asa Gray's 1860 textbook makes a point to characterize the shapes of the general outline, the base, and the apex of the leaf in three separate sections, applying labels to the forms of different sections of the leaf. For example, he writes of a base shape: "*Hastate, or halberd-shaped, when such lobes at the base point outwards, giving the leaf the shape of the halberd of the olden time*"⁴ and of an apex shape: "*Retuse, with the rounded summit slightly indented, forming a very shallow notch.*"⁵ Gray also provides images of *just* leaf tips and bases alone in order to explain their forms. By contrast, "hastate" was just one of many other fully rendered leaves in Linneaus' collected list (#15, Rose translates it as "Spear-shaped").

⁴ Gray, 59.

⁵ Gray, 60.



**Figure 2: A few modern laminar shape descriptors.
(*Manual of Leaf Architecture*, 23)**

One thing the vision system in FormaLeaf does is apply a basic, overall shape label to the input leaf automatically by taking a precise computational measurement. The terms in Figure 2 are determined by the location of the widest point on the lamina, using the terminological method of naming shape through measurement as opposed to heuristics of resemblance. For example, if the widest part of the leaf falls in the bottom $2/5^{\text{th}}$ s of its lamina, it is labeled *ovate*. This term developed analogically (meaning “like an egg”) and only later was defined quantitatively.

Because computers like specifics, the labeling system of my project uses the quantitatively well-defined terminology as set forth in the 2009 *Manual of Leaf Architecture* published by Cornell University Press, which was itself based off of Hickey’s 1979 update of von Ettinghausen’s 1861 system. This manual was in part developed to assist in modern paleobotanical research efforts. Leaves are the most common fossilized remains of ancient plants,⁶ so careful labeling systems are important for understanding and organizing the

⁶ Ellis *et. al* (2009).

fossil record (and thus the evolutionary history) of angiosperms. Large-scale organization and labeling is generally more efficient when done by some objective criteria. Where Gray gave an image and description of a “hastate” base shape, the *Manual of Leaf Architecture* gives a photograph, a description, and a range of base lobe angles (90° - 125°) to which it could apply.

A terminological division of a leaf into two main parts is usually done by separating it into the lamina and the *petiole*, also called leaf-stalk or stem.

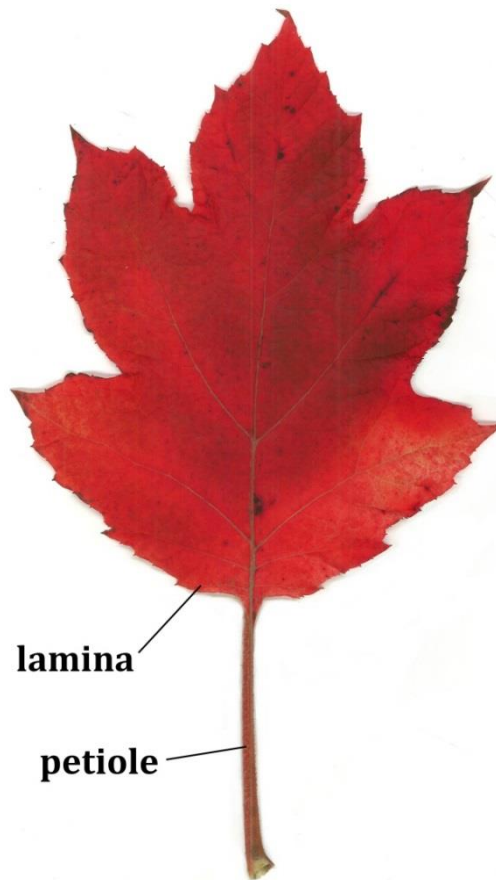


Figure 3: Parts of a leaf. The site of connection between the petiole and lamina is called the *insertion point*.

The petiole connects the leaf blade to the rest of the plant, and the style of this connection is yet another place where leaves express different spatial characteristics. As Theophrastus writes of the variable attachment styles of leaves,

[...]the means by which they are attached may be a leaf-stalk, or they may be attached directly; and there may be several leaves attached by the same leaf-stalk.⁷ Leaves attached by a leaf-stalk are today known as *petiolate*, while those attached directly (and without a petiole at all) are called *sessile*. When Theophrastus mentions that “there may be several leaves attached by the same leaf-stalk,” he’s pointing out the existence of *compound* leaves, which have many individual leaflets on one petiole.

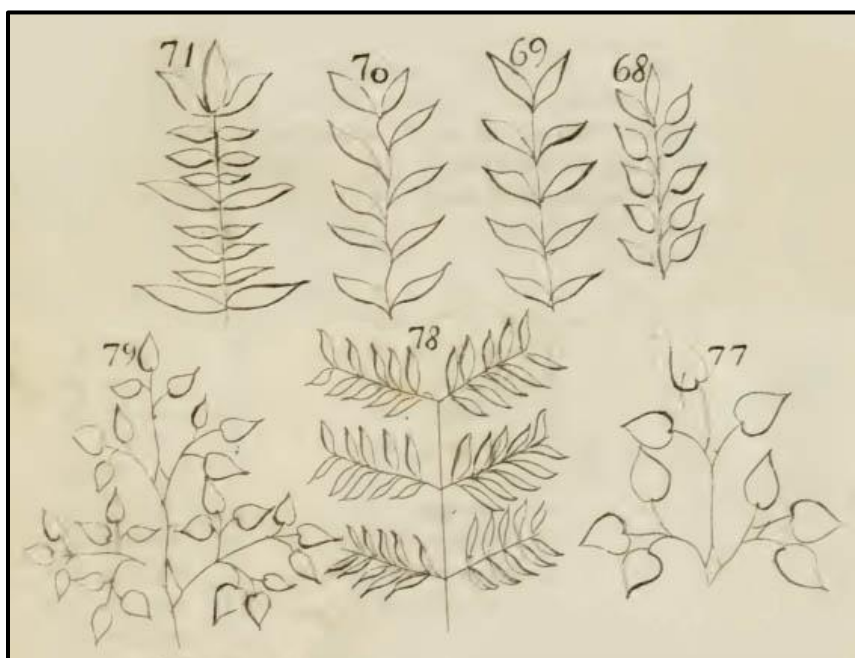


Figure 4: Drawings of compound leaves in Linnaeus' *Philosophia Botanica* (Tab. II, 290). The stems which attach the leaflets are *petiolules*.

Contrary to compound leaves are *simple* leaves, consisting of just a single laminar area—like the leaf in Figure 3. Though it is hard to tell once they've been removed, I believe most of the leaves in my sample set to be simple petiolate leaves. It is very likely that some are actually leaflets from a compound leaf, but in the end it doesn't make any difference since

⁷ Theophrastus, I.X.8, 77.

the system seeks to reproduce whatever lamina it's been presented with—hence it models what could either be a simple leaf or a compound leaflet. The system pays no regard to the status of the petiole as it is manually removed from the scanned picture during the image preparation phase.

The myriad causes behind the enormous variety of leaf shape is a huge, largely unsolved puzzle which crosses the boundaries of every scientific discipline. A 2011 review of possible theories of leaf shape significance by Adrienne Nicotra *et. al* summarizes as follows:

The theories about leaf shape are many, and not mutually exclusive: thermoregulation of leaves especially in arid and hot environments, hydraulic constraints, patterns of leaf expansion in deciduous species, mechanical constraints, adaptations to avoid herbivory, adaptations to optimize light interception and, given that leaves are hypothesized to be developmental homologues of floral organs, and it has even been suggested that leaf shape reflects the effects of selection on flower form. Finally, there is the chance that leaf shape variation has little functional or adaptive significance and instead reflects random variation within the context of phylogenetic history. However, given the importance of the leaf we believe the latter option rather unlikely.⁸

Clearly, there are many factors which affect leaf shape. Everything about the surrounding environment appears to matter—it becomes especially complicated because leaves of the same species frequently assume different forms depending on their specific climatic conditions. Keep in mind also that “leaves are hypothesized to be developmental homologues of floral organs,” which will come up again in the section about leaf development.

Before moving on to leaf function, a shape-based clarification is in order. As far as I can tell, the leaves under computational examination in this project are all from flowering plants, or *angiosperms*. This distinction is in contrast to *gymnosperms*, which include

⁸ Nicotra *et. al*, (2011), 536.

conifers and Ginkgo trees (among other plant groups). Both angiosperms and gymnosperms fall under the category of vascular plants, which are those species which transport water and nutrients through special conducting tissues—xylem and phloem. Angiosperms and gymnosperms gain their respective names from their different styles of reproduction. With *gymno* meaning “naked” and *angio* meaning “vessel,” the terms indicate whether the seed (*sperm*) of the plant is either exposed and hanging out on the leaves or a cone or is otherwise enclosed in an ovary which then develops into a fruit. Pictured are some examples of gymnosperm vs. angiosperm leaves.



Figure 5: *Left*—Gymnosperm leaves. Scale, needle, and the planar Ginkgo.
Right—Angiosperm leaves.

My project's vision system deals with angiosperms instead of gymnosperms because of their respective shapes. Conifers tend to have thin needle or scale-like leaves as opposed to the broad, planar leaves of most angiosperms. A vision system to devise an L-system representation of conifer leaves would be possible but would require work in a different direction due to this basic structural difference—at the very least, it would require a different way of using (if used at all) the polygonal interpretation of the L-systems. Ginkgo leaves are planar but their unique fan shape is strange for other reasons, as *Ginkgo biloba* is one of those peculiar ancient fossil species that still happens to be around. Another reason I personally focused on angiosperms is that it's an angiosperm leaf that comes into my mind when I hear the word "leaf." It didn't even occur to me at the outset of my sample collection that pine needles too may bear the moniker—despite the difference in form they are scientifically considered to be simple, single-veined leaves because of their function. Angiosperm leaves are, in comparison to every other leaf-like structure in evolutionary history, enormously complex in their often reticulate venation patterns. It has been proposed that this complexity of venation is also linked to the massive varieties of angiosperm leaf shapes—write Nicotra et. al: "Because of their much greater transpirational capacities[...]flowering plants have far greater leeway than other plants regarding the size and shape of their leaves."⁹ Perhaps due in part to this metabolic leeway, angiosperms have had enormous ecological success on planet Earth.

⁹ Nicotra et. al (2011), 542.

Leaves also take on highly modified forms to fulfill specific anatomical roles unique to certain species—for example, fly traps, bug-catching pitchers, climbing tendrils, spathes, and spines, among other things. This potential for performing structural and responsive functions outside of just the nutritive function of photosynthesis points to the leaf's developmental versatility as a plant organ.



Figure 6: The spathe of the jack-in-the-pulpit (*Arisaema triphyllum*) is a modified leaf which surrounds the spadix.

In conclusion, there are lots of leaf shapes and leaf terminology has become more precise over time. Aside from being useful for identification purposes, the terminology surrounding leaf form becomes a lexicon of what is noticed. The move from many heuristic labels for different leaf shapes towards a more mathematically precise and partitionally specific system of terminology follows a general scientific trend towards both numeric exactness and a greater standardization of terms describing smaller and smaller pieces of whatever is under observation. Terminological systems like the one found in *Manual of*

Leaf Architecture are precise in their measurements in the hopes of increasing standardization across research and communicability across researchers. Terminology is also invaluable for helping us to pay conscious attention to pieces and distinctions we would otherwise ignore. On the flip side, the assumption that everything's already been labeled in the best possible way perhaps makes it harder to see what has yet to be named, and also that an improved conceptual scheme for the relation between existing labels might exist.

The next section concerns the puzzle of leaf function and the role comparison in understanding what they do.

Leaf Function: The Puzzle of Photosynthesis

While the varieties of leaf shape are quite apparent to the human observer, their function is far less immediately obvious. The true biological purpose of the leaf was unknown even after the basic functions of other plant parts (such as the roots and fruits) were for the most part understood. Hence early botanists are understandably mistaken about what leaves are actually doing, although they are generally aware that it has something to do with water and sunlight. Nicolaus of Damascus, supposed tutor of Antony and Cleopatra's children, writes in his 1st century B.C. botanical treatise *On Plants*¹⁰ that leaves

have no other purpose except the attraction of moisture and to serve as a protective covering for the fruit from the excessive heat of the sun. At the same time leaves are not so essential as fruit[...]¹¹

Not quite—but he is onto something. Nicolaus recognized the sun's role in the upwards movement of moisture as well as its necessity for plant growth:

When the sun strikes it and causes the moisture therein to move, it heats up the spot by the movement which arises[...]when the heat of the sun begins to scatter the particles of water, the sun draws the particles of moisture upwards[...]¹²

Leaves are the sites of diffusive moisture movement, which give plants, as Nicolaus says, “a power of attraction which draws the moisture from the earth.”¹³ *Transpiration* is the process by which water moves up from the soil, through the plant, and out into the atmosphere. The evaporation occurs at the leaves, where the drier air outside the leaf

¹⁰ Commonly misattributed to Aristotle, who was far more a zoologist.

¹¹ *On Plants*, II.VII 217.

¹² *On Plants* II.VII, pg. 215.

¹³ *On Plants*, II.I, 187.

causes a diffusion gradient.¹⁴ The sun's heat further hastens evaporation by heating both the leaf itself and by making the surrounding air less humid.

Understanding Nicolaus's conception of leaf function requires looking a little closer at how he saw plants in general—he describes plant growth as the coalescence of moisture which is then refined by the heat of the sun and the expansion of the surrounding air:

For when the juices are compressed, their nature grows hot and hurries on to the ripening stage, and so branches will take shape and leaves grow[...]¹⁵

The continuation of this process—its “proper end” being the production of fruit—he refers to as “ripening,” which he also points as the cause of differentiation in animal parts:

A third form of ripening takes place in the animal; for this form of ripening only occurs through the division of the limbs and the natural differences of one part from another.¹⁶

Hence the developmental process which shapes the plant is seen as akin to what causes differentiation of animal parts. This is a common theme running through morphological thought. What is less clear than visual resemblance is whether the function of the parts they develop are similar as well. Does homologous form imply analogous function?

Nicolaus's assertion that the leaves serve as a covering for fruit demonstrates an awareness for how leaves are structured so as to intercept sunlight. Agnes Arber furthermore suggests that the shade theory was “a natural reaction to the southern brilliance of the Mediterranean climate.”¹⁷ Others suggest that it came out the fact that Mediterranean plants tend not to have very large leaves. Curiously, while contemporary research does not suggest that leaves are fulfilling any important protective function for

¹⁴ MacAdam, 130.

¹⁵ On Plants, II.VIII, 223.

¹⁶ On Plants, II.VIII, 221.

¹⁷ Arber, 28.

fruit, plants do have their own kind of “sunscreen”. For example, the protein receptor (UV-B resistance 8) initiates a stress response as a protective measure against excessive radiation, which is especially harmful to genetic information.

Botanists for most of human history understood the sun as providing the heat which raises the moisture of the plant into its formation. They consistently recognize that the sun is an integral part of growth and often observe that plants which get less sunlight do not grow as much. Theophrastus was aware of the *heliotropism* of leaves when he wrote that “Most leaves turn towards the sun[...]¹⁸ However, the nutritive importance of the solar rays was not known and they instead saw all nutrition as coming from the Earth.

Early botany understandably located the nutritive function of plants in the roots, with Nicolaus of Damascus calling them “the intermediary between the plant and its food,[...]the source of life.”¹⁹ Nicolaus probably got this from Aristotle, who wrote in his *On the Parts of Animals* that plant roots are analogous to the animal mouth. Plants were seen as being inverted life-forms with their heads stuck down in the earth:

[...]they take in their nourishment from below[...] the under parts come in them to be above, and the upper parts to be below.²⁰

Like much of Aristotle, this stuck around due to his enormous philosophical and scientific influence. He didn’t write much about plants (he left that up to Theophrastus), but what he did write was incorrect in its conclusions arrived at through zoomorphic comparison. All the way in 1682, John Ray wrote that “plant seize and drink all their nutrition through the roots, just as animals seize and drink all their nutrition with their mouths.”²¹ Analogies

¹⁸ Theophrastus, I.X.2, 69.

¹⁹ On Plants, I.IV, 167.

²⁰ Aristotle, Book IV, VII.

²¹ Ray, 47.

between plant and animal anatomy throughout botanical history are enormously common due in part to a deeply rooted Aristotelianism. Though Aristotle was wrong about plants, it is important to grasp that puzzling through botanical functioning through conceptual analogies does not necessarily lead to misunderstanding.

While roots *are* absorbing water and nutrients (like nitrogen) from the soil, if one had to point to the main “food” of plants it would be the glucose they synthesize during photosynthesis. The primary nutritive function of leaves for a long time escaped botanists—the leaf as an organ is assumed to be of lesser importance throughout the works of Albertus Magnus in the 13th century and Andrea Cesalpino in the 16th, though these writers at least make the effort to describe the varieties of leaf shape. The misunderstanding of the relationship between leaves and the sun persists even into the 18th century, where even Linnaeus maintains that they exist to provide shade.²² Botanist and historian Julius von Sachs credits the late 18th century plant physiologist and anatomist Marcello Malpighi as one of the first to suspect that leaves are a primary nutritive organ. Malpighi arrived at this theory by a comparison of their vascular tissue to the blood of animals²³—that is, that vascular fibers are performing resource delivery of important nutrients. Vascular branching as a universal phenomenon will come up a few times throughout this project. Malpighi’s is a situation where understanding of plant function was furthered through analogical reasoning based off homology of form.

Though Malpighi finally pointed to the leaf as nutritive, understanding the role of the sun in puzzle was essentially impossible until microscopes, which allowed researchers to see chlorophyll. Photosynthesis wasn’t understood chemically until C.B van Niel

²² Linnaeus, 66.

²³ Von Sachs, 457.

formulated the reaction in 1931. It's important to point out that a huge reason leaf function took so long to grasp is because all of the clues to what's going on are so small.

Hence, leaf functioning became clearer once botanists started looking at plants at the cellular level. Though Antoine van Leeuwenhoek's technical advances in microscopy and Robert Hooke's discovery and coining of the term "cell" happened in the late 17th century, it wasn't until the mid-19th century that the cell was studied as the principle individual unit of biological life. In particular, Matthias Jakob Schleiden focused his microscopic cellular studies on plants.

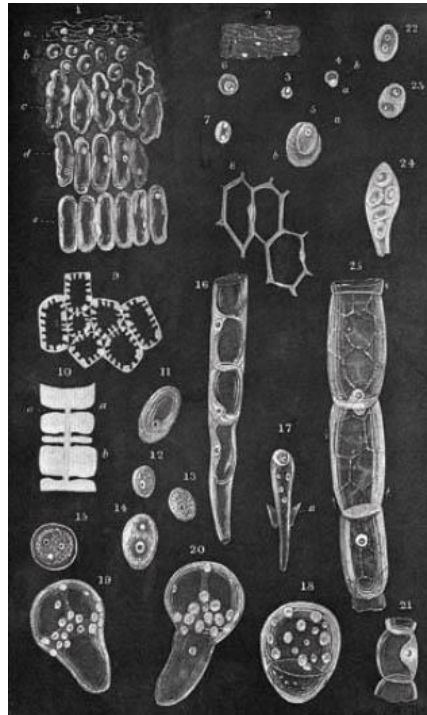


Figure 7: Schleiden's observations of cells.
(Contributions to Phytogenesis, Plate 1)

Without getting into the chemical details of photosynthesis, recall that it mainly takes place within the *chloroplasts*, organelles within the cell which themselves contain stacks of disk-shaped *thylakoids*. Chloroplasts are thought to have come from bacterial micro-organisms which were taken in by eukaryotic cells as an evolutionary adaptation (a

process known as *endosymbiosis*). Credited with making the functional connection between these organelles and light intensity is Julius von Sachs, whose book “History of Botany (1530-1860)” was an indispensable resource for Part I of this project. I had been reading this book for a long time in a search for pioneers of photosynthetic understanding, entirely unaware that the person I was perhaps looking for had written it.

Analogy at the cellular scale assumes a different form. When I was taught about plant cells for the first time in 7th grade (I recount this as it appears to be a common way of conceptualizing it) the whole cell was explained as operating something like a city. The chloroplasts are in this context acting like solar power plants. The city analogy highlights the cell’s productive capacity while also making it simpler to conceive of it as a self-contained unit of life.

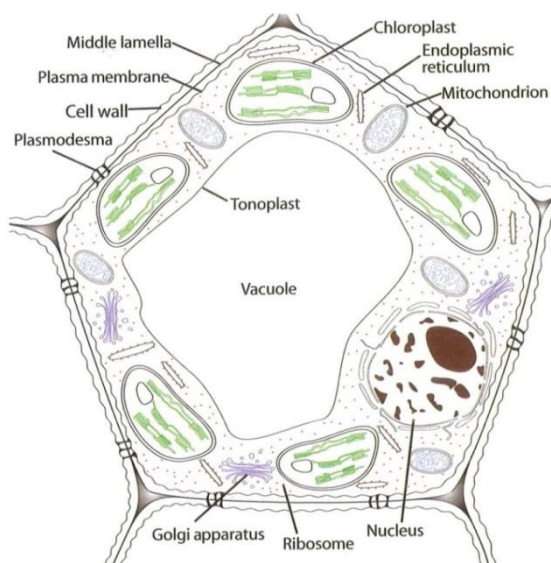
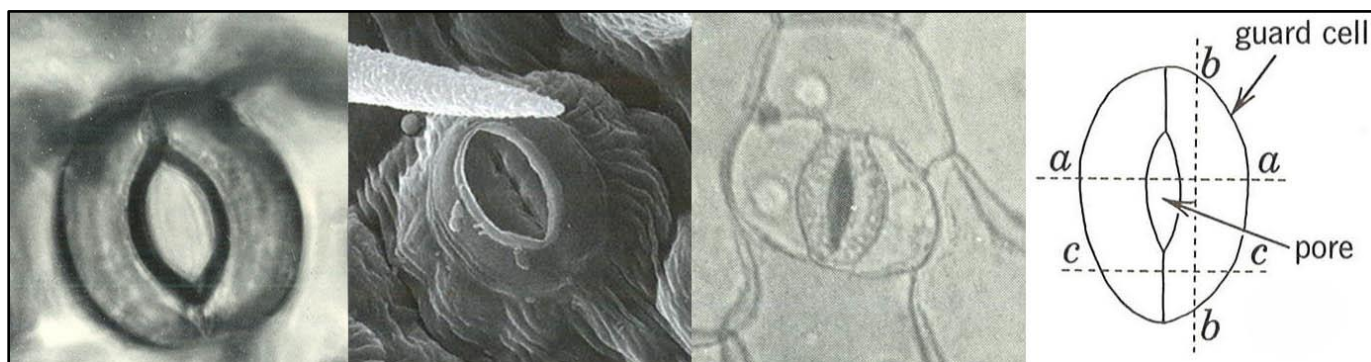


Figure 8: Plant cell diagram.
(MacAdams, *Structure & Function of Plants*, 2)

While cell diagrams like the one in Figure 8 are helpful in understanding the kinds of organelles a microscopic observer can expect to find in a plant cell, it also has the unfortunate potential to make us think of all cells as looking like a perfect regular pentagon.

Katherine Esau writes that *parenchyma* cells (the cells making up the inside of a leaf) have an average of 14 faces but are best approximately imagined as being shaped like a rhombic dodecahedron.²⁴ However, this is just for parenchyma tissue—there lots of different kinds of cells and they assume different forms depending on their function. This is also perhaps something the city analogy misses—that is, the *shape* of the cell really matters for what part it plays in the larger organism.

The shape of individual cells contributes to their functioning. This is especially interesting in vascular plants in which the entire structure of the plant is held rigid by osmotic turgor pressure within the vacuoles. R. O. Knight calls the plant cell an “osmotic unit.”²⁵ As an example of how turgor pressure (and thus the movement of water) contributes to cell shape and functioning, we look to a specific type of cell which performs a unique function entirely based in the adjustment of its internal turgor pressure.



**Figure 9: Assorted stomatal imagery.
(1.Knight 2.Vogel 3.Knight 4.Esau)**

Consider as an example of osmotic mechanism the *guard cells*, found in the leaf epidermis in pairs around a pore. Together, the guard cells and the pores make up the many *stomata*, the sites of gas exchange in the leaf. By diffusion, carbon dioxide goes in;

²⁴ Esau, 189.

²⁵ Knight, 24.

water vapor and oxygen go out. It is by turgor pressure that the guard cells make the stomata open and close. Guard cells at full pressure mean an open stoma.²⁶

Concerning gas exchange in plants, botanists knew about it for a while but the chemical details took longer. Linnaeus reported that leaves “transpire and draw the air.” Even earlier than this is Nicolaus’s reference that “Anaxagoras maintained that plants do breathe.”²⁷ Still, plants aren’t breathing the same way animals are—it’s really the opposite, with carbon dioxide coming in and oxygen and water vapor going out. Who better to grasp this than the man who himself first discovered oxygen? While Joseph Priestley was wrong about oxygen’s role in combustion,²⁸ he is rather on point when it comes to plant respiration. After performing some experiments which involved placing mice within oxygen-poor (“noxious”) enclosures both with and without growing sprigs of mint, he discovered that the mice survived much longer when the plant was present. “I presently had the most indisputable proof of the restoration of putrid air by vegetation,”²⁹ he reports in his 1774 *Experiments and Observations on Different Kinds of Air*. He concludes that plant respiration is a reversal of animal respiration:

This observation led me to conclude, that plants, instead of affecting the air in the same manner with animal respiration, reverse the effects of breathing, and tend to keep the atmosphere wholesome, when it is become noxious, in consequence of animals either living and breathing, or dying and petrifying in it.³⁰

Noticing that the open, common air does not become so noxious through animal respiration or the burning of candles that candles can no longer be lit, Priestley supposes that

²⁶ “Stoma” is from the Greek word for mouth—as sites of gas exchange on the principle nutritive organ stomata surely make better analogical mouths than roots do!

²⁷ On Plants, I.I pg.151.

²⁸ Priestley believed in the long-standing theory that combustible bodies contained ‘phlogiston’ which was released when they were burned, calling oxygen “dephlogisticated air.” Antoine Lavoisier proved otherwise: combustion is the *addition* of oxygen.

²⁹ Priestley, 88.

³⁰ Priestley, 87.

“[...]the growing vegetables, with which the surface of the earth is overspread, may, for any thing that appears to the contrary, be a cause of the purification of the atmosphere sufficiently adequate to the effect.”³¹

And of course, plant carbon-fixation does purify the atmosphere. This is one reason why deforestation is inauspicious as fuck when it comes to keeping a planet healthy. In any case, the direction of plant respiration was crucial to understand before photosynthesis (which requires carbon dioxide) could be comprehended.

Information processing comes up with the coordination of stomatal opening and closing in order to control gas exchange. Peak *et. al* (2004) presents evidence that leaves are performing emergent, distributed computational processes to optimize their CO₂ uptake (which they need for photosynthesis) while minimizing water loss through evaporation. They found that the spatial and temporal statistics of stomatal dynamics closely resemble those of certain types of *cellular automata* (CA), which are discrete systems where the subsequent state of each “cell” is determined by its own current state and those of its local neighbors. In particular, Peak *et. al* figure that this would solve the puzzle of the “patchiness” of stomatal dynamics (or the tendency for stomata to become synchronized over extended patched areas), as this is common in distributed computation.

They write:

In summary, we have demonstrated that the dynamical properties of stomatal opening and closing on a leaf are essentially identical to those some CA that perform emergent, distributed computation. Our analyses are only a first step, of course, in connecting computation and plants[...] Evolution may have found an elegantly parsimonious computational technique in which input, output, and processing are all accomplished by using the same hardware.³²

³¹ Priestley, 269.

³² Peak *et. al* (2004), 921.

If true, then leaves are not only incredible solar panels but computers, too! Although, having just considered many instances in which plant functioning was not what it first appeared to be, I also wonder whether it is enough for a natural process to bear *resemblance* to computational processing to say that something is “performing computation.” Perhaps CA are just a very good “model” of stomatal dynamics. Is this another analogy which could help brighten the dark room of such distributed processes while at the same time placing blinders due to our definitions of computation? It’s impossible to tell—once you take the premise that “computation” exists outside of “computers” it seems hard not to arrive at the conclusion that the whole universe is a computational process.³³ It probably all depends on how one’s definition of computation relates to and conceives of informational representation.

In conclusion, when considering the alien strangeness of photosynthesis due to both its physical subtlety and the lack of a comparable function in animals, it isn’t at all surprising that it took a while to figure out what leaves were doing. We mentioned before that it was partly the influence of Aristotelianism that made animal comparisons so common in the naming and description of plant parts. Even Theophrastus was aware that this could pose problems to understanding. Writing of plant veins, he says

Muscles [fibre] and ‘veins’ have no special names in relation to plants, but, because of the resemblance, borrow the names of the corresponding parts of animals. It may be however that, not only these things, but the world of plants generally, exhibits also other differences as compared with animals: for, as we have said, the world of plants is manifold.³⁴

³³ Argued by digital physicists like Fredkin, Wolfram, Zuse, etc. This position tends to end up with a view of the universe as being discrete.

³⁴ Theophrastus, I.II.3, 19.

Similar to how the names of leaf shapes were based on their resemblances to other forms found in daily human life (such as spears and eggs), the names of plant anatomical parts are given for their resemblance to “the corresponding parts of animals.” However, Theophrastus reminds us that while names tend to arise from homologous forms, this does not always imply analogous function. He saw a manifold, separate uniqueness to the whole botanical realm whereas his colleague Aristotle only really wrote of plants as upside-down animals.

Still, while certain comparisons to animal form and functioning has caused some confusion concerning unique botanical functions and hence obscured for a long time the importance of leaves, comparison also been the primary means by which uncharted biological territory is first mapped. Theophrastus continues:

However, since it is by the help of the better known that we must pursue the unknown, and better known are the things which are larger and plainer to our senses, it is clear that it is right to speak of these things in the way indicated: for then in dealing with the less known things we shall be making these better known things our standard, and shall ask how far and in what manner comparison is possible in each case.³⁵

Theophrastus reminds us that understanding the world from the frame of our own position is the only way to proceed towards the unknown—we simply must do so carefully, always questioning if our “comparison is possible in each case.” Here is a lucid call to pair the open human mind with critical thinking.

³⁵ Theophrastus, I.II.3, 19.

Leaf Development: Hormones and Metaphysics

As the 2009 textbook *Plant Biology* says, “The leaf has a limited potential for growth—in other words, it is a determinate organ.”³⁶ Francis Ponge writes more poetically: “Vegetable time resolves into vegetable space, the space plants gradually occupy on a canvas forever preordained.”³⁷ Whether this determinacy is ultimately eternal or not who can say—either way, a leaf’s development follows its genetic programming. It is by virtue of both their environmental conditions and their genotypes that leaves unfold and expand into the massive variety of shapes we addressed earlier.

Having just come off a discussion of cell function, we look to a crucial developmental tissue: the *meristem*, named by the Swiss botanist Karl Wilhelm von Nägeli. Meristems are the sites of undifferentiated cells and hence the areas of new growth, with the most important above-ground meristem being the shoot-apical meristem (SAM). Dividing cells assume their identities early—hence we can imagine the meristem containing “unassigned” cells which may become any number of different plant organs, including the leaf. Like all plant organs, leaves begin at the meristem as *primordia*. Early leaf growth happens through cell division (primary morphogenesis) which ceases and gives way to growth due to cell expansion (secondary morphogenesis).³⁸ Subsequent foliar cell division and expansion in the different axial dimensions (the *lateral* axis being the width, the *proximo-distal* axis being the length) is controlled by complex combinations of activated genes of which we have smattered understanding. Generally, research in this area proceeds by the isolation of

³⁶ Smith *et. al* (2009), 335.

³⁷ Ponge, 73.

³⁸ Scarpella *et. al* (2010), 2.

specific genes in specific species. They are revealed as being somehow important when mutating them changes how the leaf develops. For example, Nicotra *et. al* write:

The leaf length : width ratio is regulated by polar-dependent cell expansion and cell proliferation/distribution. Several key genes for its regulation have been identified from *Arabidopsis*: ANGUSTIFOLIA (AN) and ROTUNDIFOLIA3 (ROT3) regulate the shape of cells [...] Loss-of function mutations of AN and AN3 result in narrower leaves[...] a vast array of genes is known to influence leaf area [...] As yet, we do not know whether the above patterns hold in other non-model species.³⁹

In other words, it's complicated. It is a colossal, enormous abstraction (often made in the papers of those that use genetic algorithms to evolve L-systems), but the symbols or parameters of an L-system grammar can be imagined as "genotypical" with the generated structure as the expressed "phenotype". Using sliders to mutate the parameters in the template leaves of this project changes expressed characteristics (such as the length : width ratio) by controlling the lengths and growth-rates of various structural vein segments.

If you had to give a one-word answer as to what makes leaves develop and grow, it might be "auxin," the most crucial plant growth hormone. Auxin is unique among phytohormones in that rather than diffusing passively throughout the plant it is actively "pumped" through cells in specific directions, a process known as "polar auxin transport," which is especially important for leaf vein formation. One way in which auxin is theorized to affect cell expansion is by causing a reaction which loosens the structures of the cell-wall carbohydrates, thereby allowing the cell to enlarge through turgor pressure.⁴⁰ Other diffusive phytohormones (such as gibberellic acid) also play a part in regulating development. As in animals, vascular tissue is crucial for hormone transport.

³⁹ Nicotra *et. al* (2011), 539.

⁴⁰ MacAdam, 216.

Having briefly addressed the biology of leaf development, we look to its metaphysics. As it has been conceived by writers, this is a different kind of “development”—it is more spatial than it is temporal although it assumes temporal significance with evolutionary theory. Abstracted leaf development as a subject of inquiry was studied earlier by anatomists like Marcello Malpighi but assumed a unique metaphysical form with the beginnings of German plant morphology. Johann Wolfgang von Goethe’s morphological import is demonstrated by him being the one credited with coining the term “morphology” in the first place.⁴¹ Known best for his prolific literary output (notably *Faust*, *The Sorrows of Young Werther*, and *Elective Affinities*, among other things), Goethe also wrote a number of scientific treatises on subjects ranging from a theory of light and color, animal anatomy, weather, and geology. In the spring of 1790 he published the *Metamorphosis of Plants*, a book presenting his observations of plants with a focus on comparing the forms of different plant organs. While certainly his most well-known botanical work, many of his views on plant morphology are also expressed in letters to friends, travel journals, and other separate treatises.

The scholarly milieu surrounding Goethe borders on obsessive and his scientific work and method are controversial. Some scientists and historians disparage his scientific efforts for being overly Romantic, unempirical, and often factually mistaken. C.W Wardlaw is especially critical, writing that the study of plant morphogenesis did not really begin until Schleiden “cast off the fetters of Goethe’s Theory of Metamorphosis.”⁴² Other

⁴¹ Commonly stated but possibly debatable. Goethe published “On Morphology” in 1817—a cursory Google N-gram search shows the word appearing a bit earlier in both the German and English corpuses; two cases see it applied to non-living things like coins and volcanoes. Regardless, it was Goethe who opened up the word and established it as an actual field of study.

⁴² Wardlaw, 1.

scientists and writers of phenomenology are enthusiastic about Goethe’s method, which holds intuitive perception as being a valid way of engaging with the world objectively. As is common with such a polarizing issue, the middle-ground is likely the most fertile. We here focus on those of his ideas derived from his observations of plants which are both shared by other plant morphologists and which anticipated later botanical discoveries and approaches.

This brings us to the idea of the leaf as the “universal” plant organ and appendage, a viewpoint which gained its first significant advancements with Caspar Wolff and Goethe. While Goethe may have been the first to call what he was investigating “morphology,” Caspar Wolff preceded him in practice by about 20 years. Wolff and Goethe share a view of the leaf as the universal organ of the whole plant, with both believing that all plant organs—the petals, the sepals, the calyx, the corolla—are “modified” leaves. Goethe writes during a trip to Italy,

While walking in the Public Gardens of Palermo, it came to me in a flash that in the organ of the plant which we are accustomed to call the *leaf* lies the true Proteus who can hide or reveal himself in all vegetal forms. From first to last, the plant is nothing but leaf, which is so inseparable from the future germ that one cannot think of one without the other.⁴³

For Goethe, the idea of the leaf as a developmental “Proteus” came out his observations that as you move up the stem, the organs/appendages resemble the leaf but in expanded and contracted form. His conception of “leaf” is more of an ideational concept than a literal leaf; hence it can be a universal unit.

What is meant by *modified* has to do with an abstract understanding of metamorphosis. Metamorphosis proceeds as a sequence changing up the plant shoot—it’s

⁴³ Goethe, *Italian Journey*, 366.

not something quite as temporal as the way we now imagine the word “metamorphosis.” Similar views of plant form come up in the morphologists Arber, Oken, and de Candolle—this will come up again with the discussion of the template leaf’s design in Part II.

The foliar nature of some plant organs has been supported by genetic discoveries.

Nicotra *et. al* write:

Recent genetic data identifies several genes that have roles in both leaf and flower form[...]genetic changes in both leaf length to width ratio and leaf size influence not only leaf proportions, but also that of the floral organs in *Arabidopsis*[...] some floral characters as well as other metabolic pathways may now be closely linked with regulation of leaf shape and size.⁴⁴

Hence floral organs are theorized to have evolved from leaves. Despite this, neither Wolff nor Goethe made the connection that their intuitions could have brought the constancy of species dogma under fire and thus it wasn’t until Darwin that evolutionary theory was brought to light. This is largely due to the abstract nature of their approaches; a physical/temporal justification was not seen as necessary to talk about “changes” and “transitions” between the forms of separate organs on the same plant.

If the leaf was ignored for too long in botanical science for a belief in the roots as the primary nutritive organ, Goethe is guilty of entirely ignoring the roots for a focus on the leaf—or, rather, only the plant organs found above-ground and visibly apparent to casual observation. Furthermore, his *Metamorphosis of Plants* looks not at all plants but really only at dicotyledonous annuals. These enormous exclusions (which have frustrated their share of botanists) are intimately related to his goal of abstraction. He conceived of an archetypal plant—the Urpflanze—which he seems to have believed to be an actual plant he could find in the world until Schiller talked some sense into him. Rudolf Magnus writes that “To

⁴⁴ Nicotra *et al.* (2011)

Goethe the archetypal plant had now become the scheme or, as he later called it, the *type* to which all plant form could be reduced by comparison. It is the *structural plan* all plants have in common.”⁴⁵

In his notion of the Urpflanze, Goethe anticipates what morphology would later become in the hands of scientific modelers. He writes:

The Primal Plant [Urpflanze] is going to be the strangest creature in the world, which Nature herself shall envy me. With this model and the key to it, it will be possible to go for ever inventing plants and know that their existence is logical; that is to say, if they do not actually exist, they could, for they are not the shadowy phantoms of vain imagination, but possess an inner necessity and truth. The same law will be applicable to all other living organisms.⁴⁶

A generative principle for inventing logical forms? Formal grammars are up to the job. The Urpflanze can be viewed as a formalized, abstracted model, albeit one with a decidedly Romantic bent. Peter Antonelli’s 1992 book review (in the *SIAM Review*) of Prusinkiewicz and Lindenmayer’s indispensable plant-modeling volume *The Algorithmic Beauty of Plants* even goes so far as to say that

200 years ago Goethe evidently had the “rules of syntax” on “universal plant grammar” which the followers of Lindenmayer and Prusinkiewicz now seek[...]

He then points out that L-systems run into issues because while good at modeling phenomena like branching, they have a harder time modeling biological nonlinearity.

Antonelli continues:

The chapter on fractals is a step in this direction. Such results bring us one step closer to what Goethe knew about plant universal grammar but could not tell us because his method was as much subjective as it was objective.⁴⁷

⁴⁵ Magnus, 73.

⁴⁶ Goethe, *Italian Journey*, 310.

⁴⁷ Antonelli, 143.

Interesting stuff, for sure. A history of formalized morphology of a more explicitly systematic/symbolic nature will be explored in detail in the Intersection—Goethe isn't included there as he wasn't big on mathematics, but he is a clear forerunner of such abstract, model building morphological approaches.

That C. W. Wardlaw—a botanist in the 1960's who specialized in plant morphology and morphogenesis—finds Goethe's work frustrating and unscientific suggests also that “morphology” does not necessarily denote the same thing. As always, there are innumerable ways to approach a topic and an abstracted approach to form is just one—Caspar Wolff was looking at leaf embryos under a microscope while Goethe was writing plant poems:

*The plant-child, like unto human kind—
Sends forth its rising shoot that gathers limb
To limb, itself repeating, recreating,
In infinite variety; 'tis plain
To see, each leaf elaborates the last—
Serrated margins, scalloped finger, spikes
That rested, webbed, within the nether organ—
At length attaining preordained fulfillment.
Oft the beholder marvels at the wealth
Of shape and structure shown in succulent surface—
The infinite freedom of the growing leaf.*
(From *The Metamorphosis of Plants* (Poem))

Patterns of Venation: Vascular Branching and the Unity of Phenomena

I was first attracted to individual leaves from a structural and infrastructural perspective. I am especially intrigued by the relationship of the venation patterns to overall leaf shape. Leaves present to our attention a bounded space spanned by a complex system of veins which serve the dual purpose of both resource delivery and structural support. Furthermore, venation patterns exhibit not only divergent branching but networked reconnection as well. We now take a closer look at different patterns of venation, the relationship between venation and leaf shape, and some hypotheses as to how they are formed. Following this, we conclude Part I with the recognition of branching as a common pattern in the world.

Following the efforts of Andrea Cesalpino in the 16th century to categorize plants by the characteristics of their fruits and seeds specifically,⁴⁸ botany as a science (when it wasn't absorbed in pharmacological pursuits and the authoring of herbals) largely focused on classificatory systemization based on comparative anatomy. These systems tended towards artificial distinctions which had the effect of grouping together species which had little natural affinity outside of the selected organizational principle. Systematic botany came to a head with Carl Linnaeus, who from the 1730s through the 1750s devised and applied a clean artificial classification/naming scheme based entirely off of the number of sex organs on the plant.⁴⁹ Linnaeus was aware of the need for discovering groupings based on natural affinities as well and hence viewed his artificial sexual scheme as a practical tool for plant description and naming. Before this, however, John Ray in 1682 devised part of

⁴⁸ Von Sachs, 53.

⁴⁹ For an entire epic poem about anthropomorphized plant sexuality inspired by the work of Linnaeus, check out Erasmus Darwin's (grandfather of Charles) *The Loves of the Plants*. It's surprisingly boring considering the subject matter, but of course it stirred up controversy in the 1790s.

his own organizational system along an important classificatory line that turned out to be more than just an artificial distinction—a particular property on which he focused indicated an actual evolutionary divergence. Referring to the later enormous success of Linnaeus’ taxonomy, Ray biographer Charles Raven states that “it could easily be argued that Ray in fact laid down lines of classification more in accord with genuinely scientific and evolutionary principles than those of his illustrious successor.”⁵⁰ This is true to the extent that Ray recognized the classificatory importance of a property that was partly along natural lines—but again, Linnaeus was well aware of his sex system’s artifice, and Ray’s scheme itself was otherwise largely (and knowingly) artificial, being based mostly off of plant form. His most general division places a given plant into one of four categories: Trees, bushes, shrubs, and herbs.⁵¹ What then was the *natural* property Ray saw, and what does it have to do with patterns of venation?

