IN THE GARDEN OF VIDA: A SIMPLE, SPATIALLY EXPLICIT TREE-GROWTH MODEL ABLE TO SIMULATE INDIVIDUAL AND COMMUNITY DYNAMICS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Sean Hammond

January 2011

IN THE GARDEN OF VIDA: A SIMPLE, SPATIALLY EXPLICIT TREE-GROWTH MODEL ABLE TO SIMULATE INDIVIDUAL AND COMMUNITY DYNAMICS

Sean Hammond, Ph. D.

Cornell University 2011

To test whether population and community attributes, such as self-thinning and competitive displacement of one species by another, are the direct consequence of the physical characteristics of individual plants or the emergent properties of populations and communities, a simple computer model, called Vida, was created and tested. Vida is a spatially explicit, individual-based model that allows one to simulate the growth of individual trees, populations of trees, or mixed communities composed of different species. Despite using geometrically simple individuals to compete for light and space, Vida is able to accurately simulate the growth of populations of real species, and it is able to reproduce a number of community dynamics and relationships observed by field ecologists. Vida, therefore, provides a powerful heuristic tool with which to explore a large number of ecological principles and theories.

BIOGRAPHICAL SKETCH

After reading Theodore Sturgeon's *Microscopic God* in 1987, Sean T. Hammond turned his attentions away from computers and toward molecular biology. Nearly 10 years later, he graduated from the Rochester Institute of Technology with a Bachelor's degree in Biotechnology. In 1999 he received a Master's degree in Applied Molecular Biology from the University of Maryland, Baltimore County for doing work on expressing a cystine rich *Dictyostelium discoideum* gene product in a soluble form by targeting the expressed protein to the periplasm of *E. coli*. He entered the Plant Cell and Molecular Biology program at Cornell University in 2002 and received a second Master's degree in 2005 for investigating the movement of transproteins from transgenic *Vitis vinifera* tissues grafted onto unmodified vines. After leaving the transprotein project in 2007, Sean joined the lab of Karl J. Niklas to continue his pursuit of a PhD. The basis of his dissertation—an investigation into the implications of a novel spatially explicit, individual-based tree model—is described here.

Earth and Earth and Earth and Earth and. . .

ACKNOWLEDGMENTS

The Author gratefully acknowledges his family—who encouraged him to keep going; Dr. William Provine—who provided excellent council and hit upon a wonderful solution; Dr. Karl Niklas—who took a big risk and who was the perfect advisor for him; Dr. Thomas Owens and Dr. Richard Rand—for offering advice and guidance; Giles Hall and Steven Crim—for providing assistance in spite of his pig headedness, and to the Imaginary Bridges Group for providing financial, material and emotional support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D	Two dimensional
3D	Three dimensional
AutoCAD	Computer Assisted Design program originally released by the company Autodesk
CI	Confidence interval
CFDG	Context Free Design Grammar
cm	Centimeter
CPU	Central Processing Unit
CSV	Comma Separated Values
CUDA	Compute Unified Device Architecture
DBH	Diameter at Breast Height. Typically between 1.3 and 1.5 m above the ground
DXF	Drawing Exchange Format
GPU	Graphics Processing Unit
kd-tree	k-dimensional space
kg	Kilogram
NN	Nearest Neighbor
OS X	A computer operating system sold by Apple Computer
PNG	Portable Network Graphics
SIU	*Systeme International d'Unités*, International System of Units
SMA	Standardized Major Axis
WEB	West, Enquist, Brown
yr	Year

LIST OF SYMBOLS

a	Major axis of an ellipsoid
A	Plant age
b	Normalization constant; minor axis of an ellipsoid
c	Coefficient of cephalization
e	Encephalon (brain); eccentricity of an ellipsoid
g	Gram
g	Acceleration of gravity on Earth
i	Iteration
j	Iteration after i
k	Allometric coefficient
m	Meter
m	Growth memory length
n	Number
r	Coefficient of relation
s	Second
s	Soma (body)
t	Time
v	Velocity
x	Dimensionality of an entire organism
y	Dimensionality of a given organ of an organism; dependant variable
z	Vertical axis
B	Metabolic rate
E	Light energy, Young's Modulus
N	Total number
A_L	Surface area of a canopy
$A_{Loblate}$	Surface area of an oblate spheroid
$A_{Lprojected}$	Projected area of a canopy
$A_{Lprolate}$	Surface area of a prolate spheroid
D_P	Diameter of a spherical propagule
D_S	Diameter of a tree trunk (stem)
E_T	Light energy transmitted through the canopy
G_{Hmax}	Maximum growth in height
$G_{H\mu}$	Average growth in height
G_L	Growth of the canopy
G_S	Growth of the stem
G_T	Total growth (canopy and woody parts)
H_L	Thickness of a uniformly thick photosynthetic tree canopy
H_S	Height of the woody parts (stem) of a tree.
$H_{Scritical}$	Height of the woody part of a tree (trunk) at which it buckles
$H_{Smature}$	Height of mature tree trunks
H_{Syoung}	Height of young tree trunks

M_L	Mass of the canopy
$M_{Lmature}$	Mass of mature tree canopies
M_{Lyoung}	Mass of immature tree canopies
M_P	Mass of a propagule
M_{Pideal}	Idealized mass of a single propagule
M_{PT}	Mass of all propagules on a single tree
M_S	Mass of the woody and/or herbaceous, above ground portions of a plant
$M_{Scritical}$	Mass of tree at which it buckles
M_T	Total mass of an organism
N_P	Number of propagules on a single tree
R^*	Resource level at which growth rate biomass loss
R_L	Radius of a canopy spread
T_C	Fraction of light passing through a canopy
T_S	Fraction of light needed for survival
$V_{Loblate}$	Volume of an oblate spheroid
$V_{Lprolate}$	Volume of a prolate spheroid
Y_1	Interdependent variable1
Y_2	Interdependent variable2
α	Allometric coefficient
β	Normalization constant
ϕ	Fraction
θ	Angle at which a propagule leaves the parent plant
π	Pi
ρ_L	Density of a uniformly think, hemispherical canopy
ρ_P	Density of a propagule
ρ_S	Density of the woody parts of a tree

Chapter One

Beginning in the late 1800s, biologists strove to quantify degrees of differences within and between organisms. By 1897, researchers began to use a power law formula taking the form of $y=bx^{\alpha}$. *The broad utility of the power law was illustrated by Huxley, and would eventually be generally referred to as the allometric formula. Despite criticisms questioning the exclusive use of power formulae to describe scaling relationships, and whether these relationships were of biological significance, the use of power formulae has continued. Research into the underlying causes for ubiquitous scaling relationships led to the West, Enquist and Brown theory, which attempts to explain the origin of scaling relationships. In an effort to ascertain whether population-level scaling relationships, such as self-thinning, are the direct result of how individual plants grow, the Vida software model is introduced*

Eugenics and the desire to quantify

In the latter part of the 19th century, Francis Galton faced a problem. Despite his greatest efforts, he had not been able to quantitatively associate the physical characteristics of humans with criminality, subservience, intelligence, or a host of other social traits he and his fellow eugenicists were interested in. Galton wasn't the first to attempt to use the language of mathematics to describe the natural world; the entire course of Western science's mechanistic world view has used the language of mathematics to describe the universe.

> *Philosophy is written in this grand book - I mean universe - which stands continuously open to our gaze, but which cannot be understood unless one first learns to comprehend the language in which it is written. It is written in the*

> *language of mathematics, and its characters are triangles, circles and other geometric figures, without which it is humanly impossible to understand a single word of it; without these, one is wandering about in a dark labyrinth.* (Galileo, 1623)

The shape of life, however, rarely makes use of simple geometric figures, and as such, can be difficult to describe. For 20 years after his 1869 publication of *Hereditary Genius*, Francis Galton worked to bolster his arguments in favor of the selective breeding of humans. To do so, he strove to scientifically describe the "villainous irregularities" observed in parts of the population, which could serve as telltale markers of individual fitness in a civilized world. Despite his best attempts, in 1889, he expressed frustration that

> *It is strange that we should not have acquired more power of describing form and personal features than we actually possess. For my own part I have frequently chafed under the sense of inability to verbally explain hereditary resemblances and types of features, and describe irregular outlines of many different kinds....*(Galton, 1889)

After decades of work and an increasing library of human mensuration, Galton and his colleagues failed to produce a definitive means of measuring criminality or nobility. Despite this, the observations and metrics recorded by the eugenicists of the 19th century— specifically the investigations of the anthropological criminologists—were the first to approach a mathematical means of describing changes in form.

A Rose by any other name

In the latter part of the 19th century, groups of anthropologists began shifting their

work away from facial characteristics (a practice that smacked of phrenology) and instead focused on "cerebral biometry". By 1897, still searching for the key to unlocking the power of eugenics, the Dutch researcher Eugene Dubois worked out a quantitative means determining how 'evolved' a given organism was by comparing the mass of the brain ("*e*", for "*encephalon*") with the mass of the body("*s*", for "*soma*") using the formula:

$$e=cs^r \qquad \text{(Equation 1.1)}$$

where *c* is the "coefficient of cephalization" and *r* is the "coefficient of relation."(Dubois, 1897, p. 368) A year later, the French researcher Lapicque applied Dubois' formula to the study of the brains of various varieties of dogs and, in 1907, published a log-log graph of his—and Dubois's—work (Figure 1.1). (Lapicque, 1907)

This power law equation would come to be generally known as an "allometric equation," and would be most closely associated with Julian Huxley's work in the first part of the 20th century. The most famous example of Huxley's 'heterogenic development'(what would eventually come to be called allometry) is the relationship he observed in male *Uca pugnax* (the fiddler crab)(Huxley, 1924). Huxley illustrated how the mass of the male's large chela in relation to the total body mass could be closely approximated by the formula[1]

$$y=bx^{\alpha} \qquad \text{(Equation 1.2)}$$

where the relative growth of an organ (*y*) is compared to some metric of the entire

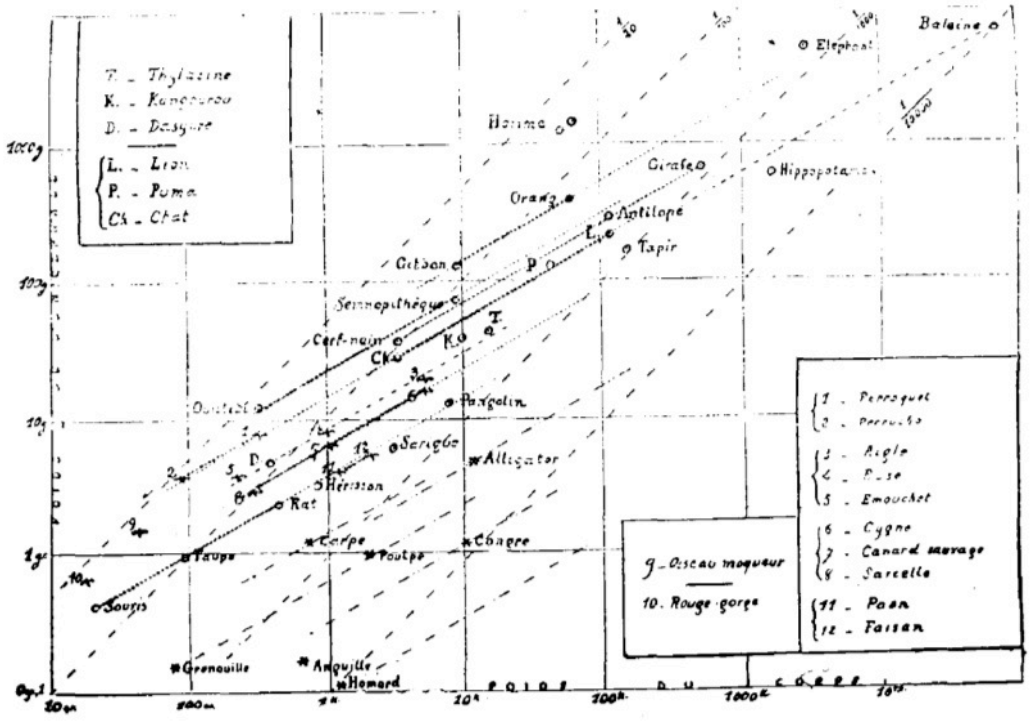
[1] Huxley's initial notation, published in 1924, took the form

$$y=bx^k \qquad \text{(Equation 1.3)}$$

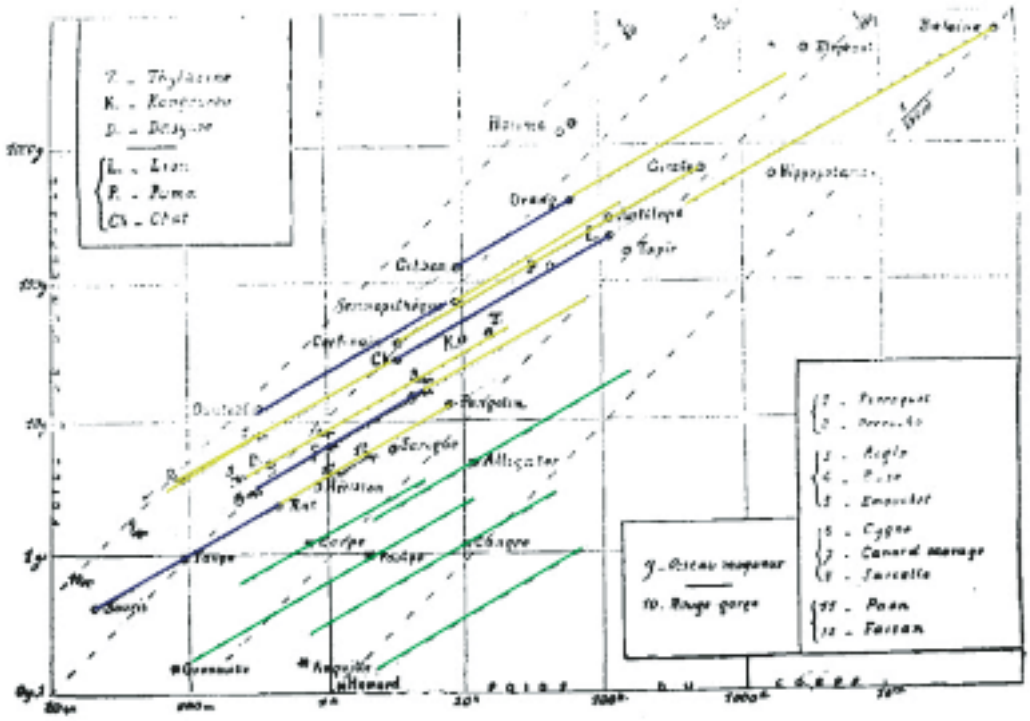
but took on the current form in 1936 as Huxley and Teissier worked to standardize the terminology used when discussing relative growth. (Huxley et al., 1936a, 1936b)

Figure 1.1: Figure reproduced from Louis Lapicque's 1907 work. Bold lines show the reproduction of Dubois' 1897 data. Dashed lines with a slope of 1.0 show hypothetical cases where the body/brain ratio would be proportional. Dashed lines in the lower portion of the figure represent hypothetical lines for frogs, birds, and other non-mammals, making the assumptions that the exponent would be the same as measured in mammals (calculated as being 0.56 by Lapicque). B.) Dubois' 1897 work has been overlain with blue lines, Lapicque's research with non-primate mammals shown in yellow, and Lapicque's hypothetical lines for non-mammals shown in green. The slope of all the highlighted lines (α) approximates 0.56, where as the normalization constant (ß) differs from species to species.

A


T. - Thylacine
K. - Kangourou
D. - Dasyure
L. - Lion
P. - Puma
Ch. - Chat
Baleine
Elephant
Hyène
Girafe
Hippopotame
Orang
Antilope
Tapir
Gibon
Sémnopithèque
Cerf-noir
Ouistiti
Porcelin
Sarigue
Alligator
Hérisson
Rat
Taupe
Serpe
Poulpe
Chèvre
Souris
Grenouille
Anguille
Homard
9 - Oiseau moqueur
10. Rouge-gorge
1 - Perroquet
2 - Perruche
3 - Aigle
4 - Oie
5 - Emouchet
6 - Cygne
7 - Canard sauvage
8 - Sarcelle
11 - Paon
12 - Faisan
POIDS DU CORPS

B


T. - Thylacine
K. - Kangourou
D. - Dasyure
L. - Lion
P. - Puma
Ch. - Chat
Baleine
Elephant
Hyène
Girafe
Hippopotame
Orang
Antilope
Tapir
Gibon
Sémnopithèque
Cerf-noir
Ouistiti
Porcelin
Sarigue
Alligator
Hérisson
Rat
Taupe
Serpe
Poulpe
Chèvre
Souris
Grenouille
Anguille
Homard
9 - Oiseau moqueur
10. Rouge-gorge
1 - Perroquet
2 - Perruche
3 - Aigle
4 - Oie
5 - Emouchet
6 - Cygne
7 - Canard sauvage
8 - Sarcelle
11 - Paon
12 - Faisan
POIDS DU CORPS

organism (x). The normalization constant, b, may or may not differ among individuals, species or taxa, and α is the allometric coefficient, which, often corresponds to the slope of the line on a log-log plot. To use Dubois' work as an example, the brain size of various primates (e in his nomenclature, y in Huxley's), is compared to the total body mass of the organism (s in the 1897 work, x in Huxley's).

Though Huxley quotes later work by Dubois, Lapicque in his *Problems of Relative Growth*,(p 17), he fails to cite the earlier 1897 brain studies by Dubois in the development of his power law. Reviewers have questioned his academic honesty for not giving explicit credit to others,(Gayon, 2000) but Huxley himself rightly states

> *"...others have pointed out that certain organs increase in relative size with the absolute size of the body which bears them; but so far as I am aware, I was the first to demonstrate the simple and significant relation between the magnitudes of the two variables."(Huxley, 1932, p 4)*

Regardless of which researcher first hit upon using the power law to describe physical relationships among various organs, Huxley's inspiration was to abandon attempts to describe forms through the use of ratios of size. Instead, Huxley showed the power of describing things in terms of ratios of growth rates, and understood the implications and usefulness of such techniques. As more researchers began using the concept, terminology and mathematical notation varied widely. Huxley was able to help standardize the terminology and notation with the help of his French contemporary, Teissier. Together, they published papers in English and French, establishing the use of the terms 'isometry' and 'allometry,'[2] thus unifying the English

[2] The term 'allometry'was first used by Osborn in his 1924 paper *The Origin of Species* (Osborn, 1924) to discuss his 1914 concept of 'allometrons'. Huxley and Teissier cite Osborn in their joint paper (Huxley et al., 1936) saying:

and French literature on the topic. (Huxley et al., 1936a; Huxley et al., 1936b)

Scaling Relationships versus Allometric Relationships

Before the 1936 joint paper published by Huxley and Teissier, there were two general groups of literature discussing the same material, but with different vocabularies and mathematical notations: the French literature, citing the Dubois and Lapicque brain/body work and using the terms 'harmony' and 'dysharmony,' and the English literature, citing Pezard's work using the terms 'isogomy' and 'heterogomy.' (Table 1.1) To help unify the two separate sources of literature, Huxley and Teissier published their joint paper in both

---

> *Allometry has the advantage of recalling the allometrons of Osborn, those gradual changes in proportion observed in evolution, which according to the work of Hersch and Robb do proceed according to our fundamental law of allometric growth.*

Osborn's concept of allometry is so similar to what Huxley and Teissier would advocate (and yet differs enough), that Osborn is extensively paraphrased here:

1) Allometrons are general changes of proportion, not localized.
2) Allometrons are allomorphs or heteromorphs, or new changes of proportion, quantitatively new characters
3) Allometrons may be totally independent of ancestral affinity
4) Allometrons apparently may have adaptive significance throughout (e.e., changes of limb proportion)
5) The origin of allometrons in animals of the same ancestry may be either convergent or divergent
6) Allometrons give rise only to analogies, never "homologies" in any sense
7) Allometrons may be produced experimentally in ontogeny
8) Allometrons or allomorphs are universal generic and specific phenomena
9) Allometrons constantly afford indicies and ratios (Osborn, 1924)

Table 1.1: A condensed table showing the evolution of the idea of allometry, and the emergence of the terms ‘isometry’ and ‘allometry’

Year	Author	Term for similar growth	Term for differential growth
1897	Dubois	NA	NA
1907	Lapicque	NA	NA
1914	Pezard	*isogony*	*heterogony*
1924	Huxley	*isogony*	*heterogony*
1924	Champy	*harmony*	*dysharmony*
1926	Teissier	*harmony*	*dysharmony*
1936	Huxley/Teissier	*isometry*	*allometry*

English and French[3] which standardized the vocabulary, as well as the mathematical notation previously mentioned (Equation 1.2). Thus, the two literatures had a common language.

Broadly stated, allometry focuses on changes in the relative dimensions of the parts of an organism as correlated with changes in the overall dimension of an organism. One example is the relationship between the mass of a tree's canopy with the mass of the tree's trunk and branches.

$$M_L \propto M_S^{3/4} \qquad \text{(Equation 1.4)}$$

For the next three decades, the use of the term 'allometry' came to dominate the literature, supplanting the use of the terms 'heterogomy' and 'dysharmony' as more and more researchers began to use the technique of comparing relative growth (Figure 1.2). In his 1966 paper on allometry, Gould identified several different categories of allometric study:

1) Ontogenetic allometry, which refers to the relative growth of individuals over the course of their lifetimes.
2) Phylogenetic allometry, which refers to constant differential growth in lineages; allometry among members of a single population at the same growth stage, but of different sizes.

[3] The two papers differ by a single sentence. Huxley and Teissier disagreed as to the significance of the '*b*' value in the power law equation. Huxley was of the opinion that differences in '*b*' were of no importance, whereas Teissier felt it had biological significance. For this reason, Teissier inserted the following sentence into the French version of their paper: "From a statistical point of view [*b*] represents the mean value of the ratio y/x for all the observed individuals."(Huxley et al., 1936b)

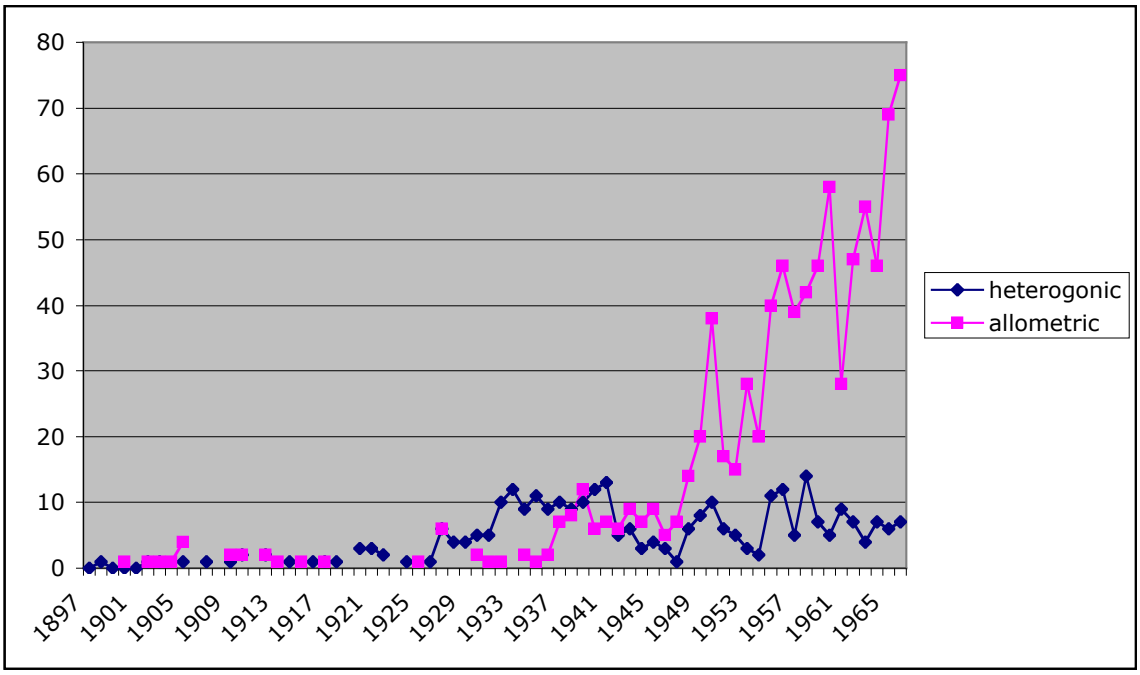


Figure 1.2: Rough estimate of the number of papers and books published using competing terms, generated by data mining scholar.google.com from 1897 to 1966. Search terms were (heterogonic OR heterogony), (allometric OR allometry). The terms *disharmonic*, *disharmony*, *disharmonic*, and *disharmony* were excluded due to multiple (and historically changing) meanings.

3) Intraspecific allometry, which refers to adult individuals within a species or a given local population; allometry among species of a single variety or subspecies of a single species at the same growth stage, but of different sizes.
4) Interspecific allometry, which refers to the same kind of phenomenon among related species; allometry among species of a single genus at the same growth stage, but of different sizes.(Gould, 1966)

By the 1970's, a new sort of allometric-like ecological relationship had been identified, and was being studied. These relationships fit the power law equation used in classical allometry, yet to label ecological relationships "allometric" stretched the usage of the word (though not its Greek definition[4]). For example, relationships between the total mass (M_T) of individuals on a given plot were found to be related to the absolute number (N) of organisms on that same plot.

$$M_T \propto N^{-4/3} \qquad \text{(Equation 1.5)}$$

One could argue that N, in this case, is the overall dimension of a superorganism (a forest), and the mass of the forest was the dimension of an organ of the superorganism. Doing so weakens the conceptualizations of Huxley and Teissier in 1936. By a similar token, relationships between two organs of an individual organism—relationships between the height of a tree and its trunk diameter, for example—could be called allometric relationships.