The distinction John Ray drew came from his careful examinations of the early growth of embryonic leaves—what his contemporary Marcello Malpighi termed “cotyledons.” In his *A New Classification of Plants*, Ray writes that

In some kinds of seeds, the seminal plant does not consist of double seed-leaves, a little root and a bud: but either consists of a stalk alone without leaves, or a leafy stalk, or a single leaf without a stalk[...]A general distinction of plants is able to be deduced from this division of seeds, and this distinction, in my judgment, is the first and best by far: between those that have a double-leaved or double-lobed seminal plants, and those which have a seminal plant analogous to the adult plant.⁵²

That is, some plants have double-cotyledons while others have just one. This is the difference between dicots and monocots, which vary not only in their embryonic structure but also in their leaf venation. Goethe wrote later that “we may infer that the point where

⁵⁰ Raven, 200.

⁵¹ Ray, 53.

⁵² Ray, 42.

the cotyledons are attached is the first true node of the plant.”⁵³ We might imagine the cotyledon as the very start of branching.

As always, the modern scientific picture makes the monocot/dicot dichotomy more complicated. Calling something dicotyledonous is actually a classificatory anachronism and the phylogenetically correct term for most of them would now be “eudicot,” as not all dicotyledons have a common ancestor. Monocots, on the other hand, are monophyletic and thus still referred to as a group. However, we here retain the common term dicot as its general meaning is essentially clear. When it comes to leaf venation, monocots usually have parallel venation while dicots usually have a more complex (not to mention variable) venational pattern.

⁵³ Goethe, *Metamorphosis of Plants*, 12.

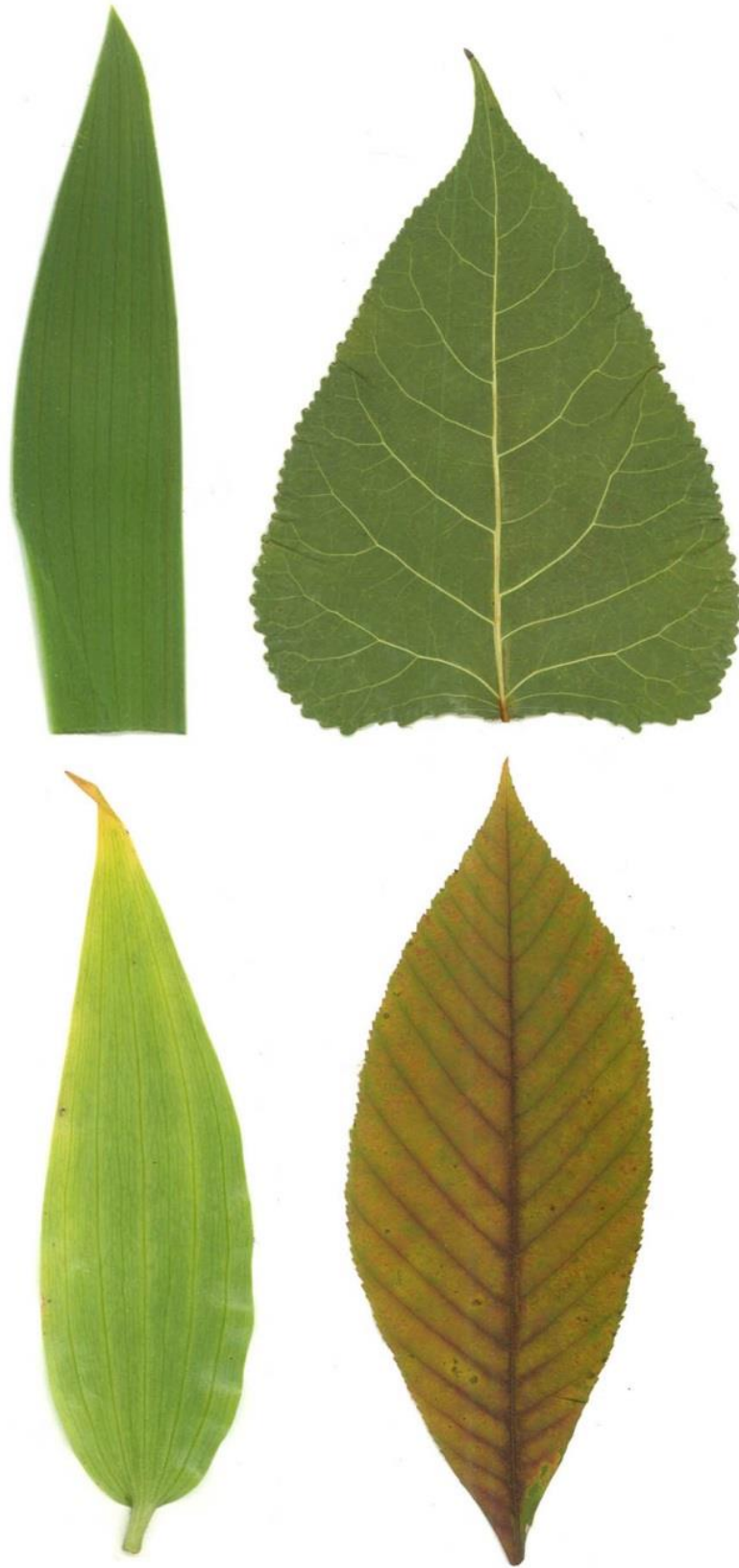


Figure 10: *Left*—Monocot leaves with parallel venation. Top sample cut from tulip. *Right*—Dicot leaves with reticulate venation.

John Ray's discovery provides a handy distinction between the parallel venation of monocots and the more complicated reticulate venation of dicots which holds in the vast majority of cases—Inamdar *et al.* (1983) give twelve species within seven different monocot families with venation more characteristic of dicots. For example, the jack-in-the-pulpit in Figure 6 is a monocot species with reticulate venation. Other than the exceptional cases, a reasonable way to deduct something about a plant's embryo and its possibly place in evolutionary classification is by looking at its vein features. Asa Gray writes:

So that a mere glance at the leaves of the tree or herb enables one to tell what the structure of the embryo is, and to refer the plant to one or the other of these two grand classes,—which is a great convenience. For generally when plants differ from each other in some one important respect, they differ correspondingly in other respects as well.⁵⁴

The system of my project models non-parallel venation as the collected samples are almost all (based purely on their venation) dicotyledonous.⁵⁵

Why would parallel venation be helpful ecologically? Monocots are commonly found close to the ground—many herbs and grasses, for example. Parallel venation can serve as a defense against herbivory because the veins run in the same direction as that of a hungry animal's bite.



Figure 11: Gory mess resulting from underestimation of yucca plant's anti-herbivory defenses.

⁵⁴ Gray, 56.

⁵⁵ The bottom left leaf in Figure 10 was perhaps the only parallel-veined leaf I happened to collect—excluding grass (a monocot), dicot leaves are more common.

The reticulate venation patterns of the dicots appear in many structural varieties. The most basic division is between leaves with *pinnate* venation (also 'feather-veined') and those with *palmate* venation (also 'digitate,' 'radiate').

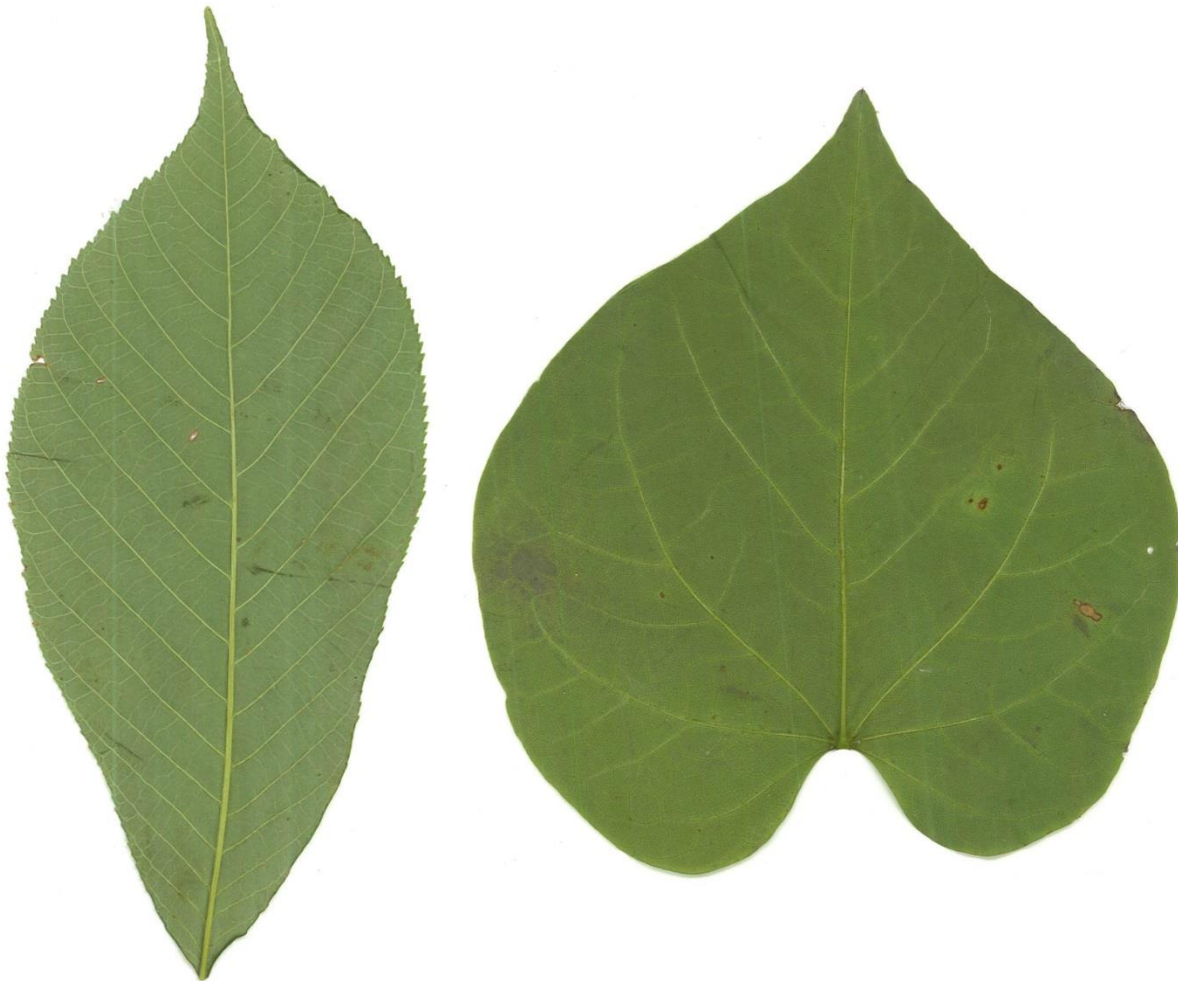


Figure 12: Left—Pinnate venation. Right—Palmate venation.

This distinction will become very important later with the different leaf venation templates. Pinnate venation means the lateral/secondary veins are coming off a single primary mid-rib. Palmate venation means there are multiple primary veins all radiating from the base of the leaf. Notice that the veins in both patterns have veins coming off of them as well, forming a complex network pattern.

Because the veins must both support and service all of the laminar area, the scheme of a leaf's venation is tied intimately to its shape. In his 1860 textbook, *First Lessons in Botany and Vegetable Physiology* American botanist (and friend of Darwin) Asa Gray makes a number of interesting observations about this relationship. Firstly, he points out that:

Since the general outline of leaves accords with the framework or skeleton, it is plain that *feather-veined* [pinnate] leaves will incline to elongated shapes, or at least will be longer than broad; while in *radiate-veined* [palmate] leaves more rounded forms are to be expected[...]Whether we consider the veins of the leaf to be adapted to the shape of the blade, or the green pulp to be moulded to the framework, is not very material.

Gray recognizes that external leaf shape is clearly related to the framework of the internal structure. His observation that pinnate leaves tend to have a greater length:width ratio when compared to palmate leaves held up in my own observations of my samples—barring one or two exceptions, palmate leaves had length:width ratios < 1 while the ratios of pinnate leaves were consistently greater than 1. However, while it may seem like this criteria should be enough to allow the system to label a leaf one way or the other, it doesn't do so simply because there is no guarantee. Instead the system was built to try both and sees which pattern gets the resulting L-system closer to the input shape.

Here Gray also argues that an understanding of precisely how leaf shape and venation are related developmentally is not necessary to being able to accurately observe a structural relationship—whether the veins follow the shape or the shape follows the veins “is not very material.” He continues:

Either way, the outline of each leaf corresponds with the mode of spreading, the extent, and the relative length of the veins. Thus, in oblong or elliptical leaves of the feather-veined sort, the principal veins are nearly equal in length; while in ovate and heart-shaped leaves, those below the middle are longest; and in leaves which widen upwards, the veins above the middle are longer than the others.⁵⁶

⁵⁶ Gray, 57.

Gray here describes how varying vein lengths in different parts of the lamina correspond to different outline shapes. Goethe writes similarly in *The Metamorphosis of Plants*:

But further development spreads inexorably from node to node through the leaf: the central rib lengthens, and the side ribs along it reach more or less to the edges. These various relationships between the ribs are the principal cause of the manifold leaf forms.⁵⁷

I find these to be especially interesting passages in light of how parametric L-systems can build the growth rates of different vein segments into their grammars. These venation parameters in turn affect the final outline shape because the veins serve as a structural framework for the leaf's polygonal geometry.

One of the most intriguing sub-phenomena of leaf venation is *anastomosis*, or when veins reconnect to form a network. Hence many systems of foliar venation have the organizational properties of both “trees” and “networks”—trees being hierarchical systems without reconnecting branches and networks being the systems in which anastomotic reconnections are common. Typically the lower order, larger, more primary veins will not exhibit this reconnection while higher order, smaller veins do, forming an intricate network visible only when one looks closely.

In most dicotyledonous leaves vein reconnection only occurs at the higher orders of venation (that is, smaller veins you must look close to see), but in some kinds of leaves even the easily visible 2nd-order (lateral) veins form anastomotic



⁵⁷ Goethe, *Metamorphosis of Plants*, 16.

loops. This is known as brochidodromous venation.

That being said, in our discussion of leaf venation we do well to heed the reminder of botanist P.B. Tomlinson, who writes in “Branching is a Process, Not a Concept” that

Existing terminology tends to be static i.e. concerned with the existing plant body as an end product, rather than with a concern for the dynamics of the branching process itself.⁵⁸

Tomlinson calls for the use of a flexible language. All the labels in the world won’t lead to understanding unless paired with awareness for developmental process. So what causes the formation of these branching patterns in leaves?

In the section on leaf development we addressed polar auxin transport—this is an essential part of vein formation. Many theories about the physics of vein formation point to elastic tensorial stresses—hence many comparisons are made between cracking patterns in drying mud and leaf venation.⁵⁹ Leaf vascular tissue comes in two varieties: the unidirectional xylem (transports water up from the roots), and the bidirectional phloem (distributes nutrients).

⁵⁸ Tomlinson (1987), 55.

⁵⁹ Couder (1999), in *Branching in Nature*.



Left—Regular reticulation pattern in a leaf. (*Manual of Leaf Architecture*, 87)
Right—Baltimore, aerial. (Bing Maps)



Left—“Electrical treeing,” studied by Georg Lichtenberg. Lightning is a natural Lichtenberg figure. (Wikimedia Commons)
Right—Lichtenberg writes also in an aphorism of the crystalline and arboresque growth of “ice ferns on the windowpane.”



Cracking patterns form by stress within a tensorial fields. The tensorial stress caused by an expanding leaf lamina is hypothesized to be one component of the physical explanation for leaf vein formation. *Left*—Bark. *Right*—Cracking mud.



Tivoli Bays demonstrating scaling of branching water patterns. Bottom picture is a zoomed-in piece of the middle of the top image. (Pictometry)

Leaf venation patterns bear a universal significance that extends beyond the mesophyll in which they are embedded. The preface to the 1999 *Branching in Nature*, a collection of scientific papers on various types of branching morphologies, states that the similarity of these patterns has been scientifically underexplored:

Hence, the idea of a unity behind amazingly different systems has remained latent for two thousand years. However, if we except the work of Scheuchzer, very little true progress was made on the scientific issues linked to branching morphogenesis until very recently. The specialization and segmentation of the scientific fields, which is one characteristic of modern science, has turned each branching pattern into a specific scientific object.⁶⁰

Perhaps the science has had less development⁶¹, but it isn't as though humans haven't been noticing this unity. The morphological similarity of branching phenomena has been poetically expressed not only in the human names for things but also in the keen observations and metaphors of writers across time. At the conclusion of the first year of his lifestyle experiment at Walden Pond, Thoreau took notice of the rivulet patterns formed in sand during the spring thaw:

You find thus in the very sands an anticipation of the vegetable life. No wonder that the earth expresses itself outwardly in leaves, it so labors with the idea inwardly. The atoms have already learned this law, and are pregnant by it. The overhanging leaf sees here its prototype.[...]The whole tree itself is but one leaf, and rivers are still vaster leaves whose pulp is intervening earth, and towns and cities are the ova of insects in their axils.⁶²

In the venation patterns of a leaf we see “something” which permeates the inorganic, organic, and human universes. If the branching trees and anastomotic networks of our data structures count too, it extends, perhaps, even into the realm of the

⁶⁰ Fleury, v.

⁶¹ Or it did in 1999, anyway.

⁶² Thoreau, 547.

informational. As the editors of *Branching in Nature* write, “There exist universal paradigms.”⁶³ The branching patterns inside my skin resemble those outside my window.

Concluding Remarks

The close reading of old, “outdated” textbooks and treatises is hardly a waste of time—when supplemented by contemporary information, tracing the route of knowledge and discovery gives the modern scientific picture⁶⁴ a crucial contextual depth otherwise absent.

Ever-astute, Agnes Arber writes in the preface to *The Natural Philosophy of Plant Form*:

I began by thinking of this subject quite simply as a branch of natural science, but I have come finally to feel that it reaches its fullest reality in the region of natural philosophy, where it converges upon metaphysics, to which it brings its own, distinctively visual contribution[...]⁶⁵

Arber states that the study of plant form brings a “visual contribution” to metaphysics—I am inclined to agree, though if you asked me for a definition of metaphysics I couldn’t provide one. As the project moves in towards Part II, the focus shifts from looking at leaf form to reproducing it graphically. To this end, the Intersection presents a broad historical overview of some past work done at the intersection of morphology and formalized representation.

⁶³ Fleury, v.

⁶⁴ Which, realistically, I’ve barely touched on. But I did kind of try! Hopefully this whole thing is mostly accurate.

⁶⁵ Arber, vii.

Intersection

Past Work in Formalized Morphology

Much work has been done attempting to understand morphology and morphogenesis through the lens of formal representation. I include under the umbrella “formal” mathematical, computational, and symbolic. This review is in no way intended to suggest that the present project is actually building off of any of these landmark works other than the efforts of Lindenmayer and Prusinkiewicz in applying L-systems to plant morphology. Rather, I wrote this section because the work done in this area over the course of history is absurdly interesting—formal morphology seeks, above all, interpretations of phenomena that are satisfactory to human intuition. A common theme throughout this review is also the creation of visual models from formal representation. A more typical “past work” summary comes later in the section, where I discuss the specific research upon which this project builds most directly.

A 2012 review of work done at the intersection of plant morphology and symbolic representation by Przemyslaw Prusinkiewicz and Adam Runions begins by citing a statement in Theophrastus’ *Enquiry into Plants* concerning the numbers of petals typically found on roses. The quotation is offered as representative of “the longest historical link between observations and a mathematically flavored research problem in developmental plant biology.”¹ While their point certainly has more to do with establishing the early recognition of common numerical configurations of plant organs rather than affirming some absolute historical origin of mathematically inspired plant models, my own review of

¹ Prusinkiewicz and Runions (2012), 549.

similar ground means the opportunity to pick a different starting place simply because it's possible to do so (and the Greeks get enough credit already).

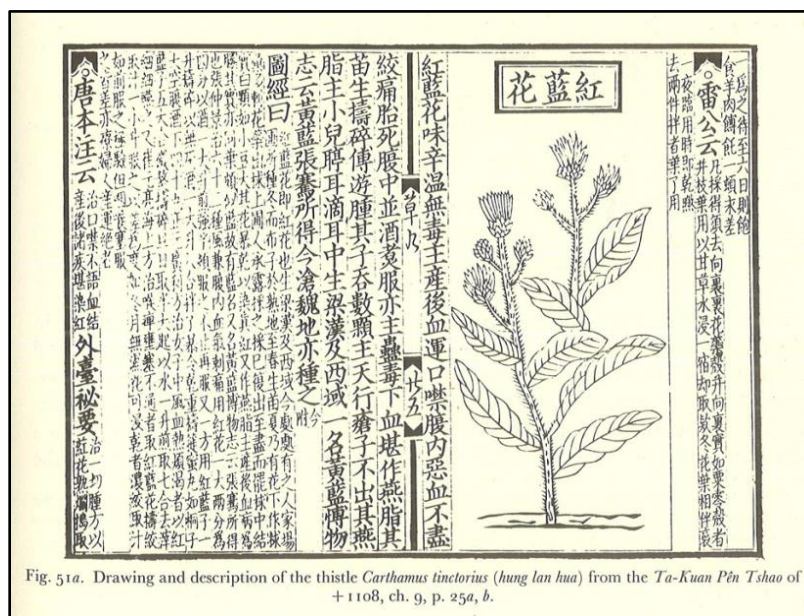


Fig. 51a. Drawing and description of the thistle *Carthamus tinctorius* (hung lan hua) from the *Ta-Kuan Pên T'shao* of +1108, ch. 9, p. 25a, b.

Figure 13: Not quite a treatise on formalized morphology, but a fine drawing. (*Science and Civilization in China*, 285)

That being said, the ancient Chinese were engaged in scientific inquiry during roughly the same (broadly considered) time period as the Greeks and similar morphological observations of plants are found in their writings. While there was not an authoritative, single-authored, comprehensive botanical tract comparable to the one produced by Theophrastus, there is no shortage of extant technical investigations of plant form by the Chinese. Huang Thing-Chien (1090) writes about the tendency of orchids to have either 1, 5, or 6 flowers while Liu Mêng (1104) describes in his book on chrysanthemums the possibility of “a doubling of the petals, and a duplication of the flowers themselves on their peduncles, not to mention how sometimes a transformation of flowers into the ‘thousand-petalled’ varieties occurs.”² What Battjes *et. al* (1993) refer to

² Needham, 418, 413.

as “numerical canalization”—a preference for certain numbers of organs—was observed in plants by the Greeks and Chinese alike. There are also botanical entries in the *Erh-Yah* (an ancient encyclopedia/dictionary from the 3rd century BC) describing different kinds of branching structures.³ Those with any interest whatsoever in Chinese science should look into Joseph Needham’s incredible *Science and Civilization in China*—the entirety of Vol. 6.i deals with botany.

Moving west and forward to the Renaissance, a notebook page of the notoriously prolific Leonardo Da Vinci presents a handful of mathematical relationships describing the branching structures found in trees.

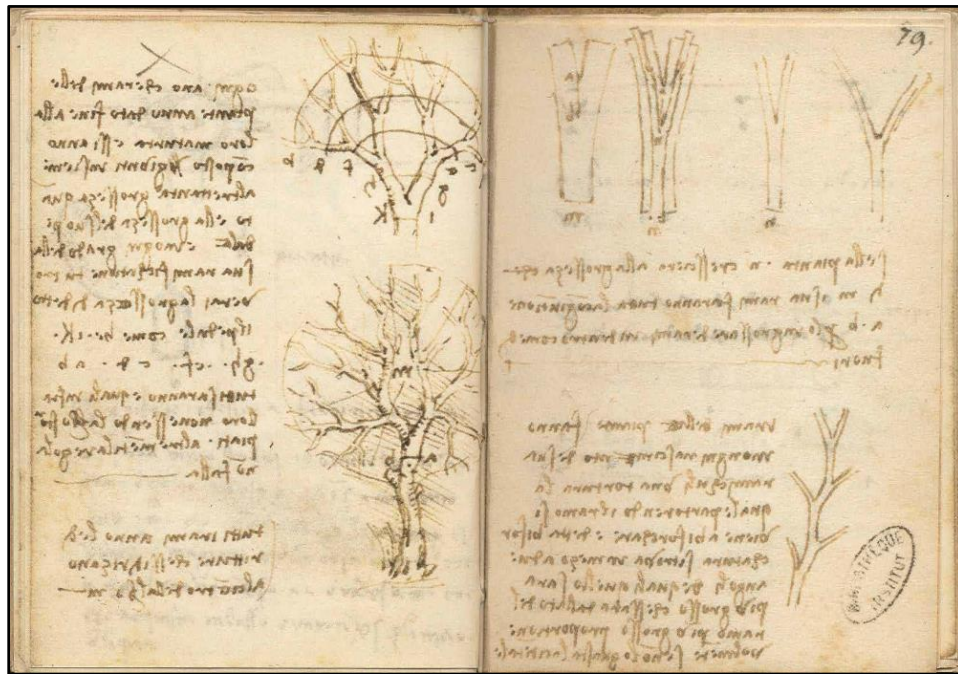


Figure 14: Tree studies by Leonardo Da Vinci, *Paris Manuscript M. fol. 78 v, 79 r.* (1490-1500)

³ Needham, 128.

Edward MacCurdy's translation of the left folio reads as follows:⁴

Every year when the branches of the trees have completed their growth,[...]at each stage of their ramification you will find the thickness of the said trunk as in *ik, gh, ef, cd, ab*. They will all be equal to each other if the tree has not been pollarded; otherwise the rule will not fail.⁵

Da Vinci's treatment of branching morphology is interesting as his method and approach resemble (as well as ink and paper might) later morphological attempts in that his observations are supported by the hand-drawn images—the earliest form of graphical modeling—alongside symbolic association. He is also confident that his posited rule holds true outside of extenuating circumstance, writing that “the rule will not fail.” He has defined what he sees as an ideal rule reliably found in nature provided the specimen hasn't been pruned. On the next page he gives another rule about branching angles and asserts that it is true so long as “no accident has marred the specimen.” Does he have confidence in an absolute, unfailing applicability of his model, or does he recognize it as an abstraction? This demonstrates a wrestling with the difficulty in modeling natural processes which find their sources in the countless accidents and unforeseen variables by which they are constituted. Within those inclined to develop systematic rules to describe the world there might also be found an aversion to ultimately undeniable teratological realities, as the mere existence of “abnormalities” point to the system's insufficient descriptive power.⁶

So what is the rule he proposes here? It is his law describing the thickness of tree branches in relation to their parent branch. He writes it again on another notebook page:

“All the branches of trees at every stage of their height, united together, are equal to the

⁴ If the characters in the image don't seem to match the text, keep in mind that (for whatever reason) Leonardo wrote his notes backwards. The folio on the right side mostly deals with branching angles.

⁵ Da Vinci, 306.

⁶ For a discussion of the significance of teratological concerns, see (Arber, 5). There are benefits to not shying away from abnormal forms and instead using them to better understand typical developmental processes.

thickness of their trunk.”⁷ Essentially, the total cross-sectional area is preserved at every branching stage. Imagine a tree trunk with a diameter of 3 ft. The cross-sectional area would be ~ 7.07 ft. Da Vinci’s rule states that at a point of branching, the cross-sectional areas of the child branches will add up to that of the parent. Let’s say the two child branches coming off this trunk are of equal thickness, although they don’t have to be to satisfy the rule. In this case they would each have a cross-sectional area of around ~ 3.53 ft. The diameter of each child branch would thus be around ~ 2.12 ft. The whole area-preserving relationship can be put in terms of diameters, with $2.12^2 + 2.12^2 \approx 3^2$.

Da Vinci’s rule can thus be expressed as a power law relating parent and child diameters:

$$d_0^2 = d_1^2 + d_2^2$$

To illustrate what this looks like, we can use a graphical model. Here’s a preview of what a parametric L-system can do. The following L-system—taken and modified slightly from one in section 6.3 of Prusinkiewicz *et. al* (1997)—models a tree whose branch thickness follows Da Vinci’s bifurcation rule.

⁷ Da Vinci, 306.



Figure 15: Tree with branch thickness following Da Vinci's bifurcation law.

Da Vinci Tree L-system:

Axiom: $A(110,60)$

Production: $A(s,w) \rightarrow !(w)F(s)[+(30)A(s*0.62, w*0.5^e)][-(66)A(s*0.73, w*0.5^e)]$

Parameters: $e = 1/2$

A description of the parametric L-system mechanism is given in Part II. For now, the most important symbols here are w and e , with w indicating the stroke weight and e being the exponential application of the bifurcation rule. The stroke weight is the thickness of the line and hence stands in for diameter. The two sets of square brackets stand for the left and right branches respectively—the child branch weight is calculated in the expression “ $w*0.5^e$.” The 0.5 on both sides makes each child branch of equal weight—if it were, for example, 0.7 on the left and 0.3 on the right, the left branch of each bifurcation would always be thicker. The trunk begins with a stroke weight of 60. At the first bifurcation, because $e = 1/2$, we get

$$60 * \sqrt{0.5} \approx 42.42$$

Each daughter branch at the first ramification hence has a stroke weight of around 42.42.

Just to confirm that the above L-system checks out, we see that

$$42.42^2 + 42.42^2 \approx 3598.9$$

$$60^2 = 3600$$

At the next point of branching on each side, 42.42 will be the parent diameter, and so on.

Hopefully this example demonstrates what Leonardo's rule expresses and how it can be encoded in a procedural system.

So does Da Vinci's branching rule manifest in actual trees? In "Twigs, Trees, and the Dynamics of Carbon in the Landscape" Henry S. Horn describes his results of testing Leonardo's rule against the real world. After measuring the branching allometry of five different species of tree, he finds that

[...] the twigs and smaller branches are generally thicker than Leonardo's area-preserving rule predicts. Further interpretations must await more precise measurements, structured to disentangle the causes of variation at the small end of the scale. Toward the trunk, however, all species seem to obey Leonardo's rule, regardless of the different hydraulic permeabilities of their wood.⁸

Horn measurements find that it holds for the thicker branches of at least five species and an assumed extrapolation to certain other species is not unwarranted. Most books and papers which reference Da Vinci's bifurcation rule agree that it is a fair abstraction which tends to match reality—Mandelbrot mentions that it also applies to the widths of rivers⁹, which is an assertion found elsewhere in Da Vinci's notebooks. But what of other branching structures? In Part I we discussed arterial branching. Do diameters of artery branches behave like the diameters of tree branches?

⁸ Horn, 204. In *Scaling in Biology*.

⁹ Mandelbrot, 157.

Not quite—while they both bifurcate, arteries aren't solving exactly the same problem as are tree bodies so they don't ramify in the same way. Instead they follow a bifurcation law with a larger exponent (call it γ) which instead of just preserving the total cross-sectional area results in a *larger* total area relative to the parent vessel. This results in thicker diameters. Concerning the value of this exponent, Schreiner *et al.* write that

The appropriate choice of γ has been thoroughly discussed in the literature, and the theoretical arguments as well as experimental measurements indicate that only values in the range $2 \leq \gamma \leq 3$ are physiologically reasonable.¹⁰

Murray's cube law—which deals with the flow rate within the blood vessel—puts the optimal energy preserving value at 3, although as always the real situation is a bit more complex. Following Murray, we now have

$$d_0^3 = d_1^3 + d_2^3$$

Hence, the parameter ϵ in the above L-system becomes $1/3$ and the resulting tree looks a bit different:

¹⁰ Schreiner et. al (2000), in *Scaling in Biology*, 147.



Figure 16: Arterial “tree” with branch thickness based off Murray’s approximations of hydrodynamic optimality.

Why do arteries—unlike the bodies of trees—have branches whose combined cross-sectional areas are greater than that of the parent branch? D’Arcy Wentworth Thompson, the next morphologist in this review, summarizes it nicely:

The increasing surface of the branches soon means increased friction, and a slower pace of the blood travelling through; and therefore the branches must be more capacious than at first appears. It becomes a question not of capacity but of resistance; and in general terms the answer shall be equal in every part of the system, before and after bifurcation, as a condition of least possible resistance in the whole system; the total cross-section of the branches, therefore, must be greater than that of the trunk in proportion to the increased resistance.¹¹

It’s all about conductance—wider tubes mean less pressure loss from resistance.¹² Da Vinci’s tree rule has led to some interesting lines of inquiry but sadly we can’t keep burrowing down the vessel diameter rabbit hole forever.

¹¹ Thompson, 954.

¹² As per the Hagen-Poiseuille law. Also, the Rall model puts the neuronal dendrite diameter exponent at 3/2.

Our focus now moves into the 20th century. Texts dealing with morphology written after 1917¹³ are indebted to D'Arcy Wentworth Thompson's *On Growth and Form*, from which we just read an explanation of arterial diameter. Thompson's book is astounding in its scope and comprehensiveness, integrating broad morphological knowledge from many languages¹⁴ and countries into a single two-volume work. In the realm of the botanical, *On Growth and Form* has sections on the mechanical efficiency of the heights of trees, phyllotaxis, and leaf shape.¹⁵ In his section on phyllotaxis (which is below addressed briefly alongside Alan Turing), Thompson quotes the statement of Nehemiah Grew (plant anatomist and famous contemporary of Marcello Malpighi) that "from the contemplation of Plants, men might first be invited to Mathematical Enquiries."¹⁶ Plants are, as they say, a gateway drug.

¹³ The expanded second edition was published in 1942—quite a bit later and after the formulation of quantum theory.

¹⁴ Primarily German, French, some Italian, not to mention his frequent citations of older Latin works. He also doesn't bother to translate many of the lengthy included quotations, apparently assuming polyglotism of his readers.

¹⁵ Thompson, 28-29, 912-933, and 1041-1047, respectively.

¹⁶ Thompson, 912.

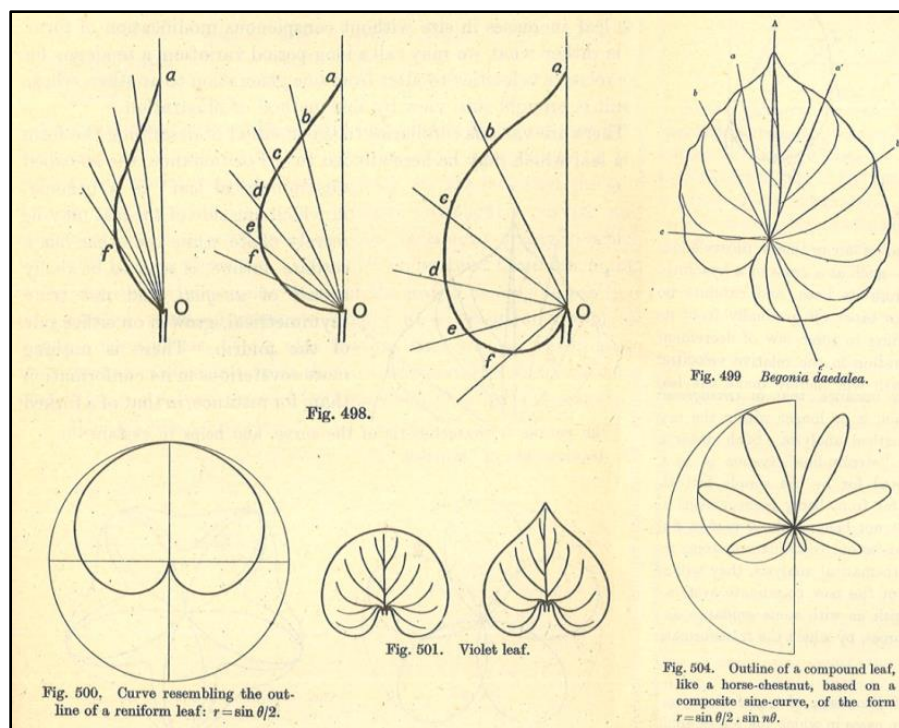


Figure 17: Assembled examples of mathematical analysis of leaf morphologies. D'Arcy Wentworth Thompson, *On Growth and Form* (1952 reprint).

One reason *On Growth and Form* makes interesting reading is the philosophical position of its author. Thompson was looking to apply mathematics to biomechanical explanations of morphological processes, as he thought this was the most precise way to gain a deep understanding. He writes that “In the morphology of living things the use of mathematical methods and symbols has made slow progress”¹⁷—if the subsequent impact of this book is any indication, Thompson surely did his part to hasten the application of mathematical methods to biology. Thompson laid out detailed comparisons of related forms using deformed Cartesian grids, his own method of morphometrical analysis and a doorway to a spatial understanding of differences in organic morphology.

However, Thompson’s overall point-of-view appears mildly inconsistent throughout the text and thus is difficult to parse. Concerning his motivations, he writes:

¹⁷ Thompson, 1028.

My sole purpose is to correlate with mathematical statement and physical law certain of the simpler outward phenomena of organic growth and structure or form, while all the while regarding the fabric of the organism, *ex hypothesi*, as a material and mechanical configuration. This is my purpose here.¹⁸

In Thompson's view, organic forms work by the same mechanical laws as does inanimate matter and they should hence be understood by the same means. As such, he has been understood as being an anti-vitalist. It is strange, however, that he at the same time derides the theory of natural selection as harkening back "to a school of mystical idealism."¹⁹ He appears to believe that the mechanical realities of mathematical law (or, perhaps, the mathematical realities of mechanical law) are enough for Form to manifest the way it does. Despite his scientifically stated "sole purpose" at the very beginning, the Epilogue to the massive text would suggest that the whole effort was all along in pursuit of beauty:

For the harmony of the world is made manifest in Form and Number, and the heart and soul and all the poetry of Natural Philosophy are embodied in the concept of mathematical beauty[...]

Not only the movements of the heavenly host must be determined by observation and elucidated by mathematics, but whatsoever else can be expressed by number and defined by natural law. This is the teaching of Plato and Pythagoras, and the message of Greek wisdom to mankind.²⁰

While there are passages throughout the text where Thompson explicitly distances himself from such a "dreamy" philosophic position (at one point even chastising an "inexcusable Pythagorisme" in the face of the Golden Mean²¹), he in the end embraces Pythagoras.

Returning to the theme of abnormalities brought up with Da Vinci, Thompson's treatment is characteristic of many morphologists. He begins a paragraph "Omitting the "abnormal" cases, such as we have seen to occur in a small percentage of our cones of the

¹⁸ Thompson, 14.

¹⁹ Thompson, 933.

²⁰ Thompson, 1097.

²¹ Thompson, 932.

spruce[...]"²² and then presents a mathematical rule which held for all other examples. What I find most interesting here is that by placing the word abnormal within quotations, Thompson reveals that he is well aware that exceptional forms which do not follow general trends are only circumstantially “abnormal.” From the viewpoint of a man like Thompson there may presumably be some other more broadly considered law which would account for their existence—but again, abstraction and omission go hand in hand. This is perhaps why Alan Turing describes his mathematical model of a growing embryo as “a simplification and an idealization, and consequently a falsification.”²³

In 1952, computer science pioneer Alan Turing published “The Chemical Basis of Morphogenesis”, his well-known paper positing an explanation of biological development and pattern formation based on the diffusion of hypothetical chemical “morphogens” through tissue. This became known as the “diffusion-reaction” theory of morphogenesis. Turing demonstrates how a system which begins with a homogeneous distribution in equilibrium ends up forming regular heterogeneous patterns following the onset of subtle instability and presents six possible mathematical outcomes of what he calls morphogen “wave” patterns. Turing describes that the workings of a particular example found in the paper—an isolated “ring” of cells—is closest biologically to the tentacles of the fresh-water organism *Hydra* and to leaves arranged in a pattern of whorled phyllotaxis.²⁴ He also notes that these “waves could arise in a tissue of any anatomical form.” Botanist C. W. Wardlaw

²² Thompson, 923.

²³ Turing, 519.

²⁴ Turing, 556.

compares Turing's approach to morphogenesis to that of D'arcy Wentworth Thompson's, as both men root their models in physico-chemical laws.²⁵

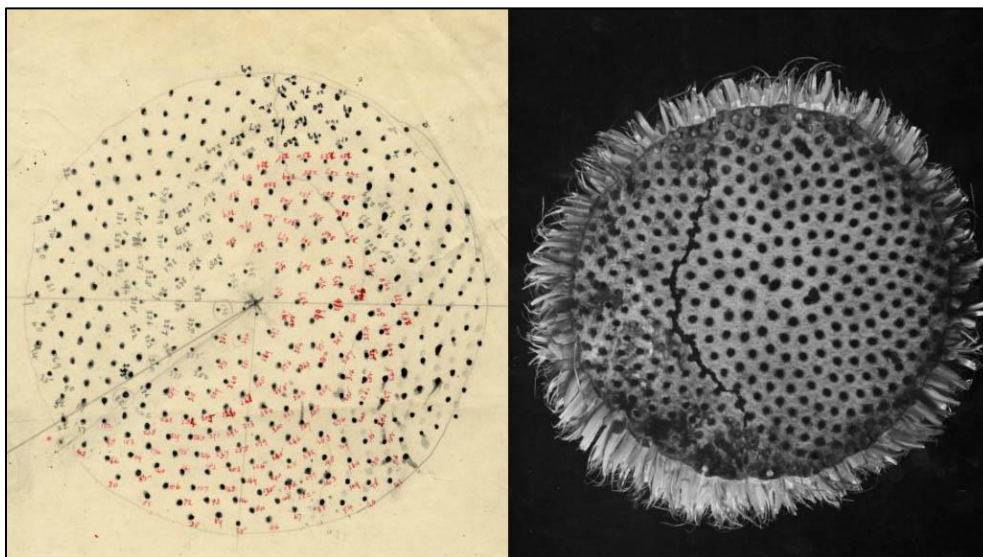


Figure 18: Sunflower florets annotated by Turing alongside photograph from the same archive folder. (Turing Digital Archive, AMT/C/25, images 95, 96)

Found within Turing's drafts and unpublished manuscripts are further investigations into phyllotaxis, or the spatial arrangement of plant organs on the stem. As suggested by Turing's drawing in Figure 18, the spiraled placement of florets on a sunflower can be analyzed mathematically. The term "phyllotaxis" has come up a number of times throughout this project and here will be given treatment. Turing follows a long line of curious investigators of this botanical phenomenon. In 1754 Charles Bonnet and Jean-Louis Calandrini first named and described phyllotaxis in its different forms. Goethe brought attention to the 'spiral tendency' he saw in all plants; there were later investigations by Schimper, Hofmeister, and Braun, among others. A.H. Church's 1901 *On the Relation of Phyllotaxis to Mechanical Laws* is particularly notable. Turing's draft of an

²⁵ Wardlaw, 124.

unpublished paper (aptly titled “A Morphogen Theory of Phyllotaxis”²⁶) applies his morphogen theory towards an understanding of this phenomenon. The morphological work of Turing presented here was done towards the end of his life, as a tragic early death cut short what could have been a career of even further influence—hard to believe of an already gargantuan figure.