Rather than sacrifice the word *allometry*, one can instead use the phrase *scaling relationship*. In a strictly mathematical sense, a scaling relationship is any relationship between two variables of interest (Y_2 and Y_1) that can be mathematically described by a

[4] To, again, quote Osborn:
*[*Allometry*] is a convenient abbreviated derivative of the Greek* αλλοιοσ, different*, and* μετρον, that by which anything is measured.(Osborn, 1924)

slope (scaling exponent, α) that quantifies the change in one variable of interest (Y_2) with respect to the change in another variable of interest (Y_1) provided that the slope ($\alpha = Y_2/ Y_1$) is statistically (and biologically) meaningful. By this definition, most quantifiable biological and ecological processes and patterns involving two or more variables of interest can be expressed as one or more scaling relationships. Thus, trends and patterns such as numbers of individual species per area ("species-area" relationships), numbers of individuals per area ("species abundance" curves), individual plant size versus number of conspecifics of equivalent size ("size frequency" distributions) as well as a host of other allometric phenomena are scaling relationships. To be explicit, all allometric relationships are scaling relationships, but not all scaling relationships are allometric.

Botanical Allometry

Thanks in large part to its anthropological and zoological origins, the study of scaling relationships was mainly applied to animals during the early part of the 20th century. A researcher in the 1930s commented that, "Botanists have been slower applying the Principle of Similarity to the study of plants,"(Bower, 1930), a criticism one could argue could be generalized. It soon became apparent that scaling relationships were of great utilitarian worth to foresters, and large tables were compiled that would help a given logger estimate the total mass of a given tree based of the diameter of the trunk, or even the total number of trees present in a given area given a sampling of several tree's trunk diameters.

At the same time, botanists began to notice trends in tree mensuration that their zoologist colleagues had noticed in animals. The assumption among zoologists dating back to Dubois was that the slopes of lines would be multiples of 1/3 due to Euclidean scaling. For example, Dubois assumed the slope of the lines in his brain/body studies

would be 0.6; instead, most biological scaling relationships exhibit 'quarter power' values ($^1/_4$, $^3/_8$, $^3/_4$, etc.). For example, the metabolic rate (B) of organisms shows the relationship

$$B \propto M_T^{3/4} \qquad \text{(Equation 1.6)}$$

versus that which would be expected if organisms showed Euclidean scaling:

$$B \propto M_T^{2/3} \qquad \text{(Equation 1.7)}$$

In addition to the scaling relationships seen among individual organisms or between various species, scaling relationships among diverse populations (such as the relationship between the total mass of individual trees and the population density of a given forest) also exhibited quarter power values.

One obvious difficulty facing researchers was that the quarter power rules were strictly empirical, and there was no underlying theory as to why and how such relationships exist. This was not a new problem faced by individuals studying scaling relationships; as early as 1932, an otherwise positive reviewer observed that Huxley's allometric formula was:

> *…necessarily empirical. Of the causes of differential growth we have little knowledge; their investigation is the problem at issue. A variety of possible relations, in fact, reduce approximately to this formula. But it is not the object of the formula to establish the correctness of a particular hypothesis as the cause of differential growth; it merely expresses the observed facts with considerable accuracy in a simple way, so that many very significant features emerge which would not otherwise do so.*(Pantin, 1932)

Later, D'Arcy Thompson, to whom Huxley dedicated his 1932 book, dismissed Huxley's claims that the power laws were biologically significant, saying:

> *...the formula is mathematical rather than biological; there is a lack of either biological or physical significance in a growth-rate which happens to stand, during part of an animal's life, at 62 per cent. compound interest.*
>
> *Julian Huxley holds, and many hold with him, that the exponential or logarithmic formula, or compound-interest law, is of general application to cases of differential growth-rates. I do not find it to be so: any more than we have found organ, organism or population to increase by compound interest or geometrical progression, save under exceptional circumstances and in transient phase. Undoubtedly many of Huxley's instances shew increase by compound interest, during a phase of rapid and unstinted growth; but I find many others following a simple-interest rather than compound-interest law.* (Thompson, 1942)

In general, critics of the study of allometry, and scaling relationships in general, have cited two concerns. The first, is whether log transformed data is the best means to analyze data. Numerous researchers, starting with Thompson, have urged for a broader approach to the study of relative size (Thompson, 1942; Sholl, 1950; Yates, 1950; Zuckerman, 1950; Gould, 1966; Smith, 1980; Harvey, 1982; Chappell, 1989) and have argued that power formulae do not always provide the best fits of the data (for review, see Smith, 1980 and Niklas, 2004).

The second criticism, explicitly mentioned by both Pantin and Thompson, is whether the relationship has biological significance. Without an underlying *a priori* explanation as to how empirically measured scaling relationships arise, their study is akin to the work done by the criminal anthropologists in the late 1800s.

The very fact that quarter power values continue to emerge from data implies that there must be some underlying mechanism at work, even if the mechanism itself is not

always immediately apparent (Niklas, 1994; Farnsworth,1995). That similar trends appear in vastly different species, inhabiting diverse ecological systems, further implies that the explanation lies in something fundamental to life on the planet.

A Theory of Everything

One of the most widely cited theories to explain the origin of many scaling relationships is that of West, Enquist and Brown (West et al., 1997), denoted here as the WEB theory. This theory focuses on the individual organism and posits the existence of an internal, fractal-like transport and delivery system that has evolved to optimize the time and energy required to distribute materials throughout the organism (West et al., 1997). The WEB theory has made numerous predictions, spanning animal and plant communities, and has been elaborated in numerous ways to provide metabolic optimization scenarios for plant communities (e.g., Enquist et al., 1998, 2007, 2009; Ernest et al., 2003). Despite its many successes, this theory and all of its more recent variants are based on a number of biological and physical assumptions that have been criticized on theoretical as well as empirical grounds (Dodds et al., 2001; Kozlowski and Konarzewski, 2004; Makarieva et al., 2008). To the best of our knowledge, no one has unequivocally shown that simple biophysical principles, acting at the level of an individual plant, scale up *a priori* to establish the properties typically observed for real plant populations or communities. Nor has anyone shown that the properties of real plant populations or communities are the direct result of the size-dependent characteristics of their representative individual plants or species.

The absence of any such demonstration cannot be taken as proof that the WEB theory is fatally flawed. Attempts to demonstrate that the attributes of the individual ultimately lead to community-level relationships, even for very simple ecological systems, are confounded by the tremendous spatiotemporal heterogeneity that typically

characterizes real environments and by the often dramatic physiological, morphological, and reproductive differences that set one species apart from another. The fact that the WEB theory has successfully predicted aspects of community dynamics argues that it cannot be easily dismissed, and that hierarchical population and community interactions may be the direct consequence of the properties of the individual.

In this context, a variety of empirical-field and theoretical-modeling approaches have been used to resolve whether "canonical" scaling relationships do in fact exist (West et al., 1997; Ernest et al., 2003; Enquist et al., 2007) in the face of well documented species-specific variation in response to different environmental conditions (Bolker et al., 2003; Diez and Pulliam, 2007; Lichstein et al., 2007). The empirical-field approach uses detailed measurements taken over ecological time-frames to evaluate how species grow, interact, and respond to abiotic and biotic factors operating at multiple spatial and temporal scales. The theoretical-modeling approach uses mathematical and computational tools to predict relationships when abiotic and biotic factors are too complex to dissect experimentally, or when relationships operate in time-frames that preclude direct experimentation. Each of these approaches has its strengths and drawbacks. Empirical-field studies are the most direct, but they can require impractically large expenditures of time and effort. Theoretical approaches can provide detailed predictions about complex phenomena that may comply reassuringly well with observations, although sometimes for the wrong reasons (for example, Ptolemy of Alexandria's *Almagest* accurately predicted the movements of the stars and the known planets, despite being a geocentric model). A balanced juxtaposition of both the empirical and theoretical approaches is advisable if long-standing debates are to be resolved about the height-structured competition for light, the relative roles of niche-based versus stochastic processes affecting species distributions, community composition and stability, and other ecologically and evolutionarily important aspects

of plant ensemble dynamics (Tilman, 1982; Purves and Pacala, 2008).

Vida

In an attempt to test whether population or community attributes, such as self-thinning and competitive displacement of one species by another, are the direct consequence of the characteristics of their individual plant components, *in silico* environments were generated using a custom built piece of software called Vida, which was designed to evaluate how different species compete for light and space in a world-space whose physical attributes, such as the direction and intensity of incident sunlight, are clearly defined. Because each variable required to parameterize a species can be varied and manipulated individually, Vida provides a venue for running controlled experiments (individual computer simulations) to assess the influence of each variable on the dynamical behavior of a population or community. Vida therefore allows us to explore the consequences of competition for light and space as a population or community grows *in silico*, under rigidly specified conditions in an environment that can be made homogeneous or heterogeneous depending on the type of computer simulations required.

In the following chapters, simulations conducted with Vida attempt to:

1) Demonstrate that the biological fidelity of Vida is sufficient to accurately reproduce the growth of an individual, real-world tree.
2) Show how populations of simulated trees interact and determine whether simulations are able to reproduce scaling relationships seen in real-world populations, and
3) Determine the effects of spatially and temporally heterogeneous simulation worlds have on population dynamics of multiple simulated species.

REFERENCES

BOLKER, B. M., S. W.PACALA AND C. NEUHAUSER. 2003. Spatial dynamics in model plant communities: What do we really know*? American Naturalist* 162:135 – 148.

BOWER, F. O. 1930 *Size and form in plants*. Macmillan, London.

CHAMPY, C. 1924. *Sexualité et Hormones*. G. Doin, Paris.

CHAPPELL, R. 1989. Fitting bent lines to data, with applications to allometry. *Journal of Theoretical Biology*. 138: 235-256.

DIEZ, J. M. AND H. R. PULLIAM. 2007. Hierarchical analysis of species distributions and abundance across environmental gradients. *Ecology* 88: 3144 – 3152.

DODDS, P. S., D. H. ROTHMAN, AND J. S. WEITZ. 2001. Re-examination of the "3/4-law" of metabolism. *Journal of Theoretical Biology* 209: 9 – 27.

DUBOIS, E. 1897. Sur le rapport de l' encépha;e avec la grandeur du corps chez les Mammiferes. *Bulletins de la Société d'anthropologie de Paris*. 5: 337-374.

ENQUIST, B. J., J. H. BROWN AND G. B. WEST. 1998. Allometric scaling of plant energetics and population density. *Nature* 395: 163 – 165.

ENQUIST, B. J., A. J. KERKHOFF, S. C. STARK, N. G. SWENSON, M. C. MCCARTHY, AND C. A. PRICE. 2007. A general integrative model for scaling plant growth, carbon flux, and functional trait spectra. *Nature* 449: 218 – 222.

ENQUIST, B. J., G. B. WEST AND J. H. BROWN. 2009. Extensions and evaluations of a general quantitative theory of forest structure and dynamics. *Proceedings of the National Academy of Science* 109: 7046 – 7051.

ERNEST, S. K. M., et al. 2003. Thermodynamic and metabolic effects on the scaling of production and population energy use. *Ecology Letters* 6: 990 – 995.

FARNSWORTH, K.D AND K. J. NIKLAS. 1995. Optimization, from and function in Branching Architecture in Plants. Functional Ecology. 9: 355-363.

GALILEO, G. 1623. *Il Saggiatore.*

GALTON, F. 1889. Personal Identification and Description. *The Journal of the Anthropological Institute of Great Britain and Ireland.* 18: 177-191.

GAYON, J. 2000. History of the Concept of Allometry. *American Zoologist.* 40: 748-758.

GOULD, S. J. 1966. Allometry and size in ontogeny and phylogeny. Biological Reviews. 41: 587-638.

HARVEY, P. H. 1989. On rethinking allometry. Journal of Theoretical Biology. 95: 37-41.

HUXLEY, J. S. 1924. Constant differential growth-ratios and their significance. *Nature*. 114: 895-896.

HUXLEY, J. S. 1932. *Problems of relative growth.* The Dial Press, New York.

HUXLEY, J. S. AND G. TEISSIER. 1936. Terminology of relative growth. *Nature*. 137: 780-781.

HUXLEY, J. S. AND G. TEISSIER. 1936. Terminologie et notation dans la description de la croissance relative. *Comptes rendus des séances de la Société de de biologie et de ses filiales*. 121: 934-937.

KOZLOWSKI, J. AND M. KONARZEWSKI. 2004. Is West, Brown and Enquist's model of allometric scaling mathematically correct and biologically relevant? *Functional Ecology* 18:283 – 289.

LAPICQUE, L. 1907. Tableau general des poids somatiques et encéphaliques dans les espéces animales. *Bulletins de la Société d'anthropologie de Paris*. 9: 248-269.

LICHSTEIN, J. W., J. DUSHOFF, S. A. LEVIN, AND S. W. PACALA. 2007. Intraspecific variation and species coexistence. *American Naturalist* 170: 807 – 818.

MAKARIEVA, A.M., V. G. GORSHKOV., B. L. LI, S. L CHOWN, P. B. REICH., AND V. M. GAVRILOVE. 2008. Mean mass-specific metabolic rates are strikingly similar across

life's major domains: Evidence for life's metabolic optimum. *Proceedings of the National Academy of Sciences (USA)* 105:16994 – 16999.

NIKLAS, K. J. 1994. Plant Allometry: The Scaling of Form and Process. University of Chicago Press, Chicago.