Phyllotaxis has excited mathematicians for a long time in part because the geometry of the most common type of phyllotaxis (spiral phyllotaxis) is related to the Golden Angle and the Fibonacci sequence. As the primordial elements of a plant demonstrating spiral phyllotaxis grow, they are each placed at a constant divergence angle (137.5°, the Golden Angle) from the one before. Counting the number of clockwise spirals (called *parastichies*) and then counting the number of counter-clockwise spirals yields a pair of consecutive numbers in the Fibonacci sequence. Figuring out *why* so many plants do this is trickier—the catch-all answer seems to be “self-organization!”

One of the consequences of the phyllotactic patterns of leaf placement is on the amount of sunlight available to leaves lower down on the shoot. Nicotra *et. al* (2011) write:

Computer simulations of mathematically generated shoots to assess the influence of leaf shape, size, and phyllotactic patterns on the ability to intercept direct solar radiation show that differences in phyllotaxy significantly influence light interception[...]

First, notice that computer models are cited as a valid way of testing and measuring the effects of various biological forms, which is an otherwise difficult independent variable to manipulate. Second, more points for Da Vinci, who wrote that

²⁶ Collected, edited, and printed in the 2013 *Alan Turing: His Work and Impact*, ed. Cooper and van Leeuwen and also available in draft form on the Turing Digital Archive: <http://www.turingarchive.org/viewer/?id=124&title=1>

[...]leaves are arranged on the plants in such a way that one covers another as little as possible, but they lie alternately one above the other as is seen with the ivy which covers the walls. And this alternation serves two ends; that is in order to leave spaces so that the air and the sun may penetrate between them, and the second purpose of it is that the drops which fall from the first leaf may fall on the fourth, or on to the sixth in the case of other trees.²⁷

This goes back to Part I, where the relationship of leaves to water and sunlight was a consistent question among observers of nature. Like his contemporaries, Da Vinci saw the sun/leaf relationship as one of drawing moisture and sap upwards (again, the movement of moisture is quite true, but not the whole story). He sees the gaps allowed by phyllotactic patterns of leaf arrangement as being there to let through air, sun, and water droplets. Whatever leaves are really trying to let through, Da Vinci was right to notice that “one covers another as little as possible.” Nicotra *et. al* continue:

They also show that comparatively small differences in leaf shape can compensate for the negative effects of leaf overlap resulting from virtually any phyllotactic pattern. For example, lobed leaves or pinnifid compound leaves facilitate light penetration through shoots bearing densely pack leaves.²⁸

Back to leaf shape again! More light for the leaves below is one benefit of lobation. These computational simulations suggest that leaf shape and leaf arrangement are tied evolutionarily. Nicotra *et. al* view phyllotaxy as “a developmental limiting factor that can drive compensatory changes in morphological features such as shape[...],” which encourages us to think of leaf shape in the context of the form and arrangement of the entire plant.

Mathematician René Thom’s book *Structural Stability and Morphogenesis: An Outline of a General Theory of Models* was first published in French in 1972. In his development of “catastrophe theory,” Thom sought a generalized mode of qualitative theorizing of

²⁷ Da Vinci, 302.

²⁸ Nicotra *et. al* (2011), 543.

morphogenesis based off of differential topological analysis, the titular “catastrophe” being the point at which a phenomenon experiences a sudden discontinuity and change in form. Thom does not see *qualitative* as meaning unmathematical—rather, he’s writing about “the tendency of the mind to give to the shape of a graph some intrinsic value; it is this tendency that we shall develop here to its ultimate consequences.”²⁹ As a book it’s remarkably strange and entertaining, full of musings like the following:

We might say, in this sense that *geometry is successful magic*. I should like to state a converse: is not all magic, to the extent that it is successful, geometry?³⁰

Thom also compares his theory to the philosophy of the pre-Socratics, writing that “all the basic intuitive ideas of morphogenesis can be found in Heraclitus: all that I have done is to place these in a geometric and dynamic framework[...].”³¹ If only math textbooks were so enchanting.

Thom’s theorizing leads him to both the classification of topological singularities into seven well-defined “elementary catastrophes” as well as to descriptions of what he calls “generalized catastrophes.” An example of the former would be the “swallow’s tail” catastrophe (as illustrated by Dalí in Figure 19), the extremities of which he saw as modeling the blastopore furrow found in embryological development.³² The generalized catastrophe he applies to phenomena ranging from human sexuality to delirium to the origin of language.³³ For Thom, morphogenesis is not limited to just the bodily forms of organisms but rather to anything with a formed structure. Hence, morphogenesis is present in essentially everything. Defending Anaximander and Heraclitus’ uses of

²⁹ Thom, 4.

³⁰ Thom, 11.

³¹ Thom, 10.

³² Thom, 67.

³³ Thom, Chapter 13. From Animal to Man: Thought and Language.

anthropological words like ‘conflict’ and ‘injustice’ to describe the appearance of the physical world, he writes:

“[...]the dynamical situations governing the evolution of natural phenomena are basically the same as those governing the evolution of man and societies[...]³⁴”

Or, as Heraclitus would say, all is flux.

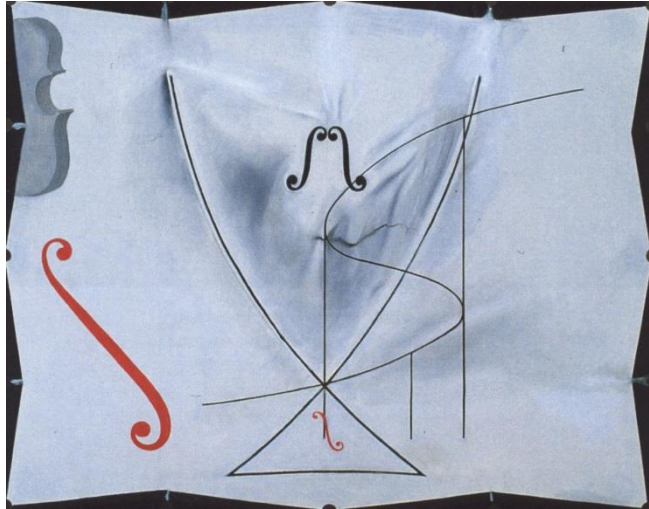


Figure 19: Salvador Dalí's final painting, *The Swallow's Tail* (1983).

By his own admission, Thom's scheme of using topology to address morphogenesis offers little explicit predictive benefit but rather a qualitative geometrical description of formative processes. His primary focus is on, as the title of the book suggests, the benefit of models. In comparison to the other mathematical approaches to morphogenesis described here, not a whole lot appears to have directly come out of Thom's catastrophe theory. After much hype in the 1970s it fell out of favor. However, it remains an intriguing and unique way of conceptualizing sudden changes and his idea of "catastrophe" as a formative tipping point has become part of the common lexicon.

Benoit Mandelbrot caused a stir in geometrical intuition with his studies in the late 1970's of irregular, frequently self-similar scaling patterns he placed under the umbrella

³⁴ Thom, 323.

term *fractals*, from the Latin *fractus*, meaning interrupted, irregular, or broken. These investigations culminated in the 1982 publication of *The Fractal Geometry of Nature*. Calling it “a manifesto and a casebook,”³⁵ Mandelbrot presents precise mathematical explanations of patterns classical Euclidian geometry deemed “formless”: that is, many of the shapes we see in the natural world all around us. The book opens with a reminder that “clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line.”³⁶ Mandelbrot examines non-linear constructions which had been labeled by mathematicians as “monstrous,” such as Koch curves, the Peano curve³⁷, and Osgood curves, among others. Likening the latter to vascular systems, he writes “Lebesgue-Osgood fractal monsters are the very substance of our flesh!”³⁸ There are numerous methods for generating fractal graphics and L-systems are one of them.³⁹

³⁵ Mandelbrot, 24.

³⁶ Mandelbrot, 1.

³⁷ A variant of the Peano curve, the Hilbert curve was used by Sapoval *et al.* (1999) to model optimal mammalian acinus morphology (“acinus” meaning the gas exchange surface of the lungs, where the alveoli are), ultimately demonstrating that, as a result of the behavior of diffusion currents, “for good efficiency of the diffusive transfer of oxygen to blood, the unit transfer system, namely the lung acinus, should not be too large[...]the lung has to be divided into a large number of small efficient units[...]As the lung is space filling, the air access to these units has to be a branched geometry.” In *Branching in Nature*, 225.

³⁸ Mandelbrot, 150.

³⁹ That is, using the looser definition of fractal which considers technically finite curves to be approximations of infinite fractals.

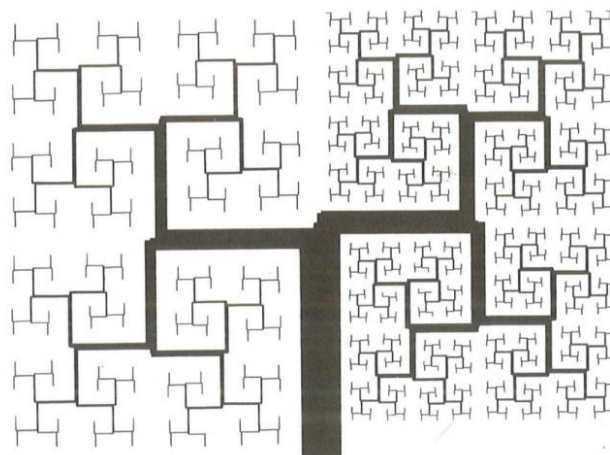


Figure 20: “Plane-filling recursive bronchi,” from Benoit Mandelbrot, *The Fractal Geometry of Nature*, Plate 164.

Mandelbrot was a believer in the power of human vision to make sense of the world. He thought that the best way to test the validity of a scientific model is to see if it produces something that looks right to a human subject:

Graphics is wonderful for matching models with reality. When a chance mechanism agrees with the data from some analytic viewpoint but simulations of the model do not look at all “real,” the analytic agreement should be suspect.

This is a rather bold statement, as it places a more immediate trust in the instinctual judgment of the human perceiver than something arrived at through an analytic method. It would not be every researcher’s first hunch to question the numerically validated model before their own opinion—although, in the case of a poor simulation result, the first suspect is probably the programmer. Mandelbrot continues:

A formula can relate to only a small aspect of the relationship between model and reality, while the eye has enormous powers of integration and discrimination.⁴⁰

Sense perception allows for a more comprehensive and all-at-once view of phenomena than a specific formula on its own. Put another way, “in the theory of fractals ‘to see *is* to believe.’”⁴¹

⁴⁰ Mandelbrot, 22.

⁴¹ Mandelbrot, 21.

A well-known method of fractal generation using “Iterative Function Systems” was first developed by John E. Hutchinson and later expanded on and popularized by Michael Barnsley in his book *Fractals Everywhere* (1988) and its successor *Superfractals* (2006). One of the most well-known images to come out of Barnsley’s work with IFSs is the fractal resembling a black spleenwort fern (often called the “Barnsley fern”) as shown at the top of Figure 21. The Barnsley fern is appealing aesthetically and also because it makes the self-similar structure of this particular kind of plant mathematically and visually apparent.

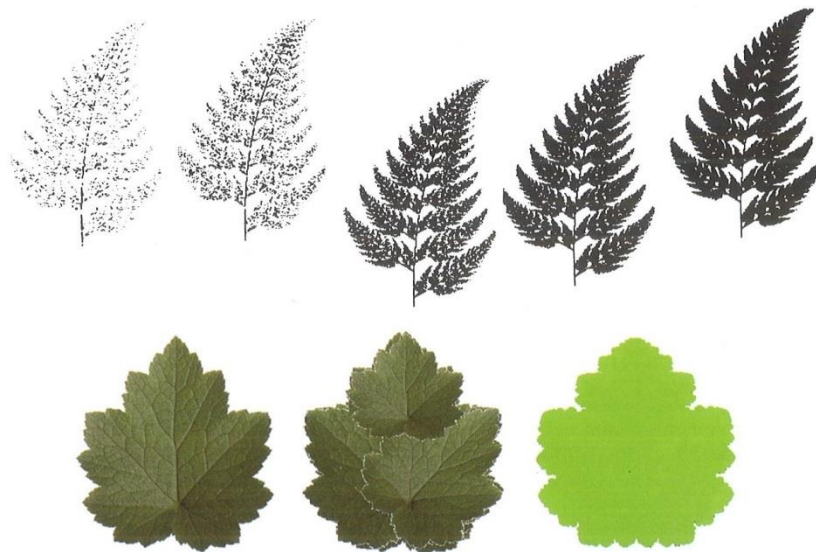


Figure 21: Some plant-like fractals by Michael Barnsley. *Top*—A Barnsley fern generated using the randomized Chaos Algorithm (*Fractals Everywhere*, 92). *Bottom*—leaf shaped attractor, right, generated using the Collage Theorem on target set, left (*Superfractals*, 329).

As an abstraction, strict self-similarity only captures certain aspects of plant form and ferns are rather unusual (as far as plants go) to the degree they express it. The basic IFS mechanism results in constructions that are *strictly* self-similar, frequently not the case in organisms. Thus research done with IFSs has resulted in expanded systems which have relaxed self-similarity requirements, such as recurrent IFSs (RIFS) or language-restricted

IFSs (LRIFS). The collage theorem as shown at the bottom of Figure 21 can also be used to generate attractors that resemble objects that are only approximately self-similar.

The relationship of IFSs to L-systems has been an area of interest for researchers. Chapter 8 of *The Algorithmic Beauty of Plants* presents an example of how to proceed from a certain kind of parametric L-system to an IFS which generates an equivalent fractal structure—however, the method described here only works if the form to be modeled has “constant branching angles as well as fixed proportions between the mother and daughter segments.”⁴² Further attempts to understand the formal relationship between the two systems can be found in Prusinkiewicz and Hammel (1994), which considers the more generalized/relaxed LRIFSs, and Ju *et. al* (2004), which presents a proof of equivalency between recursive turtle programs (RTPs—non-bracketed L-systems with one production rule) and iterated affine transformations (IATs—IFSs with only affine transformations).

While IFSs can produce attractive images of ferns, their application to biological modeling more generally is complicated. Prusinkiewicz wrote in 1998 that

To date, applications of iterated function systems to the modeling of plants have been investigated mainly from the computer graphics perspective. Their relevance to biology is yet to be determined.⁴³

It appears that this is not just because the most basic kind of IFS requires complete self-similarity but also due to the way the algorithms which generate them proceed. Simcha Lev-Yadun argues in “Fern leaves and cauliflower curds are not fractals” (2012) that common modes of understanding organic forms through fractal geometry are biologically irrelevant and misleading for two reasons. First, the self-similarity of ferns and cauliflower curds is only superficial and does not scale down very far. Second, these organisms develop

⁴² Prusinkiewicz and Lindenmayer (1990), 189.

⁴³ Prusinkiewicz (1998), 121.

from the “inside out” while common methods of fractal generation do not. However, he only addresses one kind of fractal generation algorithm⁴⁴, referencing the randomized chaos algorithm (see Figure 21) as being entirely unlike the fern’s actual developmental process. He writes that

Dissecting developmental processes, such as the formation of fern leaves and cauliflower curds, into stages that are mathematically manageable and developmentally and structurally correct would make the mathematical procedures more relevant for biologists. A set of mathematical procedures that reliably describe the development of fern leaves, cauliflower curds or any other plant organ will not be just an elegant visual demonstration, but probably an important lesson in developmental biology. I look forward to seeing it.⁴⁵

I wonder if he’s heard of L-systems! As will be explained in Part II, L-systems develop iteration by iteration through their productions—in this sense, they can develop from the “inside out.” Developmental models which grow larger every step are easy to encode.

Thus it might be argued that the way L-systems interface with spatial development make them more generally suited to the modeling of biological phenomena than IFSs as they are currently understood and formulated. If anything, this argument is best supported by the fact that L-systems have been used extensively for biological modeling while IFSs generally haven’t. *The Algorithmic Beauty of Plants* opens by explaining its purpose in exploring *two* factors that “organize plant structures and therefore contribute to their beauty.” These are the “elegance and relative simplicity of *developmental algorithms*, that is, the rules which describe plant development in time” and “*self-similarity*.”⁴⁶ L-systems can hence express fractal self-similarity where it appears but are not as strict in their

⁴⁴ There do exist deterministic generation algorithms for IFSs.

⁴⁵ Lev-Yadun (2012), 534.

⁴⁶ Prusinkiewicz and Lindenmayer (1990), 189.

mathematical requirements as are most IFSs, and it is also simpler to construct them so that their iterative “development” resembles something like biological growth.

Whether or not IFSs are especially useful for biological modeling purposes, Barnsley sees something inherently botanical about them. In *Superfractals* Barnsley makes the metaphorical comparison of mathematical code space to the meristem of a plant:

There is a remarkable set, called a code space, which consists of an uncountable infinity of points and which can be embedded in the tiniest real interval. A code space can be reorganized in an endless variety of amazing geometrical, topological, ways, to form sets that look like leaves, ferns, cells, flowers and so on. For this reason we think of a code space as being somehow protoplasmic, plastic, impressionable and capable of diverse re-expressions, like the meristem of a plant[...] This idea is a theme of this chapter and of the whole book.⁴⁷

The common morphological quest of divining some ultimate Protean unit reappears once again. For Wolff and Goethe it was the leaf, and in Barnsley’s abstract fractal world, it is a code space. Weirdly enough, Barnsley isn’t even the only mathematician to make this kind of comparison—M.A. Peterson writes of the singularities of Laplacian growth as “mathematical meristems.”⁴⁸ Barnsley takes the botanical metaphor even further with his definition of a mathematical set (relating to his intriguing “superIFSs”) he calls a “V-variable code tree,” made up of precisely defined “limbs”, a “trunk”, and “branches.”⁴⁹ Finally, *Superfractals* uses some kind of plant form for nearly every graphical example in the book. Vegetation is nothing if not iterative.

L-systems came up in the above discussion of development for a good reason: it’s what they were designed to model from their conception. Aristid Lindenmayer first introduced L-systems in 1968 with the publication of “Mathematical models for cellular

⁴⁷ Barnsley (2006), 8.

⁴⁸ Peterson (1999), 449.

⁴⁹ Barnsley (2006), 435.

interaction in development” in the *Journal of Theoretical Biology*. Originally intended to model the development of simplistic multi-cellular organisms, their formal properties and extensibility make them also very suitable for modelling plant form and growth.

Lindenmayer’s research has been continued most notably by Przemyslaw Prusinkiewicz, who at the time of writing runs the Biological Modeling and Visualization research group at the University of Calgary.

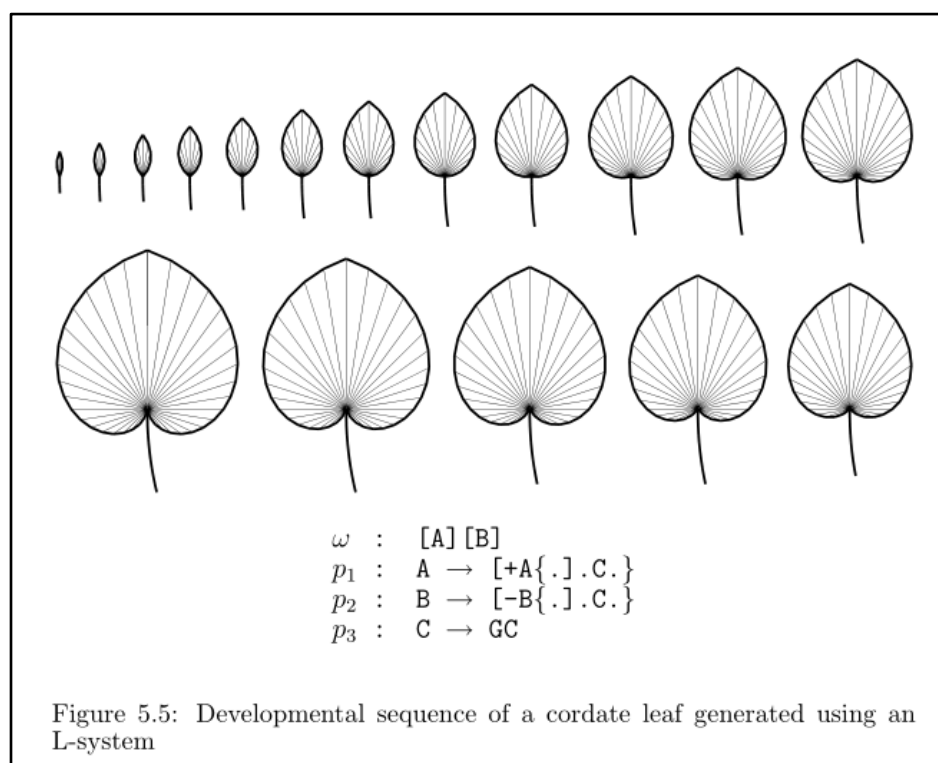


Figure 22: From *The Algorithmic Beauty of Plants*, 123. The L-system framework given here resembles the parallel venation pattern of a monocot cordate leaf, such as that of the wild yam (*Dioscorea villosa*).

1990—the year following Lindenmayer’s death—saw the publication of *The Algorithmic Beauty of Plants*, a compilation of the research, results, and efforts of Prusinkiewicz and Lindenmayer up to that point. *ABOP* offers an extensive exposition of L-systems and explores many dimensions of their possible applications to plant modeling

while assuming of the reader no prior familiarity with the subject.⁵⁰ It is cited extensively in subsequent literature and offers an excellent example of how years of rigorous computational research can be effectively organized and presented in an accessible book format. It was an indispensable resource for this project.

I find *The Algorithmic Beauty of Plants* intriguing also for its title—before you even open it the book makes an aesthetic argument. How does thinking of beauty as “algorithmic” change or enhance our understanding of both aesthetics and computation? Is *all* beauty somehow algorithmic or is “algorithmic beauty” just one of many types of beauty? What would Kant say? I don’t know—I tried to read *Critique of Judgment* but it gave me a headache.

Having sketched one possible historical trajectory of the interdiscipline, we now look at more recent research relevant to the project at hand.

The single paper upon which my project builds most directly is Rodkaew *et al.*'s “Modeling leaf shapes using l-systems and genetic algorithms,” which can be found in the 2002 proceedings of the *Plant International Symposium on Plant Growth Modeling, Simulation, Visualization and their Applications*. Bringing together genetic algorithms, L-systems, and leaf shape, Rodkaew *et al.* evolved the parameters of a simple skeleton framework so the resultant shape matched the input leaf’s outline satisfactorily. The genetic algorithm was performed on a set of tag-functions in order to adjust their parameters, meaning the evolution was not being done on the L-system symbols themselves. For papers with genetic algorithms which use the L-system alphabet symbols instead of their parameters as the genotype, see Jacob (1994) and Ochoa (1998).

⁵⁰ It is also available for free in full on the internet! URL: <http://algorithmicbotany.org/papers/abop/abop.pdf>

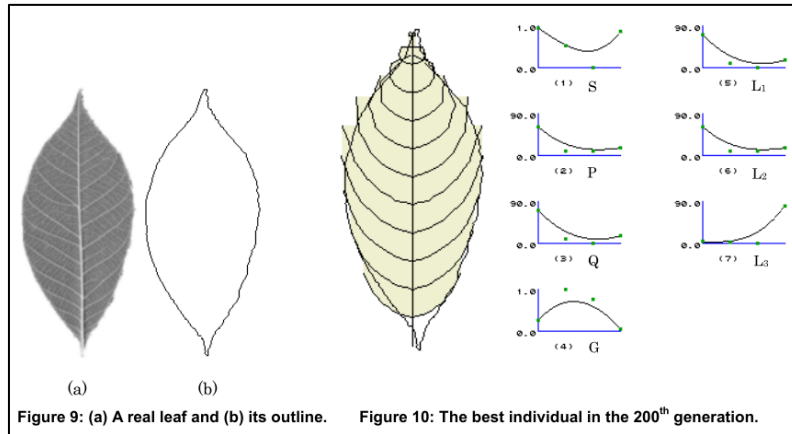


Figure 23: Results from Rodkaew *et. al.*

The primary similarity is between the basic aims of these two projects—using computational search techniques, find a parametric L-system which results in the same shape as the input leaf, thus in the end getting information about a *plausible* internal structure. I am unsure if the resulting L-system of Rodkaew *et. al* can also show a reasonable developmental sequence throughout the system iterations, but this was an additional motivation on my part.

There are a number of differences between this paper and my project. They do not use internal definition of geometry while I use a polygonal L-system definition where the veins serve as a framework for the whole leaf shape. The mutation of their venation skeleton results in an upwards bending due to how they have structured their productions. This bend appears to be what their parameters (mostly angle changes) control, not growth rates. They end up with a result with arcuate venation as the curves bend to approximate the shape. Their skeleton model is limited to a pinnate unlobed leaf. The fitness function measures the distance of the outlines of the input leaf and the L-system output. For every coordinate, it computes $(\mathbf{x}_i^t - \mathbf{x}_i^o)^2$ where t is the outline of input leaf (target) and o is the outline of the L-system output. What I don't understand is how they know there will be the

same number of points in each outline or otherwise how a varying number of points is accounted for. A final difference is that they actually implement a genetic algorithm. Their search technique actually works.

Part II: FormaLeaf

**An Interactive System for Generating L-system
Representations of Leaf Shape and Structure**

The Language of L-systems: Grammars of Growth

In the interest of making it clear what's under discussion, this section opens with an explanation of what L-systems are and how they work before delving into further details. I've written the basic explanation with the hope that it might be mostly comprehensible to those without a background in computer science. These images were generated using the same L-system implementation which runs in FormaLeaf. Also, the footnotes in the subsequent sections will not longer be highlighted green as there isn't as much citational information.

The Basics: Anatomy and Mechanism

L-systems are parallel string rewriting systems. The mechanism by which they work is quite simple. We first consider a purely symbolic/linguistic example sans any graphical interpretation in order to demonstrate how a string is rewritten in parallel over the course of a few iterations.

Succinctly, at every step an L-system takes a string (a sequence of characters) and applies an appropriate replacement rule to every single character before moving on to the next iteration. These replacement rules are defined in the set of production rules. The beginning string is called the "axiom." The axiom serves as the starting point. Consider the following example:

You decide to play a game wherein each round you replace every character in a sequence according to a specific rule.

Suppose you had this sequence of characters: **ABCD**

ABCD is your *axiom*.

Reading the rulebook for this game, you see that:

A becomes **D**

B becomes **BB**

C becomes **AC**

While the book lists no rule for **D**, you see a note that this means the symbol is just replaced with itself. You play this game for a few rounds:

Round 0: ABCD

Round 1: DBBACD

Round 2: DBBBBDACD

Round 3: DBBBBBBBDDACD

Round 4: DBBBBBBBBBBBBBBBDDDDACD

Round 5: DBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBDDDDACD

That's getting to be a lot of **B**'s! Wary of wheat and chessboards, you decide to stop playing.

That's parallel string rewriting in a nutshell. The "rulebook" in the above example does not list a rule for **D** because this is normal for the definition of any L-system's productions—the rule that rewrites a character as itself is implied if this character does not appear on the left-hand-side (LHS) of any production in the set. Using a more typical formal notation, the above example can be written like so:

ω : **ABCD**

p_1 : **A → D**

p_2 : **B → BB**

p_3 : **C → AC**

So how do you get from a string of characters to an image? Simply read every symbol in order and interpret each symbol as a drawing instruction. L-systems generally use a LOGO-Turtle interpretation, which means that the instructions control the local state

of a Turtle moving around the space and drawing things. That is, if a symbol stands for “Turn right 30°,” the Turtle will rotate to *its* right and adjust the way it is facing. Hence, the only drawing instruction really needed for most images is “Move forward one unit and draw a line,” as drawing in a new direction is done by placing an angle turn command before the draw instruction. Implementations also may have a command to move forward without drawing, which allows for composing images without necessarily connected lines.

Two crucial symbols are the turn commands:

Turn left: +

Turn right: -

Let’s say that **F** means “Move forward one unit and draw.” A Logo Turtle which begins facing North when presented with the string **F-F-F-F** and told to make all its turns 90° would draw a box.

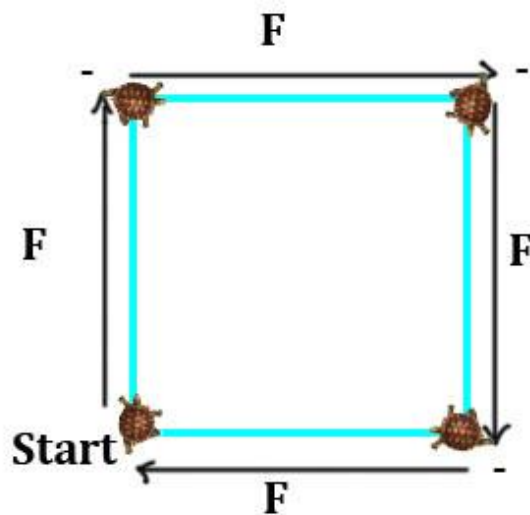
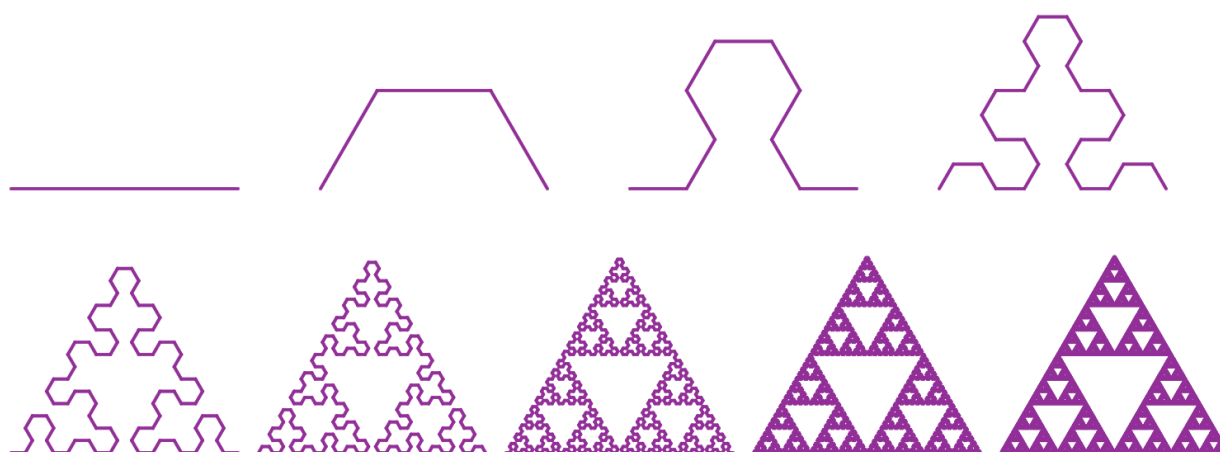


Figure 24: LOGO-Turtle interpretation of a string.
(Turtle image from <http://small-pets.lovetoknow.com>)

After a string has been rewritten according to the production rules, the Turtle interpreter reads and follows the whole thing in sequence. Changing the iteration (number of rewrites) means the Turtle is interpreting a new command string.

Pictured below is the axiom and first 8 iterations of the Sierpinski triangle as generated by an L-system. The sequence below is also scaled down each iteration.¹ Imagine that the Turtle begins facing East.



ω : **A**

p_1 : **A** \rightarrow **+B-A-B+**

p_2 : **B** \rightarrow **-A+B+A-**

Interpretation:

A : Go forward one unit while drawing.

B : Go forward one unit while drawing.

+ : Turn Left 60°

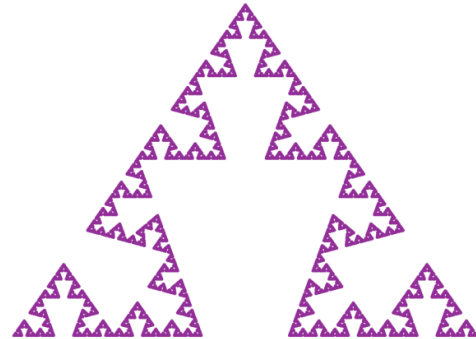
- : Turn right 60°

There are two line drawing commands here due to how the L-system is structured. Both commands do the same thing but in order to have the two unique productions there are two draw symbols. Sometimes L-systems have symbols with *no* defined Turtle interpretation—these symbols are included to give grammatical structure.

¹ Were it not scaled down, the triangle would grow to an enormous size (that is, every segment would be as long as the line at the top-left).

In the above case, the angle and scaling factor are extraneous to the L-system and are supplied at the time of graphical generation. The classic Sierpinski triangle is constructed with a 60° turning angle.

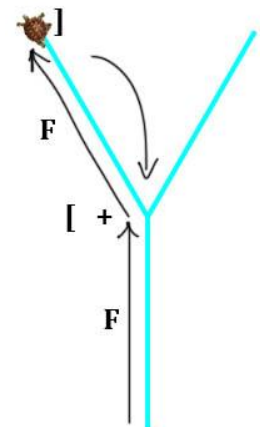
Pictured to the right is the same L-system after 9 iterations constructed instead with a rotation angle of 55° . This small change in the angle has a clear ripple effect throughout the subsequent iterations, making the final shape noticeably different from the 60° form.



Bracket Notation and Branching:

The basic L-system mechanism described above can be extended with bracketing, which uses stack operations and is useful for modeling branching structures. When it encounters a left bracket ([) as it reads the string, the Turtle knows to push its own state onto a stack (basically write it down and save it for later). The Turtle will then interpret all the symbols within the brackets. Once it reaches the closing bracket (]), it pops its saved state off the stack and thus its internal state returns to what it was before it embarked on its interpretive bracket journey.

This is helpful for modeling branching structures, as the Turtle can go ahead and draw the first branch of structure and then use its magic turtle powers to teleport back to the place of bifurcation before drawing the second one. The figure to the right shows the interpretation of the string $F[+F][-F]$. Assume a branching angle of 30° . The arrows show how



the Turtle draws the left branch and returns to the point where first it encountered an opening square bracket.

After applying the L-system production/rewriting mechanism to bracketed strings, it is possible to generate branching structures which appear to “grow” over the course of their iterations. The **X** symbol has no Turtle interpretation but is there to structure the strings which are created. **F** again means move forward one unit while drawing. This L-system has recursive production rules. It is scaled down slightly each step in order to fit on the screen, but the growth effect remains.

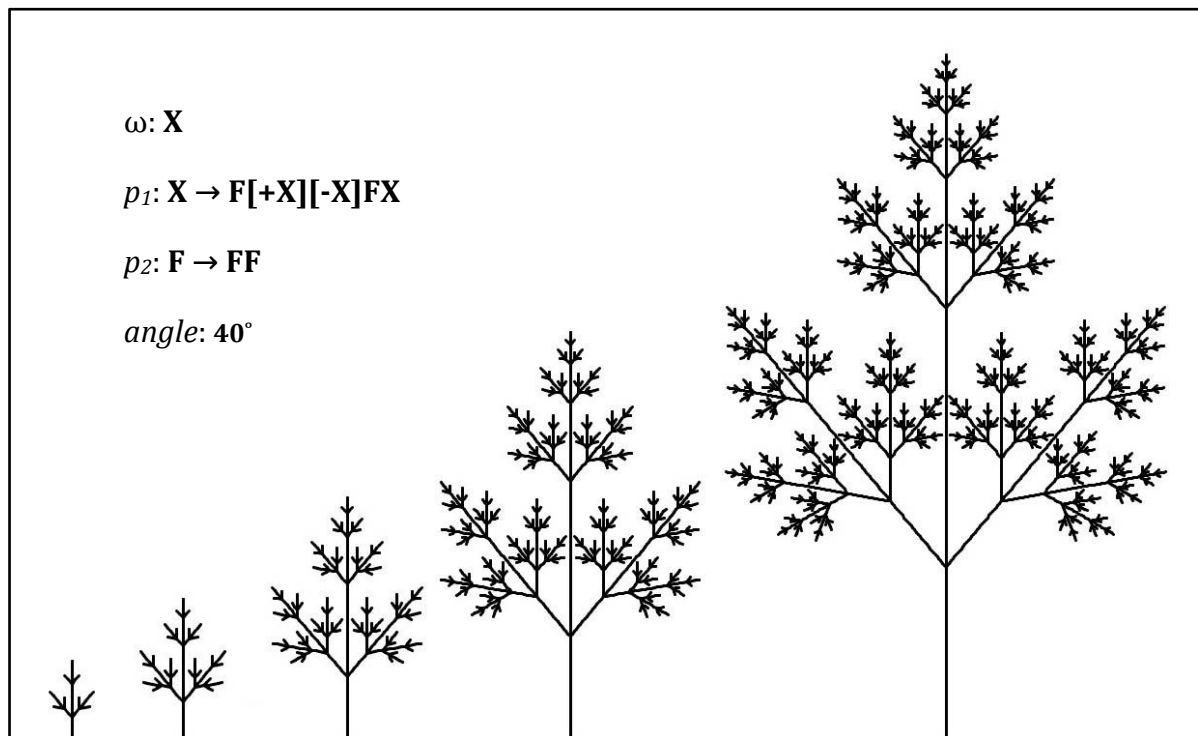


Figure 25: L-system adapted from *The Algorithmic Beauty of Plants*, 25.

Keep an eye out for how brackets are used to indicate branching veins during the discussion of the template leaf L-systems in the Method section. Branching L-systems are integral to this project!

Also, a slight modification to the above tree L-system gets a really surprising result. I wonder whether or not it is a coincidence that the little hole towards the bottom vaguely resembles the Mandelbrot set.

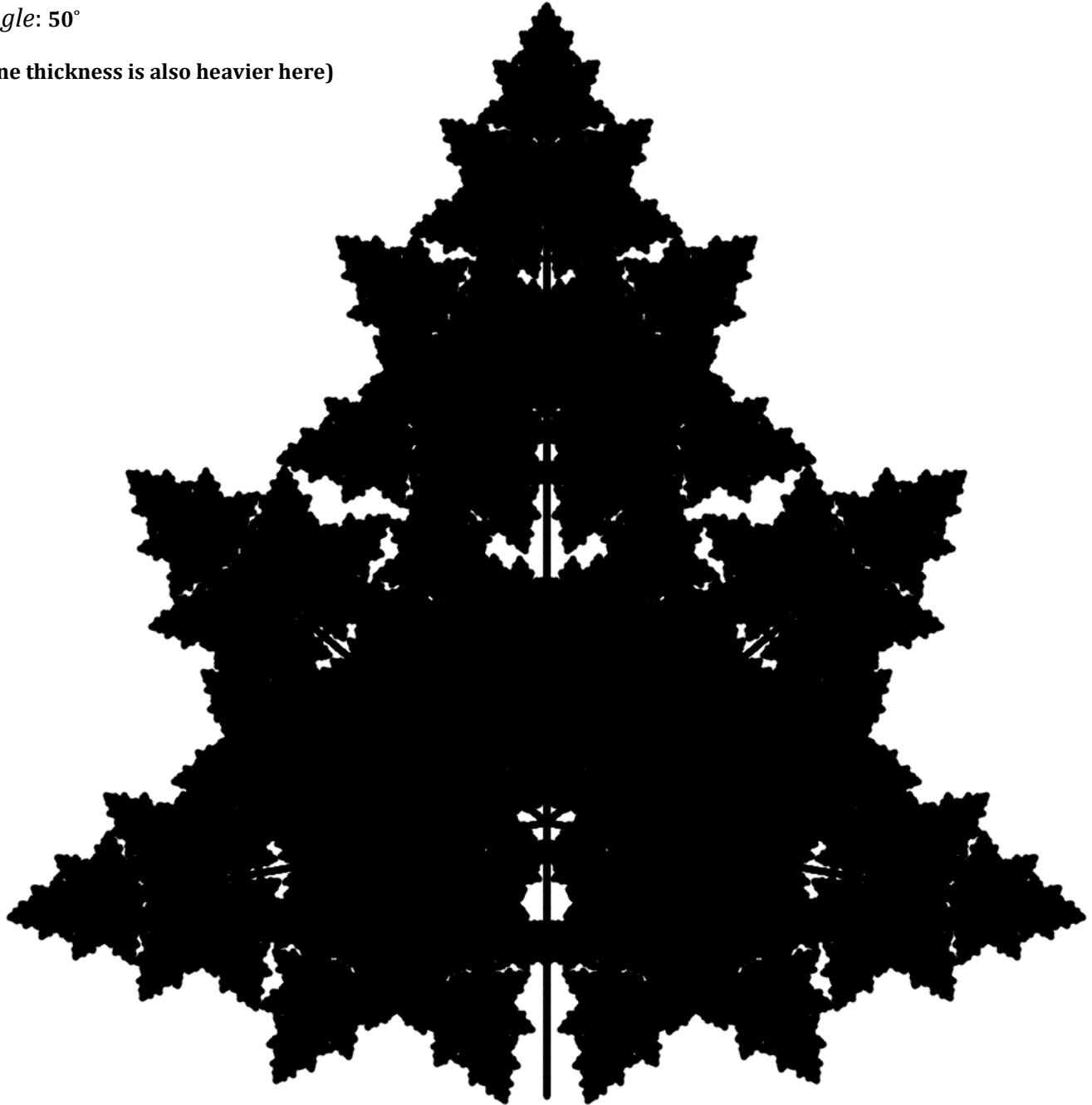
ω : X

p_1 : X \rightarrow F[++X][+X][-X][--X]FX

p_2 : F \rightarrow FF

angle: 50°

(line thickness is also heavier here)



Parametric L-systems:

There are many types of L-systems.² All examples given so far are DOL-systems, which are the most basic kind. They are deterministic and context-free. The “D” stands for “deterministic” and the “0” means that the system is completely unaware of a given character’s context when executing production rules—it never takes into account the characters on the left or right of the current symbol.

Another kind are parametric L-systems, used extensively in this project. These associate numerical parameters with the symbols in the grammar. In a few of the above examples the Turtle interpretation for a symbol is explained as “Go forward one unit and draw.” A parametric L-system could have a draw command with a unit *parameter*. Hence **+F(1)-F(3)+** is a possible parametric string fed to the Turtle. The first draw command would probably mean “Go forward one unit” while the second would be interpreted as “Go forward three units.” You could also parameterize angle: **+(40)F(1)-(60)F(3)+(90)**. This way, angle is not defined outside of the grammar. It also changes between symbols and thus the angle value here is variable (unlike all examples so far). A full parametric L-system could look something like this:

$$\begin{aligned} \omega &: \mathbf{X(3, 40)} \\ p_1 &: \mathbf{X(u, a) \rightarrow F(u)[+(a)X(u,a)][-(a)X(u,a)]F(u)X} \\ p_2 &: \mathbf{F(u) \rightarrow F(u)F(u)} \end{aligned}$$

Hence the 3 and the 40 get passed as inputs into the system—the u and a then take on these values (standing for ‘unit’ and ‘angle’ respectively) in the resulting productions.

² Two kinds not used in this project are context-sensitive L-systems (one or two-sided, productions only fire when contextual rule is true) and stochastic L-systems (productions fire probabilistically).