NIKLAS, K. J. 2004. Plant allometry: is there a grand unifying theory? *Biological Reviews*.79: 871-889.

OSBORN, H. F. 1915. 1915 Osborn Origin of Single Characters as Observed in Fossil and Living Animals and Plants. *The American Naturalist*. 49: 193-239.

OSBORN, H. F. 1924. The Origin of Species. *Proceedings of the National Acadamey of Sciences*. 11: 749-752.

PANTIN, C. F. A. 1932. Form and Size. *Nature*. 129: 775-777.

PEZARD, A. 1918. Le conditionnement physiologique des caractéres sexuels secondaires chez les Oiseaux. *Bulletin biologique de la France et de la Belgique*. 52: 1-176.

PURVES, D. AND S. PACALA. 2008. Predictive models of forest dynamics. *Science* 320: 1452 – 1453.

SHOLL, D. A. 1950. The Theory of Differential Growth Analysis. *Proceedings of the Royal Society of London. Series B, Biological Sciences*. 137: 470-474.

SMITH, R. J. *Rethinking Allometry*. 1980. Journal of Theoretical Biology. 87: 97-111.

THOMPSON, D'A.W. 1942. *On Growth and Form*, 2nd ed. Cambridge University Press, Cambridge.

TILMAN, D. 1982. *Resource competition and community structure*. Princeton University Press, Princeton, New Jersey, USA.

WEST, G. B, J. H. BROWN, AND B. J. ENQUIST. 1997. A General Model for the Origin of Allometric Scaling Laws in Biology. *Science*. 276: 122-126.

YATES. F. 1950. The Place of Statistics in the Study of Growth and Form. *Proceedings of the Royal Society of London. Series B, Biological Sciences*. 137: 479-488.

ZUCKERMAN, S. 1950. The Pattern of Change in Size and Shape. *Proceedings of the Royal Society of London. Series B, Biological Sciences*. 137: 433-443.

Chapter Two

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.

-Mandelbrot, in his introduction to *The Fractal Geometry of Nature*

"Imagine the cow is a sphere…."

-Punchline to a joke about physicists.

The most simple representation of a tree is the child-like image of a circular, photosynthetic surface supported by a perpendicular column. In general, the relationships of the above-ground organs of a tree can be described using the relationships $M_S \propto M_T^{\alpha}$, $M_L \propto M_S^{\alpha}$, $D_S \propto M_S^{\alpha}$, *and* $H_S \propto D_S^{\alpha}$. *After basic mathematical relationships were determined, a game-like system was established in which each tree or propagule was treated as an object in a simulated world-space, where its ability to grow and survive was determined by the ability to harvest light.*

Code Considerations

When conceptualizing how to model real-world events on a computer, an obvious difference between computers and reality is how events are handled. In the world we exist in, the universe can be seen as being massively parallel, with multiple things apparently happening at the same time. While there are various ways one can give the appearance that computers can do multiple things at once, in the absence of multiple processors, computers deal with things in a serial fashion, meaning that multithreading and other techniques for parallelization are just programmatic illusions (Neuburg, 1999, Tanenbaum, 1992). Such threads of execution occur when a program is forked into

more than one task. On a single processor, this multithreading occurs when the processor switches between different threads, incrementing each by a small amount. The amount of time spent incrementing each thread varies from language to language. For example, the Python programming language will, by default, execute 100 bytecode instructions before switching to the next thread (Martelli, 2005).

A fast enough computer can cycle through threads with such rapidity, executing each thread for a short period of time before stopping it and moving to the next, that the user has the illusion of parallelization (Neuburg, 1999). The effect is an illusion; effectively a zoetrope or praxinoscope.

This point may seem trivial and obvious, but it affects how one considers constructing the logic of a computer-based simulation, and has interesting philosophical implications. For example, we perceive the universe as being massively parallel with each and every subatomic particle behaving in ways independent from other subatomic particles (ignoring entanglement) (Figure 2.1). If, however, there were a conscious Actor existing within a serialized simulation, the Actor's perception of his universe would be that the universe was behaving in a massively parallel fashion. The simulation would cycle through each and every object in the simulated universe, making individual step-wise changes to each object's state. The Actor's perception, then, would last only as long as it was his 'turn'. The gaps of time between their perception would never exist from their relative perspective. The time between his 'turns' would be below his Planck time, a sort of persistence of consciousness.

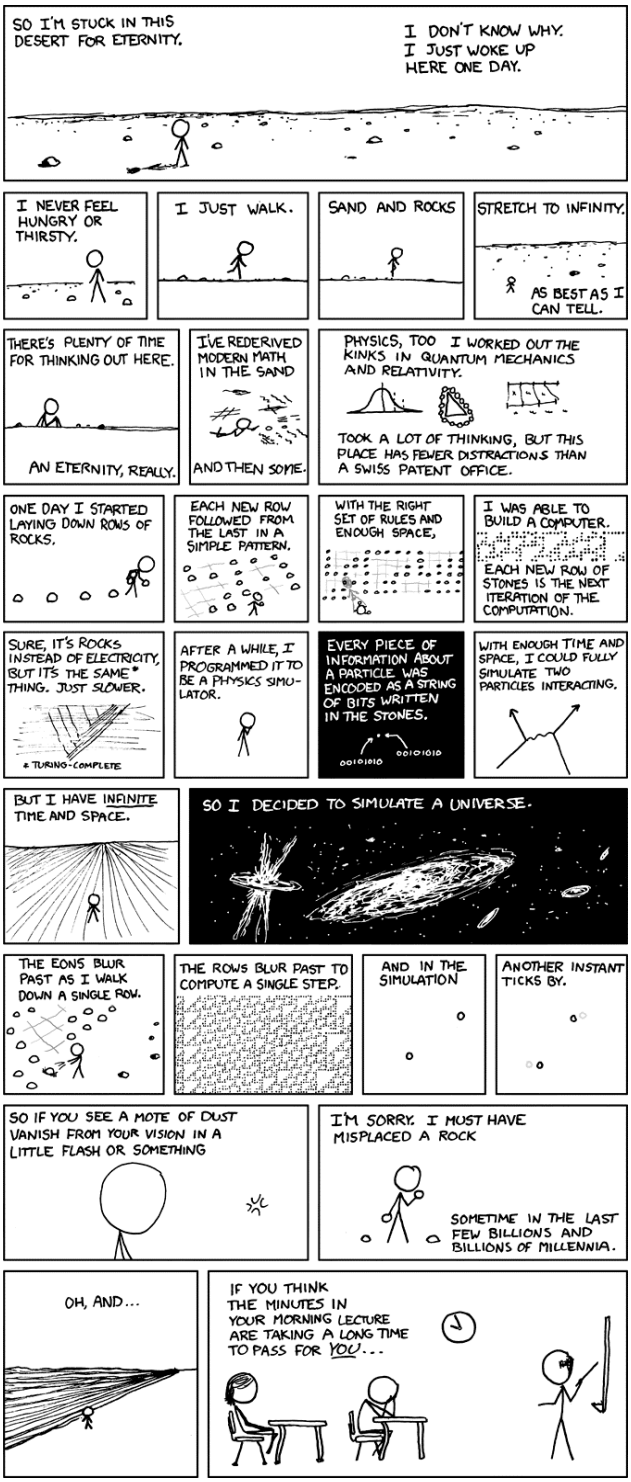


Figure 2.1 "A Bunch of Rocks". Copyright Randal Monroe, XKCD. Used with permission.

In practice, implementing threads on a single processor, or implementing genuine parallel processing using multiple processors is a non-trivial situation. The third edition of "Programming Perl" makes the following analogy :

> *Imagine taking a recipe from a cookbook and converting it into something that several dozen chefs can work on all at the same time. You can take two approaches.*
>
> *One approach is to give each chef a private kitchen, complete with its own supply of raw materials and utensils. For recipes that can be divided into parts easily, and for foods that can be transported from kitchen to kitchen easily, this approach works well because it keeps the chefs out of each other's kitchens.*
>
> *Alternately, you can just put all the chefs into one kitchen, and let them work things out, like who gets to use the mixer when. This can get messy, especially when the meat cleavers start to fly (Wall, 2000).*

The former, individual kitchen model, is meant to represent the use of multiple processors, whereas the latter tries to model the multithreaded method using a single processor. While the number of personal computers that have multiple processors is increasing, the vast majority of older computers use a single processor. For this reason, the multithread model is the most common for now.

Some of the previous attempts at creating software to simulate individual and community interactions made use of SWARM, a multithreaded platform allowing individuals to create individual, interacting agents using code written in objective C (Enquist and Niklas, 2001a, 2001b; Pringle, 2001). Relatively simple in its implementation, the application referred to as PLANT was not parameterized using real world species data.

Rather than build upon this previous work, it was decided that all work should start from scratch, including envisioning how the program would deal with each object in the simulated world. To keep development relatively simple and somewhat easy to troubleshoot, it was decided to ignore parallel processing and multithreaded models. Instead, the program would run as a single thread, where each object in the world would be dealt with in a serial fashion.

Conceptualizing a simulation as running in a series of individual turns is quite easy to do, thanks in large part to the ubiquity of turn-based games in our culture. Envisioning a tree/forest-growth simulation in terms of a turn-based game allows one to sketch out a logical flow of events.

The Game of Life

Breaking down how an individual plant grows, one sees discrete steps:

1) Germination
2) Photosynthesis
3) Allocation of carbon to organs
    3a) Vegetative growth
        3i) Reproduction
        3ii) Seed dispersal

Simulating how an individual plant grows in its lonely platonic world is useful and informative, but one needs interactions between individuals if one wants to examine the results of competition for light and space on growth. When it comes to thinking about how plants in a simple world interact, the options are pretty limited: they kill one another.

Obviously a photosynthetic plant will die when it is deprived of light due to excessive shading. Plants in a simulation must also obey physical laws, e.g., they

cannot occupy the same space, and they cannot exceed their Euler-Greenhill maximum height.

Incorporating mortality into the simple flow chart yields:

1) Germination
    1a) Death due to failure to germinate
2) Photosynthesis
    2a) Death due to lack of light
3) Allocation of carbon to organs
    3a) Vegetative growth
        3i) Death due to Euler-Greenhill buckling.
        3ii) Reproduction
        3iii) Seed dispersal
4) Other causes of death
    4a) Death due to the stem extending off the simulated world space.
    4b) Death due to being crushed
    4c) Death due to occupying the same space as another object.
    4d) Death due to senescence.
    4e) Stochastic death.

Creating realistic mortality algorithms is not a trivial process. A review of mortality algorithms by Hawkes showed just how complicated models can become (Hawkes, 2000). The source of this complication may arise from the fact that it is sometimes difficult to measure exactly why individuals die in natural, uncontrolled habitats. One can measure the number of trees that die within a hectare over a 10 year period, but what sort of bias does the data contain? Is the researcher only measuring the death of

trees that have a DBH (i.e. they are at least as tall as one's breast)? Do they somehow measure seedling mortality?

Even if an enterprising team of graduate students were able to record every tree death in a hectare, their ability of saying exactly why any given tree died would be difficult. If a tree is sickly, growing slowly in the shade, and dies due to a fungal infection, what is the cause of death? Lack of light or the fungus, or something else entirely?

In short, there are many, many ways to die in this world, and only a few ways to succeed.

To use the game analogy, one can imagine a software simulating growing trees as a modified Monopoly game. At the start of each player's (tree's) turn, they 'pass go' and collect a certain amount of money (carbon), based on the surface area of all the houses and hotels they have on the board. The player converts houses to hotels (allocates carbon to organs of the tree) and maybe places new houses on the board (reproduces). After each player has finished his turn, each player then picks up a gun from the Chance pile and plays Russian roulette to see who dies (due to stochastic death). The survivors then begin the next round.

With a simple game analogy in mind, the underlying relationships behind Vida were assembled.

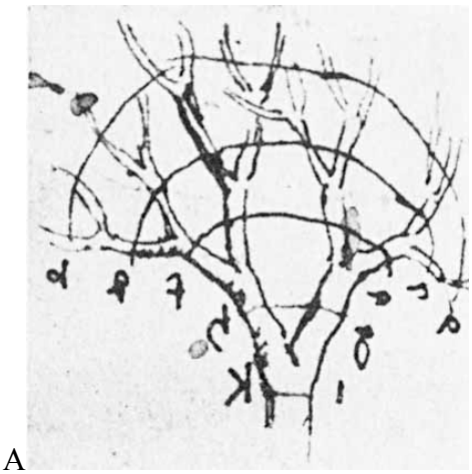
Modeling Very Simple Trees: Allometric Ideals and Spherical Cows

With the eye of an artist and a desire for symmetry, Leonardo da Vinci formulated a means whereby he could generate realistic looking trees and plants in his works. In so doing, his observations are the first recorded instance of an individual using mathematical relationships to describe the form of a tree:

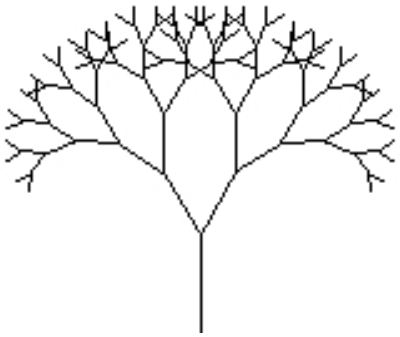
> *All the branches of a tree at every stage of its height when put together are equal in thickness to the trunk [below them]….*
>
> *Every year when the boughs of a plant [or tree] have made an end of maturing their growth, they will have made, when put together, a thickness equal to that of the main stem; and at every stage of its ramification you will find the thickness of the said main stem; as: i k, g h, e f, c d, a b, will always be equal to each other; unless the tree is pollard—if so the rule does not hold good.* (Figure 2.2) (Richter, 1939)

This concept would be revisited and formalized in 1964 by Shinozaki et al., where they called Leonardo's rule-of-thumb the "pipe model" (Shinozaki et al., 1964a, 1964b). In the years that followed, thanks in large part to the work done by Mandelbrot (1982), the pipe model and the concept of iterative branching led to increasingly more realistic-looking computer generated images of trees and other plants (Weber, 1995; Lintermann, 1998; Deussen, 1998).