Parameterizing various parts of a grammar really opens up what can be done with it. The FormaLeaf interface has multiple parameter sliders which allow the user to manipulate the values being plugged into the parametric L-system which defines the generated leaf's structure and shape.

Internal Geometric Information and Modeling Polygons:

In their original conception, L-systems did not have internal geometric information pertaining to the angles and directions of the segments and branches (in bracketed cases). Prior to automation, L-systems were interpreted intuitively by draftspeople who drew by hand a graphical representation of the axioms and productions. Because the interpretation was done by humans, there was less need for standardized geometrical information to be included in the systems themselves. The resultant hand-drawn graphics were thus one possible interpretation of ambiguous information.

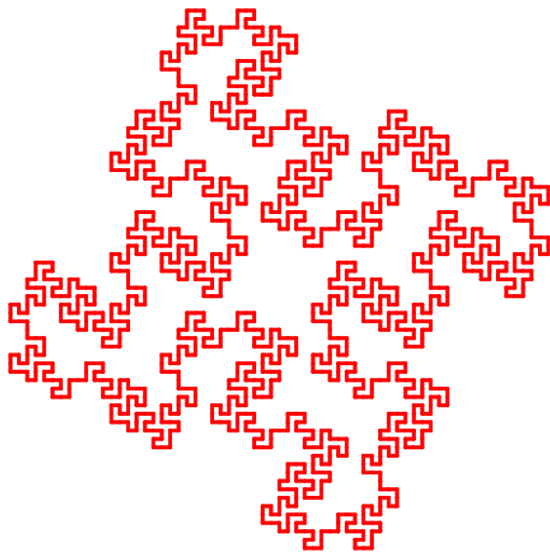
Automated interpretation by a computer requires the elimination of ambiguities. Geometric rules to make automated interpretation unambiguous were at first global, external, and not part of the specific system, which resulted in certain structures being unspecifiable. Alongside the LOGO style graphics interpretation in the late 1980s came additional alphabetical symbols used to specify angle direction—typically [-,+] are used, although which one is designated “turn left” and which one “turn right” appears to vary arbitrarily across implementations.

Similarly, the geometric information that allows for closed, polygonal structures can be made internal to the system as well. One of the chief goals of this project (and one way in which it does something different from previous automated modeling efforts) is that the geometric shape information of the generated leaf is internal to the generated L-system.

Fractals and L-Systems:

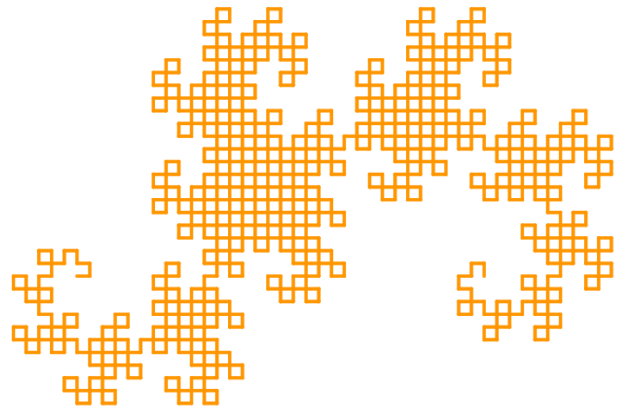
By virtue of their potential to encode self-similarity within their productions, L-systems are able to generate many classic fractal structures. Some examples are given below.³ The number of iterations varies by the complexity of the constructions, as some will crash or freeze the program after exceeding just 4 or 5 iterations while others don't get interesting until upwards of 8.

Quadratic Koch Island



ω : F-F-F-F
 p : F \rightarrow F+FF-FF-F-F+F+FF-F-
 F+F+FF+FF-F
 angle: 90° iterations: 2

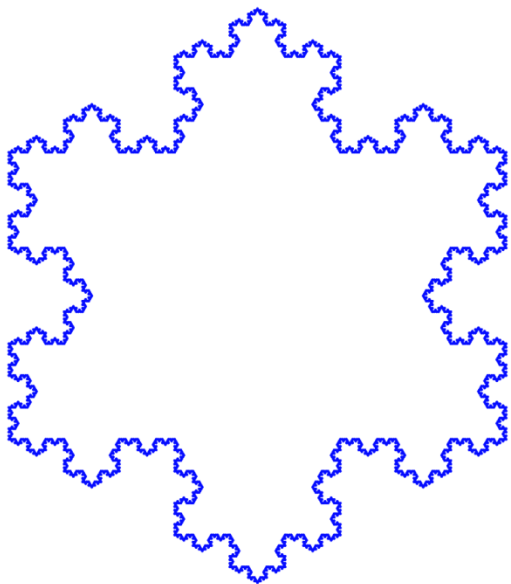
Dragon Curve



ω : FX
 p_1 : X \rightarrow X+YF+
 p_2 : Y \rightarrow -FX-Y
 angle: 90° iterations: 10

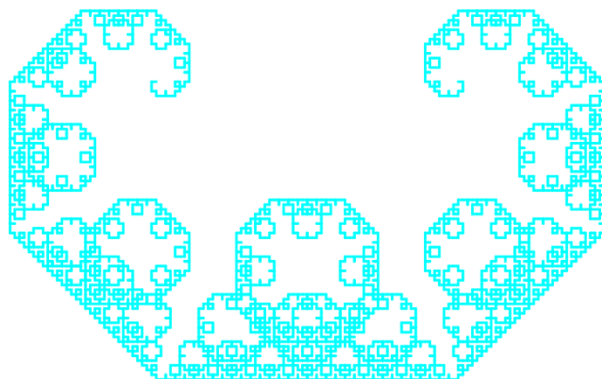
³ The L-system definitions for the first four examples were found in *The Algorithmic Beauty of Plants* or on Wikipedia. These two particular shapes made out of the Cesàro curve (a very simple angle modification of the Koch curve) I found through experimentation.

Koch Snowflake



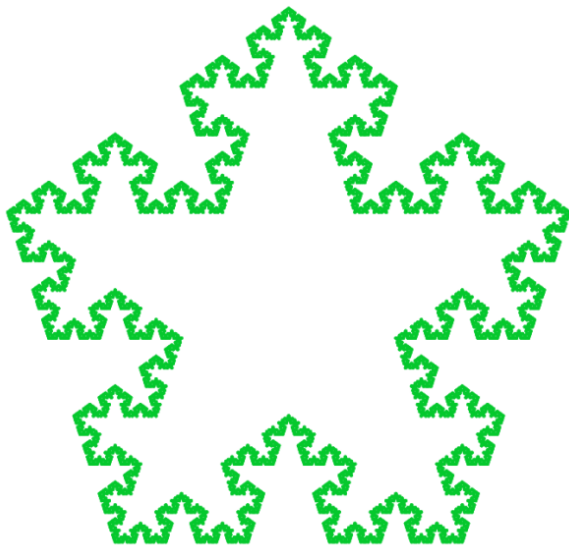
ω : F++F++F
 p : F \rightarrow F-F++F-F
 angle: 60° iterations: 5

Lévy C curve



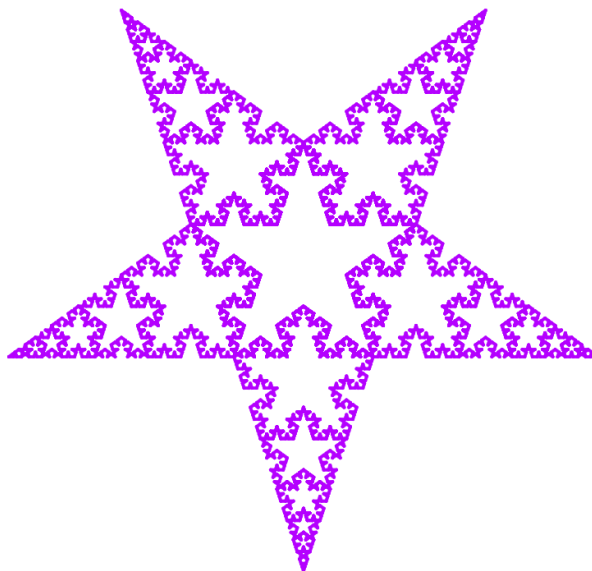
ω : F
 p : F \rightarrow +F--F+
 angle: 45° iterations: 12

Cesàro Curve Pentagon



ω : F-F-F-F-F
 p : F \rightarrow F-F++F-F
 angle: 72° iterations: 5

Cesàro Curve Star



ω : F--F--F--F
 p : F \rightarrow F-F++F-F
 angle: 72° iterations: 5

L-systems as a Formal Language:

The absolutely crucial difference between L-systems and normal Chomsky grammars is that at each step a production rule is applied to *every* symbol in the string. In Chomsky grammars only one production is applied at a time. As it says in *The Algorithmic Beauty of Plants*, “This difference reflects the biological motivation of L-systems.”⁴ If the state of an organism can be said to be ‘updating,’ it is doing so in parallel. As can be seen by Figure 26, this parallelism affords extra power. A context-free L-system can hence generate languages that a context-free Chomsky grammar cannot.

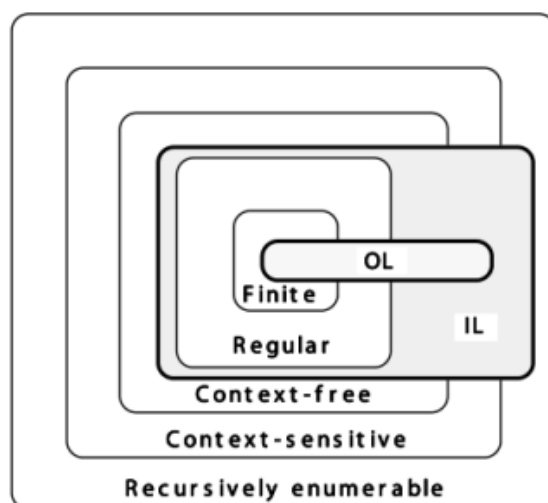
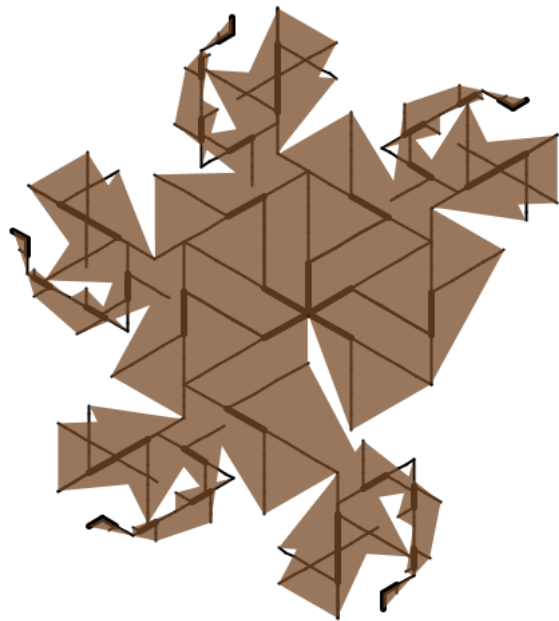
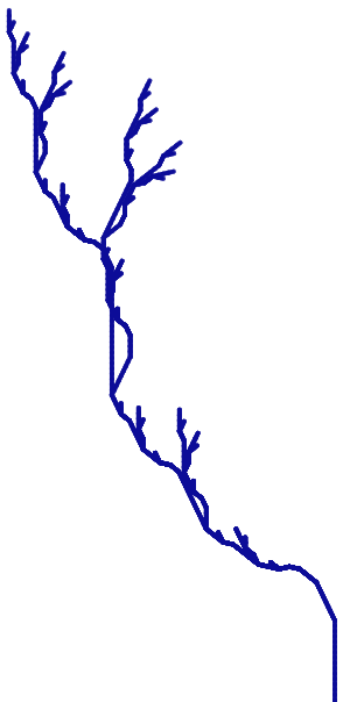
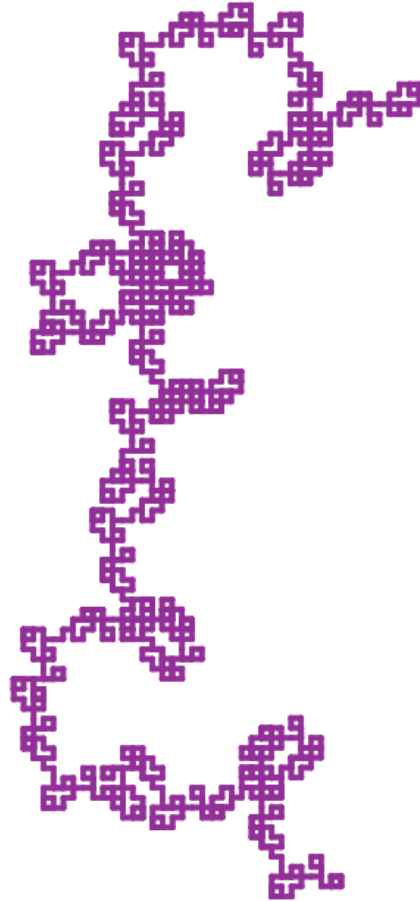
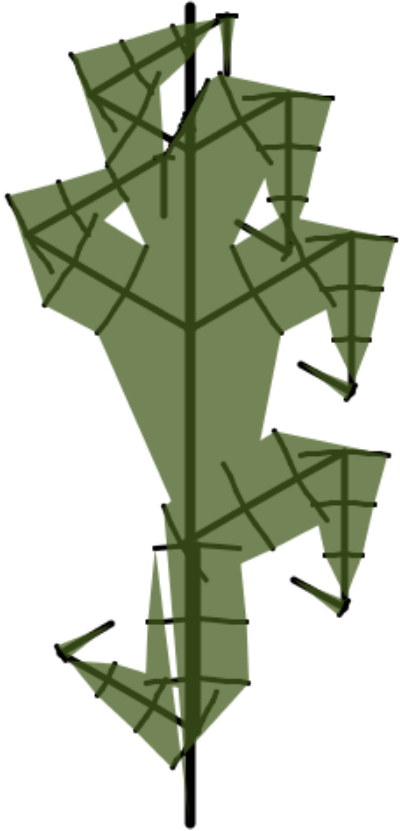


Figure 26: Context-free L-systems (0L) and one-sided context-sensitive L-systems (1L) as they exist in the formal language hierarchy.

⁴ Prusinkiewicz and Lindenmayer (1990), 3.

“Abnormal” L-systems Gallery:

Sometimes things don't go as planned.



Method: Approach, Algorithms, and Tools Used

This section explains in detail the method of my approach and the workings of the FormaLeaf system with information concerning tools used given throughout. The explanation is divided into four phases: Phase One consists of the collection and preparation of input leaves. Phase Two explains how computer vision techniques were applied for shape analysis. Phase Three concerns the parametric, polygonal L-System representation and the idea of “template leaves.” Phase Four describes the details of Search Mode, including how similarity (fitness) is evaluated. Throughout the following explanation I will provide some detail as to my implementation, usually in the form of mentioning which classes take care of what and what functions they use to do it. Unless otherwise mentioned, the classes are from my code—see the Appendix.¹

Some screenshots of the interface taken throughout development in order to explain certain parts of the program in this report may have unfinalized elements—this generally accounts for U.I. inconsistencies or strange looking values in the images.

Before explaining the phases in detail, I present a broad and brief system overview, a description of my overall development strategy, and some comments on why Processing was an appropriate platform for this project.

¹ Of the code included in the Appendix, all written classes are my own except for the Slider class, which was copied and modified slightly from an example on Processing.org.

System Overview:

The four phases are not sequential but rather feed into each other at different points. Here is flowchart illustrating how the system as a whole proceeds.

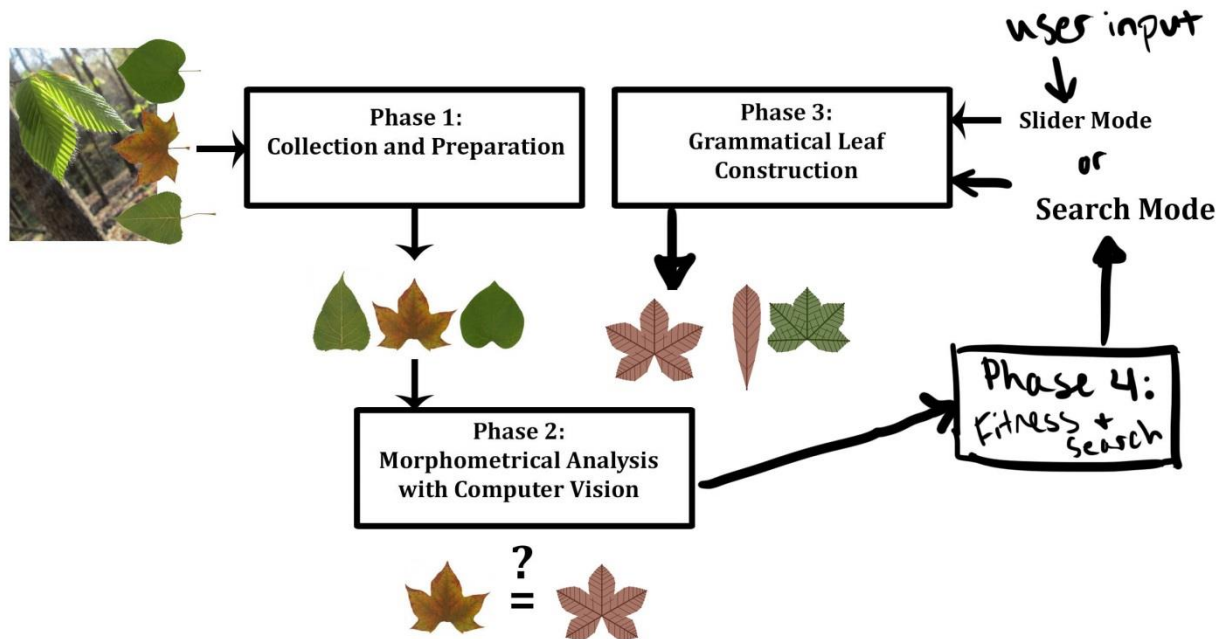


Figure 27: Flowchart illustrating how the Phases described in the report interact.

As the flowchart shows, the construction of the L-system leaves can be left up to the computer or done by manual input.

Project Development Strategy:

Actually getting things done is hard. The main project strategy I conceive of as “interface development.” The idea behind this is that by focusing my coding efforts on increasing my ability to manually tweak and control what appears on the screen, in the process I ended up building a platform on which I am able to:

1. Easily produce images with which to explain things in the write-up. This streamlines the writing process, which is the hardest part. One of a thousand and one reasons why Processing is awesome.
2. Learn through experimentation how best to try and automate something that looks either good or reasonable. By being able to view and adjust in real time and tandem both the graphic results and their associated quantitative measurements in Slider mode, hypotheses concerning what adjustments to make to the whole system are easier to come by.
3. Stare at leaves in two different visual modes—the normal way which shows the leaf image and the L-system, and in Vision mode, which shows what the computer sees.

Pithily, a fun way to approach leaf development is through software development!

Processing as Project Platform:

Processing² was an excellent choice for my project, as it is at once simple, flexible, and powerful. Its draw function serves as a built-in redraw cycle, which is necessary for any interactive system. It makes a number of interface-related requirements very easy, such as mouse and keyboard input, drawing shapes to certain parts of the screen, and loading and saving images. Because it was built for making graphical programs, it has well-designed functions to manipulate visual properties. The developers take pains to make both the rendered graphics and the code aesthetically pleasing. It's got a slick IDE (I used Processing3) and extensive library support. And of course, it's all based in Java—all the goodies of the official API are just an import statement away. Although I am not aware of many attempts to build interface-based research systems like FormaLeaf in Processing (though they surely exist), I found it to be a platform well-suited to my purposes. This is also due to my familiarity with the language, as Processing was my first introduction to programming. Obviously I'm biased, but I think it's unparalleled as an educational tool.



I began on Processing 2 but switched to the more recent Processing 3 shortly into my development cycle. The program currently uses the default 2D renderer.³

² <https://processing.org/>

³ FX2D may have some benefits but I need to do more testing. Using P2D (the 2D OpenGL renderer) results in egregious graphical issues.

Phase One: Collection and Preparation

Step 1: Specimen Collection

The majority of leaves were collected during the fall semester. Samples were either taken off of living plants or from the ground. More often than not, leaves were removed directly from plants because leaves from a living plant were in better condition than fallen leaves, which had often started decomposing. I thought carefully about each sample before removing it so as to not cause unnecessary damage to the plants. At least by removing the leaves during autumn the plants were already preparing metabolically to shed them—leaves cost lots of resources to produce so I would feel a bit worse removing them in spring before they've even had the opportunity to soak up the summer sunlight! Plants were thanked for their generosity when I remembered to do so.

Step 2: Scanning

All leaves were scanned in a Canon® imageRUNNER 3245i⁴ as color JPEGs at 300x300 DPI. Both sides of each leaf were scanned for consistency. Prior to working out details of the project it was undetermined whether or not some sort of visual line detection would be attempted on the venation patterns

themselves. This would have been easier to do with the *abaxial* side (under-side) of most leaves, as the veins are

more distinct in these images. As Theophrastus says, “In most trees the upper surfaces are greener and smoother, as they have the fibres and veins in the under surfaces, even as the



Figure 28: Canon® imageRUNNER 3245i

⁴ Also the scanner I used for all book-sourced images in this project.

human hand has its ‘lines’[...].”⁵ Though line detection was ultimately not used, it is still nice to have both sides of every sample. Some of these sample images were used as illustrative examples in Part I.

Step 3: Digital Image Preparation

All leaf images were rotated, cropped and cleaned up in Adobe Photoshop CS6 with the help of a Wacom Bamboo Pen tablet. All leaf images were rotated so the apex is pointing up with the mid-vein as straight as possible. Cropping was done by eye, attempting to make the leaf take up nearly the whole resulting image with only a small amount of white space on the top, bottom, and sides. The closer the crop, the larger the leaf itself appears in the final Processing display of the results. The images were only cropped, not resized to be consistent with each other—a smaller leaf means a smaller image.⁶ Stray marks, smudges, and most⁷ artifacts from the scanner were removed with the Brush Tool, as was the petiole of each leaf.



⁵ Theophrastus, I.X.2, 69.

⁶ However, leaf images are ultimately scaled down before the contours are found—see discussion of size and scaling issues in Phase Three.

⁷ There are still some subtle gunky shadows left from the scanning on some of these images—this may show up during printing.

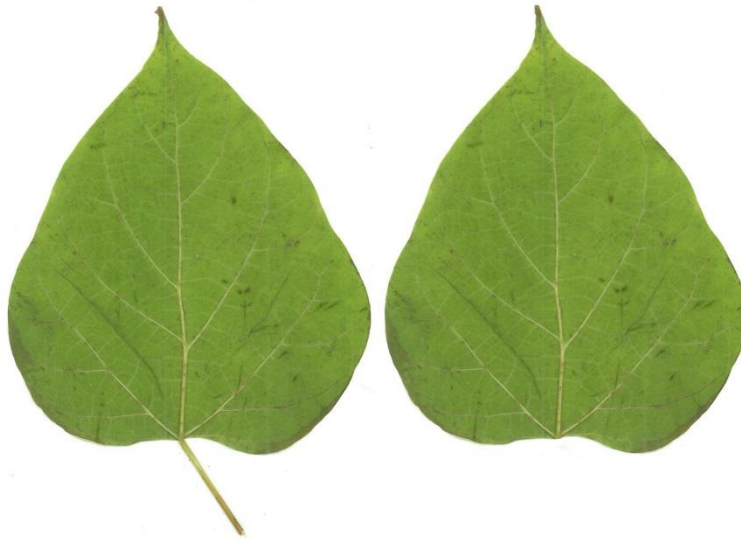


Figure 29: Leaf image before and after petiole removal.

Though digitally removing the petiole from the scanned image seems almost criminal, it simplifies the vision and L-system generation process. The system discerns the lamina shape, so it would have taken some extra work to get the computer to locate the petiole so it knows to ignore it for certain measurements. This is certainly not impossible—in fact, the visual system of the mobile leaf classification application Leafsnap has been programmed to ignore the petiole by looking for thin protrusions.⁸ Leafsnap also works on leaf images at any rotational angle, as a mobile classification application must be as flexible with input as possible. However, I decided not to build this into my system and instead chose to manually fix up my input images in order to focus my coding efforts on parts of the problem more central to my purposes. Another option would have been to not ignore the petiole but instead analyze it and build it into the resulting L-system. If the petiole was kept, the resulting L-systems all would have been a little different^{**}($t > \text{positive num}$). One reason I decided against this was due to the inconsistency of the sample quality. Some petioles were entirely intact because I ripped them off of the shoot right at the node where they were attached. Others were torn in an arbitrary place. Hence, it isn't as if a generated

⁸ Kumar *et. al.* (2012), 6.

system which matches one of these arbitrarily torn and scanned petioles is really accurately representing the leaf and its entire petiole length. As a final reason for removal, the petiole is the connection site to the rest of the plant—yet we choose here to look at the leaf as an individual entity! Erasing the petiole from the image is like cutting an umbilical cord: the leaf becomes self-contained, with the omphalic base being the only indication it was ever the part of something larger.⁹

I will also say that while there is an impulse of computer science to automate as much of a process as possible in order to ‘save time’, there is also a benefit to working with and looking closely at each sample. This is what manual cleanup and preparation allowed. Obviously this is only feasible when the sample set is very small, but well-cleaned data makes a world of difference.

⁹ Just kidding, it was for practical reasons.

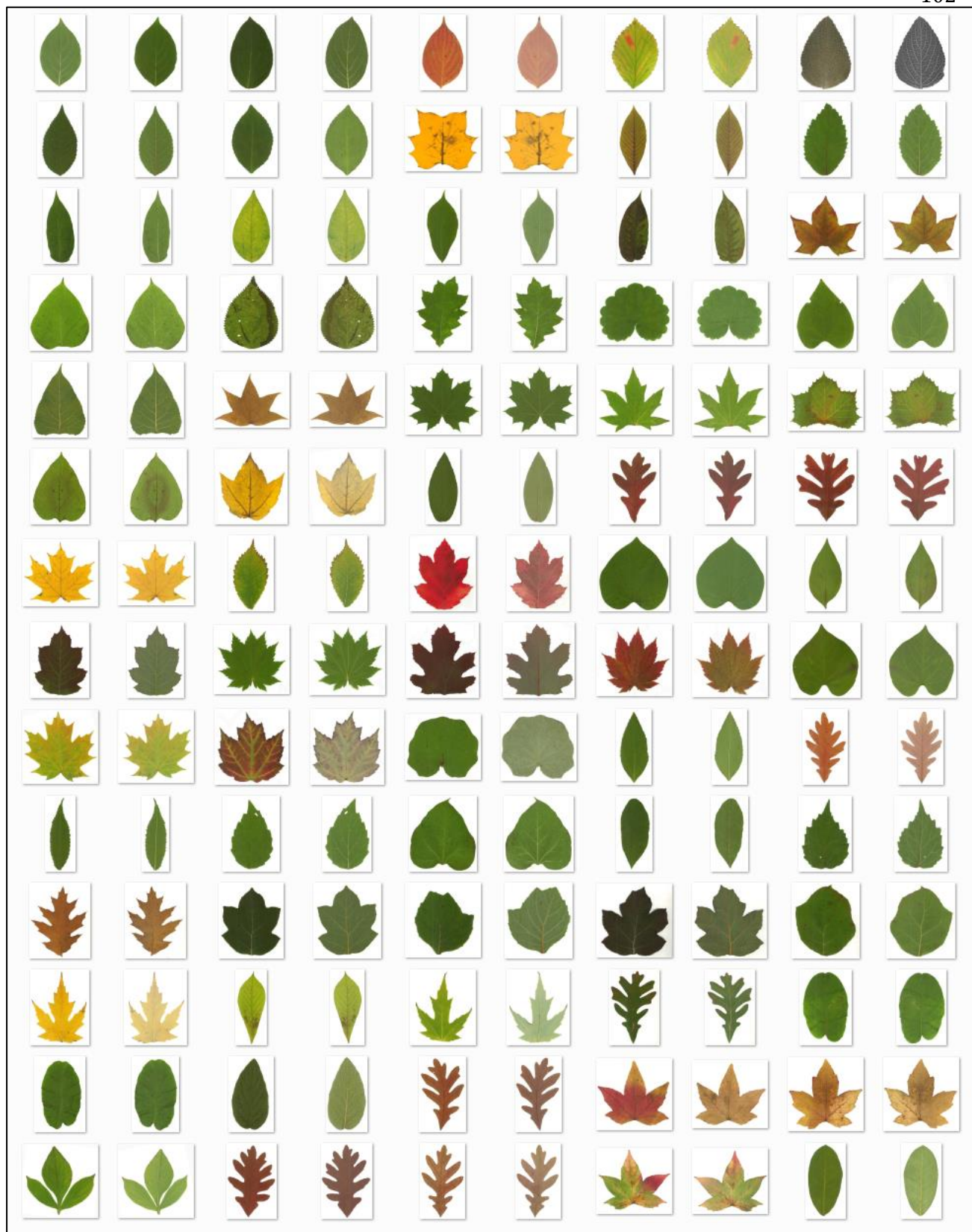


Figure 30: Final sample pool of 70 leaves in no particular order.

Phase Two: Morphometrical Analysis with Computer Vision

Phase Two pertains to the analysis of leaf shape done using computer vision techniques. All of the described measurements are done on both the input leaf image as well as on the image of every candidate leaf generated by an L-system over the course of the search process. This part of the system is handled by the ImageProcessor class, which performs the exact same visual analysis when given an image of a real leaf or an image of a fake one. As the ImageProcessor discovers aspects of the shape under analysis, it stores this information in a Leaf object. It is two Leaf objects—one in the form of the subclass SysLeaf—which are ultimately compared in order to assess shape similarity (fitness). The morphometrical information is also reported in the panel on the left side of the screen. Vision Mode allows the user to see illustrative representations of what the computer is “seeing,” such as contours, convexity defects, bounding boxes, etc. Many of the figures here in Phase Two were made in Vision Mode.

The tool used for visual analysis was OpenCV, a robust open-source computer vision library started in the 1990’s by Intel Research and now supported by its own non-profit foundation. While OpenCV is more commonly used with C++ or Python there are (thankfully) also Java bindings. In order to both install OpenCV for use with Processing as well as convert Processing PImages to the OpenCV image matrices I used Greg Borenstein’s “OpenCV for Processing” library.¹⁰

It is important to stress that the computer vision portion of this project is an *application* of pre-existing and pre-implemented techniques to the specific problem leaf shape analysis. The majority of applications of



¹⁰ <https://github.com/atduskgreg/opencv-processing>

computer vision to leaf shape have been for species classification purposes. This is most commonly done with machine learning algorithms. While I will sometimes attempt to explain how certain important OpenCV functions are working, my main focus will be on the details of how I used them.¹¹

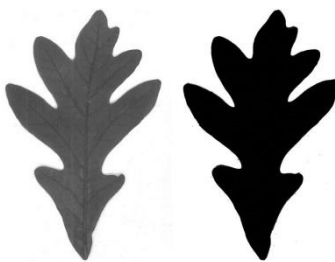
Step 1: Finding Contours

The contour of a shape is essentially its outline.



Finding the contours in the leaf image first requires a binary mask, which is made by running the OpenCV `ImgProc` threshold function¹² on a grayscale version of the leaf image.

$$\text{outputImage}(x, y) \begin{cases} 255 & \text{inputImage}(x, y) > \text{threshValue} \\ 0 & \text{otherwise} \end{cases}$$



¹¹ The most difficult part of using OpenCV is dealing with its idiosyncratic data structures. Many lines of code are dedicated to getting coordinate data in the right form of a list. The functions take and return bizarre combinations of these many structures. OpenCV is ridiculously useful but also annoyingly unintuitive to program with.

¹² Technically the code calls Borenstein's "OpenCV for Processing" thresholding function, which itself just calls the OpenCV `ImgProc` function. It could just as well have been done by calling the normal OpenCV functions directly, but my program arbitrarily thresholds the image before doing matrix operations. Once the matrices are set up, all image processing is handled by Java OpenCV functions until a final `Mat` to `PIImage` conversion (done with Borenstein's library) is done to display the morphometrically processed information.

A threshold value of 220 is used because this value proved sufficient for all the inputs and the requirements of contour extraction—a more complex adaptive threshold was not necessary. Pixels which exceed the threshold value are turned white, while all others are turned black. Following this, OpenCV runs Canny edge detection on the thresholded image in order to find the contour.

Because OpenCV's `findContours` function returns *all* of the contours in the input image, the contours which are not the leaf must be ignored. My simple solution to this was to sort the found contours by area and pick the second largest one, as the entire image's outline (going around the image border) was always of greater area than the leaf. Sorting by area also means small contours from holes in the lamina or scanning artifacts are ignored—thus the leaf's contour is isolated successfully. The contour is stored in the associated Leaf object and can thus be used for later calculations of in the fitness function which require direct comparison of contours.

Step 2: Measuring Lamina Dimensions

The next step is to take basic dimensional measurements. Because the leaf images are scaled down before visual processing, these measurements are all in pixels.¹³ The most useful tool for this is a bounding box, a rectangle the boundaries of which are determined by the vertical and horizontal extremities of the leaf contour.

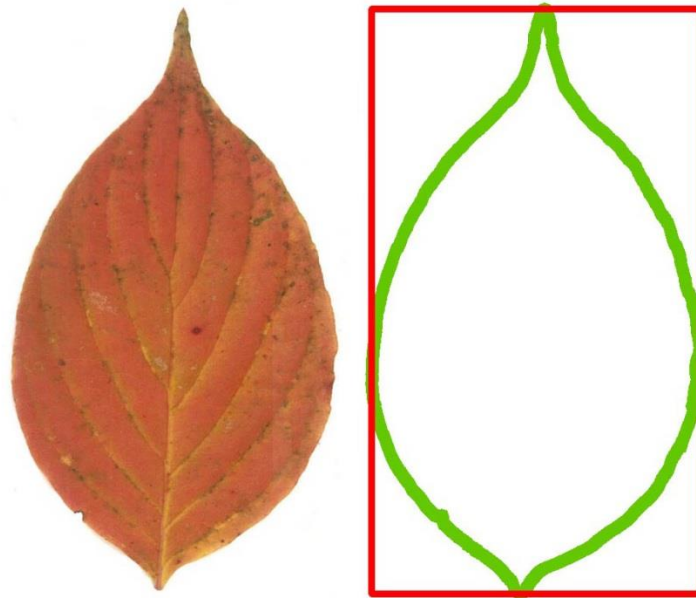


Figure 31: Bounding box shown in red. Using the dimensions of the rectangle, the system stores and displays the following information about the leaf on the left:

Lamina Length: 771.0

Lamina Width: 435.0

Lamina L:W Ratio: 1.7724138

In this way, lamina length and width are determined using OpenCV's boundingRect function and from these the L:W ratio is calculated. It was mentioned in Part I (see the section on patterns of venation) that the average palmate leaf tends to have a lower L:W

¹³ Prior to my decision to have the program scale down the leaf images before contour extraction (the rationale behind this is explained later), I toyed with the idea of displaying the actual leaves' measurements in inches or centimeters—the system would just have to convert from pixels to the chosen unit. The images were all scanned at 300 DPI, so this wouldn't have been too hard. However, this measurement would have been purely for the scientific curiosity of the user, as it is far less useful for L-system comparison purposes. As the system is now, the exact values of the pixel measurement depend on the size of the window (by default 1280x800).

ratio in comparison to the average pinnate leaf. This ratio is thus a valuable metric for assessing similarity so at the very least the correct venation template is found.

While the bounding box can be used for measuring lamina length and width, determining leaf area is not done by multiplying these as there is obviously non-leaf space within the rectangle. Luckily, lamina area is not difficult to determine as OpenCV has a contour area function (it uses Green's theorem). Similarly, OpenCV's `arcLength` function was used to get the contour perimeter.

Step 3: Determining Shape Class

As addressed in Part I, parts of modern leaf terminology are defined quantitatively. One of the first things I wanted the system able to do is apply terminological labels to the input leaf. A simple label to apply is that of the overall shape—the 2009 *Manual of Leaf Architecture* defines some different shape classes by where on the lamina the widest point falls. The *Manual* gives five shape classes: Ovate, Elliptic, Obovate, Oblong, and Linear. See Figure 32 for an image of the first four. I decided not to include the “oblong” label as not only is determining parallelism a bit more complicated than just finding the widest section of the leaf, but also because almost none of my leaves were very oblong anyway. Similarly, none of my collected leaves had a L:W ratio high enough ($\geq 10:1$) to be “linear.” However, the parametric L-system leaves do sometimes reach this value so it might be worth including at some point.

As it is, the system labels every leaf as being Ovate, Elliptic, or Obovate. The `ImageProcessor`'s `findShapeClass` function finds the “widest fifth” of the leaf and saves it to the `Leaf` object. The `Leaf` object uses this integer to determine the appropriate label and

stores it as a String attribute. As for why the Leaf object is the piece of the system actually applying the label, I did this mostly because I wanted the ImageProcessor object to return quantitative information. The “widest fifth” also gives more information than does the label (as it can be any of five possibilities instead of three) so it’s hypothetically useful for this to be stored in the Leaf object were it used as a more precise fitness metric.

The location of the widest point on the leaf is determined by first finding the points of the contour that lie on the left and right sides of the bounding box, as these points are horizontal extremities. For every extremity point on the left and right side, a point lying on the contour on the exact opposite side of the leaf is also recorded. Following this, the Euclidean distance between every extremity point and its opposite point is measured. The pair with the longest distance is selected as the widest area of the leaf. Finding the widest point is surprising convoluted (in my implementation, anyway).

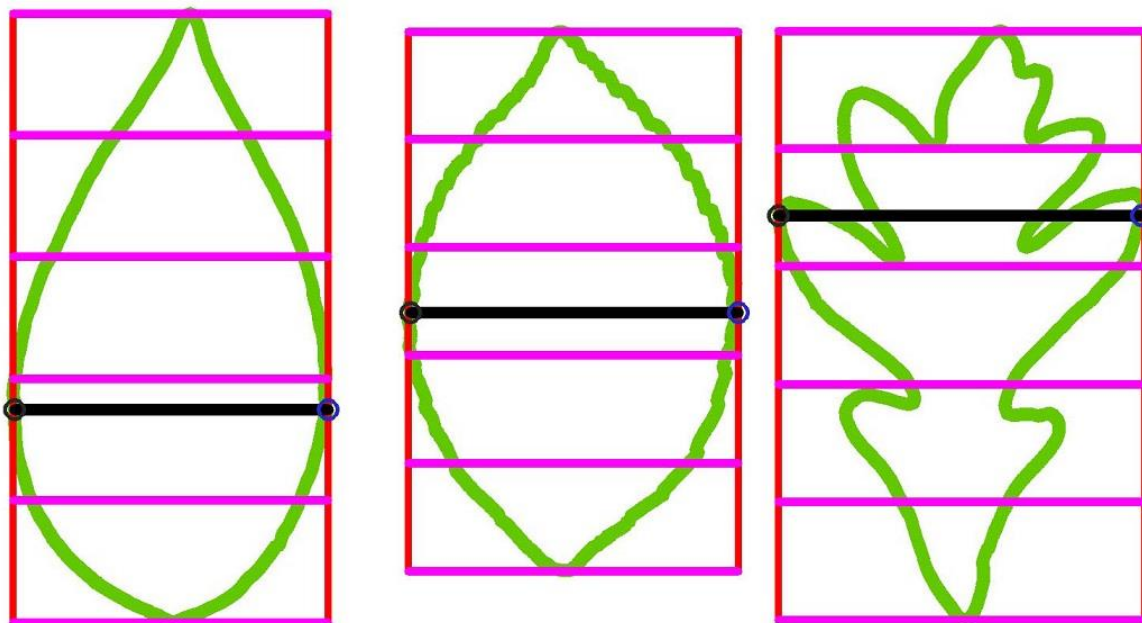


Figure 32: *Left—Ovate: greatest width in bottom 2/5th*
Center—Elliptic: greatest width in middle 1/5th
Right—Obovate: greatest width in top 2/5th

The lamina is divided into fifths as the purple lines in Figure 32 show. The program checks each section for the widest point (either a left or right extremity, all that matters is the y-coordinate) until it is found, at which point it returns an integer representing which fifth contains it. 1 designated the top section and 5 designate the bottom. The Leaf object then determines the label based on this integer as per the criteria described above.

The greatest width line and the left and right points circled in Figure 32 are displayed in Vision Mode. The code which draws the bounding box and fifth divisions, however, is commented out as they are less directly informative to the human viewer. Another improvement to the whole program would be to make Vision Mode more uniquely interactive—the user could press different keys or click on things in order to toggle which measurement visualizations appear.

Step 4: Approximating Apex and Base Location

It's not quite accurate to say that this step finds either the apex or the base because it's such a simplistic approximation. However, it is this "base" value of the real leaf image which is then used to determine the vertical placement of the L-system leaf's actual base on the right-hand canvas, so Step 3 will retain its title. This is an obvious area for a more nuanced analysis in the future—finding the *actual* location of the apex and base points (probably by using convexity defects for leaves with indented apices or bases) would make it easier to apply an apex/base shape label to an input leaf. While this would make the labeling side of the system more robust and give another qualitative point of comparison when evaluating similarity, it was not an important priority as it would make L-system placement only marginally more accurate.

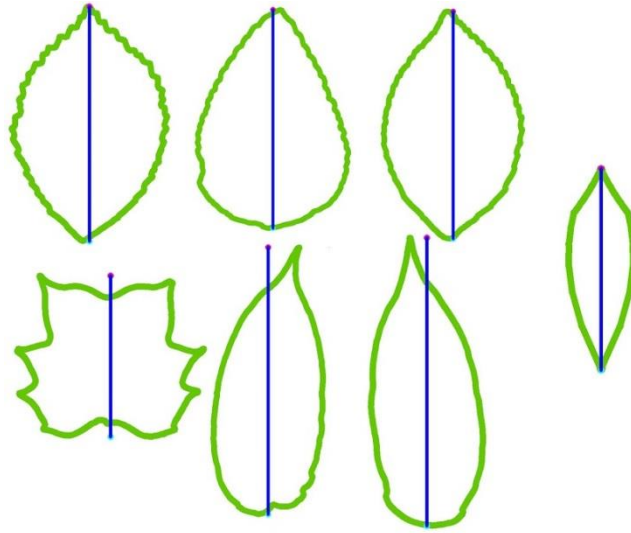


Figure 33: The described approach is a decent apex and base approximation for the three leaves up top but is not so good for the three on the bottom. It works perfectly on the L-system generated Pinnate template leaf to the right.

Hence, the rudimentary apex/base location simply divides the bounding box in half with a vertical line right down the center, saving the top point of the line as the apex and the bottom point as the base. It thus assumes a perfectly straight mid-rib on a perfectly symmetrical leaf with an apex and base which lie on the bounding box outline.¹⁴ Again, this is painfully rudimentary but the only thing this measurement is used for in the end is the vertical placement of the L-system leaf's base on the right-hand PGraphics canvas. While this vertical line does not *really* represent the mid-rib, if the system could find the actual apex and base points and draw a more accurate connecting line¹⁵ it could measure the angle or distance between the “ideal” mid-rib and the one closer that of the leaf, which could be a useful metric.