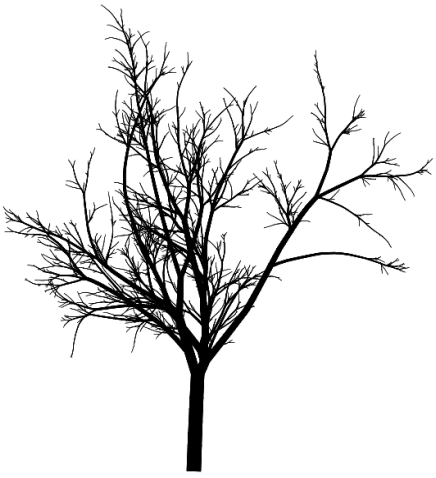
When one steps back from attempting to achieve realistic images of trees, an abstract "spherical cow" tree is exactly what a child might draw: a circular, photosynthetic surface supported by an elongated, perpendicular structural member. While a child's drawing might lack realism, it does capture the essence of a tree in its most basic, Platonic tree-ness. Attempting to split the difference between photorealistic images of trees and overly simple Platonic trees is problematic because robust information about the geometry of a tree canopy geometry is typically lacking in the datasets that also record information related to total tree mass, canopy mass, tree height, and other physical properties.



A



B



C

Figure 2.2: A.) Sketch from Leonardo's sketchbook. Labels (a-b, c-d, e-f, g-h, i-k) are in Leonardo's idiosyncratic backward handwriting. B.) A simple fractal tree. C.) A realistic looking, computer generated tree making use of fractal branching and reductions in stem diameters following the pipe model.

For the sake of simplification, if one assumes a tree's canopy acts as a uniformly thick surface, of which only one side is photosynthetically active, one can use easily obtained physical data and calculate the surface area according to the formula

$$A_L = \frac{M_L / \rho_L}{H_L} \qquad \text{(Equation 2.1)}$$

where A_L is the surface area , M_L is the mass of the canopy, ρ_L is the density of the photosynthetic surface, and H_L is the thickness of the photosynthetic surface. If this area constitutes a flat surface parallel to the surface of the earth, the tree resembles a roofing nail. The canopy spread (R_L) is arrived at using the equation:

$$R_L = \sqrt{\frac{A_L}{\pi}} \qquad \text{(Equation 2.2)}$$

While a wonderfully simplified tree, modeling the photosynthetic surface as a flat disc results in a canopy diameter that is relatively large. If one treats this same area as a hemisphere versus a flat disc, the radius is described as

$$R_{L\,Hemisphere} = \sqrt{\frac{A_L}{2\pi}} \qquad \text{(Equation 2.3)}$$

If one assumes that light energy (E) for photosynthesis comes from directly above each model tree, the area available to a 'roofing nail' type model is far greater than that available to the area available to a 'bumbershoot' type model, where the area available for photosynthesis is the projected area of the hemisphere.

$$A_{L_{Projected}} = \pi\left(R_{L_{Hemisphere}}\right)^2 \qquad \text{(Equation 2.4)}$$

or

$$A_{L_{Projected}} = \frac{M_L}{2\rho_L H_L} \qquad \text{(Equation 2.5)}$$

Put another way, the area available for photosynthesis of a bumbershoot is half that

of a roofing nail.

One could more accurately model the geometry of a tree using prolate or oblate spheroids (or as cones in the case of some conifers) and achieve more realistic canopy spreads, but a hemispherical model is relatively easy to work with mathematically if one is striving for the most simple model.

The mass of the canopy and the structures that elevate it are governed by the following relationships:

$$M_S \propto M_T^{\alpha}$$

$$M_L \propto M_S^{\alpha}$$

$$D_S \propto M_S^{\alpha}$$

$$H_S \propto D_S^{\alpha}$$

In the early part of the 20th century, Huxley observed that such relationships are often linear when plotted on log-log paper, and took the form of $Y = \beta\ X^{\alpha}$ (where beta is a species-specific constant and alpha is the slope of the line) and is similar despite differences in species (Huxley, 1932). While the observation is interesting, an underlying reason as to why these relationships exist has remained elusive (Niklas, 1994, 2004; also see Chapter 1). Huxley's generalization has allowed for the creation of practical databases, however. Foresters and ecologists have made use of various formulae that allow them to estimate a given tree's total above-ground mass, mass of the trunk and stems, mass of the canopy, and height, based on a tree's DBH (Jenkins, 2004; Zianis 2005; Navar, 2009).

$$M_S = \beta_1 D_S^{\alpha_1} \qquad \text{(Equation 2.6)}$$

$$M_L = \beta_2 D_S^{\alpha_2} \qquad \text{(Equation 2.7)}$$

$$H_S = \beta_3 D_S^{\alpha_3} \qquad \text{(Equation 2.8)}$$

The vast majority of studies examining the allometry of trees fall into two

categories: those that attempt to show underlying allometric trends among all species, and those that are interested in the practical application of allometry to help predict standing biomass. In the former case, species encompassing numerous genera are often grouped together, while in the latter case, specific formulae are constructed for each species and rely on the DBH as the dependant variable. Relationships based on the DBH are useful in a practical sense in that a tree trunk's diameter is the easiest physical characteristic that can be measured in the field in a nondestructive fashion. However, reliance upon DBH creates a situation that does not reflect a logical flow of carbon in an organism.

The following formulae are better suited to show the flow of carbon within a tree:

$$M_S = \beta_4 M_T^{\alpha_4} \qquad \text{(Equation 2.9)}$$

$$M_L = \beta_5 M_S^{\alpha_5} \qquad \text{(Equation 2.10)}$$

$$D_S = \beta_6 M_S^{\alpha_6} \qquad \text{(Equation 2.11)}$$

$$H_S = \beta_7 D_S^{\alpha_7} \qquad \text{(Equation 2.12)}$$

While generally correct, Equation 2.12 is observably incorrect. A tree's diameter will continue to increase over the entire course of its life (thus the ability to use tree rings to determine a tree's age), but trees do not continue to grow in height indefinitely. Even the tallest species of trees have limits on their height imposed by their ability to transport water to the highest boughs. Careful observation has shown that the relationship between diameter and height is more accurately portrayed by

$$H_S = \beta_7 \, D_S^{\alpha_7} - \beta_8 \qquad \text{(Equation 2.13a)}$$

and

$$H_S = \beta_9 + \beta_{10} \ln D_S \qquad \text{(Equation 2.13b)}^5$$

[5] Interestingly, the right hand side of the equation for height is essentially the same as Ludwig Boltzmann's formula for entropy: $S = l \ln W$.

where Equation 2.13a models a growth early in a tree's life when the amount of secondary xylem is minimal within the entire tree, and Equation 2.13b models an increasing amount of secondary xylem and dead heartwood. Because very small values of D_S can result in negative values of H_S in equation 2.13b, the transition point from equation 2.13a to 2.13b occurs when equation 2.13b results in a value for H_S that is greater than or equal to the value for H_S arrived at by using equation 2.13a. (Figure 2.3)

Another slightly more complex relationship involves the relationship $M_L \propto M_S$. While the general equation $Y = \beta X^{\alpha}$ is still valid, it appears there are at least two different relationships (as shown in Figure 2.4):

$$M_L = \beta_5 M_S^{\alpha_5} \qquad \text{(Equation 2.14a)}$$

$$M_L = \beta_{11} M_S^{\alpha_8} \qquad \text{(Equation 2.14b)}$$

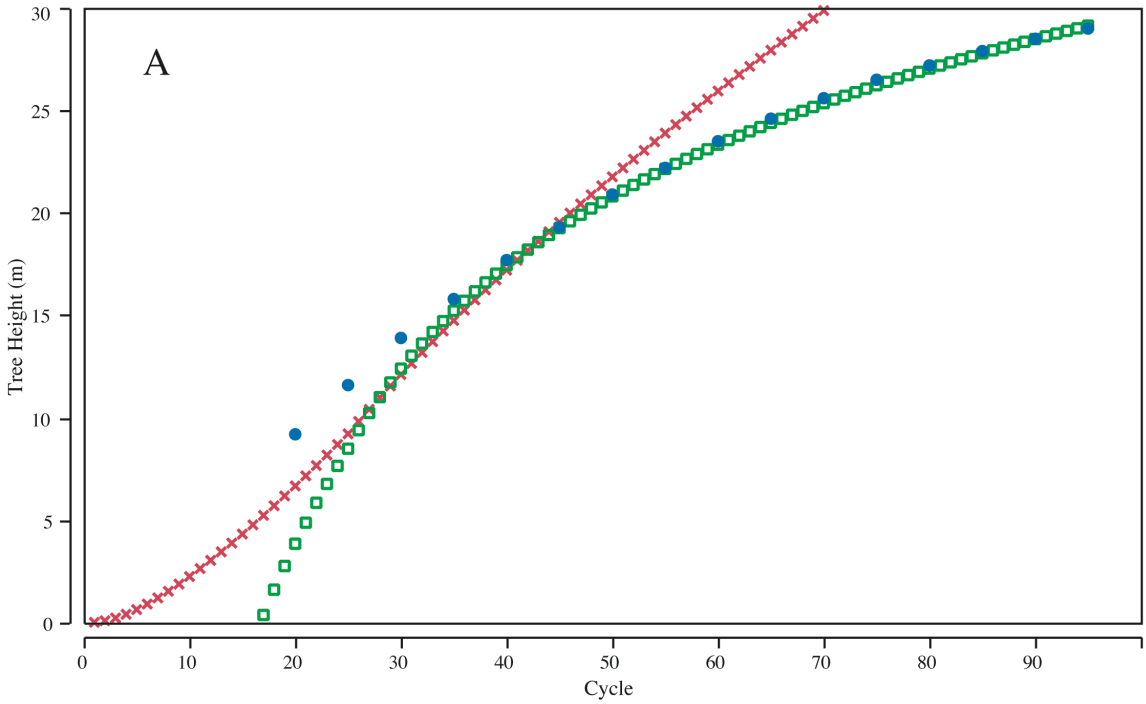
A similar trend was reported by Huxley (1932) for male *Uca pugnax* (fiddler crabs), in his seminal "Problems of Relative Growth." Huxley illustrated how the mass of the male's large chela in relation to the total body mass was closely approximated by the formula $Y = \beta X^{\alpha}$:

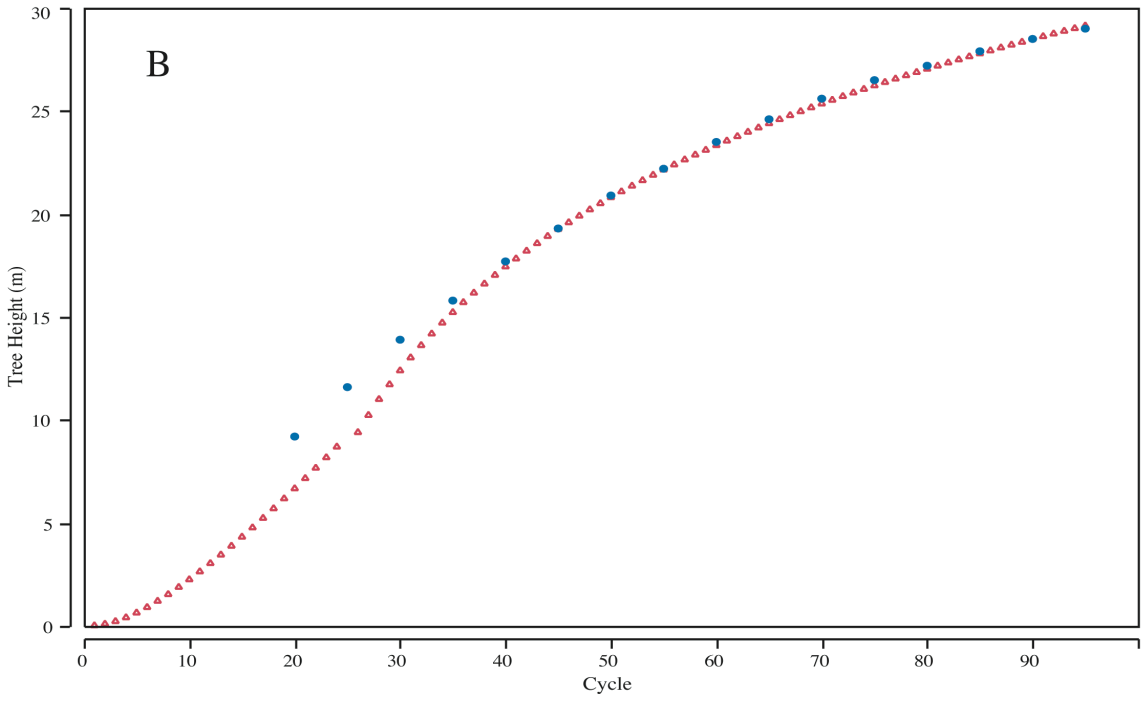
> *The best worked-out example of this law so far concerns the large chela of male fiddler-crabs,* Uca pugnax. *This obeys the law of constant growth-ratio from crabs of only about 60 milligrams total weight to the largest found, weighing sixty times as much; the value for k[6], however, changes quite abruptly at about 1.1g total weight, a point which probably denotes the onset of sexual maturity, decreasing here to less than 80 per cent. of its value for the earlier growth-phase.* (Huxley, 1932)(Figure 2.5)

---

[6] Huxley's early work designated the variable *k* as being the exponent in the power law equation he became associated with. It wasn't until 1936 that Huxley advocated using the form $Y=bX^{a}$. See Chapter 1.