¹⁴ This actually means it works perfectly for any L-system generated leaf without a concave base (true of the Pinnate and Palmate templates).

¹⁵ Though this line would be straight it might suggest the presence of a curved mid-rib.

Furthermore, the OpenCV Point object in which the “base” is saved gives the coordinates *within* the input image. This doesn’t cause a problem when it comes repositioning tall/long leaves, as their bases end up at the bottom of the screen anyway. Very wide leaves, however, end up repositioning the L-system base a little higher than desirable (but nothing so off as to make it unusable, especially when using the Palmate template). Fixing this would just require finding the right way to take input image size (and the base location within it) into account while finding the absolute window location. Doing the repositioning at least gets the generated leaves for the wide leaves into the center of screen where they have more space for lobes which droop below the base, so it isn’t a huge issue.

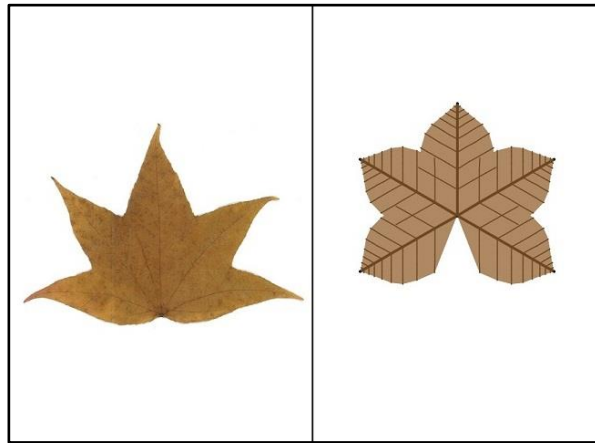
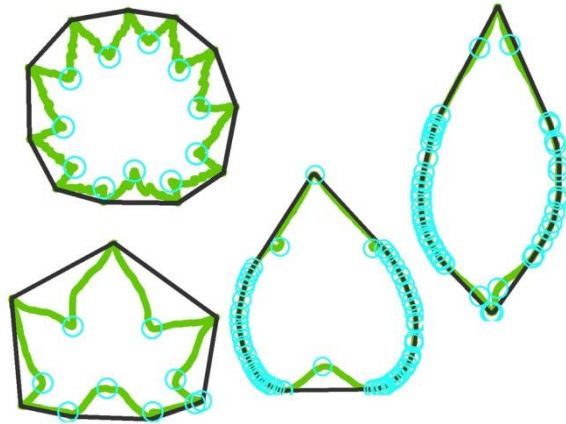


Figure 34: For wide leaves, the base of the generated leaf does not properly line up with the bottom of the input leaf.

Step 4: Counting Lobes

Lobe counting is so inaccurate in its present state that the system doesn't yet report it in the left-hand information panel (which is actually pretty silly, to be honest). The lobe "estimate" however does factor into the fitness function, as leaves with more lobes usually end up with a higher number. I'll explain how lobe detection is currently working and what steps can be taken to make it better.

The OpenCv function `convexityDefects` finds the points which are concave in relation to the convex hull of the contour. The blue circles indicate that the function

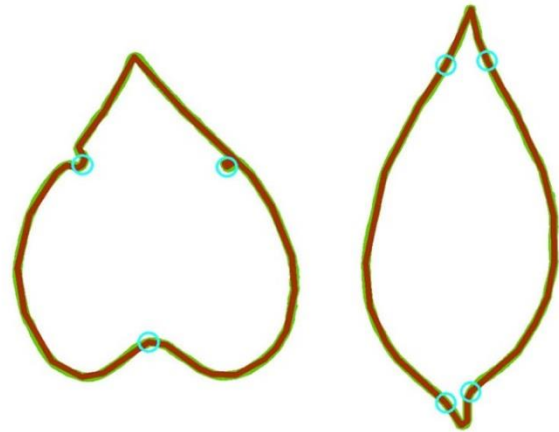


found a defect. As is clear, by default it works much better on actual lobed leaves than it does on smooth, entire margins. The convex hull for these leaves hugs the contour quite closely (that is, there are lots of convex points in the hull) so many points are thought to be concave.

The step taken to reduce these extra points (those right two leaves *should* be labeled as unlobed) is to use a polygon approximation as the contour which is passed to the convex hull function. By making the actual contour and thus the resulting convex hull blockier, the number of possible convexity defects to be found is reduced since it checks between convex

points. This helps a lot with those leaves with rounded margins, as the image to the right indicates.

The next step (which the program does not take) would mean going through each of the convexity defects and filtering them based on some kind of measure—probably distance from the convex hull point at a certain angle from the point. Hence as it



is it can't really discern lobed from unlobed leaves, which is really more crucial than the number of lobes when it comes to evaluating template similarity. Also, it consistently underreports lobe number for the PinLobed template for some reason.

Working through OpenCV's utterly complicated data structures often means looking for help on the internet, where lots of helpful folks post tutorials, demos, and examples of their image processing endeavors. I found it interesting that the most common application of OpenCV's convexity defect detection functionality is to use it to count the number of fingers on a hand being held up to a camera. Applying it to leaf lobing is a similar task morphologically—it is not for nothing that certain leaves are known as “palmate” or “digitate!” Thoreau writes fittingly:

Is not the hand a spreading *palm* leaf with its lobes and veins? [...] Each rounded lobe of the vegetable leaf, too, is a thick and now loitering drop, larger or smaller; the lobes are the fingers of the leaf; and as many lobes as it has, in so many directions it tends to flow[...]¹⁶

¹⁶ Thoreau, 548.

These online examples are helpful as I try to figure out how best to filter the convexity defect points to just the important ones (such as trying polygon approximation). It's a shame this part of the system doesn't work as well as it could yet, as I think it is one of the most useful metrics for assessing the similarity of a template to an input leaf. If lobe number was accurately discerned and the weighted highly in the comparative fitness function, the system would be far more likely to reject unfitting templates on the basis of their lobation. Boxy dimensional measurements really aren't enough to see structure. At least leaves with *more* lobes get labeled with a higher lobe count—hence it does factor in to the present fitness evaluation in some form.

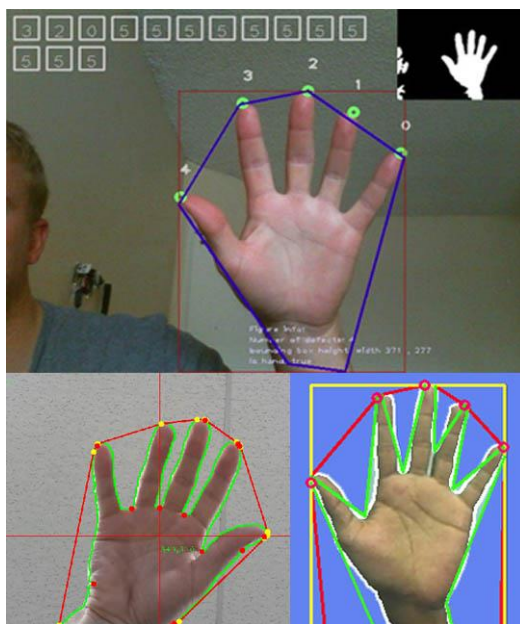


Figure 35: Fingertip counting with OpenCV¹⁷

¹⁷ *Top:* <http://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/>
Left: <http://stackoverflow.com/questions/18143077/computer-vision-filtering-convex-hulls-and-convexity-defects-with-opencv>
Right: <http://www.codeproject.com/Articles/782602/Beginners-guide-to-understand-Fingertips-counting>

Phase Three: Grammatical Leaf Construction

Phase Three deals with how the L-system leaves which appear on the right-hand side of the screen are generated. This section deals with how the parametric L-system grammars were designed to represent leaf structure and shape and also how these generated leaves were made to appear in window.

As was mentioned in the section on L-Systems, the geometric information for the model leaf's surface polygon construction is built directly into the L-system. This means the grammar contains specific symbols which designate when in the interpretation a polygon does the following:

begins:	{
ends:	}
has a vertex:	.

Hence, the drawn shape which appears on the screen changes in tandem with the structure of the grammar and the values of the parameters because its geometry is an intrinsic part of the system. The resultant shape is visually analyzed in the exact same manner as the input leaf image in order to grab morphometric information about the generated leaves. That is, everything explained in Phase 2 applies to these leaves as well.

Template leaf grammars are structured in such a way as to build more complex leaf shapes out of simpler ones. The mid-rib for the Pinnate template is used as one of many primary veins of the Palmate template while in the Pinlobed template it becomes the secondary veins. Because simple/entire leaves can be used to model lobes, for convenience I refer to these overlapping surfaces as "lobelets," a portmanteau of "lobe" and "leaflet," leaflets being the separate laminar areas found on compound leaves. That is, in these models, the leaf form is so universal that even *leaves* are made of leaves.

These grammars also contain multiple parameters, with complexer leaf shapes having more. Because the L-systems consist of strings, parsing the parameters requires the evaluation of mathematical expressions. Java can't do this on its own so Peter Lager's QScript library¹⁸ was used for this purpose. The parameter parsing is easily one of the buggiest parts of the whole system.

There are three leaf templates "officially" built into the system: *Pinnate*, *Palmate*, and *PinLobed*. That is, while there are some other templates in the code, only these three are generated during search and without editing the code only these three can be viewed and manipulated in Slider mode. The Results and Future Work sections will discuss some other possible templates as a way of demonstrating the process of template creation. Here, however, we look at the workable templates constructed for three common leaf forms. Each one has some interesting structural characteristics that give an idea of how L-systems can be designed to represent different forms.

¹⁸ <http://www.lagers.org.uk/qscript/>

Pinnate Template

The Pinnate template is an unlobed leaf with pinnate venation. Shown below are some images of leaves on which the template was based alongside some random candidates generated from the template.

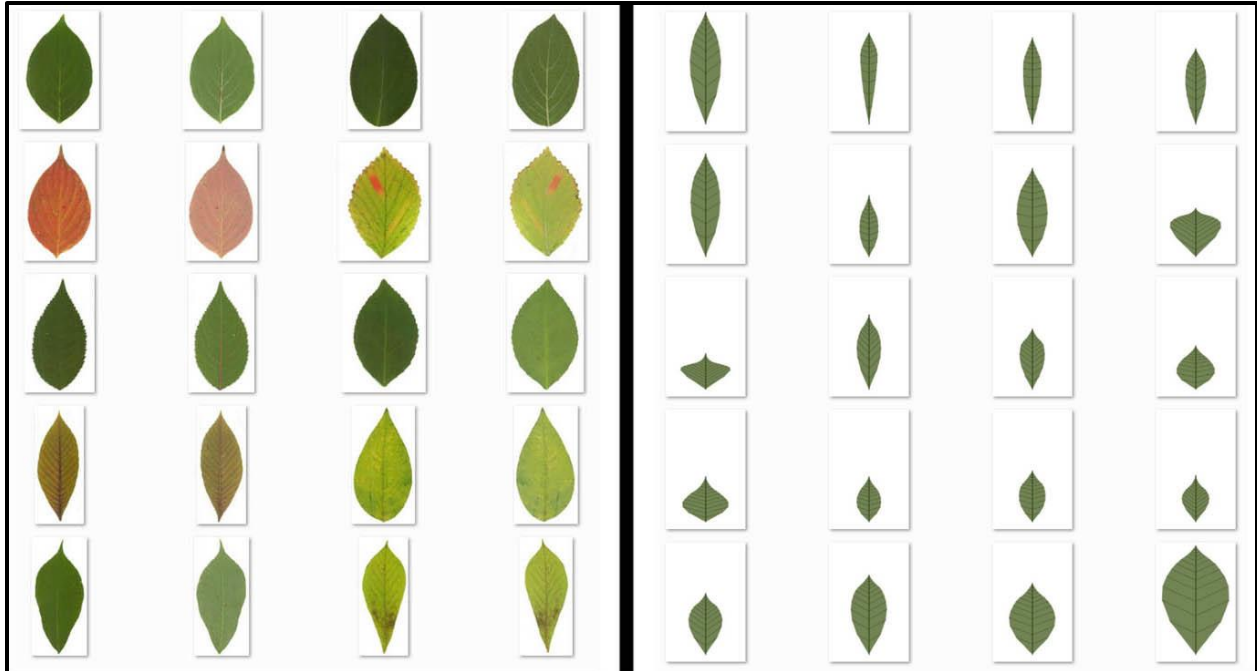


Figure 36: *Left*—Some unlobed pinnate samples. *Right*—Randomly generated candidates from Pinnate template

Because the leaves on the right are random, some have parameter value combinations that make them especially small or strangely shaped. Hence, the whole purpose of the search process is to compare the generated leaves against the input leaf image and then mutate the candidate leaf pool in hopes of getting closer. As it is, this image shows some of the different forms that can arise from this template. The biggest improvement to the Pinnate template would be to modify it to get a greater variety of apex and base shapes. It has a hard time approximating rounder apices and bases.

The Pinnate template was the first one implemented in FormaLeaf. My starting point for all efforts in leaf grammar construction was the following example from *The Algorithmic Beauty of Plants*:

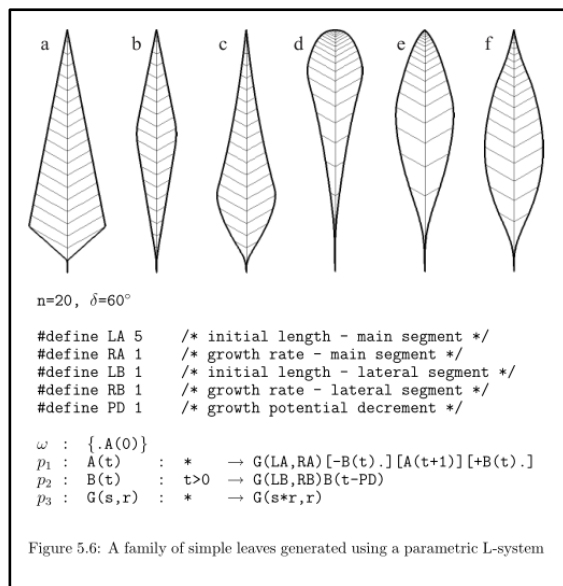
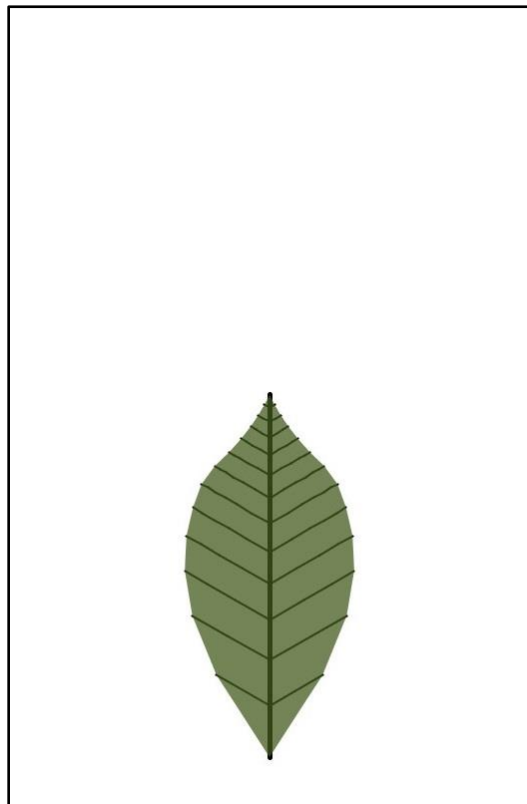


Figure 37: Parametric leaf surface models.
The Algorithmic Beauty of Plants (124).

This example also had a table of different parameter values to get the six different leaf forms above. While I retained the structure of this grammar, I modified all of the parameter ranges and constraints so the kinds of forms I get are quite different. Furthermore, the leaves in this example are all at 20 iterations, while my Pinnate template is capped at 14—this actually makes a big difference in both how the parameters affect the shape as well as the number of lateral veins on the leaf.

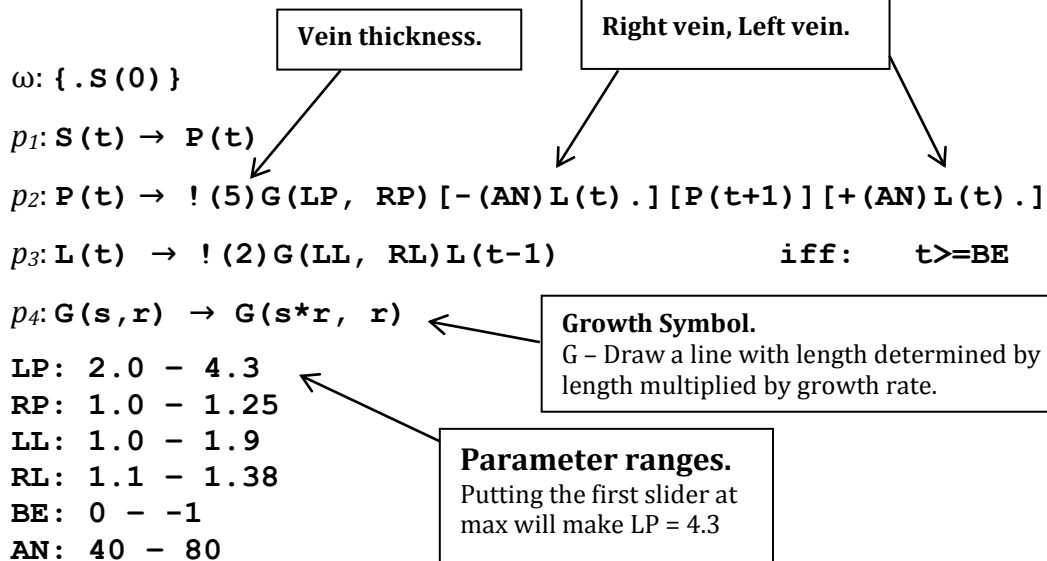
What follows is the L-system and parameter ranges for the Pinnate template, along with some explanatory notes.

Pinnate Template



Median leaf.

Templates start with median parameter values by default. Hence, leaf is small before slider tweaks/search.



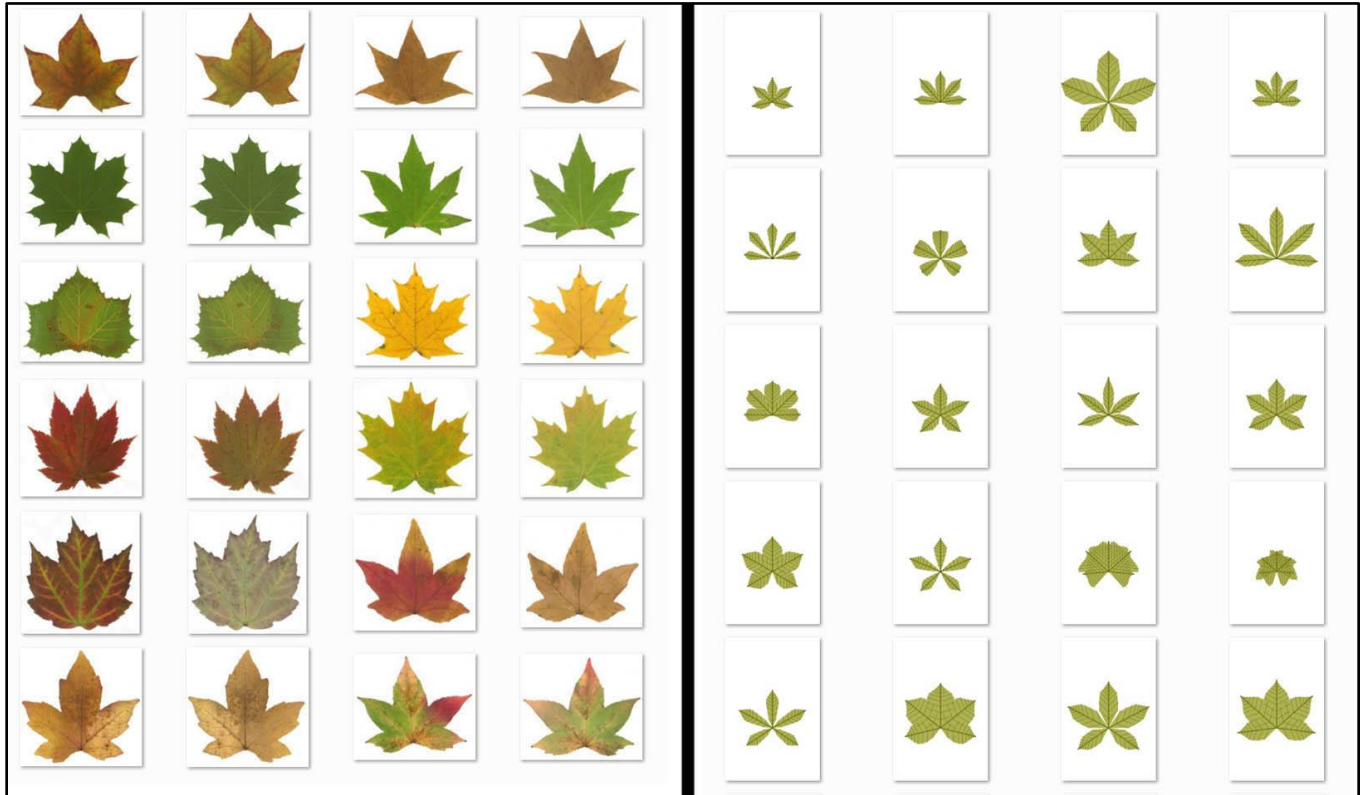
Parameters:

LP: Primary vein length. **RP:** Primary vein growth rate.
LL: Lateral vein length. **RL:** Lateral vein growth rate.
BE: Affects basal extension. **AN:** Vein branch angle.

Palmate Template

A better name for this template would make it clear that it is intended to be a palmate lobed leaf. However, sometimes the parameters end up making something less lobed and it's the only working palmate template anyway, so the name sticks.

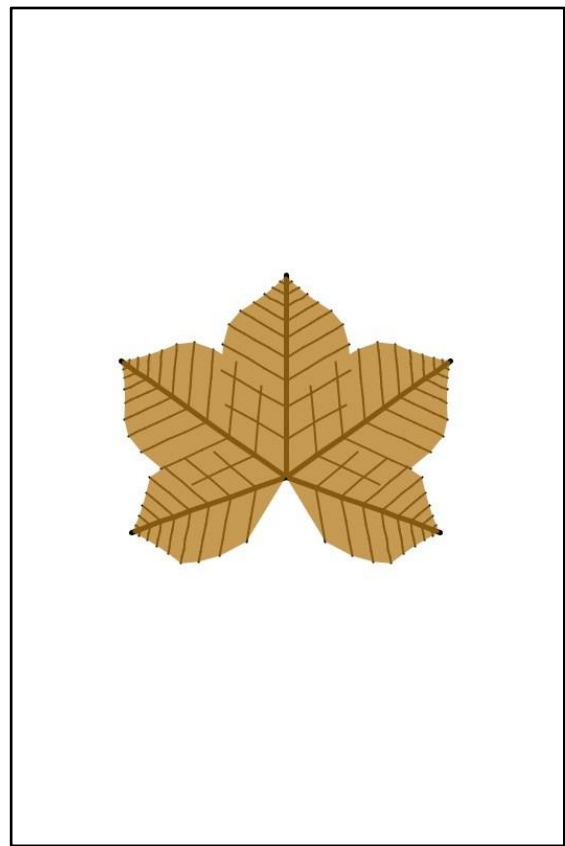
The main idea behind this template is using the previously developed Pinnate template as the lobes, which are arranged in a radiated fashion around the base. Since the lobes are modeled with another leaf model, I refer to these surfaces as "lobelets." The mid-ribs of the Pinnate lobelets hence become the multiple primary veins of the Palmate template. I found that overlapping surface models of entire leaves is the simplest way to model non-entire laminas ("entire" being the common botanical term for simple unlobed leaves with smooth margins). The biggest improvement to the Palmate template would be to parameterize the number of lobes, as right now it puts exactly five on every leaf.



**Figure 38: *Left*—Some lobed palmate samples.
Right—Some randomly generated candidates from Palmate template.**

The grammar was modified so the bottom two lobes are scaled down. As the samples suggest, the lower lobes are usually smaller. This was done by adding an extra production step which only the bottom lobelets must fire. This ‘delays’ their development relative to the others by a full iteration, which in this case acts as an effective scalar. This could also probably be parameterized somehow alongside the number of lobes.

Palmate Template



$\omega: \{ . S (0) \}$

$p_1: S (t) \rightarrow [+ (ANN) + (ANN) B (t)] . [+ (ANN) P (t)] . [P (t)] .$

Top Right Lobe

$\rightarrow [- (ANN) P (t)] . [- (ANN) - (ANN) B (t)]$

Bottom Right Lobe

$p_2: P (t) \rightarrow ! (5) G (LP, RP) [- (AN) L (t) .] [P (t+1)] [+ (AN) L (t) .]$

$p_3: B (t) \rightarrow P (t)$

$p_4: L (t) \rightarrow ! (2) G (LL, RL) L (t-1) \quad \text{iff: } t \geq BE$

$p_5: G (s, r) \rightarrow G (s*r, r)$

Extra Delay Production

LP: 1.2 - 3

RP: 1.1 - 1.25

LL: 1.1 - 2

RL: 1.0 - 1.5

BE: 0 - -4

AN: 40 - 80

ANN: 40 - 70

Parameters:
LP: Primary vein length. **RP:** Primary vein growth rate.
LL: Lateral vein length. **RL:** Lateral vein growth rate.
BE: Affects basal extension. **AN:** Lateral vein branch angle.
ANN: Primary vein branch angle (Lobelet arrangement angle).

Center Lobe

I find the Palmate template and its resulting models morphologically interesting for two reasons. Notice how when the Pinnate surface models overlap each other as Palmate lobelets, their lateral/second-order veins form crisscrossed patterns. While in the L-system model these veins are simply overlapping visually, it makes me wonder if it shows something about how anastomosis could proceed. That is, those sites of intersection could perhaps vaguely predict how/which secondary veins would connect. A comparison of Figure 39 to the leaf models above with this in mind is intriguing.

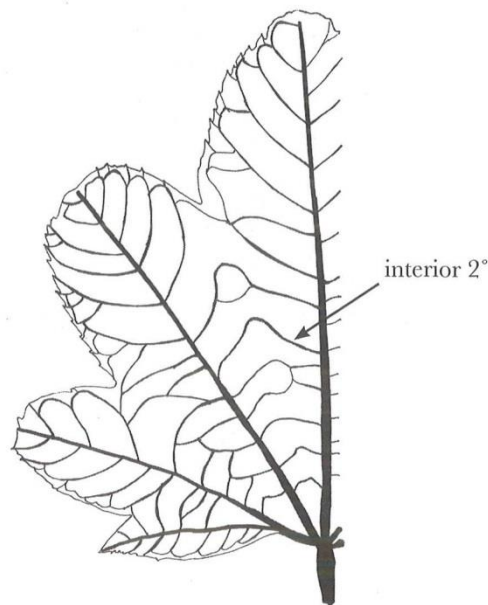


Figure 39: Vein anastomosis of second-order veins of lobed palmate leaf.
(Manual of Leaf Architecture, 54)

Secondly, the overlapping surface approach leads to an interesting way of thinking about the form of lobes themselves. Agnes Arber writes,

The analogy between leaf and shoot has been obscured by the technique of leaf description, which is based on the idea of the leaf as a member with an entire margin, which may be more or less indented or deeply cut, as if a pair of scissors had been employed upon it. De Candolle long ago pointed out that this method of visualizing lamina-form is liable to create a wrong impression[...] On this view, De Candolle would describe a pinnatifid leaf as showing fusion, for half their length, of

the lobes associated with the pinnate lateral veins. He goes so far as to suggest that this theory may possibly be applied to all leaves, even those that are quite simple and undivided.¹⁹

The method of modeling lobes I use for these templates embodies this perception of the leaf lamina. Regarding this analogy being “obscured by the technique of leaf description,” *The Manual of Leaf Architecture* defines lobes as being extrusions that have a certain proportional sinus (indentation) depth, with smaller sinuses being indicative of “teeth” instead of lobes. If you’re just looking at the margin (like the vision system of FormaLeaf) this is really all you can do to point to lobation. However if you look at the venation, you can see lobation expressed as fused foliar appendages coming off the mid-rib of the leaf, here acting like a shoot. Arber continues:

The point that matter to us is that de Candolle saw, even if dimly, that the leaf is a system comparable with a shoot, but in which the main and lateral veins and their associated leaf surfaces form a united whole, instead of being separable entities, such as the main axes and the lateral branches with their individual leaves, which in the aggregate makeup the shoot.²⁰

One way to imagine a leaf is as a flat tree with a horizontal network weaving a surface fabric in between its hierarchy. Venation is hence a crucial part of morphological understanding. In leaves they are internal and structural but are clearly correlated to the outline shape.

¹⁹ Arber, 83-84.

²⁰ Arber, 84.

PinLobed Template

This template is for pinnate lobed leaves and was based off of an oak leaf. The most successful part of this template is that the number of lobes is parametrized, which is great. Otherwise I am somewhat disappointed with it as leaf template, as it's getting close but the growth pattern of the lateral lobelets are really off—this makes the mutations hardly ever look like pinnately lobed leaves do. I think has something to do with the structures of the conditionals, the decrements, and the BE parameter. Oak leaves tend to be obovate, but more often than not this template makes elliptic leaves since the lobes closest to the base are always wider. This wouldn't be such a problem if the search wasn't frequently matching this template to unlobed pinnate leaves because they're so often elliptic!

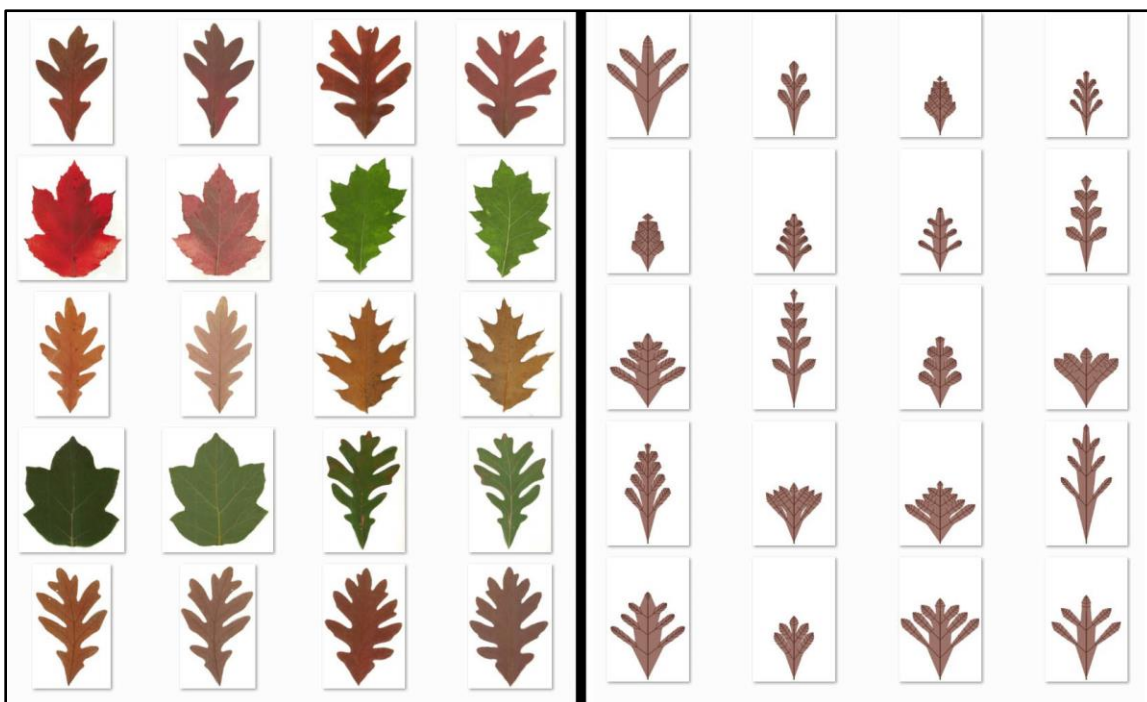
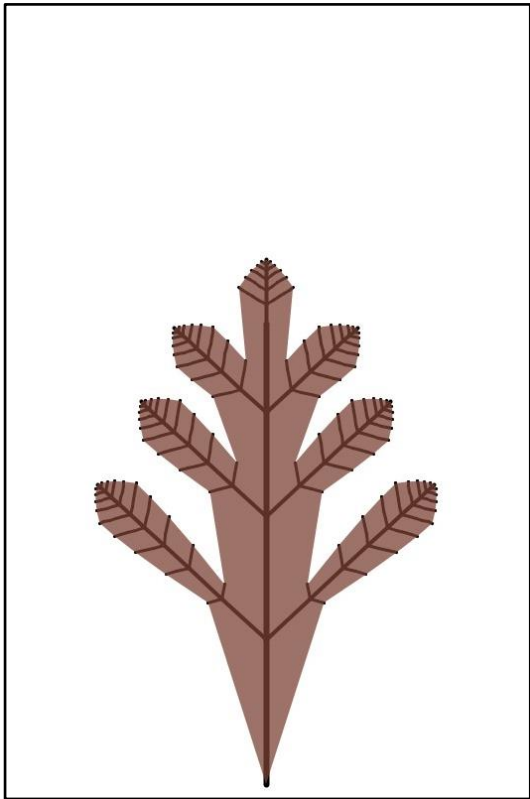


Figure 40: *Left*—Samples of pinnately lobed leaves, most oak.
Right—Randomly generated PinLobed leaves.

PinLobed Template



$\omega: \{ .S(0) \}$

$p_1: S(t) \rightarrow P(t, LO)$

$p_2: P(t, i) \rightarrow !(6)G(LP, RP)[- (AN)A(t)]$
 $[P(t+1), (i-1)] [+ (AN)A(t)]$

Draws i lobe pairs

iff: $i \geq 0$

$p_3: P(t, i) \rightarrow !(6)G(LP, RP)N(t)$

iff: $i < 0$

Extension for tip lobe

$p_4: A(t) \rightarrow !(4)G(LL, RL)[- (ANN)L(t-1) .] [A(t+1)]$
 $[+ (ANN)L(t-1) .]$

iff: $t \geq BE$

Draws tip lobe

$p_5: L(t) \rightarrow !(3)G(LT, RT)L(t-1)$

iff: $t \geq BE$

$p_6: N(t) \rightarrow !(4)[A(t+1)]$

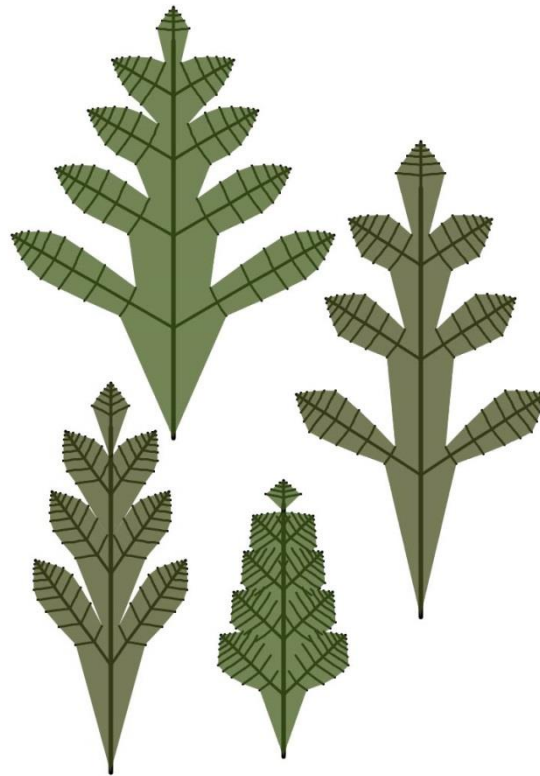
iff: $t \geq BE$

$p_7: G(s, r) \rightarrow G(s*r, r)$

- LP: 6.0 - 8.0 AN: 35 - 65
- RP: 1.1 - 1.26 ANN: 30 - 85
- LL: 1.0 - 1.4 LO: 1 - 4
- RL: 1.2 - 1.45
- LT: 1.0 - 1.4
- RT: 1.1 - 1.2
- BE: -1 - -4

Parameters:
 LP: Primary vein length. RP: Primary vein growth rate.
 LL: Lateral vein length. RL: Lateral vein growth rate.
 LT: Tertiary vein length. RT: Tertiary vein growth rate.
 BE: Affects basal extension. AN: Lobe branch angle
 ANN: Tertiary vein angle (Veins of the lobelets)
 LO: Number of lobe pairs.

However, while this template doesn't make for very good *leaf* models, the randomly generated candidates often look like interesting whole *shoots*—that is, they look like an abstract plant stem with multiple appendages. I find this morphologically interesting as it demonstrates how the leaf form can transition to resemble a whole plant (albeit, a very abstract flat 2D version). Plants might not be strictly self-similar, but branching structures do scale and we can see how the form of a leaf can become that of a shoot—or any other foliar form, really.



I attempted to address this in Part I when I wrote about the “universal leaf” and transitional forms and what not, but I think it’s clearer with the examples of the PinLobed template and the Palmate template given above. I really think this is part of the impetus behind the attempts of so many plant morphologists (Wolff, Goethe, de Candolle, Oken, and Arber, to name a few) to conceive of some abstract “universal” plant appendage. Arber

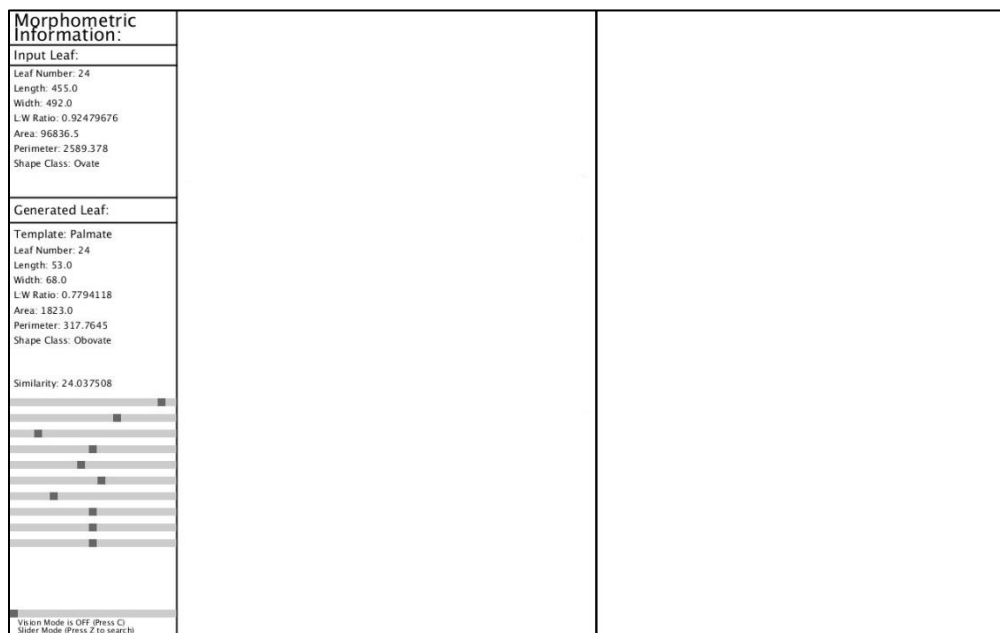
quotes Oken, who writes “The leaf is a tree of special form, a tree the branches or veins of which all lie in one plane.”²¹Arber bases part of her entire morphological approach on likening the leaf to a partial shoot and we saw above how this leads to a view that sees lobed leaf laminas as potentially decomposable into smaller foliar units.

When parameters are taken into account, that’s when we get abstracted *metamorphosis*. Contraction, expansion, all of the qualitative “movements” and changes seen by Goethe as he examined different leaf forms growing up a stem—these can be thought of as similar to the form transitions caused by parametric manipulation, and that’s why for so many morphologists the whole idea of metamorphosis doesn’t have to be conceived temporally/historically. It exists in the possible form space, which is what Search mode is meant to be exploring. “Transitional” forms exist along a continuum within this space. That may be the whole idea behind computational search techniques like hill-climbing.

²¹ Arber, 87.

Grammars on the Screen: Sizing and Placement

Some problem solving went into figuring out how size and scaling factored into the entire system. I was initially hesitant to scale down the leaf images before analysis as I figured more spatial detail was preferable, especially for something like the roughness of the leaf margin (which I never ended up trying to analyze anyway). Ultimately, however, I decided to scale down the images of the larger input leaves prior to visual analysis as it made it easier to evaluate shape similarity—smaller contours are simpler to compare with the generated L-system leaves, which I programmed to always fit within their allotted space in the window. Most leaf images are made to be $5/12^{\text{th}}$ s of the sketch window, with especially tall leaves instead scaled down to fit within the vertical boundary. The $5/12^{\text{th}}$ portion number was arrived at because the first $1/6$ of the horizontal screen space is taken up by measurement information and control sliders. The remaining $5/6^{\text{th}}$ of the screen is split in half between the leaf image and the L-system.



Because the input leaves are scaled down prior to finding the contour, the L-system is hence looking to match the leaf exactly as it is seen on the left side of the screen—bigger leaves do not actually result in bigger L-systems.²² Furthermore, because window size is variable and the input image scales to the window (and because the unit length of the lines and the iteration for each template are held constant), this also means that the program will theoretically behave somewhat differently on the same input leaf at different window sizes. The program is optimized for 1280x800 resolution and should be run that way for the best results. With a bigger window, it takes longer to generate candidates and the template parameter constraints also must to be adjusted to become as big as the input leaf picture (which scales automatically). I felt 1200x800 was a good size also because it seemed strange to have the generated leaves be much bigger than the actual leaves were when I collected them—as it is, most of the input and generated leaves on the screen look just bit over the average size of a real leaf.

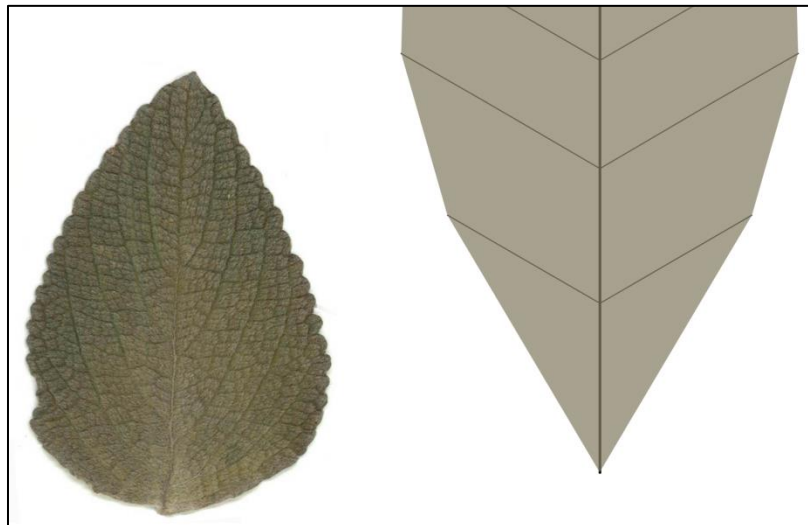


Figure 41: Sometimes generated leaves greatly exceed the sketch window.

²² That is, a leaf which is physically bigger than the computer screen will not result in an L-system bigger than the screen.

Special care was taken to get the resultant L-system leaves to fit in the sketch window. It happens very frequently (my very rough estimate would say at least 60% of the time) that the generated parameters result in a surface which exceeds the screen. Further constraint of the ranges of possible parameter values is not a good solution, as it is particular parameters in *combination* with each other which causes such behavior. The chief culprit is usually the growth rate of the main axis in conjunction with a large initial length. Constraining parameters like these in order to stop generated leaves from going out of bounds would make other legitimate shapes impossible—a shape which fits in the window just fine could have, for example, the same growth rate as a shape 100 times larger. It is rather the interaction *between* parameters as they influence the behavior of the whole L-system that determines the final surface.

On the technical end, an-out-of-bounds candidate causes the ImageProcessor object to not be able to find the leaf contour. One approach I attempted was to make the canvas (a PGraphics object) on which the Turtle drew the L-system very large with the intention of later scaling it down for display, but this just made the program laggier without even solving the problem—when the leaves were too big, they were *really* too big. Furthermore, scaling down a PGraphics object isn't as simple as is scaling an image. This approach was abandoned alongside the possibility of generating larger L-systems for physically larger leaves. Instead, the unusable, out-of-bounds leaves are generated but discarded. A second approach took advantage of how these out-of-bounds leaves would cause a glitch which would crash the entire program, as the fact that the program could not handle these shapes made it very easy to detect such exceptions. However, this glitch only occurred on very tall leaves which were not also out of bounds horizontally—that is, usually on the Pinnate

template leaves instead those from the Palmate template. The final approach which worked nicely ended up being very simple and stupidly obvious. As the Turtle draws each vertex, it first checks if the vertex coordinates are outside the bounds of the PGraphics canvas. If so, the Turtle drawGrammar function returns a null PGraphics object. Hence the program will have the Turtle attempt to draw the generated candidate until a valid, in-bounds one is created before moving on to the next space in the population pool.²³ If, in Slider mode, the selected template, input leaf (which affects base placement), and parameters make the resulting leaf out-of-bounds, a message is reported to the screen.

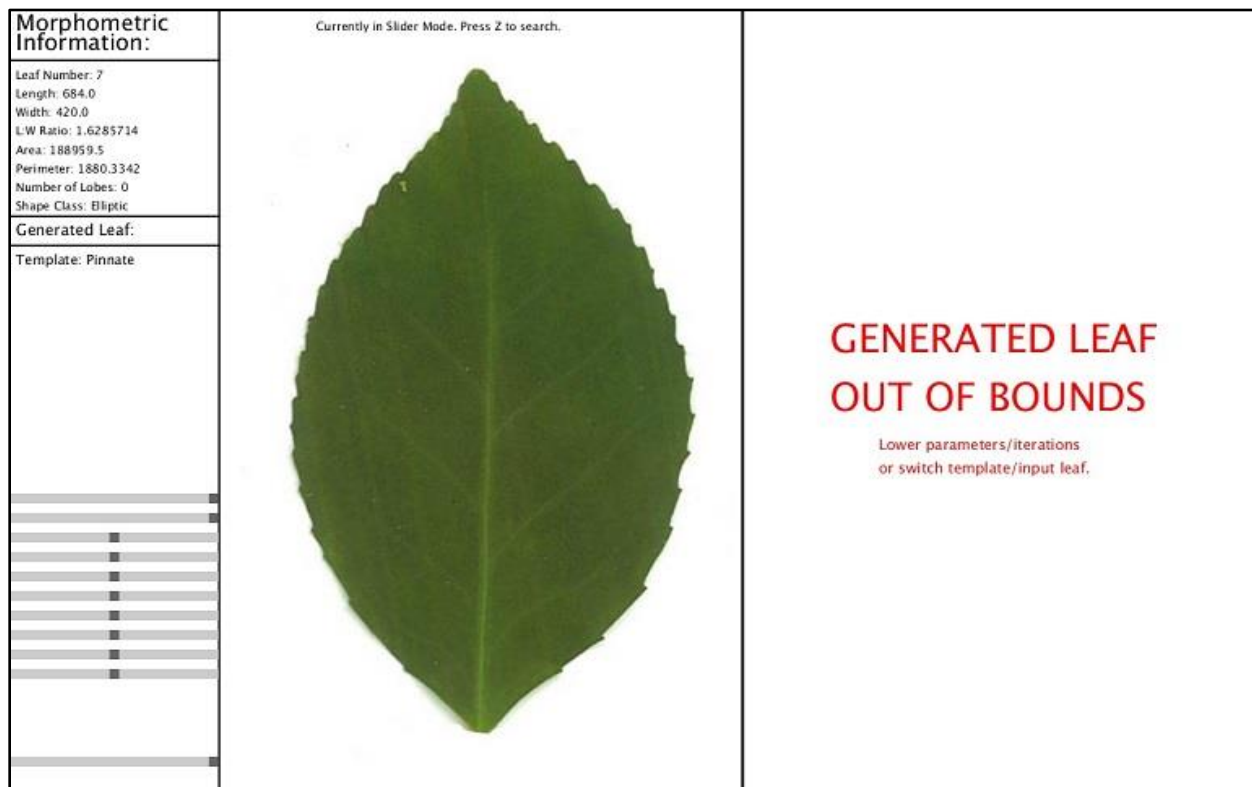


Figure 42: Out-of-bounds message.

²³ At first the system just discarded the invalid candidate and didn't generate another to replace it—hence if you asked it make 300 leaves it would spit out way fewer. This was silly.

By moving the vertical position of the L-system's base closer to the input leaf's base location, it also makes the search process more likely to end up with the correct template as less appropriate templates are more likely to be out of bounds—Imagine a long leaf with its base at the very bottom of the screen and then imagine a palmate leaf whose lobes reach down below the concave base. Hence, this is a type of “hard” constraint which makes the candidate pool closer.

An out-of-bounds candidate is also more common the higher the iteration of the L-system. This is to be expected—the growth rate multiplier is applied more times! Because the growth of different templates proceeds faster than others in certain dimensions, I ended up deciding on a maximum appropriate iteration for each template through trial and error. For example, an average generated palmate leaf will hit the horizontal boundary in fewer iterations than an average pinnate leaf will due to both less available horizontal space and a smaller length:width ratio. Hence I determined a reasonable maximum iteration number for each template.

During search, the generated candidates are all drawn by the Turtle at the maximum iteration for their template. Hence, “iteration” is *not* a parameter on which search is performed. The system (in particular the parameter constraints relative to the default window size) was instead optimized through manually tweaking to work with the template's maximum iteration. The bottom slider, however, is always dedicated to adjusting the *displayed* iteration, which means the user can easily control and view the “development sequence” of the generated leaf.

Leaf Color:

The color of the generated leaf is that of the pixel 15 pixels to the left of the center pixel of the input leaf image. The 15 pixel buffer was added to avoid the mid-rib, usually a different shade from the rest of the lamina. The L-system leaf color is displayed with an alpha value of 185 so that the black venation lines can be seen beneath. While making the color of the generated leaf closer to the input leaf in this manner is rudimentary and plays no further part in the process, I find that it increases what I perceive as resemblance by a surprising amount.

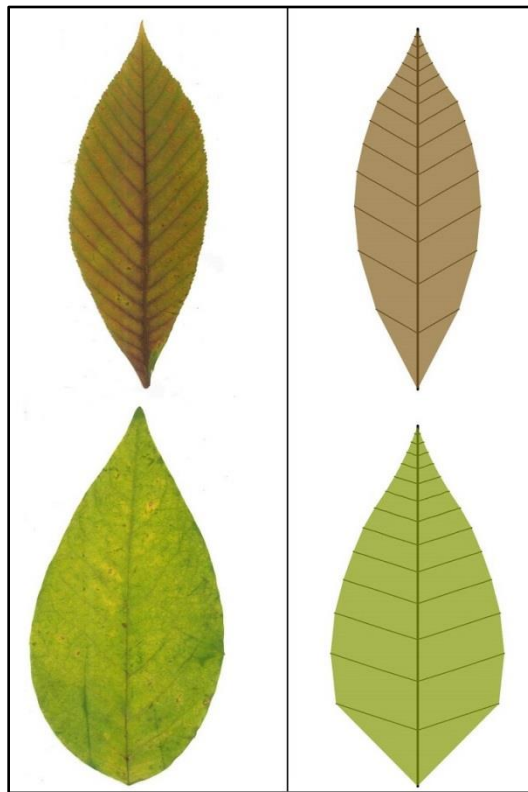


Figure 43: Generated leaf color samples a pixel in the input leaf. The L-system leaves above were matched to the images manually using parameter sliders.

It also occurs to me that one of the most useful things about having a wide variety of input leaves is that if I need to generate a bunch of leaf images I have an automatic natural

color palette right at my fingertips and don't have to go screwing around the code picking and changing RGB values (which because of impatience usually results in extreme colors—255's all around). Instead, if I want to make a bunch of red L-system leaves (regardless of the template or the input leaf shape) I can just navigate in the program to one of the red input leaves and get a natural looking shade on the L-system leaf automatically. Generating blue leaves however means changing the code.

Phase Four: Searching Parameter Space for an Adequate Model

Phase Four concerns Search mode and how the system looks for a generated leaf which resembles the input leaf. It is probably the least developed part of the whole project.

Assessing Fitness

Fitness is currently assessed using a weighted sum of different metrics which were gathered during Phase 2. They are all normalized to be between 0 and 1, but I think the way I did the normalization was really arbitrary and probably not the right way to approach it. As the weights demonstrate, fitness is at maximum 100.

```
float lwR = 1-map(abs(lwRatio - other.lwRatio),0,3,0,1);
float len = 1-map(abs(laminaLength-other.laminaLength), 0, 300, 0, 1);
float wid = 1-map(abs(laminaWidth-other.laminaWidth), 0, 300, 0, 1);
float area = map(abs(lamArea - other.lamArea),lamArea, 0, 0,1);

float lobe = map(abs(lobeNum-other.lobeNum), 0, 6, 1, 0);

float sc;
if(shapeClass.equals(other.shapeClass)){
    sc = 1;
}
else{
    sc = 0;
}

float fitness = lwR*25 + len*10 +wid*10 + area*15 + lobe*15 +sc*25;
```

I think this has the constitutive components of a successful fitness function but it needs better/more important measurements, more careful weighting, and some other way of normalizing units.

Search

This is a little ridiculous. The search was initially the whole point of this project and I never actually wrote a real one because I was having too much fun playing with leaf parameters myself. There *is* however a Search Mode, and it does at least make a rudimentary attempt to find a matching leaf template.

Here's pseudo-code for the currently implemented search function:

1. For n in candidate pool size:
 - a. Pick k random parameter values between 0 and 1 (these become mapped in the template to their unique parametric constraints).
 - b. Generates a candidate using these parameters, randomly selecting one of the three templates.
 - c. Check if candidate fits in window, if not return to **a**.
 - d. Evaluate SysLeaf candidate fitness (compare to Input leaf) and save to pool
2. Sort pool of valid SysLeaf candidates by fitness (compare each to Input leaf).
3. Display the candidate with best fitness.

Basically Search Mode finds some hills but it doesn't bother climbing them. As it is, the higher the candidate pool number the more options it has to pick from. Searching for this problem is limited because generating each candidate takes about a second, which over the course of many generations (and with a decent size candidate pool) would make the whole thing kind of slow.

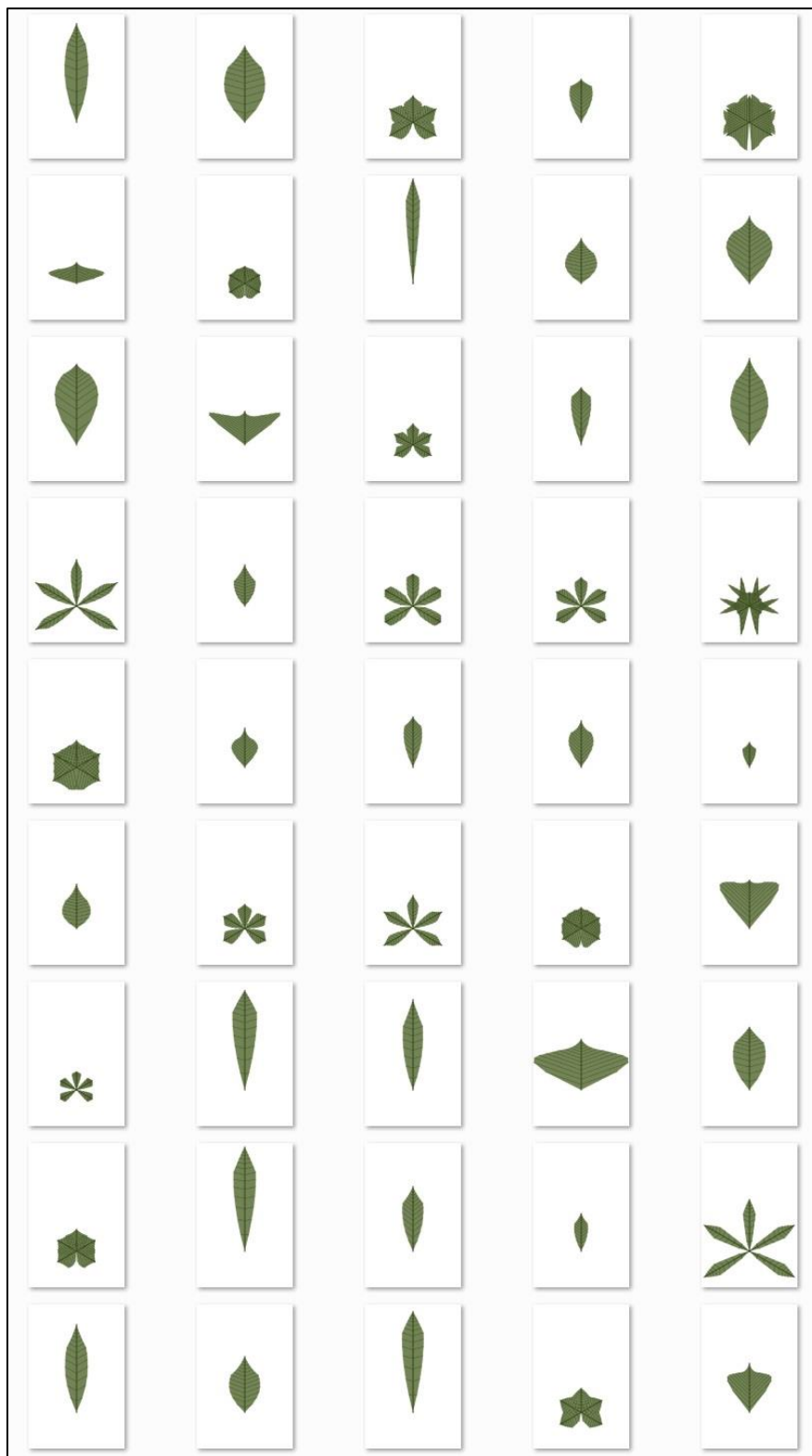


Figure 44: Screenshot from mid-development of randomly generated candidates.

I will say, however, that despite not being a good local search this function was extremely useful for this project—it runs when the user presses ‘z’ and it currently saves pictures of all the valid randomly generated L-system leaves to a folder. The filename is the fitness value. Hence, rather than automating a local search that will return the best candidate, it’s a randomized form generation algorithm which spits out a bunch of cool JPEGs. One of the neater things to come out not bothering to evolve the candidates into something fitter is that I get a lot of wacky shapes that don’t really look like leaves at all:

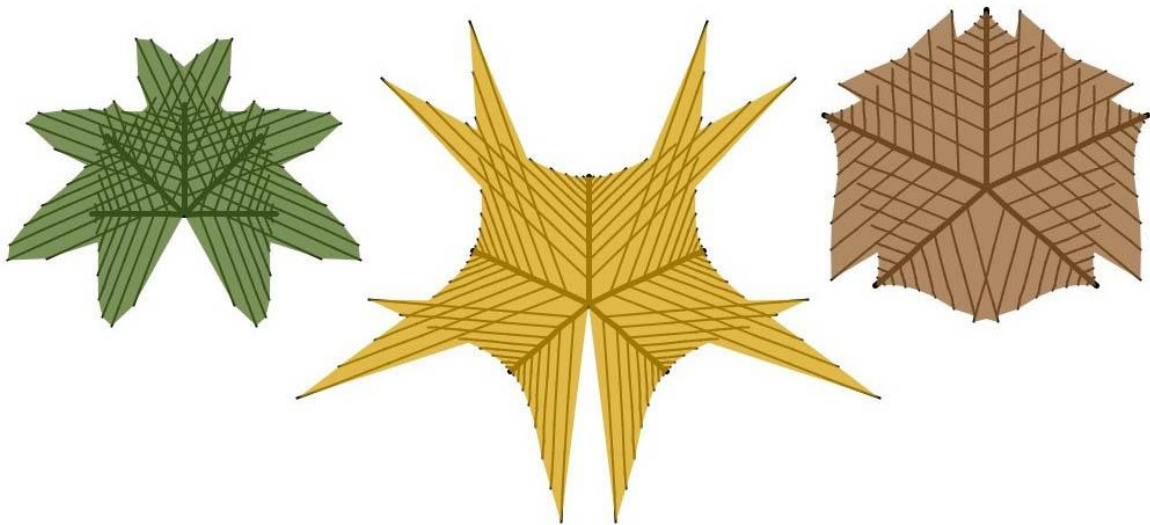


Figure 45: The Palmate template sometimes generates forms which remind me of radiolaria.

I think it would be neat to extend this random image creation to parametric L-systems which aren’t designed to look like leaves in the first place. Think of the variety!

Results and Discussion

My original goal with this project was that when given a picture of an input leaf, the system would return a graphically interpreted L-system representation with a matching shape and venation structure. A few reasons I found this interesting:

1. I was curious if information about the **external shape** alone was enough to assume something about the **internal venation** structure.
2. When it comes to representing specific biological structures, **automatic model generation** is less common than manually designed and adjusted models. Humans are plenty good at idealization—it would be neat to engineer an idealizing machine.
3. Viewing the iterative **developmental sequence** would be like having a visual time portal to a possible past of the leaf. I thought this would make a cool video effect if combined with footage of leaves.²⁴
4. I found **polygonal L-systems** geometrically exciting.
5. The problem combines **computer-vision, computer-graphics, formal grammar theory, and Nature**.

²⁴ To this end, another part of Branching Boogaloo was going to be an experimental video of plants using these and other computational effects. Work in progress.

FormaLeaf

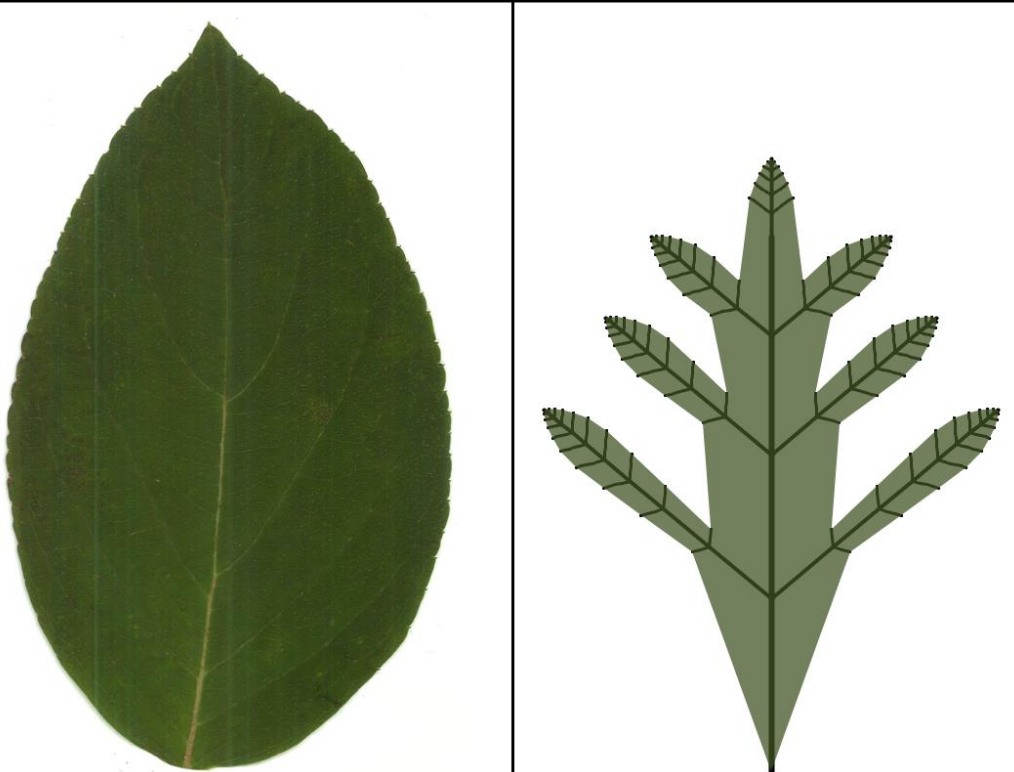
The most tangible result of working on this problem has been the entirety of the platform I built to explore it. I didn't go into the project looking to make a software interface; it just sort of happened. Software is ideational and can be an extension of our perceptive and imaginative faculties.

Building the system from the ground-up (beginning with an implementation of L-systems) helped me to understand the separate pieces and furthermore allowed me to customize it to my needs. I ended up with a program I can imagine someone else using, but realistically almost everything about it emerged around my personal patterns of use. Interactive systems are exciting to use because the user is part of the system loop. Interactive systems are even more exciting to *develop* because the developer gets control over the nature of the interaction. Being both developer and user at once means you are in an excellent position to satisfy your own desires. The way I use the system, the malleable source code is part of the interface. Grammatical definition of leaves is controlled by writing and modifying L-system code within the Processing IDE while these changes are understood and manipulated visually in the actual program window.

While FormaLeaf is not good at matching template leaves to input leaves, it excels in image generation. Hence, much of the results section will be graphic examples.


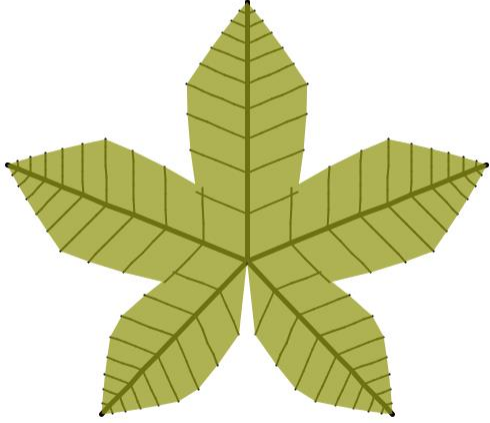
Search Results:


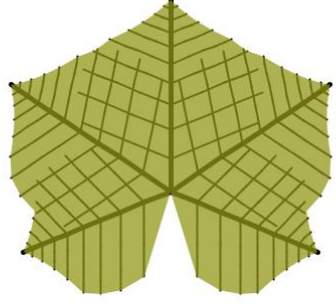
Search Mode does not get the right template very often as the fitness function and search function are inadequate. It seems to work better on Palmate leaves because this template in general has a different L:W ratio. Otherwise, Pinnate and PinLobed leaves are easily confused.

Morphometric Information:		
Input Leaf:		
Leaf Number: 2		
Length: 768.0		
Width: 460.0		
L:W Ratio: 1.6695652		
Area: 251780.5		
Perimeter: 2099.3882		
Shape Class: Elliptic		
Generated Leaf:		
Template: PinLobed		
Leaf Number: 2		
Length: 635.0		
Width: 472.0		
L:W Ratio: 1.345339		
Area: 94428.5		
Perimeter: 3298.637		
Shape Class: Elliptic		
Similarity: 73.09042		
		
<small>Vision Mode is OFF (Press C) Search Mode (Press X for Sliders)</small>		

Here is an example of incorrent template matching. The dimensions are somewhat similar, but the lobe structure is way off. At least it figured out that its got pinnate venation, though.

Here are two examples of correct template matching.
Note that the similarity of the leaf is higher.

Morphometric Information:		
Input Leaf:		
Leaf Number: 41 Length: 493.0 Width: 507.0 L:W Ratio: 0.9723866 Area: 134913.5 Perimeter: 2713.853 Shape Class: Obovate		
Generated Leaf:		
Template: Palmate Leaf Number: 41 Length: 419.0 Width: 486.0 L:W Ratio: 0.86213994 Area: 92065.0 Perimeter: 2241.2876 Shape Class: Obovate		
Similarity: 81.15062		
<small>Vision Mode is OFF (Press C) Search Mode (Press X for Sliders)</small>		

Morphometric Information:		
Input Leaf:		
Leaf Number: 41 Length: 493.0 Width: 507.0 L:W Ratio: 0.9723866 Area: 134913.5 Perimeter: 2713.853 Shape Class: Obovate		
Generated Leaf:		
Template: Palmate Leaf Number: 41 Length: 298.0 Width: 324.0 L:W Ratio: 0.9197531 Area: 68209.5 Perimeter: 1319.4083 Shape Class: Obovate		
Similarity: 77.045074		
<small>Vision Mode is OFF (Press C) Search Mode (Press X for Sliders)</small>		

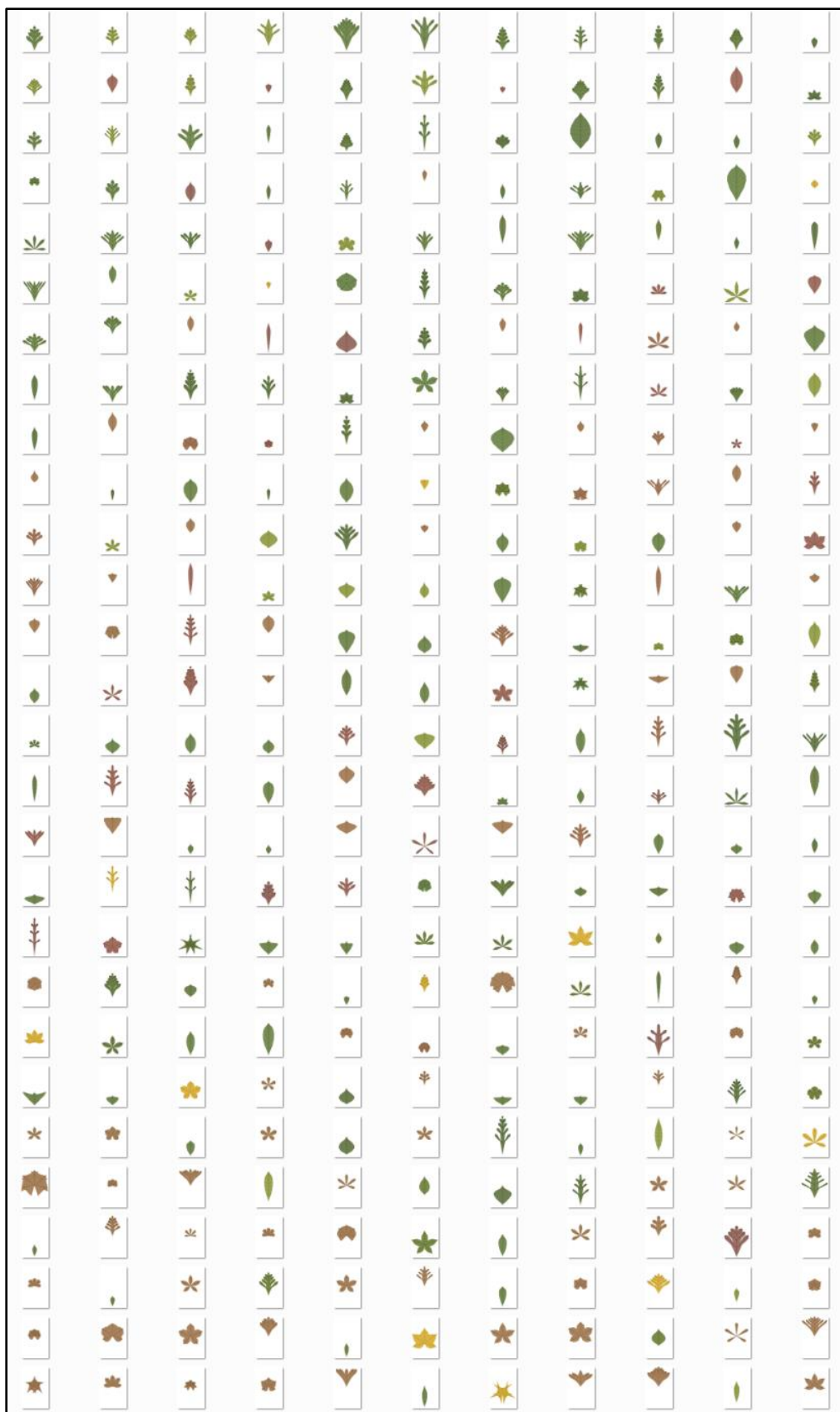
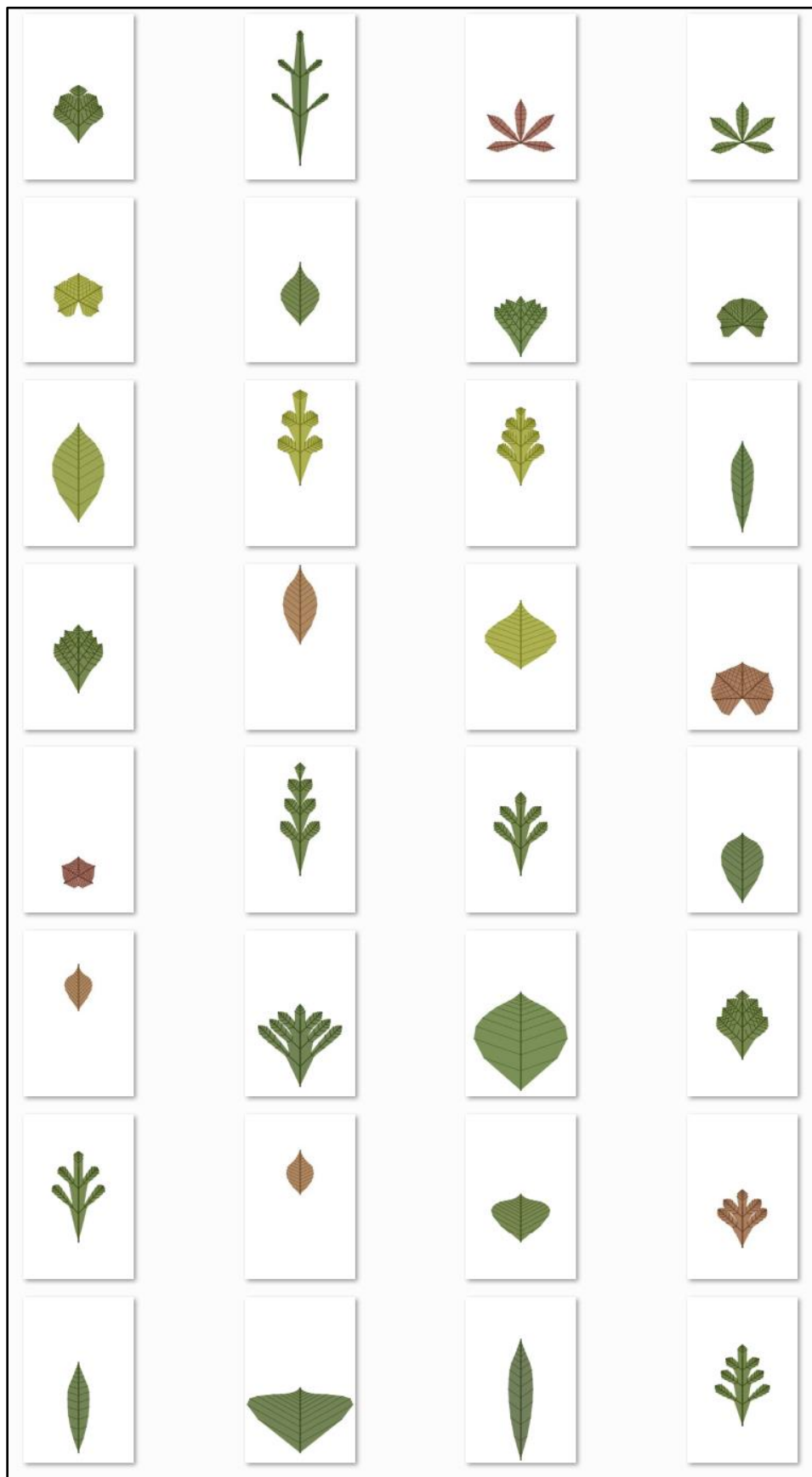


Figure 3: JPEG generation capabilities are off the charts.



Hypothetical Developmental Sequences

Viewing the successive iterations of an L-system leaf makes it look like the leaf is growing. This report is a static document so it can't display the effect in motion—I think it looks coolest as an animated gif. There is little reason to suspect that this *necessarily* resembles the actual spatial development over time of a leaf with that final shape, as leaves can assume different forms at different points in their growth. Designing grammars which resemble the more complex/earlier development would require work with more attention paid to early iterative behavior.

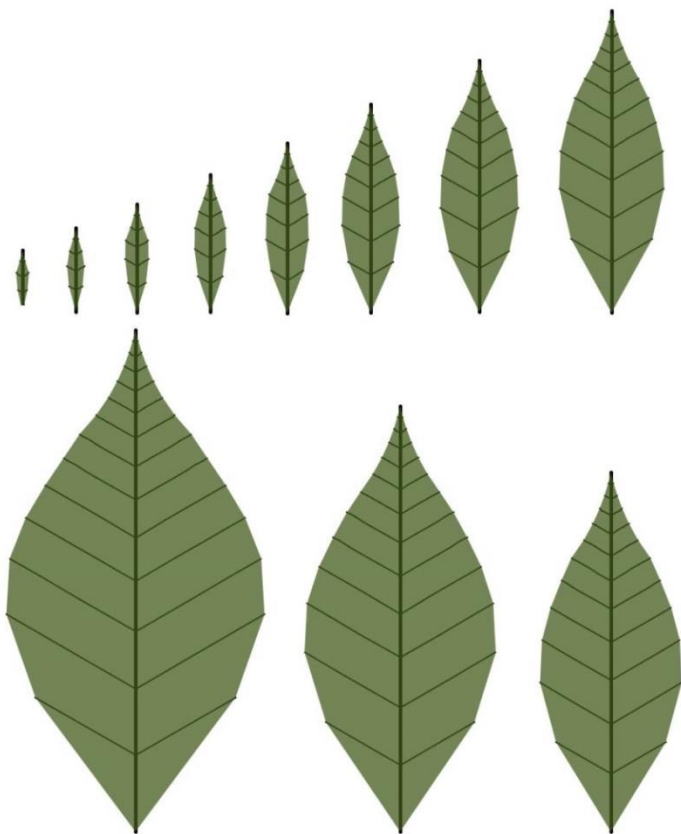
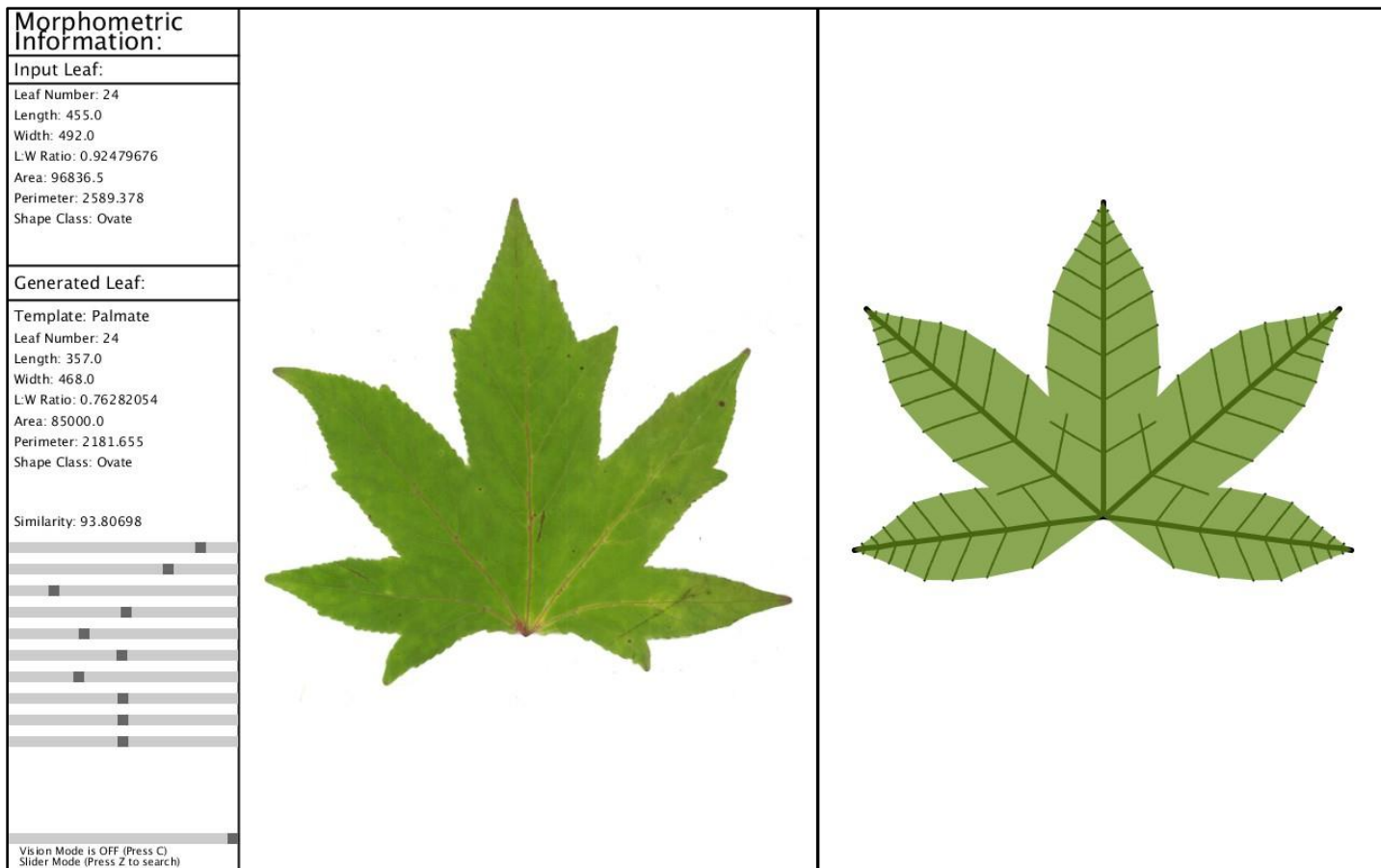


Figure 46: Iterative sequence of a Pinnate leaf.

How does the software allow for generating an image like this? Here is a manually adjusted Slider mode leaf I tweaked to vaguely resemble the input leaf:

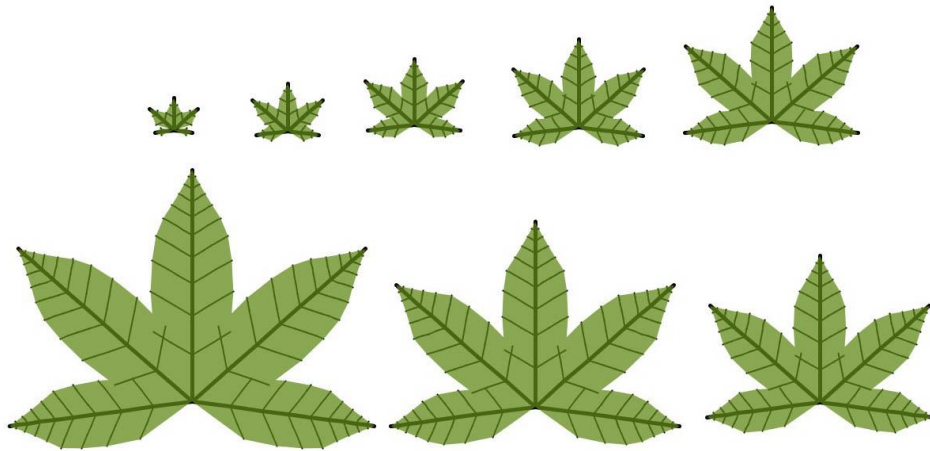


I then manually adjust the bottom-most slider to go through the iterations step by step,²⁵ and use the Processing saveFrame function (tied to key input) to save a sequence of images. After getting the frames, I use Photoshop to put the images together. It would be very cool and quite easy to automate this frame collection (and also make an oscillating animate button using the sine function for the displayed iteration), but the frame stitching

²⁵ The iteration slider actually starts at the third or fourth iteration step (depending on max iteration) since the first few are so tiny.

would be a little more involved. Maybe the best way to do it is with another offscreen PGraphics canvas which draws every iteration side by side and then just saves that image.

Putting the images together gets something like this:

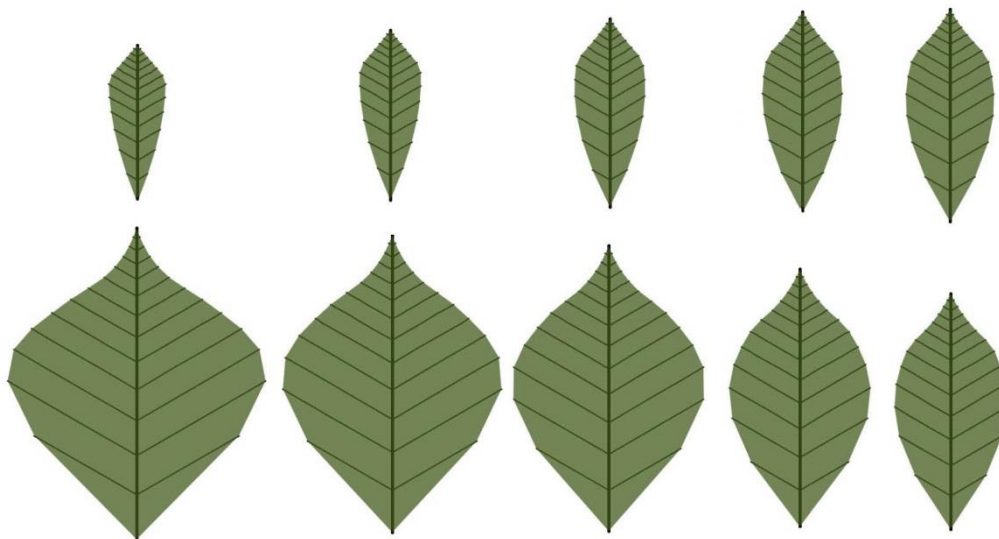


This sort of sequence isn't entirely dissimilar from what the palmate lobed leaves below look like as they grow. Compare the younger leaf in the center to the bigger ones—it's mostly a matter of scaling (and unfolding).