Figure 2.3: A) Actual growth data from a managed stand of *A. alba* plotted in blue (Cantiana, 1974). Simulated growth using a simple Microsoft Excel based simulation using Equation 2.13a plotted in red. Simulated growth using a simple Microsoft Excel based simulation using Equation 2.13b in green. B) Actual growth from a managed stand of *A. alba*, plotted in red, compared to a combined Microsoft Excel based simulation where growth transitions from using Equation 2.13a to 2.13b once the value of 2.13b exceeds the value of 2.13a.


A
Tree Height (m)
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
Cycle


B
Tree Height (m)
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
Cycle

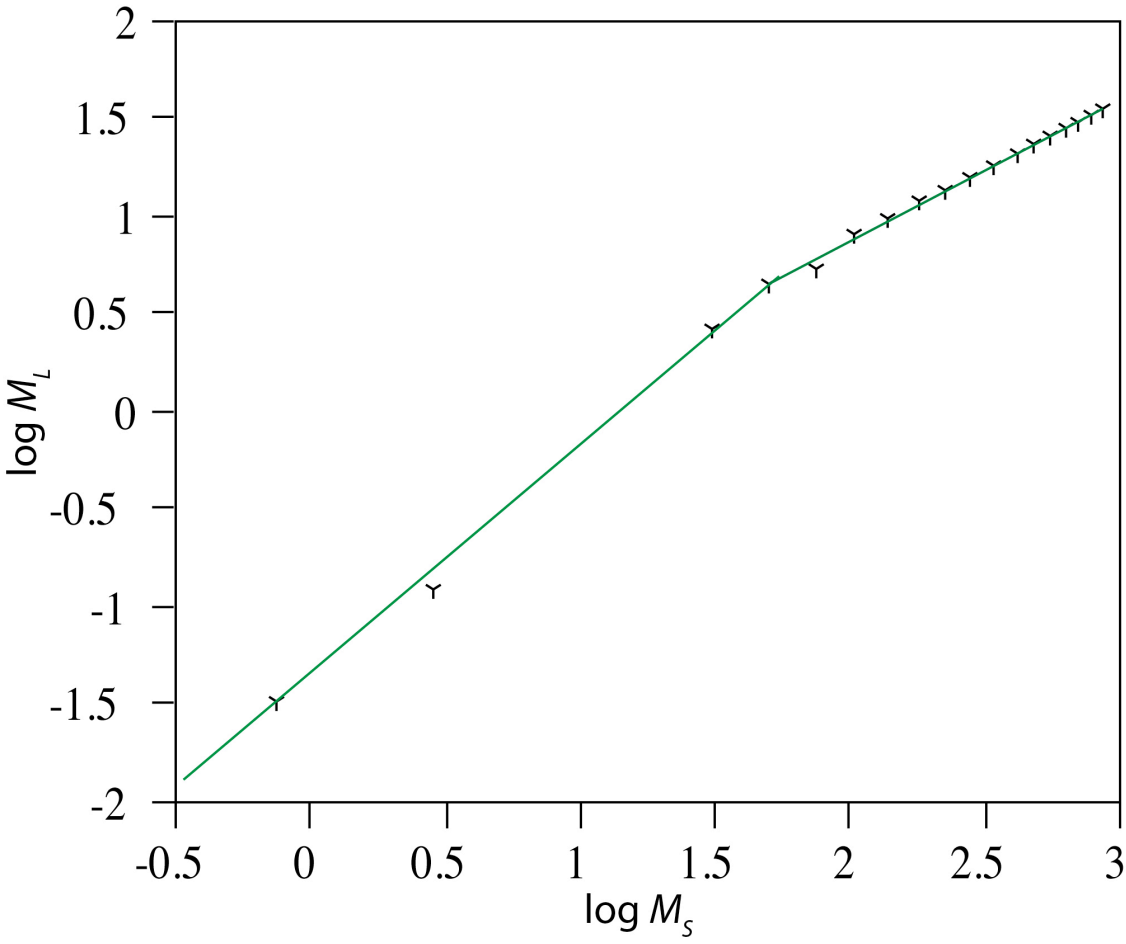


Figure 2.4: Data drawn from the Cannell data set showing the relationship between the mass of the canopy (M_L) and the mass of the trunk and branches (M_S) of a stand of *A. alba*, illustrating a change in the slope of the line.

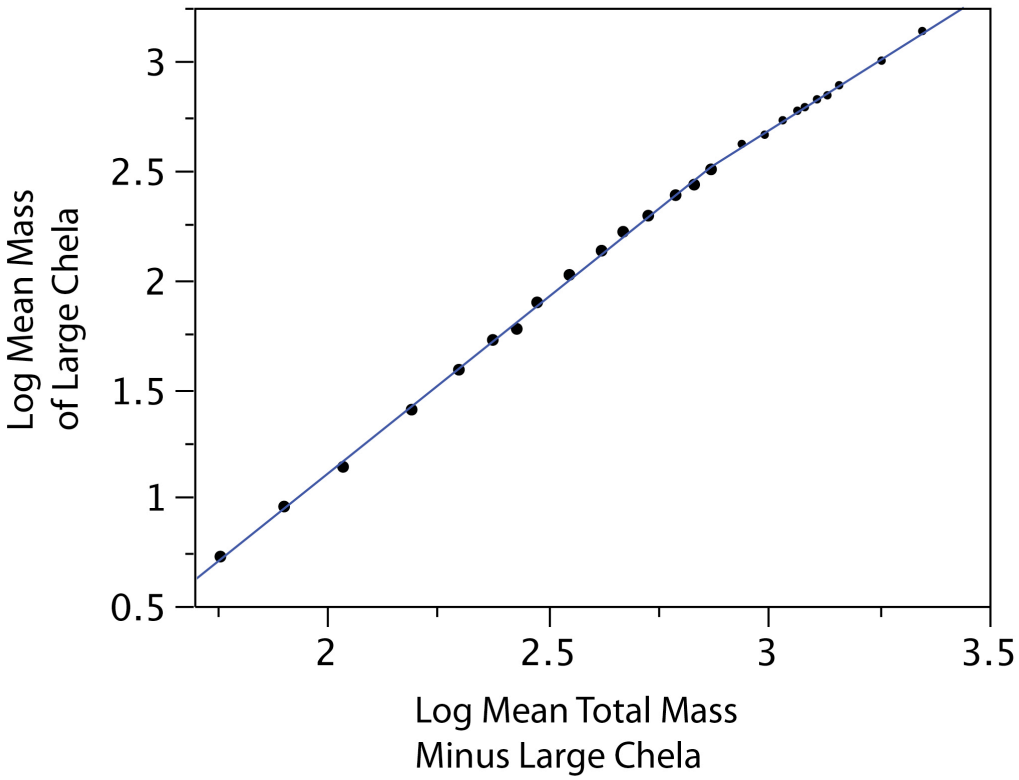


Figure 2.5: Log-log plot of data originally presented by Huxley, showing the relationship between the mass of the large chela and mass of the body of *U. pugnax*, illustrating a change in the slope of the line. (Huxley, 1932).

An individual tree's relationship between M_L and M_S is clearly not a single line but, like the *U. pugnax* chela vs. body mass relationship, it "changes quite abruptly." The exact cause for this shift within trees is not necessarily transparent when one examines large datasets, and it can be completely obscured if one combines data for populations (even of the same species) growing in differing geographic locations.

The perception that $M_L \propto M_S^{\alpha}$ or $H_S \propto D_S^{\alpha}$ are log-log linear emerges from studies in which multiple genera are grouped together. While useful to ascertain cross-species trends, differences between individual species become noise lost in the analysis. Take, for example, data on human growth provided by the United States' National Center for Health Statistics (Centers for Disease Control and Prevention, 2000). Due to sexual dimorphism, merging data for male and female growth patterns results in a poor model to approximate either male or female growth, but would allow one to conceptualize what a "human" growth model would be (Figure 2.6). Including other closely related species, such as *Pan trogloditis*, *P. troglodytes*, *P. paniscus*, *Gorilla gorilla* and *G. beringei*, one can generate a model for the living Hominidae, but such a model would obviously be poor at predicting the growth of a *Homo sapiens* male.

Thus, one can continue to add more and more data to an analysis, making it more and more inclusive. While informative as to general trends, such an amalgam of data can never accurately model any real species; instead, it shows an abstractly useful—but practically deceptive—average. In general, one can best see the transitions in $M_L \propto M_S^{\alpha}$ or $H_S \propto D_S^{\alpha}$ relationships when one examines a specific population growing in the same geographic location so that all individuals are experiencing the same environmental fluctuations.

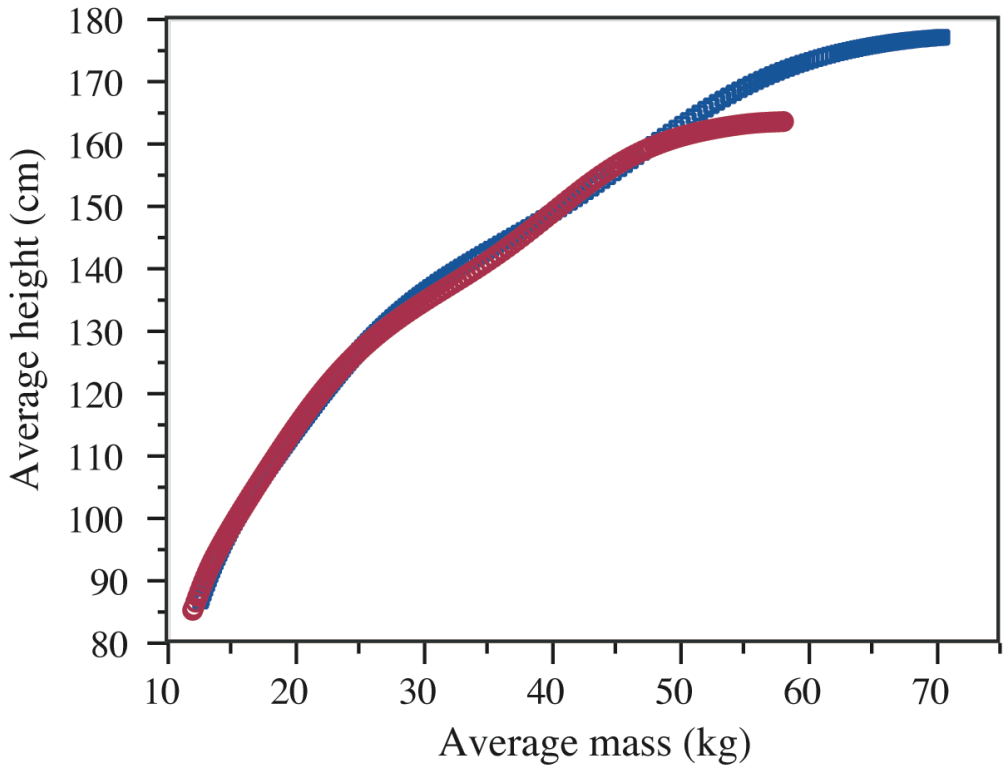


Figure 2.6: Comparison of the growth of male (blue) and female (red) *Homo sapiens* height compared to increase in mass. Combining the two datasets would result in an accurate model of human growth, but would lack information relevant to modeling sexual dimorphism within the species.