As addressed in Part I, leaves develop through both cell division and cell expansion. Speaking generally, cell division stops quite early in development and has a lot to do with determining the shape structure. Cell expansion makes the leaf grow to its adult size—hence the grammar iterations aren't a bad growing approximation for this *latter* phase of development since they are just applying the various segment growth rate multipliers to a pre-existing structure.

Of course, when we discussed development, there was also the matter of the second kind of development—that is, the non-temporal (or less *immediately* temporal, if we're thinking about actual evolving plants) metamorphosis from form to form. You can thus illustrate a metamorphical sequence using a different slider than the iteration slider, or even tie two parameters to the same slider and see how the form interacts when you change them in tandem. This is the sequence from increasing the length of the primary vein as well as rate of growth of the lateral vein. This isn't growth in time (like iterations), its parametric movement from form to form. This is not the determinate execution of the code/grammar like growth is; rather this is a continuous changing of the code itself.



Tulip Tree (*Liriodendron*):

One of the more unique samples is from a tulip tree.²⁶ It has what looks like pinnate venation, a cordate-esque base, and a concave apex. It's L:W ratio is more typical of a palmate leaf. Tulip tree leaves usually have four lobes, but this one has six.



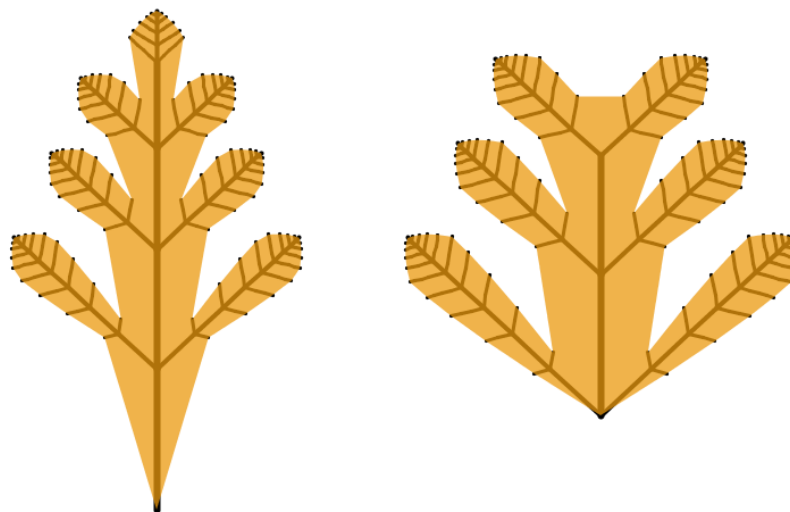
The *default* PinLobed template was based on the pinnate lobation of oak leaves and thus cannot approximate this leaf. Hence, the search and the parameters sliders won't ever really get close to this one. However, two simple structural modification to the grammar and some futzing with the parameters makes resemblance easy.

Structural Modifications:

1. In the original PinLobed template, the third production rule adds a final lobelet to the top of the leaf after the lobe number parameter has been decremented appropriately. Remove this production entirely.
2. Eliminate the bottom segment—I did this by adding a production which fires only after the start symbol which has the two lobelet branches but no segment beforehand. After this, subsequent mid-rib productions make a segment before branching like usual.

²⁶ Named so because its flowers *look* like tulips, not because it grows tulips. Tulips don't grow on trees.

This gets us from something like the left side to the one on the right:



Parametrically, it should make 3 pairs of thick, rounded lobelets (i.e, extend their lateral veins). Some slider constraints in the PinLobed template were modified to do this properly.

After some tweaking in the code and with the parameter sliders:



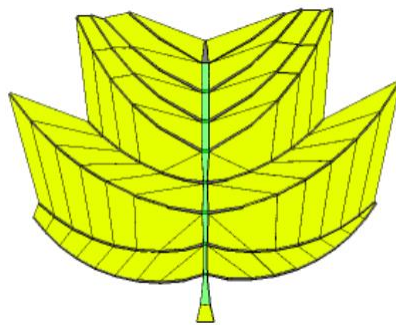
```

Axiom: {S(0)}
S(t) -> Z(t, 2.492611)
Z(t, i) -> !(6)[- (55.197044)A(t)][P(t+1, i-1)] [+ (55.197044)A(t)]      iff: i >= 0
P(t, i) -> !(6)G(7.3497534, 1.1394088)[- (55.197044)A(t)][P(t+1, i-1)] [+ (55.197044)A(t)]      iff: i >= 0
A(t) -> !(4)G(3.6009853, 1.0837438)[- (72.26601)L(t-1)] [A(t+1)] [+ (72.26601)L(t-1)]      iff: t >= -1.5123153
L(t) -> !(3)G(1.0, 1.3029556)L(t-1)      iff: t >= -1.5123153
G(s, r) -> G(s^r, r)

```

Obviously this needs some work but it's certainly getting closer. It's got a concave base and apex, although notice that the mid-rib line is not extending all the way to the top like it is in the actual leaf. In any case, I think this is an interesting case study. First of all, it's a very weird leaf. Second, it shows how a new template²⁷ for a leaf form can be manually created and that it requires editing the actual structure of the grammar. Could *this* process be automated as well? Some researchers evolve the symbols (as opposed to parameters) in the productions of non-parametric L-systems modeling branching structures, but I don't know if it's been done on parametric surface models. It also seems like a difficult problem. You would probably need to do it modularly—that is, have some premade productions that get shuffled around randomly.

As a comparison to another method of modeling tulip tree leaves with L-systems, here is a rendered leaf image from Peyrat *et. al* (2008). This paper uses a type of L-system called a 2Gmap L-system. They write that one of their aims was to model the leaf in such a way as to have accurate venation patterns, but the tertiary veins are a little funky.



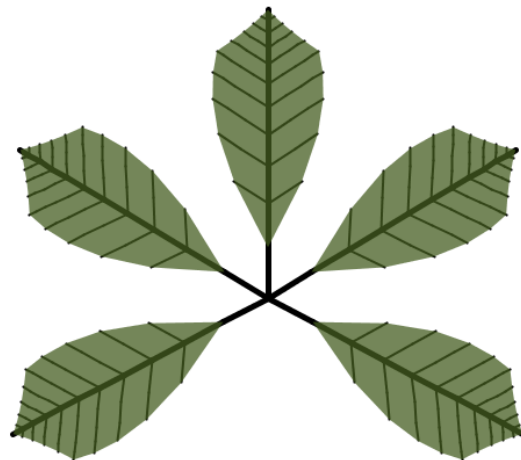
The grammar in these kinds of L-systems (here both parametric and context-sensitive) builds a 2D map and performs productions on entire geometrical faces instead of segments.

²⁷ The template for making leaves like this is present in my code as "Tulip" but isn't called in search.

Compound Leaves:

FormaLeaf deals with simple leaves or what could be compound leaflets as input. However, it is possible to generate images of compound leaves by modifying one of the parameter constraints to give the lobelets of the Pinnate or Palmate template each their own petiole. The Pinnate template's grammatical structure was based on some leaf surface models in *The Algorithmic Beauty of Plants* (see Figure 37) which had tiny petiole bits at the bottom. When I first adapted these grammars for my purposes it was one of the first things I got rid of. However, once I figured out how to make lobed leaves out of overlapping leaf surface models, bringing the petioles back made images that resemble compound leaves with leaflets attached by petiolules.

The modification is extremely simple: just adjust the parameter constraints so the BE (basal extension) parameter is a positive number. BE is checked in the conditional and I find it has unpredictable behavior since I haven't traced through on paper exactly what the conditional is checking all the time. A positive BE number makes it so the lamina starts further up.



Randomly Generated Leaves:



Future Work

As the numerous places in Section III which read something like “I could have done...” or “one possible approach would have been to...” suggest, there are innumerable ways this system could be improved. Because there are so many different pieces (not to mention different motivations) which make up FormaLeaf, my time was spent getting each portion into “adequate” shape so it could all at least come together. Literally every part of the system could be improved drastically if given focused attention and what I’ve ended up with by the end of the allotted time for Senior Project is just a start. As alluded to before, the “result” of this project is not only what the system spits out but also the whole system itself. I think this is a valid way to perform computer science.

-Lobe Counting:

I think that having an accurate count for the number of lobes would be one of the single most useful metrics possible for matching a structural template. Unfortunately my lobe detection is awful so the number it gets is really just an estimate (it’s at least usually *higher* for lobed leaves). But still, what use is it if the system thinks an obviously unlobed leaf has 2 lobes?

-Leaf Margins:

Many leaves have serrated/jagged margins, with serrated margins more common in cooler climates. The only place margin type is reflected in FormaLeaf at all is that non-entire leaves make the lobe detector even more confused. Assessing the margin visually would probably mean measuring the roughness of the contour. Generating different margins sounds trickier.

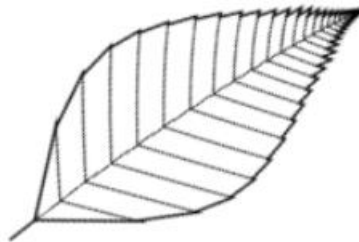


Figure 47: Rose leaf model with marginal notches.
(The Algorithmic Beauty of Plants, 126)

There is one example in *ABOP* but the polygonal encoding here is a little different than my templates. In the leaf above, each quadrilateral is a separately defined polygon. Maybe this is worth exploring. Alternatively, I found that the PinLobed template can be modified to get the L-system leaf on the right. Perhaps marginal notches and teeth can be approached with many very small overlapping lobelets. The apices of the lobelets become the teeth of the larger leaf.



-Brochidodromous Venation:

A few leaves in my sample set have looping on their 2nd-order veins. While I am unsure how the system could figure out if an input leaf is brochidodromous or not without doing line detection (since very similar leaf shapes exist with and without 2nd-order looping), it would still be interesting to model such venation using an L-system.



The L-system mechanism as described by Lindenmayer and Prusinkiewicz does not have the inherent capacity for networked connections among the string elements (unless there is some way to do this with context-sensitive L-systems, which I haven't explored yet at all). A few researchers have approached L-system network modeling with varying strategies, building indications of connections between symbols into the grammars. Boers *et. al* (1995) introduces G2L-Systems, which are graph grammars. Parish(2001) and Eilertsen (2013) papers describe methods for city road generation with L-systems.

What also really puzzles me about modeling these is that the current polygonal surface models in FormaLeaf define polygonal vertices by having the Turtle drop vertices at the leaf margins. That is, the second-order veins are extending all the way to the margin (craspedodromous). Looking at the brochidodromous example above, this isn't happening. Perhaps vertices must be dropped at the end of tertiary veins coming off the 2nd-order loops. Tricky!

-Higher Order Venation Network:

One of the things I find most fascinating about leaves is the intricate network of higher-order veins. These networks are often modeled using techniques involving Voronoi diagrams. I thought about making it so that once the best result leaf was found you could press a key or button to “Voronoiize” it, which would fill in the spaces between the lateral veins with their own Voronoi diagram. These edges would not be a part of the L-system (and hence would not follow with the iterations) and would be done purely for visual effect. Fooling around with Lee Byron’s Mesh library²⁸, this is as far as I got:

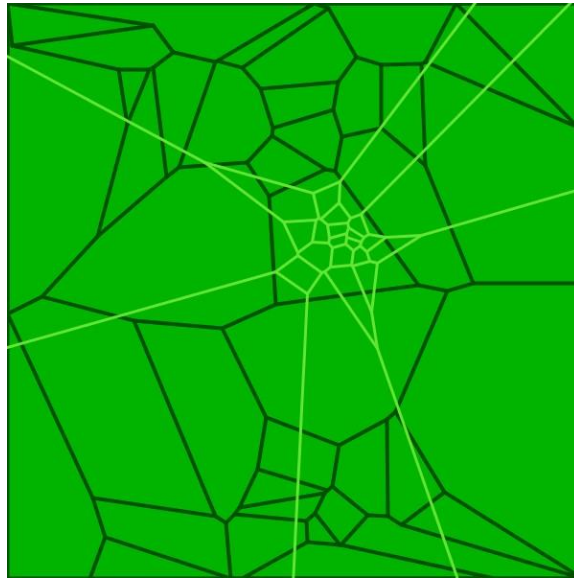


Figure 48: Voronoi diagram experiment.

Generating a smaller diagram within a region of the larger diagram was based off Shirriff (1993). While I don’t know how one would get something like this to work alongside being able to iterate through development steps (the diagram would have to be adaptive somehow), I do think it would look interesting to have the spaces between the lateral veins on a typical L-system leaf filled in in this manner even if only worked on a

²⁸ <http://leebyron.com/mesh/>

final, static iteration. However, storing the Turtle's drawn lines in a way useable with Byron's library (or otherwise implementing Fortune's Plane-Sweep Algorithm myself, which is complicated), figuring out the geometry and boundaries of arbitrary regions, and getting the system to "cap" its drawings of edges would have taken too much time for unessential payoff and hence this effort was abandoned. At least Figure 1 gives an idea of what I was thinking about.

-Arcuate Venation.

The lateral (and sometimes the primary) veins of many leaves are not straight lines. This is commonly referred to as arcuate venation. In my sample set, it is most common on the simplest of the leaves—that is, unlobed "entire" leaves. This actually makes me wonder if the curved venation is indicative of a simpler branch structure in general.



Processing has a curve function which could be worked into the Turtle implementation. It would probably work parametrically, with some way to tell the system where to place the inner points (which define the path of drawn curve) relative to the start points, end points, and specified amount of curviness.

-Non-Symmetrical Leaves:

All my generated leaves are perfectly symmetrical, which isn't true of many actual leaves. The leaf to the right has a branching scheme which the system has no way to approach. It's actually rather difficult to make heads-or-tails of this leaf even by looking at it. Would you say that the midrib is bifurcating equally or branching to the left and then bending to the right? How come the left side of the leaf is developing what look like underdeveloped lobes while the right side isn't? Even weirder, this leaf is found on the same plant as the two below. This is possibly an instance of heteroblastic development.

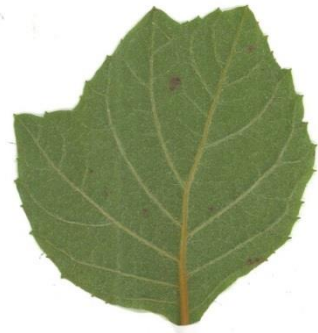


Figure 49: Oakleaf hydrangea (*Hydrangea quercifolia*).
They grow near the stairs by Kline.

-More Labeling:

The only terminological label that is applied (other than the template type) is that of the overall shape class. It would be interesting if the system could apply an apex label and a base label as well and report it in the left-hand information panel. This of course sounds like a problem more suited to machine learning approaches, but for that you need a lot of data.

I also wonder whether you could use label information to give the search process better hints of how to proceed—that is, it could use harder constraints instead of just soft weights if you could be confident that the applied label was correct. This would make the search more deductive.

-Fix Console Output:

The Processing console is useful for debugging, but the console for FormaLeaf is consistently filled with repeated print statements from the OpenCV for Processing library.

The constructor for the OpenCV object always prints

```
OpenCV for Processing 0.5.2 by Greg Borenstein http://gregborenstein.com  
Using Java OpenCV 2.4.5.0
```

which means that it goes to the console every time a new OpenCV object is created. In FormaLeaf this is done in a number of different constantly executing loops, as the constructor takes the image it will be working on. Though the library has a loadImage function, my attempts to use this to minimize the calls to the constructor did not behave as expected. I either need to reorganize some of the code and figure out how to get loadImage to work as I would like or otherwise recompile the library with the print statement commented out, which is not the best solution because it complicates the library dependency situation. In any case, it's annoying and I'd like to fix it.

-More Template Leaves and Better Grammar Design:

As can be gleaned by some examples explained in the Results section, figuring out templates for different leaf forms takes some experimentation. In particular, it would be satisfying to get a cordate template working that looks right²⁹ and actually morphs properly with the parameters. This may mean figuring out the proper proportional relationships of different parts of the leaf or otherwise structuring it differently.

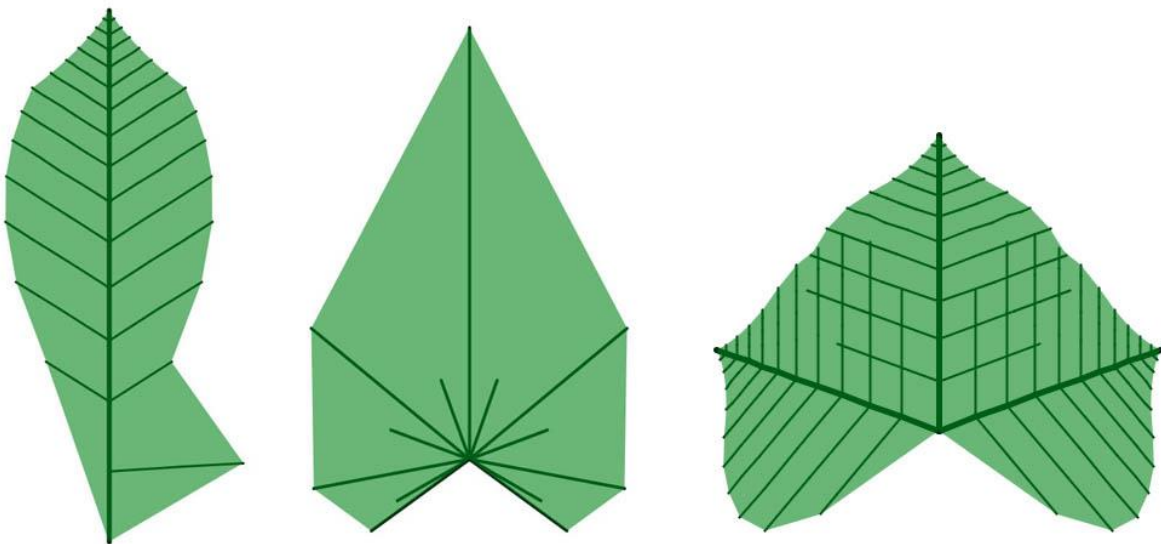


Figure 50: Attempts at a cordate leaf template were disappointing. The one on the right looks good until you touch any of the parameters or change the iteration, at which point it becomes a lobed leaf.

Furthermore, the templates which already exist could be designed better. For example, the Palmate template should be redesigned so it can take the number of lobes as a parameter like the PinLobed template. The PinLobed lobelets have inaccurate looking lengths relative to each other—the bottom-most lobes should not be the longest. I would

²⁹ You perhaps recall the image from *The Algorithmic Beauty of Plants* included in the Intersection of a cordate L-system. However, this model had parallel venation and further defines the surface model in a different way.

also like to see to what extent parameters can be consolidated or tied to each other in order to minimize the number of required inputs to the more complicated grammars.

-Improved Parameter Parsing:

Having used an external library for parameter and expression parsing, I didn't have as much control over this part of the program and hence running into issues here was especially frustrating. Problems usually occur when trying to use symbolic elements which are otherwise defined in the L-system as having interpretative meaning (such as parentheses or even + and - signs) within parametric expressions for arithmetic purposes. For whatever reason, the parser crashes if I try to use parentheses to make the order of operations within an expression clearer. I'm not sure why this happens, but I ran into a few similar cases where what could be expressed in the L-system was limited because of language parsing problems.

-Fitness and Search:

The fitness function is currently a weighted sum of a number of spatial measurements. This could be improved by taking other more important measurements and also by tweaking how they are weighted. One spatial analysis technique which might be handy is found in Thomas S. Ray (1992), "Landmark Eigenshape Analysis: Homologous Contours: Leaf Shape in *Syngonium* (Araceae)." Florindo *et. al* (2010) use a measurement called Curvature Scale Space and got good results. There's some code (I didn't write it) in the ImageProcessor class which measures the Hausdorff distance between two contours which seems useful, but it didn't seem right to use this metric without understanding how

it was working. A further option is to use a more nuanced technique for assessing similarity than just having one fitness value, which would mean modifying the search technique as well.

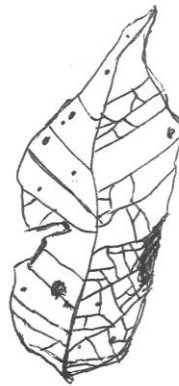
Search Mode in its current state doesn't look very hard. I would like to try a genetic algorithm, as it worked for Rodkaew *et. al* (2002) to evolve an L-system skeleton structure. There's also the appeal of "evolving" leaves "genetically." An even more interesting fitness function wouldn't try to match the leaf to an input picture but rather try to optimize some kind of simulated ecological metric.

Project Conclusion

Motivations and Summary

The eventual selection of leaf morphology as a senior project topic came out of personal interests and experiences. I had since sophomore year at Bard found foliage kaleidoscopically intriguing and had been thinking about the human backgrounding of trees since the end of junior year. With boredom and free time on my side, I sat down with a leaf the summer before senior year started. Actually bothering for once to take a closer look, I drew a picture and wrote a dopey haiku:

Leafy veins branching
 The water is delivered
 To the thirsty cells



In hindsight, this project might be nothing but an earnest attempt to unpack that haiku, that drawing, that particular leaf I picked from the front lawn grass of a janky summer sublet in Tivoli—the house was set back from the road so there was lots of green space out front, catalpa trees dropping their cigars on the sidewalk, in the back a real shadow jungle set beyond a desiccated swimming pool. Cool air and clear moons. Once we set the porch on fire by accident.

There was also a purely practical aspect to the choice of the leaf—collecting and generating spatial information is much easier in two dimensions. Putting a leaf in the

library's flatbed scanner is a lot more feasible than 3D-scanning an entire tree, and programming L-systems is a lot simpler when there is one fewer dimension to account for in your code and in your head. While I began my project with the vague intention of modeling entire trees with L-systems, I did not officially decide to narrow the focus to leaves until fall break. While I had toyed with this idea earlier, it finally occurred to me that if I wanted my project to interface with the real world at all, the project should be about leaves specifically due to the possibility of collecting actual spatial data and using it as input.

I now have a decent platform and, more importantly, a knowledge base upon which to conduct further explorations of leaf morphology through computational means.

Bibliography

- Antonelli, Peter L.: "Review of The Algorithmic Beauty of Plants" *SIAM Review*, Vol 34, No. 1. 1992.
- Arber, Agnes Robertson. *The Natural Philosophy of Plant Form*. Cambridge: U, 1950. Print.
- Aristotle, Trans. William Ogle, *On the Parts of Animals*. Web.
- Barnsley, Michael F. *Fractals Everywhere*. Boston: Academic, 1988. Print.
- Battjes, Johannes, Norbert O. E. Vischer, and Konrad Bachmann. "Capitulum Phyllotaxis and Numerical Canalization in *Microseris Pygmaea* (Asteraceae: Lactuceae)." *American Journal of Botany* 80.4 (1993): 419. Web.
- Brenner, Eric D., Rainer Stahlberg, Stefano Mancuso, Jorge Vivanco, František Baluška, and Elizabeth Van Volkenburgh. "Plant Neurobiology: An Integrated View of Plant Signaling." *Trends in Plant Science* 11.8 (2006): 413-19. Web.
- Brown, James H., and Geoffrey B. West, eds. *Scaling in Biology*. New York: Oxford UP, 2000. Print.
- Horn, Henry S.: *Twigs, Trees, and the Dynamics of Carbon in the Landscape*, 199-220.
 - Schreiner, W., Karch, R., Neumann, F., Neumann, M.: *Constrained Constructive Optimization of Arterial Tree Models*, 145-165.
- Da Vinci, Leonardo, and Edward McCurdy. *The Notebooks of Leonardo Da Vinci*. New York: Reynal & Hitchcock, 1939. Print.
- Ellis, Beth, Douglas C. Daly, Leo J. Hickey, Kirk R. Johnson, John D. Mitchell, Peter Wilf, and Scott L. Wing. *Manual of Leaf Architecture*. Ithaca: Cornell UP, 2009. Print.
- Esau, Katherine. *Plant Anatomy*. New York: Wiley, 1965. Print.
- Fleury, Vincent, Jean-François Gouyet, and M. Léonetti, eds. *Branching in Nature: Dynamics and Morphogenesis of Branching Structures, from Cell to River Networks: Les Houches School, October 11-15, 1999*. Berlin: Springer, 1999. Print.
- B. Sapoval, M. Filoche and E.R. Weibel. *Branched Structures, Acinus Morphology and Optimal Design of Mammalian Lungs*, 225-242.
 - Couder, Y.: *Patterns with Open Branches or Closed Networks: Growth in Scalar or Tensorial Fields*, 2-20.
 - Peterson, M.A. *Mathematical Meristems: The Singularities of Laplacian Growth*, 445-450.

- Florindo, João B., André R. Backes, and Odemir M. Bruno. "Leaves Shape Classification Using Curvature and Fractal Dimension." *Lecture Notes in Computer Science Image and Signal Processing* (2010): 456-62. Web.
- Goethe, Johann Wolfgang Von. *Italian Journey, 1786-1788*. Harmondsworth: Penguin, 1970. Print.
- Goethe, Johann Wolfgang Von. *The Metamorphosis of Plants*. Ed. Gordon L. Miller. Cambridge, MA: MIT, 2009. Print.
- Gray, Asa. *First Lessons in Botany and Vegetable Physiology: To Which Is Added a Copious Glossary, or Dictionary of Botanical Terms*. New York: Ivison and Phinney, 1860. Print.
- Inamdar, J., Shenoy, K., Rao, N.: "Leaf Architecture of Some Monocotyledons with Reticulate Venation." *Annals of Botany*. 52.5 (1983): 725-735.
- Ju, Tao, Scott Schaefer, and Ron Goldman. "Recursive Turtle Programs and Iterated Affine Transformations." *Computers & Graphics* 28.6 (2004): 991-1004. Web.
- Kumar, Neeraj, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida C. Lopez, and João V. B. Soares. "Leafsnap: A Computer Vision System for Automatic Plant Species Identification." *Computer Vision – ECCV 2012 Lecture Notes in Computer Science* (2012): 502-16. Web.
- Lev-Yadun, Simcha. "Fern Leaves and Cauliflower Curds Are Not Fractals." *Plant Signaling & Behavior* 7.5 (2012): 533-34. Web.
- Linné, Carl Von. *Philosophia Botanica*. Ed. Casimiro Gómez Ortega and Johann Andreas Murray. Matriti: P. Marin, 1792. Print.
- MacAdam, Jennifer W. *Structure and Function of Plants*. Ames, IA: Wiley-Blackwell, 2009. Print.
- Mandelbrot, Benoit B. *The Fractal Geometry of Nature*. New York: Freeman, 1983. Print.
- Needham, Joseph, and Lu Gwei-Djen. *Science and Civilization in China*. Vol. 6.i. Cambridge: Cambridge University Press, 1986. Print.
- Nicolaus of Damascus, and Walter Stanley. Hett. *On Plants (in Aristotle: Minor Works)* London: Heinemann, 1936. Print.
- Nicotra, Adrienne B., Andrea Leigh, C. Kevin Boyce, Cynthia S. Jones, Karl J. Niklas, Dana L. Royer, and Hirokazu Tsukaya. "The Evolution and Functional Significance of Leaf Shape in the Angiosperms." *Functional Plant Biology Functional Plant Biol.* 38.7 (2011): 535. Web.
- Ochoa, Gabriela. On genetic algorithms and Lindenmayer systems. *Parallel Problem Solving from Nature V*, 353-367. 1998

- Peak, D., J. D. West, S. M. Messinger, and K. A. Mott. "Evidence for Complex, Collective Dynamics and Emergent, Distributed Computation in Plants." *Proceedings of the National Academy of Sciences* 101.4 (2004): 918-22. Web.
- Peyrat, Alexandre, Olivier Terraz, Stephane Merillou, and Eric Galin. "Generating Vast Varieties of Realistic Leaves with Parametric 2Gmap L-systems." *The Visual Computer Visual Comput* 24.7-9 (2008): 807-16. Web.
- Ponge, Francis. *Selected Poems*. Trans. C. K. Williams, John Montague, and Margaret Guiton. Winston-Salem, NC: Wake Forest UP, 1994. Print.
- Priestley, Joseph. *Experiments and Observations on Different Kinds of Air. With a Continuation of the Observation on Air*. London, 1774. Print.
- Prusinkiewicz, Przemyslaw, and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. New York: Springer-Verlag, 1990. Print.
- Prusinkiewicz, P., Hanan, J., Hammel, M., and M. Radomir.: L-systems: from the Theory to Visual Models of Plants. *Plants to Ecosystems. Advances in Computational Life Sciences*, CSIRO, Collingwood, Australia 1997, 1-27.
- Prusinkiewicz, Przemyslaw. "Modeling of Spatial Structure and Development of Plants: A Review." *Scientia Horticulturae* 74.1-2 (1998): 113-49. Web.
- Prusinkiewicz, Przemyslaw, and Adam Runions. "Computational Models of Plant Development and Form." *New Phytologist* 193.3 (2012): 549-69. Web.
- Ray, John, Stephen A. Nimis, Kathleen Tschantz Unroe, Michael A. Vincent, Mark W. Chase, and Michael Black. *Methodus Plantarum Nova*. 2014, Print.
- Ray, Thomas S. "Landmark Eigenshape Analysis: Homologous Contours: Leaf Shape in Syngonium (Araceae)." *American Journal of Botany* 79.1 (1992): 69. Web.
- Raven, Charles E. *John Ray, Naturalist; His Life and Works*. Cambridge: U, 1950. Print.
- Rodkaew, Y., Lurinsap, C., Fujimoto, T., Siripant, S., Chongstitvatana, P., Chiba, N.: Modeling leaf shapes using l-systems and genetic algorithms. *Proceedings of Plant International Symposium on Plant Growth Modeling, Simulation, Visualization and their Applications*, 210-217. 2002
- Rozenberg, Grzegorz, and Arto Salomaa. *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*. Berlin: Springer-Verlag, 1992. Print.
- von Sachs, Julius. *History of Botany (1530-1860)*. Trans. Henry E. F. Garnsey. Ed. Isaac Bayley Balfour. Oxford: Clarendon, 1890. Print.

- Sack, Lawren, Christine Scoffoni, Athena D. Mckown, Kristen Frole, Michael Rawls, J. Christopher Havran, Huy Tran, and Thusuong Tran. "Developmentally Based Scaling of Leaf Venation Architecture Explains Global Ecological Patterns." *Nature Communications Nat Comms* 3 (2012): 837. Web.
- Scarpella, Enrico, Michalis Barkoulas, and Miltos Tsiantis. "Control of Leaf and Vein Development by Auxin." *Cold Spring Harbor Perspectives in Biology* 2:a001511 (2010): n. pag. Web.
- Schleiden, Matthias Jacob. *Contributions to Phytogenesis*. 1838. Print.
- Shirriff, Ken. "Generating fractals from Voronoi diagrams", *Computers and Graphics* 17, 2 (1993) 165-167.
- Theophrastus. *Enquiry into Plants and Minor Works on Odours and Weather Signs*. Trans. Arthur Hort. London: W. Heinemann, 1916. Print.
- Thom, René. *Structural Stability and Morphogenesis; an Outline of a General Theory of Models*. Reading, MA: W.A. Benjamin, 1975. Print.
- Thompson, D'Arcy Wentworth. *On Growth and Form*. Vol. 1. 1917. Cambridge, 1952. Print.
- Thompson, D'Arcy Wentworth. *On Growth and Form*. Vol. 2. 1917. Cambridge, 1952. Print.
- Thoreau, Walden. *The Portable Thoreau*. Vintage. 1970, Print.
- Tomlinson, P. B. "Branching Is a Process, Not a Concept." *Taxon* 36.1 (1987): 54. Web.
- Turing, Alan Mathison. "The Chemical Basis of Morphogenesis." 1952. *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life "plus" the Secrets of Enigma*. Ed. B. Jack Copeland. Oxford: Oxford University Press, 2013. Print.
- Vogel, Steven. *The Life of a Leaf*. Chicago: University Of Chicago, 2013. Print.
- Wardlaw, C. W. *Essays on Form in Plants*. Manchester: Manchester U.P., 1968. Print.

Appendix: Processing Code

FormaLeaf.pde:

```
/*  
  
FormaLeaf  
  
version 1.0  
  
Diana Ruggiero  
  
05/06/16  
  
*/  
  
import gab.opencv.*;  
import org.opencv.imgproc.Imgproc;  
import org.opencv.core.*;  
import java.lang.*;  
import java.util.List;  
import java.util.Collections;  
import org.qsscript.*;  
  
OpenCV opencv;  
PImage img, thresh;  
  
Leaf realLeaf;  
SysLeaf fakeLeaf;  
  
Turtle turt;  
  
ImageProcessor iProc;  
PImage realLeafProcessed;  
  
Slider slide0;  
Slider slide1;  
Slider slide2;  
Slider slide3;  
Slider slide4;  
Slider slide5;  
Slider slide6;  
Slider slide7;  
Slider slide8;  
Slider slide9;  
  
Slider iterSlider;  
  
GrammarGenerator gGen;  
Grammar startGram;  
  
PApplet sketchPApplet;  
float[] sliderVals = {0.5, 0.5, 0.5, 0.5, 0.5, 0.5,0.5,0.5,0.5};
```

```

Boolean searchMode;
Boolean visionMode;

Boolean LSinfo;

String displayTemplate;
String[] templateList = new String[3];
int t;
int leafNum;

void setup() {

    size(1280, 800);
    sketchPApplet=this;
    iProc = new ImageProcessor();

    realLeaf = new Leaf(1);
    img = loadImage("01_r.jpg");

    img.resize((width/12)*5, 0);

    println(img.width);
    //used for thresholding, converting from Mat to PImage:
    opencv = new OpenCV(this, img);

    realLeafProcessed = iProc.processImage(opencv, realLeaf);
    realLeafProcessed.resize(0, height);

    slide0 = new Slider(0, height-300, width/6, 10, 1, false);
    slide1 = new Slider(0, height-280, width/6, 10, 1, false);
    slide2 = new Slider(0, height-260, width/6, 10, 1, false);
    slide3 = new Slider(0, height-240, width/6, 10, 1, false);
    slide4 = new Slider(0, height-220, width/6, 10, 1, false);
    slide5 = new Slider(0, height-200, width/6, 10, 1, false);
    slide6 = new Slider(0, height-180, width/6, 10, 1, false);
    slide7 = new Slider(0, height-160, width/6, 10, 1, false);
    slide8 = new Slider(0, height-140, width/6, 10, 1, false);
    slide9 = new Slider(0, height-120, width/6, 10, 1, false);

    iterSlider = new Slider(0, height-30, width/6, 10, 1, true);

    searchMode = false;
    visionMode = false;

    LSinfo = false;

    templateList[0] = "Pinnate";
    templateList[1] = "PinLobed";
    templateList[2] = "Palmate";
    t = 0;
    displayTemplate = templateList[t];

    leafNum = 1;

```

```

gGen = new GrammarGenerator(sketchPApplet);
startGram = gGen.buildGrammar(displayTemplate, sliderVals);

fakeLeaf = new SysLeaf(realLeaf.getNum(), startGram, displayTemplate);
}

void draw() {
  background(255);
  Slider[] sliderList = {slide0, slide1, slide2, slide3, slide4, slide5,
slide6, slide7, slide8, slide9, iterSlider};

  image(img, width/6, height/2-(img.height/2));
  realLeaf.col = img.get(img.width/2-15, img.height/2);

  strokeWeight(2);
  stroke(0);
  line(width/6, 0, width/6, height);

  fill(0);
  textSize(23);
  textLeading(18);
  text("Morphometric" + "\n" + "Information:", 5, 20);
  line(0, 43, width/6, 43);
  textSize(16);
  text("Input Leaf:", 5, 62);
  line(0, 69, width/6, 69);
  textSize(12);
  text(realLeaf.getInfo(), 5, 84);

  float[] sliderVals = {
    map(sliderList[0].getPos(), 0, width/6, 0, 1),
    map(sliderList[1].getPos(), 0, width/6, 0, 1),
    map(sliderList[2].getPos(), 0, width/6, 0, 1),
    map(sliderList[3].getPos(), 0, width/6, 0, 1),
    map(sliderList[4].getPos(), 0, width/6, 0, 1),
    map(sliderList[5].getPos(), 0, width/6, 0, 1),
    map(sliderList[6].getPos(), 0, width/6, 0, 1),
    map(sliderList[7].getPos(), 0, width/6, 0, 1),
    map(sliderList[8].getPos(), 0, width/6, 0, 1),
    map(sliderList[9].getPos(), 0, width/6, 0, 1)};

  textSize(10);

  if (!searchMode) {
    text("Slider Mode (Press Z to search)", 10, height-5);
    Grammar slideGram = gGen.buildGrammar(displayTemplate, sliderVals);
    fakeLeaf = new SysLeaf(realLeaf.getNum(), slideGram, displayTemplate);
    updateSliders(sliderList);
  }

  if (searchMode) {

```

```

    text("Search Mode (Press X for Sliders)", 10, height-5);
}

if(visionMode){
    fill(0);
    text("Vision Mode is ON (Press C)", 10,height-15);
}

else if(!visionMode){
    fill(0);
    text("Vision Mode is OFF (Press C)", 10,height-15);
}

iterSlider.display();
iterSlider.update();

//the turtle takes the iterations, where it's drawn, the unit length, the
initial angle

turt = new Turtle(fakeLeaf.gram, int(map(iterSlider.getPos(), 0, width/6,
4, fakeLeaf.maxIt)), 4, width/2, (float)realLeaf.base.y, -PI/2);

PGraphics canvas = turt.drawGrammar(realLeaf.col);
if (canvas!= null) {

    image(canvas, width-((width/12)*5), 0);
    PImage fakeLeafImg = canvas.get();

    OpenCV sysLeafCV = new OpenCV(this, fakeLeafImg);
    PImage fakeLeafProcessed = iProc.processImage(sysLeafCV, fakeLeaf);

    if (visionMode) {
        fakeLeafProcessed.resize((width/12)*5, 0);
        image(fakeLeafProcessed, width-(width/12)*5, 0);
        image(realLeafProcessed, width/6, height/2-(img.height/2));
    }

    stroke(0);
    fill(0);

    textSize(12);
    text(fakeLeaf.getInfo(), 5, 310);
    text("Similarity: " + realLeaf.evaluateSimilarity(fakeLeaf), 5, 480);

} else {

    fill(255, 0, 0);
    textSize(40);
    text("GENERATED LEAF" + "\n" + "OUT OF BOUNDS", width/2+200, height/2-
50);
    textSize(15);
    text("Lower parameters/iterations" + "\n" + "or switch template/input
leaf.", width/2+250, height/2 +50);
}

```



```

stroke(0);
fill(0);
textSize(16);
line(0, 238, width/6, 238);
text("Generated Leaf:" + "\n", 5, 260);
line(0, 270, width/6, 270);
textSize(14);
text("Template: "+ fakeLeaf.getTemplate(), 5, 290);
strokeWeight(3);
line(width-(width/12)*5, 0, width-(width/12)*5, height);

if(LSinfo==true) {
  drawLSInfo(fakeLeaf.gram);}
}

void keyPressed() {
  if (key=='z') {
    searchMode = true;
    SysLeaf bestfakeLeaf = gGen.search(realLeaf, fakeLeaf, 15, 5);
    fakeLeaf = bestfakeLeaf;
  }

  if (key=='x') {
    //turn search lock off,visionMode off, see leaf, allows sliders
    searchMode = false;
    visionMode = false;
  }

  if (key=='c') {
    //visionmode still allows slider use
    visionMode = !visionMode;
  }

  //take screenshot
  if (key=='m'){
    saveFrame("frames/#####.jpg");
  }

  if (key==CODED) {
    if (keyCode == UP) {
      t = t + 1;
    } else if (keyCode == DOWN) {
      t = t - 1;
    }
    t = constrain(t, 0, templateList.length-1);
    println(t);
    displayTemplate = templateList[t];
  }

  if (key==CODED) {
    if (keyCode == RIGHT) {
      leafNum = leafNum + 1;
    }
  }
}

```

```

    else if(keyCode == LEFT){
        leafNum = leafNum - 1;
    }

}

//pick samples leaves with numkeys and left and right arrows
if (key=='1') {
    leafNum = 1;
} else if (key=='2') {
    leafNum = 2;
} else if (key=='3') {
    leafNum = 3;
} else if (key=='4') {
    leafNum = 4;
} else if (key=='5') {
    leafNum = 10;
} else if (key=='6') {
    leafNum = 20;
} else if (key=='7') {
    leafNum = 30;
} else if (key=='8') {
    leafNum = 40;
} else if (key=='9') {
    leafNum = 50;
} else if (key=='0') {
    leafNum = 60;
}

//displays Lsystem info
if(key=='p'){
    LSinfo = !LSinfo;
}

leafNum = constrain(leafNum, 1, 70);
String n = str(leafNum);
if(leafNum < 10){
    n = "0" + n;
}
img = loadImage( n+"_r.jpg");
realLeaf = new Leaf(leafNum);

PImage imgCopy = img.get();
imgCopy.resize((width/12)*5, 0);

if (imgCopy.height>height) {
    img.resize(0, height);
    opencv = new OpenCV(this, img);
    realLeafProcessed = iProc.processImage(opencv, realLeaf);
    realLeafProcessed.resize(0, (height));
} else {

    img.resize((width/12)*5, 0);

```

```

    opencv = new OpenCV(this, img);
    realLeafProcessed = iProc.processImage(opencv, realLeaf);
    realLeafProcessed.resize((width/12)*5, 0);
}

}

void drawLSInfo(Grammar gram) {
    //Writes lsys to display screen.
    fill(255);
    rect(width/2-15, 80, width/2-50,100);
    fill(0);
    textSize(10);
    text("Axiom:          " + gram.axiom, width/2, 100);
    for (int j=0; j<gram productions.length; j++) {
        String prod = gram productions[j][0]+" ->" + gram productions[j][1];
        if (gram productions[j].length==3) {
            prod = prod+"          iff: "+gram productions[j][2];
        }
        text(prod, width/2, 100+((1+j)*10));
    }
}

void updateSliders(Slider[] slList) {
    for (int i=0; i<slList.length; i++) {
        slList[i].update();
        slList[i].display();
    }
}

```

Grammar.pde:

```

class Grammar {
    String axiom;
    String[][] productions;
    Parameters parameters;
    float delta;
    float scaleFactor;

    /*Grammars need at least four arguments:
    (1)The definition of the axiom. The starting String.
    (2)The definition of the productions.
    It is an array made up of arrays which each contain 1 production.
    The first element of each array is the Left-Hand Side (LHS) of the
    contained production.
    The second element of each array is the Right-Hand Side (RHS) of the
    contained production.
    (3)Delta, the angle value. Give in degrees.
    (4)Scaling Factor. Give 1 to not scale.
    Can give in either decimal (0.5) or fraction with float denom (1/2.0).

    They also take parameter definitions, see second constructor
    */