Taking into account the minor changes in formulae to make them mirror reality more accurately, the four basic equations showing carbon allocation in the above-ground portions of a tree are

$$M_S = \beta_4 M_T^{\alpha_4} \qquad \text{(Equation 2.9)}$$

$$M_L = \beta_5 M_S^{\alpha_5} \rightarrow M_L = \beta_{11} M_S^{\alpha_8} \qquad \text{(Equation 2.14a and b)}$$

$$D_S = \beta_6 M_S^{\alpha_6} \qquad \text{(Equation 2.11)}$$

$$H_S = \beta_7 D_S^{\alpha_7} - \beta_8 \rightarrow H_S = \beta_9 + \beta_{10} \ln D_S \qquad \text{(Equation 2.13a and b)}$$

A physical aspect of plants that has not been touched on concerns their reproductive structures. Different plant species have evolved vastly different details in how they reproduce. Clonal reproduction via plantlets, seed-filled cones, nuts, and fruits are only a few of the myriad ways in which plants have found effective ways to reproduce. With such a diverse set of reproductive structures, the word *propagule* encompasses all of these structures.

The propagules produced on the Platonic trees described here can be very simply represented as spheres. Their diameter (D_P) is calculated using the species-specific values for propagule density (ρ_P) and mass (M_P)

$$D_P = \left[\frac{M_P}{(4/3)\pi\rho_P} \right]^{1/3} \qquad \text{(Equation 2.15)}$$

Germination —

Different plant species have evolved vastly different details in how they reproduce. Clonal reproduction via plantlets, seed-filled cones, nuts, and fruits are only a few of the myriad ways in which plants have found effective ways to reproduce. With such a diverse set of reproductive structures, the word *propagule*, as noted, will be used within this work to encompass all of these structures.

All plants simulated by Vida begin as propagules. Whether a propagule germinates depends on three factors: (1) a pseudo-random chance of death related to what fraction of seeds fail to germinate, (2) if seed mass is larger than a critical minimum value, and (3) whether a species is characterized as having delayed germination.

Factor (1) is optional in that the default setting is to ignore germination failure during the initial iteration. Factor (2) is also optional because simulations are initialized with propagules that have the correct mass for any given species. The only time factor affecting a run is when plants in the simulated world begin making and dispersing their own propagules. Factor (3) is explicitly defined for each species (most species have a germination delay set to 0, i.e., germination occurs in the first year).

Assuming a propagule can germinate, a series of simple calculations is used to determine exactly what fraction (ϕ) of its mass (M_D) is converted to the newly germinated plant (M_T), and how much of the seedling's mass is stem (M_S) and canopy (M_L), i.e.

$$M_T = \phi_1 M_P \qquad \text{(Equation 2.16)}$$

$$M_S = \phi_2 M_T \qquad \text{(Equation 2.17)}$$

$$M_L = M_T - M_S \qquad \text{(Equation 2.18)}$$

Vegetative growth —

The size of the canopy, and the extent to which it is shaded by neighboring plants, dictates the ability of the individual to harvest light and thus grow. Vida assumes that all light energy (E) comes from directly above each plant and that light interception is time-averaged over a year, i.e., the unit of time in any iteration i.

Attenuation of E as it passes through a tree's canopy (E_T) is an important component regulating whether or not understory plants can survive. At one extreme, if no light reaches the ground, no plants can survive until a gap appears in the forest due to the

death of a tree. The reality is that some light will filter through canopies, so an understory plant's ability to survive rests on its shade tolerance and just how much light passes through the canopy.

While there is information available about the shade tolerance of various species, it is often presented as a relativistic measurement (Niienemets and Valladares, 2006). What is known is that approximately 20% of full sunlight passes through a single tree's canopy and reaches the ground (personal communication with Dr. Thomas Owens, Cornell University). Therefore, is it reasonable to assume a 20% transmission rate for simulated tree canopies, and that, if a tree receives less than 20% of E in a simulation, it should die due to lack of light. Calculations on how shaded a given plant might be are complicated in that it can be partly or wholly shaded by one or more overtopping canopies.

The best solution to calculate the amount of light energy (E) reaching a plant is to use a Monte Carlo method in which a fixed number of 'photons' are moved through the z-axis (Wilson, 1983; Prahl, 1989; Hasegawa, 1991). Every time a photon encounters a canopy, there is a fixed chance whether or not it will continue to move downward in the z-axis, or be 'absorbed'. For example, if a canopy allows 20% of the light to pass through it, than, on average, two out of every ten photons in the axis corresponding to the location of a canopy will continue downward. For the sake of simplifying calculations, we can assume that there is no light scattering in the atmosphere or in tissues, so photons only move from $+z$ to $-z$.

To implement this concept, a given tree's canopy is first bounded by a box and all trees the same height or taller than the target tree are tested to see whether their canopies overlap partly or wholly. Next, a series of 'photons' are generated within the bounding box, having a z value greater than the highest tree canopy. Photons with x, y coordinates falling outside the canopy area of the test tree are removed and the z-axis of

the remaining ones are set to the tallest tree. All trees have a value defining how much light passes through their canopy (E_T) and, if a given photon intersects a given tree, it has a (E_T)*100% chance of continuing. The remaining photons are set to the next lowest canopy height and again tested. After all photons reach the target tree, one can determine how shaded the tree is by comparing the total number of photons reaching the tree with the starting number of photons. If, for example, 500 photons begin a z_{max+1} and 250 photons reach the z-axis at which the tree's photosynthetic surface is located, the tree is 50% shaded.

The maximum possible photosynthetic area per plant is the projected surface area of the canopy (A_L), and the annual growth (G_{T_i}) equals the sum of the change in leaf mass per year (dM_L /dt) and the change in stem mass per year (dM_S/dt). The conversion of E into growth per iteration is described by

$$G_{T_i} = E\ A_L\left(\beta_{12} M_L^{\alpha 9}\right) \qquad \text{(Equation 2.19)}$$

where β_{12} and α_9 denote a species-specific normalization (allometric) constant and scaling exponent, respectively, A_L is the projected canopy area, and E is a fraction of full sunlight reaching the photosynthetic surface (Figure 2.7).

Growth in biomass is subsequently partitioned into new stem and canopy mass using the formulas

$$M_{S_i} = M_{S_i - 1} + \beta_4 G_{T_i}^{\alpha_4} \qquad \text{(Equation 2.20)}$$

$$M_{L_i} = \beta_5 M_{S_i}^{\alpha_5} \rightarrow M_{L_i} = \beta_{11} M_{S_i}^{\alpha_8} \qquad \text{(Equation 2.14a and b)}$$

where the subscript $i - 1$ denotes mass in the previous iteration of growth (which for a propagule is specified when a simulation is initiated; see ***Germination***). Equations 2.19, 2.20 and 2.14(a and b) are reiterated for each growth cycle. It is important to note that using Equations 2.14(a and b) and 2.20 to calculate M_S and M_L typically stipulates

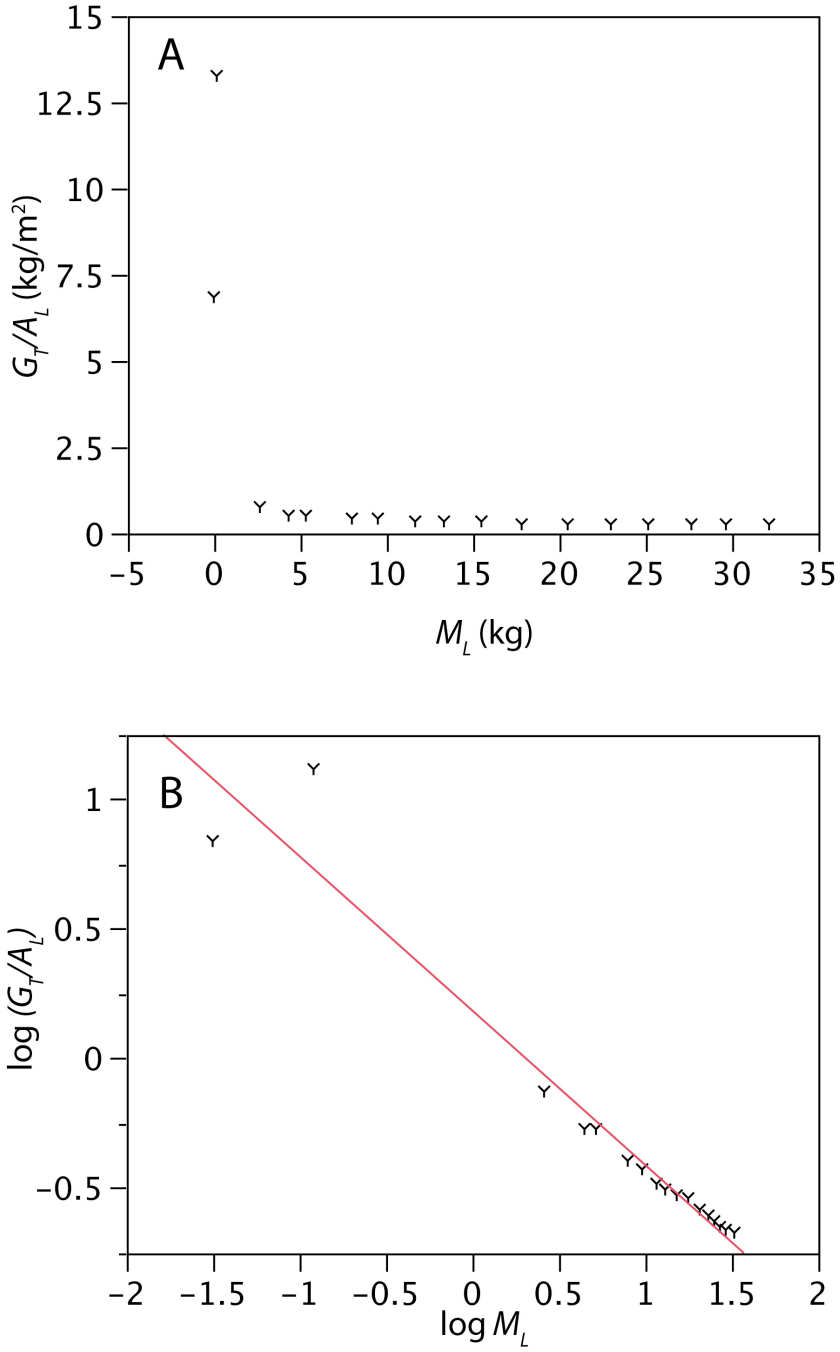


Figure 2. 7: Total growth divided by projected surface area of the canopy (G_T/A_L) verses the mass of the canopy (M_L)of a managed forest of *Abies alba*. A: simple plot of G_T/A_L versus M_L. B: log-log plot of G_T/A_L versus M_L, where the fit line is described by $G_{T_i} = E\ A_L\left(\beta_{12} M_L^{\alpha 9}\right)$ where E is equal to 1.0.

that $M_S + M_L \neq M_T$. Rounding errors and incomplete field data are the source of this error.

For each iteration, Vida uses G_{T_i} to determine changes in M_S, M_L, D_S and H_S. As Equation 2.13 indicates, there is a shift in how H_S is calculated. This shift occurs in the iteration after the condition $H_{S_b} > H_{S_a}$ has been reached. Computer runs and allometric analyses of data from real species indicate that this shift occurs when a species becomes reproductively mature.

When species become reproductively mature, a second shift in growth occurs for the mass allocated to the canopy, using Equation 2.14a for young plants and using Equation 2.14b for older plants once the software begins using 2.13b to calculate plant height.

Propagule Production, Location and Dispersal

A number of questions need to be addressed when simulating plant reproduction: what factors trigger the onset of sexual maturity? How much carbon is diverted toward reproductive efforts? What is the mass of a propagule and how many are produced? Where are the reproductive units located upon the mother plant? How are they dispersed? The answers to these questions differ from species, but insights that allow models to be made.

Onset of Maturity

As had previously been discussed, Huxley hypothesized that the abrupt change he observed in *U. pugnax* chela mass, relative to the mass of the rest of the body (Figure 2.5), had to do with the onset of sexual maturity (Huxley, 1932). The a similar change is observed in the M_L relative to M_S for individual species, as described by Equation 2.14 (Figure 2.4), and in the relationship between D_S and H_S, as described by Equation 2.13 (Figure 2.3). The point at which Equation 2.13b results in values for H_S that

exceed those from Equation 2.13a coincides with the transition described by Equation 2.14. That this transition occurs at sexual maturity is described in more detail within the section "**A Real Species**".

The transition from Equation 2.13a to Equation 2.13b is based only on the values of the two equations and not on arbitrary values. Here, then, is an excellent trigger to mark the onset of reproductive maturity and changes in the allocation of carbon to the canopy. Within simulations the onset of reproductive maturity is based on the first time Equation 2.13b is greater than or equal to Equation 2.13a, versus explicitly defining the age of reproductive maturity. The advantage of this method is that it is able to take into account environmental factors that affect when real-world plants reach reproductive maturity. With H_S dependant on the growth of a simulated tree, the greater a tree is shaded, the slower its growth, and thus the longer it takes for the tree to reach sexual maturity.