```

```

Grammar(String ax, String[][] pro, float delt, float scFa) {
    axiom = ax;
    productions = pro;
    delta = delt;
    scaleFactor = scFa;

    String[][] emp = {{}};
    parameters = new Parameters(emp);
}

Grammar(String ax, String[][] pro, Parameters par, float delt, float scFa)
{
    axiom = ax;
    productions = pro;
    parameters = par;
    delta = delt;
    scaleFactor = scFa;

    for (int i =0; i<productions.length; i++) {
        String parString = new String(productions[i][1]);
        for (Map.Entry ent : parameters.paramHash.entrySet()) {
            String parString2 = parString.replaceAll((String)ent.getKey(),
(String)ent.getValue());
            parString = parString2;
        }
        productions[i][1] = parString;
    }

    for (int i =0; i<productions.length; i++) {
        if (productions[i].length==3) {
            String condString = new String(productions[i][2]);
            for (Map.Entry ent : parameters.paramHash.entrySet()) {
                String condString2 = condString.replaceAll((String)ent.getKey(),
(String)ent.getValue());
                condString = condString2;
            }
            productions[i][2] = condString;
        }
    }
}

Grammar reWrite() {
    StringBuilder stringBuilder = new StringBuilder();
    for (int j = 0; j<axiom.length(); j++) {
        Boolean param = false;
        Boolean ruleApplied = false;
        String word = "";
        //println("value of character:" + " " +
String.valueOf(axiom.charAt(j)));
    }
}

```

```

if (j+1 > axiom.length()-1 || axiom.charAt(j+1)!='(') {
    word = word + axiom.charAt(j);
} else {
    param = true;
    int k = 0;
    while (axiom.charAt(j+k)!=')') {
        word = word + axiom.charAt(j+k);
        k = k + 1;
    }
    word = word + ')';
}

for (int i = 0; i<productions.length; i++) {

    //When symbol is not parametric, simply match symbol to production.
    if (param == false && word.equals(productions[i][0])) {
        stringBuilder.append(productions[i][1]);
        ruleApplied = true;
        break;
    }

    //When symbol *is* parametric, check conditions, and pass variable
    else if (param == true &&
word.charAt(0)==productions[i][0].charAt(0)) {
        String[] wordSym = match(word, "\\((.+?)\\)");
        String[] inputParams = wordSym[1].split(",");

        String[] prodVar = match(productions[i][0], "\\((.+?)\\)");
        String[] productionVariables = prodVar[1].split(",");

        //Check if number of # of parameters match
        if (inputParams.length != productionVariables.length) {
            break;
        }

        Boolean condition = true;
        if (productions[i].length==3) {
            String condExp = new String(productions[i][2]);

            for (int k = 0; k<inputParams.length; k++) {
                String condExp2 = condExp.replaceAll(productionVariables[k],
inputParams[k]);
                condExp = condExp2;
            }

            //construct condition with actual input num

            Result cond = Solver.evaluate(condExp);
            condition = cond.answer.toBoolean();

```

```

    }

    String passedPro = new String(productions[i][1]);
    for (int k = 0; k<inputParams.length; k++) {
        String repp = passedPro.replaceAll(productionVariables[k],
inputParams[k]);
        passedPro = repp;
    }

    if (condition==true) {
        stringBuilder.append(passedPro);
        ruleApplied = true;
        break;
    }
}

if (ruleApplied == false) {
    /*BUG ALERT !!!
    lsystem has loads of extra (0)s at the end,
    but this doesn't affect functionality */
    stringBuilder.append(word);
}
}
String finalString = stringBuilder.toString();
//println(finalString);
return new Grammar(finalString, productions, delta, scaleFactor);
}

String getStr() {
    return axiom;
}

float getAng() {
    return delta;
}

float getScaleFactor() {
    return scaleFactor;
}
}

```

GrammarGenerator.pde:

```

import gab.opencv.*;

class GrammarGenerator {

    PApplet sketch;
    OpenCV sysLeafCV;
    ImageProcessor iProc = new ImageProcessor();

    GrammarGenerator(PApplet sk) {

```

```

    sketch = sk;
}

Grammar buildGrammar(String template, float[] paramVals) {
    if (template.equals("Pinnate")) {
        String axiom = "{.S(0)}";
        String [][] productions = {
            {"S(t)", "P(t)"},
            {"P(t)", "! (5)G(LP, RP) [- (AN)L(t) .] [P(t+1)] [+ (AN)L(t) .]"},
            {"L(t)", "! (2)G(LL, RL)L(t-1)", "t>=BE"},
            {"G(s,r)", "G(s*r, r)"};
        String [][] paramDefs = {
            {"LP", str(map(paramVals[0], 0, 1, 2, 4.3))},
            {"RP", str(map(paramVals[1], 0, 1, 1, 1.25))},
            {"LL", str(map(paramVals[2], 0, 1, 1, 1.9))},
            {"RL", str(map(paramVals[3], 0, 1, 1.1, 1.38))},
            {"BE", str(map(paramVals[4], 0, 1, 0, -1))},
            {"AN", str(map(paramVals[5], 0, 1, 40, 80))};
        Parameters parameters = new Parameters(paramDefs);
        Grammar constructedGrammar = new Grammar(axiom, productions,
parameters, 60, 1);
        return constructedGrammar;
    } else if (template.equals("Palmate")) {
        String axiom = "{.S(0)}";
        String [][] productions = {
            {"S(t)", "[+ (ANN) + (ANN)B(t) .] [+ (ANN)P(t) .] [P(t) .] [- (ANN)P(t) .] [-
(ANN) - (ANN)B(t) ]"},
            {"P(t)", "! (5)G(LP,RP) [- (AN)L(t) .] [P(t+1)] [+ (AN)L(t) .]"},
            {"B(t)", "P(t)"},
            {"L(t)", "! (2)G(LL,RL)L(t-1)", "t>=BE"},
            {"G(s,r)", "G(s*r, r)"};
        String [][] paramDefs = {
            {"LP", str(map(paramVals[0], 0, 1, 1.2, 3))},
            {"RP", str(map(paramVals[1], 0, 1, 1.1, 1.25))},
            {"LL", str(map(paramVals[2], 0, 1, 1.1, 2))},
            {"RL", str(map(paramVals[3], 0, 1, 1, 1.5))},
            {"BE", str(map(paramVals[4], 0, 1, 0, -4))},
            {"AN", str(map(paramVals[5], 0, 1, 40, 80))},
            {"ANN", str(map(paramVals[6], 0, 1, 40, 70))};
        Parameters parameters = new Parameters(paramDefs);
        Grammar constructedGrammar = new Grammar(axiom, productions,
parameters, 60, 1);
        return constructedGrammar;
    } else if (template.equals("PinLobed")) {
        String axiom = "{.S(0)}";
        String [][] productions = {
            {"S(t)", "P(t,LO)"},
            {"P(t,i)", "! (6)G(LP,RP) [- (AN)A(t) ] [P(t+1,i-1)] [+ (AN)A(t) ]", "i>=0"},
            {"P(t,i)", "! (6)G(LP,RP)N(t)", "i<0"},
            {"A(t)", "! (4)G(LL,RL) [- (ANN)L(t-1) .] [A(t+1)] [+ (ANN)L(t-1) .]"},
            "t>=BE"},
            {"L(t)", "! (3)G(LT,RT)L(t-1)", "t>=BE"},
            {"N(t)", "! (4) [A(t+1)]", "t>=BE"},
            {"G(s,r)", "G(s*r, r)"};

        String [][] paramDefs = {

```

```

    {"LP", str(map(paramVals[0], 0, 1, 6, 8))},
    {"RP", str(map(paramVals[1], 0, 1, 1.1, 1.26))},
    {"LL", str(map(paramVals[2], 0, 1, 1, 1.4))},
    {"RL", str(map(paramVals[3], 0, 1, 1.2, 1.45))},
    {"LT", str(map(paramVals[4], 0, 1, 1, 1.4))},
    {"RT", str(map(paramVals[5], 0, 1, 1.1, 1.2))},
    {"BE", str(map(paramVals[6], 0, 1, -1, -3))},
    {"AN", str(map(paramVals[7], 0, 1, 35, 60))},
    {"ANN", str(map(paramVals[8], 0, 1, 30, 85))},
    {"LO", str(map(paramVals[9], 0, 1, 1, 4))}};

Parameters parameters = new Parameters(paramDefs);
Grammar constructedGrammar = new Grammar(axiom, productions,
parameters, 60, 1);
return constructedGrammar;
} else if (template.equals("Tulip")) {
/*must be tweaked with sliders to resemble tulip tree
not included in search */
String axiom = "{.S(0)}";
String [][]productions = {
    {"S(t)", "Z(t,LO)"},
    {"Z(t,i)", "! (6) [- (AN) A(t)] [P(t+1,i-1)] [+ (AN) A(t)]", "i>=0"},
    {"P(t,i)", "! (6) G(LP,RP) [- (AN) A(t)] [P(t+1,i-1)] [+ (AN) A(t)]", "i>=0"},
    {"A(t)", "! (4) G(LL,RL) [- (ANN) L(t-1) .] [A(t+1)] [+ (ANN) L(t-1) .]",
"t>=BE"},
    {"L(t)", "! (3) G(LT,RT)L(t-1)", "t>=BE"},
    {"G(s,r)", "G(s*r, r)"};

String [][]paramDefs = {
    {"LP", str(map(paramVals[0], 0, 1, 6, 8))},
    {"RP", str(map(paramVals[1], 0, 1, 1.1, 1.26))},
    {"LL", str(map(paramVals[2], 0, 1, 1, 4))},
    {"RL", str(map(paramVals[3], 0, 1, 1, 1.5))},
    {"LT", str(map(paramVals[4], 0, 1, 1, 1.6))},
    {"RT", str(map(paramVals[5], 0, 1, 1.1, 1.5))},
    {"BE", str(map(paramVals[6], 0, 1, -1, -3))},
    {"AN", str(map(paramVals[7], 0, 1, 35, 60))},
    {"ANN", str(map(paramVals[8], 0, 1, 30, 85))},
    {"LO", str(map(paramVals[9], 0, 1, 1, 4))}};

Parameters parameters = new Parameters(paramDefs);
Grammar constructedGrammar = new Grammar(axiom, productions,
parameters, 60, 1);
return constructedGrammar;
} else if (template.equals("ComPalm")) {
//compound palmate leaf, not in search
String axiom = "{.S(0)}";
String [][]productions = {
    {"S(t)", "[+ (ANN) + (ANN) P(t) ] . [+ (ANN) P(t) ] . [P(t) ] . [- (ANN) P(t) ] . [-
(ANN) - (ANN) P(t) ]"},
    {"P(t)", "! (5) G(LP,RP) [- (AN) L(t) .] [P(t+1)] [+ (AN) L(t) .]"},
    {"L(t)", "! (2) G(LL,RL)L(t-1)", "t>=BE"},
    {"G(s,r)", "G(s*r, r)"};

String [][]paramDefs = {
    {"LP", str(map(paramVals[0], 0, 1, 1.2, 3))},
    {"RP", str(map(paramVals[1], 0, 1, 1.1, 1.25))},
    {"LL", str(map(paramVals[2], 0, 1, 1.1, 2))},
    {"RL", str(map(paramVals[3], 0, 1, 1, 1.5))},

```



```

        {"BE", str(map(paramVals[4], 0, 1, 2, -4))},
        {"AN", str(map(paramVals[5], 0, 1, 40, 80))},
        {"ANN", str(map(paramVals[6], 0, 1, 40, 70))}};
    Parameters parameters = new Parameters(paramDefs);
    Grammar constructedGrammar = new Grammar(axiom, productions,
parameters, 60, 1);
    return constructedGrammar;

}

else {
    //Don't think this ever gets run, here just in case
    //old version of palmate temp, dunno what it looks like
    String axiom = "[++A(0)].+(AN)A(0).[A(0)].-(AN)A(0).--A(0)";
    String [][]productions = {"A(t)", "G(LA,RA)[-L(t).][A(t+1)][+L(t).]"},
        {"L(t)", "G(LL,RL)L(t-1)", "t>=-8"},
        {"G(s,r)", "G(s*r, r)"};
    String [][]paramDefs = {
        {"LA", str(map(paramVals[0], 0, 1, 2, 5))},
        {"RA", str(map(paramVals[1], 0, 1, 1, 1.25))},
        {"LL", str(map(paramVals[2], 0, 1, 1, 1.9))},
        {"RL", str(map(paramVals[3], 0, 1, 1.1, 1.35))},
        {"AN", str(map(paramVals[4], 0, 1, 40, 80))}};
    Parameters parameters = new Parameters(paramDefs);
    Grammar constructedGrammar = new Grammar(axiom, productions,
parameters, 60, 1);
    return constructedGrammar;
}
}

```

```

SysLeaf search(Leaf realLeaf, SysLeaf fakeLeaf, int poolSize, int
generations) {

    List<SysLeaf> candList = makeCandidates(realLeaf, fakeLeaf, poolSize);

    //candList is sorted by fitness, so return best individual.
    return candList.get(0);
}

```

```

List<SysLeaf> makeCandidates(Leaf realLeaf, SysLeaf fakeLeaf, int poolSz) {
    Grammar[] candidateGrams = new Grammar[poolSz];
    List<SysLeaf> candidateLeavesAll = new ArrayList<SysLeaf>();

    float fitness;
    String template = "Pinnate";

    for (int i=0; i<candidateGrams.length; i++) {
        //for serach, hill climbing/sim anneal? Depends on generation?
        //modify template probabilities??
        //mutate top half of candidates
        PGraphics candiCanvas = null;
    }
}

```

```

//keep generating candidate until valid, in-bounds one arrives
while (null == candiCanvas) {
    float[] paramVals = {random(1), random(1), random(1), random(1),
random(1), random(1), random(1), random(1), random(1), random(1)};
    //need to store venation/template type in sysLeaf...
    float dice = random(1);
    int itMax = 14;
    if (dice<=0.333333) {
        template = "Pinnate";
        candidateGrams[i] = buildGrammar(template, paramVals);
        itMax = 14;
    } else if (dice>0.333333 && dice<0.666666) {
        template = "Palmate";
        candidateGrams[i] = buildGrammar(template, paramVals);
        itMax = 11;
    } else if (dice>=0.666666) {
        template = "PinLobed";
        candidateGrams[i] = buildGrammar(template, paramVals);
        itMax = 12;
    } else {
        itMax = 14;
        template = "Pinnate";
        candidateGrams[i] = buildGrammar(template, paramVals);
    }

    Turtle turty = new Turtle(candidateGrams[i], itMax, 4, width/2,
(float)realLeaf.base.y, -PI/2);
    candiCanvas = turty.drawGrammar(realLeaf.col);
}

PImage sysLeafImg = candiCanvas.get();
image(sysLeafImg, 0, 0);
OpenCV sysLeafCV = new OpenCV(sketch, sysLeafImg);
SysLeaf fakeLeafCan = new SysLeaf(realLeaf.getNum(), candidateGrams[i],
template);
PImage sysLeafProcessed = iProc.processImage(sysLeafCV, fakeLeafCan);

fitness = realLeaf.evaluateSimilarity(fakeLeafCan);
fakeLeafCan.setFitness(fitness);
println(i);
//saving cadidates! filename is fitness.
candiCanvas.save("candidates/" +fitness+ ".jpg");
candidateLeavesAll.add(fakeLeafCan);
}

println(candidateLeavesAll.size());
Collections.sort(candidateLeavesAll);
return candidateLeavesAll;
}
}

```

ImageProcessor.pde:

```

class ImageProcessor {
  Core cvCore;
  Imgproc process;

  Mat thresholdMat;
  Mat holderMat;

  ArrayList<MatOfPoint> contourList;
  Mat hier;
  int contourIndex;
  Mat drawnContoursMat;

  ImageProcessor() {
    process = new Imgproc();
    cvCore = new Core();
  }

  PImage processImage(OpenCV opencv, Leaf leafy) {

    opencv.threshold(220);
    thresh = opencv.getSnapshot();

    contourList = new ArrayList();
    thresholdMat = opencv.getGray();
    drawnContoursMat = opencv.getColor();
    hier = new Mat();
    holderMat = opencv.imitate(drawnContoursMat);

    process.findContours(thresholdMat, contourList, hier, Imgproc.RETR_LIST,
    Imgproc.CHAIN_APPROX_SIMPLE);

    //seeking 2nd largest contour (largest contour is outline of picture)
    float[] areas = new float[contourList.size()];
    for (int i=0; i<contourList.size(); i++) {
      //println(process.contourArea(contourList.get(i)));
      areas[i] = (float)process.contourArea(contourList.get(i));
    }

    areas = sort(areas);
    double targetArea;
    try {
      targetArea = (double)areas[areas.length-2];
    }
    catch(IndexOutOfBoundsException e) {
      return null;
    }

    contourIndex = 0;

    for (int j=0; j<contourList.size(); j++) {
      if (targetArea == process.contourArea(contourList.get(j))) {
        contourIndex = j;
      }
    }
  }
}

```

```

    }

    cvCore.bitwise_not(holderMat, drawnContoursMat); //inverts contour image
    so background is white

    MatOfPoint leafContour = contourList.get(contourIndex);

    leafy.setContour(leafContour);

    //draw leafContour
    process.drawContours(drawnContoursMat, contourList, contourIndex, new
    Scalar(0, 200, 100), 15);

    //Calculate bounding box, draw it, save lamina length to leaf.
    Rect boundBox = process.boundingRect(leafContour);
    //cvCore.rectangle(drawnContoursMat, boundBox.tl(), boundBox.br(), new
    Scalar(0, 0, 255), 8);
    leafy.setLamLength(boundBox.size().height);
    leafy.setLamWidth(boundBox.size().width);

    //Midline of bounding box.
    Point apex = new Point((boundBox.tl().x+boundBox.br().x)/2,
    boundBox.tl().y);
    Point base = new Point((boundBox.tl().x+boundBox.br().x)/2,
    boundBox.br().y);

    //cvCore.circle(drawnContoursMat, apex, 6, new Scalar(170, 0, 255), 4);
    //cvCore.circle(drawnContoursMat, base, 6, new Scalar(255, 255, 40), 4);
    //cvCore.line(drawnContoursMat, apex, base, new Scalar(255, 0, 0), 8);

    leafy.setApex(apex);
    leafy.setBase(base);

    findShapeClass(boundBox, leafy);

    /*Convex hull on full contour
    //help from here:
    //http://stackoverflow.com/questions/18143077/computer-vision-filtering-
    convex-hulls-and-convexity-defects-with-opencv
    MatOfInt hullIndices1 = new MatOfInt();
    ArrayList<Point> hullPointList1 = new ArrayList();
    process.convexHull(leafContour, hullIndices1);
    for (int j=0; j < hullIndices1.toList().size(); j++) {
    hullPointList1.add(leafContour.toList().get(hullIndices1.toList().get(j)
    ));
    }
    MatOfPoint hullMatPoint1 = new MatOfPoint();
    hullMatPoint1.fromList(hullPointList1);
    ArrayList<MatOfPoint> hullMatPointList1 = new ArrayList();
    hullMatPointList1.add(hullMatPoint1);
    process.drawContours(drawnContoursMat, hullMatPointList1, 0, new
    Scalar(50, 50, 50), 10);

```

```

//convexity defects on full contour:
MatOfInt4 defectMat = new MatOfInt4();
process.convexityDefects(leafContour, hullIndices1, defectMat);
List<Integer> defectList = defectMat.toList();

for (int i=2; i<defectList.size(); i=i+4) {
int cIndex = defectList.get(i);
Point[] cPoints = leafContour.toArray();
cvCore.circle(drawnContoursMat, cPoints[cIndex], 24, new Scalar(255,
255, 40), 4);
//convexity defect points:
//println(cPoints[cIndex]);
}
*/

//polygon approximation for better lobe estimate.
MatOfPoint2f contourFloat = new MatOfPoint2f(leafy.contour.toArray());
MatOfPoint2f polygonApprox = new MatOfPoint2f();

process.approxPolyDP(contourFloat, polygonApprox, 4, true);

MatOfPoint approxContour = new MatOfPoint();
polygonApprox.convertTo(approxContour, CvType.CV_32S);

ArrayList<MatOfPoint> approxContourList = new ArrayList();
approxContourList.add(approxContour);

process.drawContours(drawnContoursMat, approxContourList, 0, new
Scalar(0, 50, 160), 10);

//Convex hull on polyapprox contour
MatOfInt hullIndices = new MatOfInt();
ArrayList<Point> hullPointList = new ArrayList();
process.convexHull(approxContour, hullIndices);
for (int j=0; j < hullIndices.toList().size(); j++) {
hullPointList.add(leafContour.toList().get(hullIndices.toList().get(j)));
}
MatOfPoint hullMatPoint = new MatOfPoint();
hullMatPoint.fromList(hullPointList);
ArrayList<MatOfPoint> hullMatPointList = new ArrayList();
hullMatPointList.add(hullMatPoint);
//draw convex hull--but it doesn't really show when using polyapprox
//process.drawContours(drawnContoursMat, hullMatPointList, 0, new
Scalar(255, 50, 50), 10);

//convexity defects
MatOfInt4 defectMat = new MatOfInt4();

```

```

//approxpoly freaks out at low iterations why!!!!

int lobeCount = 0;

if (hullIndices.rows() >=3) {
    process.convexityDefects(approxContour, hullIndices, defectMat);

    List<Integer> defectList = defectMat.toList();

    for (int i=2; i<defectList.size(); i=i+4) {
        int cIndex = defectList.get(i);
        Point[] cPoints = approxContour.toArray();
        cvCore.circle(drawnContoursMat, cPoints[cIndex], 17, new Scalar(255,
255, 40), 4);
        lobeCount ++;
        //convexity defect points:
        //println(cPoints[cIndex]);
    }
}

//get area
float area = (float)process.contourArea(leafContour);

//get perimeter
MatOfPoint2f conFloat = new MatOfPoint2f(leafy.contour.toArray());
float perimeter = (float)process.arcLength(conFloat, true);

leafy.setApproxPolyContour(approxContour);
leafy.setLobeNum(lobeCount);
leafy.setArea(area);
leafy.setPerimeter(perimeter);

/*minEnclosing circle on polygonApprox
Point circCent = new Point();
float[] radius = new float[1];
process.minEnclosingCircle(polygonApprox, circCent, radius);
cvCore.circle(drawnContoursMat, circCent, (int)radius[0], new Scalar(0,
0, 255), 4);
*/

//make final image, turn into PImage
PImage drawnContoursPImage = createImage(drawnContoursMat.width(),
drawnContoursMat.height(), RGB);
opencv.toPImage(drawnContoursMat, drawnContoursPImage);
return drawnContoursPImage;
}

void findShapeClass(Rect boundBox, Leaf leafy) {

```

```

ArrayList<Point> leftMostList = new ArrayList();
Point leftMost = new Point();
MatOfPoint2f contourFloat = new MatOfPoint2f(leafy.contour.toArray());
//find widest point on left
for (double y = boundBox.tl().y; y<boundBox.br().y; y++) {
    Point leftMostCan = new Point(boundBox.tl().x, y);
    if (process.pointPolygonTest(contourFloat, leftMostCan, true)==0) {
        leftMost = leftMostCan;
        leftMostList.add(leftMost);
    }
}

ArrayList<Point> rightMostList = new ArrayList();
Point rightMost = new Point();
//find widest point on right
for (double y2 = boundBox.br().y; y2>boundBox.tl().y; y2=y2-1) {
    Point rightMostCan = new Point(boundBox.br().x, y2);
    if (process.pointPolygonTest(contourFloat, rightMostCan, true)+1==0) {
        rightMost = rightMostCan;
        rightMostList.add(rightMost);
    }
}

ArrayList<Point> acrossFromLList = new ArrayList();
Point acrossFromL = new Point();
//find points across from extreme Left points
for (int i = 0; i<leftMostList.size(); i++) {
    for (double x = boundBox.br().x; x>leftMostList.get(i).x; x=x-1) {
        Point acrossFromLCan = new Point(x, leftMostList.get(i).y);
        if (process.pointPolygonTest(contourFloat, acrossFromLCan, true)==0)
        {
            acrossFromL = acrossFromLCan;
            acrossFromLList.add(acrossFromL);
            break;
        }
    }
}

ArrayList<Point> acrossFromRList = new ArrayList();
Point acrossFromR = new Point();
//find points across from extreme Right points
for (int i = 0; i<rightMostList.size(); i++) {
    for (double x = boundBox.tl().x; x<rightMostList.get(i).x-1; x=x+1) {
        Point acrossFromRCan = new Point(x, rightMostList.get(i).y);
        if (process.pointPolygonTest(contourFloat, acrossFromRCan, true)==0)
        {
            acrossFromR = acrossFromRCan;
            acrossFromRList.add(acrossFromR);
            break;
        }
    }
}

float[] distListL = new float[leftMostList.size()];
for (int i = 0; i<leftMostList.size(); i++) {
    try {

```

```

        distListL[i] = ((float)euclideanDist(leftMostList.get(i),
acrossFromLList.get(i)));
    }
    catch(IndexOutOfBoundsException e) {
        distListL[i] = 0;
    }
}
float maxLeftDist = 0;
if(distListL.length>0) {
maxLeftDist = max(distListL);}

float[] distListR = new float[rightMostList.size()];
for (int i = 0; i<rightMostList.size(); i++) {
    try{
        distListR[i] = ((float)euclideanDist(rightMostList.get(i),
acrossFromRList.get(i)));
    }
    catch(IndexOutOfBoundsException e){
        distListR[i] = 0;
    }
}
float maxRightDist= 0;
if(distListR.length>0){
maxRightDist = max(distListR);}

Point widest = new Point();
if (maxLeftDist > maxRightDist) {
    int ind = 0;
    for (int i = 0; i<distListL.length; i++) {
        if (distListL[i] == maxLeftDist) {
            ind = i;
        }
    }
    widest = leftMostList.get(ind);
    //cvCore.line(drawnContoursMat, leftMostList.get(ind),
acrossFromLList.get(ind), new Scalar(0, 0, 0), 14);
    //cvCore.circle(drawnContoursMat, acrossFromLList.get(ind), 12, new
Scalar(200, 40, 40), 4);
    //cvCore.circle(drawnContoursMat, leftMostList.get(ind), 12, new
Scalar(40, 40, 40), 4);
} else {

    int ind = 0;
    for (int i = 0; i<distListR.length; i++) {
        if (distListR[i] == maxRightDist) {
            ind = i;
        }
    }
    widest = rightMostList.get(ind);
    //cvCore.line(drawnContoursMat, rightMostList.get(ind),
acrossFromRList.get(ind), new Scalar(0, 0, 0), 14);
    //cvCore.circle(drawnContoursMat, rightMostList.get(ind), 12, new
Scalar(200, 40, 40), 4);
    //cvCore.circle(drawnContoursMat, acrossFromRList.get(ind), 12, new
Scalar(40, 40, 40), 4);
}

```



```

    }

    //Divide into 5ths
    for (int i = 0; i<6; i++) {
        Point left = new Point(bbox.tl().x, bbox.tl().y +
(leafy.laminaLength/5.0)*i);
        Point right = new Point(bbox.br().x, bbox.tl().y +
(leafy.laminaLength/5.0)*i);
        //draw 5th lines:
        //cvCore.line(drawnContoursMat, left, right, new Scalar(255, 0, 255),
9);
        if (widest.y<bbox.tl().y+(leafy.laminaLength/5.0)*i &&
leafy.wideFifth==0) {
            leafy.setWideFifth(i);
        }
    }
}

double euclideanDist(Point a, Point b) {
    Point euc = new Point(a.x - b.x, a.y - b.y);
    double toSq = euc.x*euc.x + euc.y*euc.y;
    return Math.sqrt(toSq);
}

int distance_2(Point[] a, Point[] b) {
    /*hausdorff distance helper
    I DID NOT WRITE THIS
    http://stackoverflow.com/questions/21482534/how-to-use-shape-distance-
and-common-interfaces-to-find-hausdorff-distance-in-op
    */
    int maxDistAB = 0;
    for (int i=0; i<a.length; i++)
    {
        int minB = 1000000;
        for (int j=0; j<b.length; j++)
        {
            int dx = (int)(a[i].x - b[j].x);
            int dy = (int)(a[i].y - b[j].y);
            int tmpDist = dx*dx + dy*dy;

            if (tmpDist < minB)
            {
                minB = tmpDist;
            }
            if ( tmpDist == 0 )
            {
                break; // can't get better than equal.
            }
        }
        maxDistAB += minB;
    }
    return maxDistAB;
}

```

```

double distance_hausdorff(Point[] a, Point[] b ) {
    /*hausdorff distance function
    I DID NOT WRITE THIS
    http://stackoverflow.com/questions/21482534/how-to-use-shape-distance-and-common-interfaces-to-find-hausdorff-distance-in-op
    */
    int maxDistAB = distance_2( a, b );
    int maxDistBA = distance_2( b, a );
    int maxDist = max(maxDistAB, maxDistBA);

    return Math.sqrt((double)maxDist);
}
}

```

Leaf.pde:

```

class Leaf {

    int number;
    float laminaLength= 0;
    float laminaWidth = 0;
    float lwRatio;
    float lamArea = 0;
    float lamPerimeter = 0;
    int wideFifth = 0;
    int lobeNum = 0;
    MatOfPoint contour;
    MatOfPoint approxPolyContour;

    Point apex;
    Point base;

    Imgproc pro;
    ImageProcessor myPro;

    color col;

    String shapeClass;

    Leaf(int num) {
        number = num;
        col = color(34, 200, 40, 180);

        pro = new Imgproc();
        myPro = new ImageProcessor();
    }

    int getNum() {
        return number;
    }

    void setLamLength(double len) {
        laminaLength = (float)len;
        if (laminaWidth != 0) {

```

```

        lwRatio = laminaLength/laminaWidth;
    }
}

void setLamWidth(double wid) {
    laminaWidth = (float)wid;
    if (laminaLength != 0) {
        lwRatio = laminaLength/laminaWidth;
    }
}

void setWideFifth(int fif) {
    wideFifth = fif;
    if (wideFifth==1 || wideFifth==2) {
        shapeClass = "Obovate";
    } else if (wideFifth==3) {
        shapeClass = "Elliptic";
    } else if (wideFifth==4 || wideFifth==5) {
        shapeClass = "Ovate";
    } else {
        shapeClass = "Special?";
    }
}

void setContour(MatOfPoint con) {
    contour = con;
}

void setApproxPolyContour(MatOfPoint apc){
    approxPolyContour = apc;
}

void setShapeClass(String sc) {
    shapeClass = sc;
}

void setLobeNum(int ln){
    lobeNum = ln;
}

void setApex(Point a){
    apex = a;
}

void setBase(Point b){
    base = b;
}

void setArea(float a){
    lamArea = a;
}

void setPerimeter(float p){
    lamPerimeter = p;
}

```

```

String getInfo() {
    String info = "";
    info = info +
        "Leaf Number: " + number + "\n" +
        "Length: " + laminaLength + "\n" +
        "Width: " + laminaWidth + "\n" +
        "L:W Ratio: " + lwRatio + "\n" +
        "Area: " + lamArea + "\n" +
        "Perimeter: " + lamPerimeter + "\n" +
        //won't bother displaying until lobes is more accurate
        //"Number of Lobes: " + lobeNum + "\n" +
        "Shape Class: " + shapeClass+ "\n";
    return info;
}

float evaluateSimilarity(Leaf other) {
    //this is the fitness function!

    Point[] cPoints = contour.toArray();
    Point[] oPoints;
    try{
        oPoints = other.contour.toArray();} //buggy
    catch (NullPointerException e) {
        oPoints = null;
    }

    float lwR = 1-map(abs(lwRatio - other.lwRatio), 0, 3, 0, 1);
    float len = 1-map(abs(laminaLength-other.laminaLength), 0, 300, 0, 1);
    float wid = 1-map(abs(laminaWidth-other.laminaWidth), 0, 300, 0, 1);
    float lobe = map(abs(lobeNum-other.lobeNum), 0, 6, 1, 0);

    float sc;
    if(shapeClass.equals(other.shapeClass)){
        sc = 1;
    }
    else{
        sc = 0;
    }

    float fitness = lwR*25 + len*10 +wid*10 + + area*15 +lobe*15 +sc*25;

    /*hausdorff distance, fool with this another time
    float hausdorff;
    if(null!=oPoints){
        hausdorff = 1000/(float)myPro.distance_hausdorff(cPoints, oPoints);
    }
    else
        hausdorff = 0; */

    return fitness;
}

```

```
}
```

Parameters.pde:

```
import java.util.Map;

class Parameters {

    HashMap<String, String> paramHash = new HashMap<String, String>();
    String[][] paramList;

    Parameters(String[][] pList) {
        paramList = pList;

        if (paramList[0].length!=0) {
            for (int i=0; i<paramList.length; i++) {
                addParam(paramList[i]);
            }
        }
    }

    void addParam(String[] pairToAdd) {
        paramHash.put(pairToAdd[0], pairToAdd[1]);
    }

    String getVal(String keyy) {
        String val = paramHash.get(keyy);
        return val;
    }
}
```

Slider.pde:

```
/* Slider class
I did not write this.
Modified from https://processing.org/examples/scrollbar.html
*/

class Slider {
    int swidth, sheight; // width and height of bar
    float xpos, ypos; // x and y position of bar
    float spos, newspos; // x position of slider
    float sposMin, sposMax; // max and min values of slider
    int loose; // how loose/heavy
    boolean over; // is the mouse over the slider?
    boolean locked;
    float ratio;

    Slider (float xp, float yp, int sw, int sh, int l, Boolean maxStart) {
        swidth = sw;
        sheight = sh;
        int widthtoheight = sw - sh;
        ratio = (float)sw / (float)widthtoheight;
    }
}
```

```

xpos = xp;
ypos = yp-sheight/2;

if(!maxStart){
spos = xpos + swidth/2 - sheight/2;
}

else{
spos = xpos + swidth - sheight;
}

newspos = spos;
sposMin = xpos;
sposMax = xpos + swidth - sheight;
loose = 1;
}

void update() {
if (overEvent()) {
over = true;
} else {
over = false;
}
if (mousePressed && over) {
locked = true;
}
if (!mousePressed) {
locked = false;
}
if (locked) {
newspos = constrain(mouseX-sheight/2, sposMin, sposMax);
}
if (abs(newspos - spos) > 1) {
spos = spos + (newspos-spos)/loose;
}
}

float constrain(float val, float minv, float maxv) {
return min(max(val, minv), maxv);
}

boolean overEvent() {
if (mouseX > xpos && mouseX < xpos+swidth &&
mouseY > ypos && mouseY < ypos+sheight) {
return true;
} else {
return false;
}
}

void display() {
noStroke();
fill(204);
rect(xpos, ypos, swidth, sheight);
if (over || locked) {

```

```

        fill(0, 0, 0);
    } else {
        fill(102, 102, 102);
    }
    rect(spos, ypos, sheight, sheight);
}

float getPos() {
    // Convert spos to be values between
    // 0 and the total width of the scrollbar
    return spos * ratio;
}
}

```

SysLeaf.pde:

```

class SysLeaf extends Leaf implements Comparable<SysLeaf>{

    Grammar gram;
    float fitness = 0;
    String template;
    int maxIt;

    SysLeaf(int num, Grammar g, String temp){
        super(num);
        gram = g;
        template = temp;
        if(template=="Pinnate"){
            maxIt = 14;
        }
        if(template=="Palmate"){
            maxIt = 11;
        }
        if(template=="PinLobed"){
            maxIt = 12;
        }
    }

    String getLSInfo(){
        return "um";
    }

    void setFitness(float f){
        fitness = f;
    }

    float getFitness(){
        return fitness;
    }

    String getTemplate(){
        return template;
    }
}

```

```

//comparator function in order to sort candidate leaves by fitness
int compareTo(SYSLeaf o){
    float otherFit = ((SYSLeaf)o).getFitness();
    if(this.fitness - otherFit > 0){
        return -1;
    }
    else if(this.fitness - otherFit < 0){
        return 1;
    }
    else{
        return 0;
    }
}
}
}

```

Turtle.pde:

```

import java.util.Stack;

/*Turtles need at least three arguments:
(1)The Grammar to be drawn.
(2)Number of iterations.
Setting iterations too high for certain L-Systems
may cause Processing to crash.
(3)Unit Length in pixels.

Second constructor has other inputs too.
*/

class Turtle {
    float len;
    Grammar grammar;
    float x;
    float y;
    float heading;
    TurtleState state;
    Stack<TurtleState> turtleStack;

    PGraphics canvas;

    Turtle(Grammar gram, int iterations, int size) {

        canvas = createGraphics((width/12)*5, height);
        len = float(size);
        grammar = gram;
        x = canvas.width/2;
        y = canvas.height-200;
        heading = -PI/2;

        state = new TurtleState(x, y, heading);
        turtleStack = new Stack<TurtleState>();
    }
}

```



```

turtleStack.push(state);

for (int i = 0; i<iterations; i++) {
    len = len * grammar.getScaleFactor();
}

for (int i = 0; i<iterations; i++) {
    Grammar newGram = grammar.reWrite();
    grammar = newGram;
}
}

//Second constructor with specified location, rotation (radians):
Turtle(Grammar gram, int iterations, int size, float xin, float yin, float
rotation) {

    canvas = createGraphics((width/12)*5, height);
    len = float(size);
    grammar = gram;
    x = canvas.width/2;
    y = yin;
    heading = rotation;

    state = new TurtleState(x, y, heading);
    turtleStack = new Stack<TurtleState>();
    turtleStack.push(state);

    for (int i = 0; i<iterations; i++) {
        len = len * grammar.getScaleFactor();
    }

    for (int i = 0; i<iterations; i++) {
        Grammar newGram = grammar.reWrite();
        grammar = newGram;
    }
}

PGraphics drawGrammar(color col) {

    int thickness = 2;
    PShape shape = null;

    canvas.beginDraw();
    canvas.background(255);
    canvas.strokeWeight(thickness);

    String str = grammar.getStr();
    //useful for debugging:
    //println(str + " ");

    //Turtle Interpretation of Symbols:
    /*The way parameters are read here is kinda clunky + repetitive

```

```

and I've just copied and pasted it for every parameterized symbol lol*/
for (int j = 0; j<str.length(); j++) {

    if (str.charAt(j)=='F') {
        if (str.charAt(j+1)=='(') {
            String parameters = "";
            int k = 2;
            while (str.charAt(j+k)!=')') {
                parameters = parameters + str.charAt(j+k);
                k = k+1;
            }
            float[] parameterList = parseParam(parameters);

            forward(parameterList[0]);
        } else
            forward(1);
    }

    if (str.charAt(j)=='G') {
        if (str.charAt(j+1)=='(') {
            String parameters = "";
            int k = 2;
            while (str.charAt(j+k)!=')') {
                parameters = parameters + str.charAt(j+k);
                k = k+1;
            }
            float[] parameterList = parseParam(parameters);
            forward(parameterList[0]);
        } else
            forward(1);
    }

    if (str.charAt(j) == '!') {
        if (str.charAt(j+1)=='(') {
            String parameters = "";
            int k = 2;
            while (str.charAt(j+k)!=')') {
                parameters = parameters + str.charAt(j+k);
                k = k+1;
            }
            float[] parameterList = parseParam(parameters);
            //println(parameterList[0]);
            canvas.strokeWeight(parameterList[0]);
        } else
            canvas.strokeWeight(thickness-1);
    }

    if (str.charAt(j)=='[') {
        //push
        turtleStack.push(state);
    }

    if (str.charAt(j)=='']') {
        //pop
        state = turtleStack.pop();
    }
}

```

```

if (str.charAt(j)=='{' ) {
    shape = createShape();
    shape.beginShape();
    shape.fill(red(col), green(col), blue(col), 190);
}

if (str.charAt(j)=='}') {
    shape.noStroke();
    shape.endShape(CLOSE);
    canvas.noStroke();
    canvas.shape(shape, 0, 0);
}

if (str.charAt(j)=='.' && !Character.isDigit(str.charAt(j+1))) {
    if
(state.getX()>=canvas.width||state.getX()<=0||state.getY()>=canvas.height-
1||state.getY()<=0) {
        //println("Throw this leaf away.");
        turtleStack.pop();
        canvas.endDraw();
        return null;
    } else {
        shape.vertex(state.getX(), state.getY());
    }
    //uncomment to mark vertex with ellipse:
    //ellipse(state.getX(),state.getY(),3,3);
}

if (str.charAt(j)=='+') {
    //turn left
    float angle;

    if (str.charAt(j+1)=='(') {
        String parameters = "";
        int k = 2;
        while (str.charAt(j+k)!=')') {
            parameters = parameters + str.charAt(j+k);
            k = k+1;
        }
        float[] parameterList = parseParam(parameters);
        angle = parameterList[0];
    } else {
        //println(" state get heading: "+ state.getHeading());
        angle = grammar.getAng();
    }

    heading = state.getHeading() - radians(angle);
    //print(" radians of gramm ang: " + radians(grammar.getAng()));
    //print(" new heading: " + heading);
    x = state.getX();
    y = state.getY();
    state = new TurtleState(x, y, heading);
}

if (str.charAt(j)=='-') {

```

```

//turn right
float angle;
if (str.charAt(j+1)=='(') {
    String parameters = "";
    int k = 2;
    while (str.charAt(j+k)!=')') {
        parameters = parameters + str.charAt(j+k);
        k = k+1;
    }
    float[] parameterList = parseParam(parameters);

    angle = parameterList[0];
} else {

    angle = grammar.getAng();
}

heading = state.getHeading() + radians(angle);
x = state.getX();
y = state.getY();
state = new TurtleState(x, y, heading);
}
}

turtleStack.pop();
canvas.endDraw();
return canvas;
}

void forward(float sc) {
    float scale = sc;
    canvas.stroke(0, 0, 0);

    //computing turtle movement using heading angle.
    float nx = state.getX() + (scale*len)*cos(state.getHeading());
    float ny = state.getY() + (scale*len)*sin(state.getHeading());
    canvas.line(state.getX(), state.getY(), nx, ny);
    heading = state.getHeading();
    x = nx;
    y = ny;
    state = new TurtleState(x, y, heading);
}

float[] parseParam(String parString) {
    //Uses QScript to parse expression strings
    String[] split = parString.split(",");
    float[] paramFloats = new float[split.length];
    for (int i = 0; i<paramFloats.length; i++) {
        Result parsed = Solver.evaluate(split[i]+ " + 0");
        paramFloats[i] = parsed.answer.toFloat();
    }
    return paramFloats;
}
}
}

```

TurtleState.pde:

```
/*The TurtleState class was written because the polygon
drawing in Processing doesn't allow using translate(), rotate(),
etc. within "beginShape". All coordinate plane transformations
had to be kept track of manually instead.
This class replaces the use of pushMatrix, popMatrix
for branching purposes.*/
```

```
class TurtleState {

    float x;
    float y;
    float heading;

    TurtleState(float xin, float yin, float headin) {
        x = xin;
        y = yin;
        heading = headin;
    }

    float getX() {
        return x;
    }

    float getY() {
        return y;
    }

    float getHeading() {
        return heading;
    }

    String toString() {
        return x + " " + y + " " + heading;
    }
}
```

Thank you for reading!