Allocation to Reproduction

Within the world-wide compendium for forestry data compiled by Cannell (1982), there are several datasets that record the mass of propagules (M_P). A basic model showing a relationship between propagule mass and growth (G_T) was generated (Figure 2.8), represented by the formula

$$M_{P_T} = \phi_3 \beta_{13} G_{T_i}^{\alpha_{10}} \rightarrow M_{P_T} = \beta_{13} G_{T_i}^{\alpha_{10}} \qquad \text{(Equation 2.21a and b)}$$

where M_{P_T} is the total mass of all propagules produced by a given plant during a growth cycle, β_{13} and α_{10} denote a species-specific normalization (allometric) constant and scaling exponent, respectively, and ϕ_3 is a species-specific value defining what fraction of total growth per year is dedicated to the construction of propagules. The maximum number of propagules (N_P) that a plant can produce during any growth cycle is

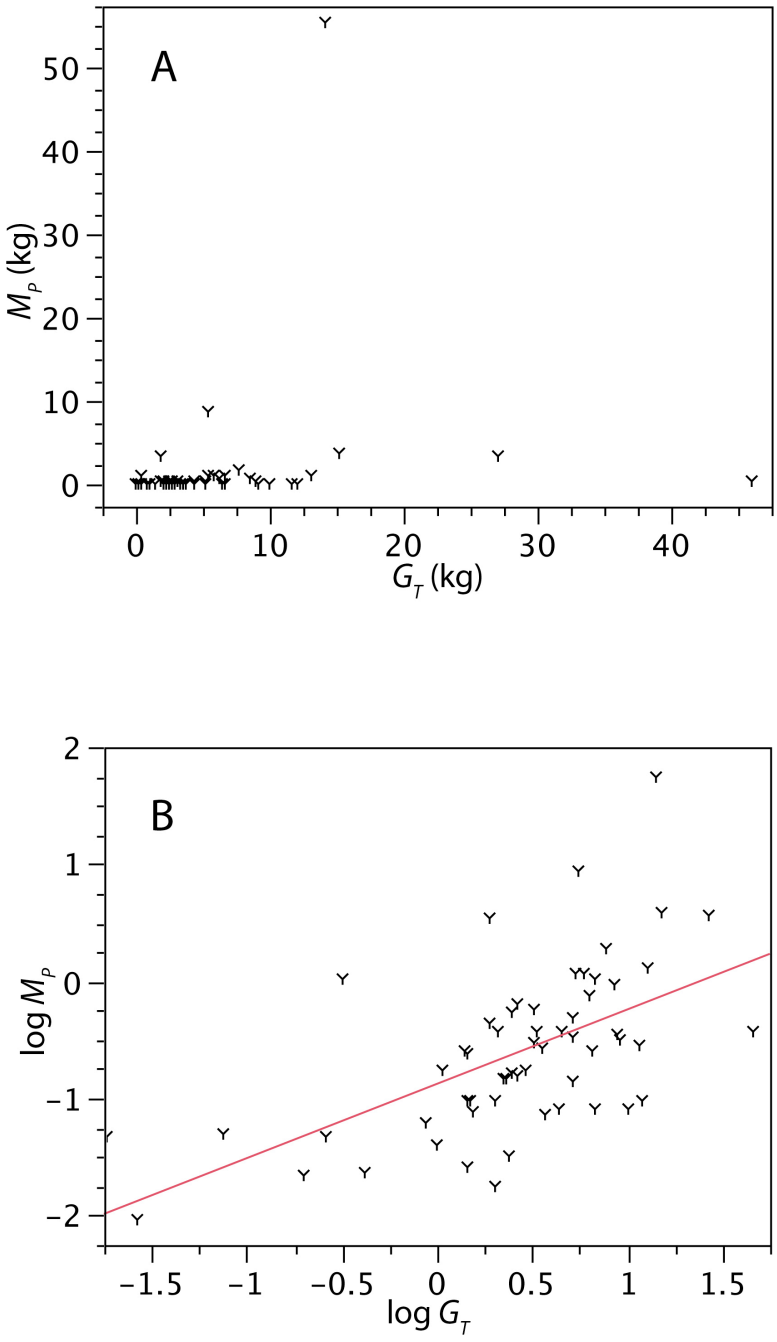


Figure 2.8: Average mass of all propagules above ground woody growth (M_P) verses total above ground growth (G_T) of all gymnosperms in the Cannell(1982) data set which have information regarding the mass of reproductive structures. A: simple plot of M_P versus G_T. B: log-log plot of M_P versus G_T, where the fit line is described by $M_{P_T} = \phi_3 0.0997 G_{T_i}^{1.0978}$, where ϕ_3 is equal to 1.0.

M_{P_T} divided by the mass of an individual propagule ($M_{P_{ideal}}$):

$$N_P = \frac{M_{P_T}}{M_{P_{ideal}}} \qquad \text{(Equation 2.22)}$$

The shift described by Equation 2.21 occurs when vegetative growth slows relative to an average value. Each species has a defined "growth memory" (*m*) that determines how many years of previous growth a plant has experienced. For each iteration of a plant's life, the average growth ($G_{T_{average}}$), given by

$$G_{T_{average}} = \frac{\sum G_{T_{i-m}} ... G_{T_i}}{m+1} \qquad \text{(Equation 2.23)}$$

is compared against the current growth cycle's growth (G_T in Equation 2.22). If $G_{T_i} < G_{T_{average}}$, the fractional difference (ϕ_2) between the two is determined:

$$\phi_4 = \frac{G_{T_i}}{G_{T_{average}}} \qquad \text{(Equation 2.24)}$$

Species can be seen as having levels of "selfishness" regarding the mass used to construct propagules. This selfishness factor (ϕ_3) represents the extent to which a plant invests more in reproduction than it requires for sustained vegetative growth. The conditions $G_{T_i} < G_{T_{average}}$ and $\phi_3 \geq \phi_4$ trigger a plant to set $\phi_3 = 1.0$ and to divert as many resources to propagule production as possible, since the sudden reduction in growth could be due to severe changes in the environment or being rapidly overtopped by neighboring plants.

In summary, the selfishness factor quantifies the extent to which a plant invests more in reproduction that it requires for sustained vegetative growth. Specifically, if a plant is rapidly overtopped, annual growth rate will rapidly decrease, indicating that the plant risks death due to light deprivation. The level of ϕ_3 is the trigger point at which

the plant will shift from using Equation 2.21a to using Equation 2.21b. For example, if a species is 50% selfish (i.e., $\phi_3 = 0.5$), a plant would begin using Equation 2.21b instead of Equation 2.21a in an attempt to produce more propagules before it died due to light deprivation.

Using information on the mass of individual seeds one can estimate the total number of propagules produced, if one assumes all the M_P is used to produce seeds and not fruit flesh or cones. For example, the average mass of an individual seed of the Silver fir, *Abies alba*, is 0.000027kg (Goudwaard, 2006). Therefore, M_P is 1kg, 37,037 seeds can be produced, an unreasonable number for use in simulations.

To account for the mass of cones and other structures not directly measured by seed mass, the concept of M_P is any structure that is directly related to reproduction. For example, in the case of *A. alba*, M_P would include the mass of cones. Reproductive units were also combined so, instead of producing hundreds of cones, each filled with seeds, propagules represent a collection of cones and seeds, and from each propagule a single plant is produced. Upon germination, it is assumed that the new seedling should have the same mass as a single seed. Therefore, a certain mass of the propagule (M_{P_W}) is lost. For example, if $M_{P_{\max}}$ is 0.6 kg and the average seed mass from the literature is 0.000027 kg, M_{P_W} would be equals 0.000045 kg.

Propagule growth and dispersal —

Propagule formation and growth on a canopy, like all aspects of Vida, are spatially explicit. Each propagule occupies a specific *x, y, z* location. Propagule location is defined by two variables (ϕ_5, ϕ_6), each defining a fraction of the radius of the canopy. The value ϕ_5 represents the outermost fraction of the canopy radius propagules can occupy. The value ϕ_6 is the innermost fraction of the projected canopy radius at which propagules can be placed, e.g., the values $\phi_5 = 1.0$ and $\phi_6 = 1.0$ result in propagules

being placed at the canopy edge; the values ϕ_6 =1 .0, ϕ_6 = 0.5 result in propagules located at the outermost 50% of the projected canopy (Figure 2.9). The default setting allows propagules to develop anywhere on the canopy surface, i.e., ϕ_5 = 1.0, ϕ_6 = 0.0.

An ideal propagule mass is defined for each species. As propagules mature (whether over several iterations, or during a single iteration), their mass increases based on Equations 2.21 and 2.22, where each propagule will receive a certain amount of mass by dividing total propagule mass by propagule number. Using this method, it is possible for the mass of individual propagules to exceed $M_{D_{ideal}}$. Once $M_D \geq M_{D_{ideal}}$, the propagule is immediately dispersed.

There are five optional modes for propagule dispersal: (1) vertically from the canopy, (2) random dispersal, irrespective of the distance from a canopy, (3) random dispersal within a circle (with a specified radius) centered on the canopy, (4) random dispersion with a specified maximum distance orthogonal to the canopy-edge, and (5) the default setting, ballistic dispersion. Ballistic dispersal is done in such a way that it is assumed that it occurs in a vacuum, and that propagules pass through canopies and stems. Dispersal distance is governed by the launch angle relative to the horizontal (θ) and the initial velocity of the propagule (v). In the default setting, $\theta = 45^{\circ}$ and v = 5 m/s. Noting that drag is neglected, the distance traveled is calculated using the formula

$$\text{Distance} = \left(v\cos\theta/g\right)\{v\sin\theta + \left[v\sin\theta^2 + 2gz\right]^{1/2}\} \qquad \text{(Equation 2.25)}$$

where g is the acceleration due to gravity (9.81 m/s^2) and z is the height of the propagule above ground.

The absolute maximum number of propagules a given tree can produce is limited by values provided at the start of a simulation. This limitation provides a cap to deal with computational limitations.

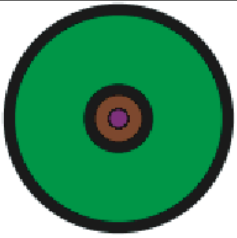
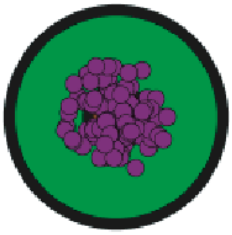
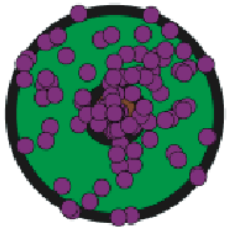
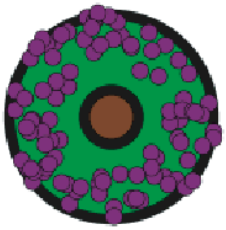
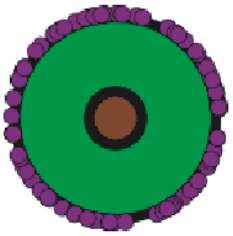
Description	Setting	Image
A	[0.0,0.0]	
B	[0.5,0.0]	
C	[1.0,0.0]	
D	[1.0,0.5]	
E	[1.0,1.0]	

Figure 2.9: Polar views of simulated trees showing examples of propagule (purple circles) formation on canopies (green circles). Stems (brown) are visible simply as a visual aid. A) Propagule formation is limited to the center. B) Propagule formation occurs on the inner 50% of the canopy. C) Propagule formation occurs over the entire surface of the canopy. D) Propagule formation occurs over the outer 50% of the canopy. E) Propagule formation occurs only along the outermost edge of the canopy.

Mortality

There are many ways in which an individual plant can die within a simulated world: (1) a propagule can fail to germinate, (2) if a propagule is dispersed outside of the boundaries of the world-space, (3) if a propagule lands on a location occupied by another plant or propagule, the smaller of the two objects (plant or propagule) dies, (4) if two stems grow and touch each other, the plant with the smaller mass dies, (5) if a plant exceeds its critical buckling height (calculated on the basis of the Euler-Greenhill formula (Greenhill, 1881; see Niklas and Spatz, 2006), (6) random death independent of propagule/plant size (to mimic stochastic processes such as tree fall or fire), (7) age-dependent death where the probability of death increases with age (to mimic increased risks of death by disease or some other age-dependent process), and (8) light deprivation resulting from overtopping canopies. Each of these methods of dying falls into one of three categories: basic physics, growth constraints, and stochastic processes.

Death due to Physical Laws

Within simulations, no two objects are allowed to occupy the same location. The exception to this is that canopies can become intermeshed. When there is a situation when two objects overlap, the object with the smaller mass dies. For example, a tiny seedling could be crushed by a massive coconut-like propagule. Likewise, a newly dispersed propagule's x and y location within the simulation might overlap with the trunk of a massive tree. In situations in which equally massed objects overlap, one is randomly chosen for death.

Because the simulated world space has defined edges, any object that extends off of the world-space dies. For example, propagules that are dispersed off of the simulation space are immediately killed, and any tree who's stem extends off the world-space is

also killed. The exception to this rule is that canopies can extend off the world-space, and continue to receive light.[7]

Trees can also die due to stem buckling, if their height exceeds the Euler-Greenhill critical buckling height (Greenhill, 1881; Niklas, 2006):

$$H_{S_{Critical}} = 0.79\left(\frac{E}{g\rho_S}\right)^{1/3}\left(D_S^{2/3}\right) \qquad \text{(Equation 2.26)}$$

where E is the Young's modulus for the stem, g is the gravity, ρ_S is the density of the stem, and D_S is the diameter of the stem.

Stochastic Mortality

Without clear data on real-world mortality, test simulations were run to manually alter the rate of stochastic death in the simulation space until no more than one tree out of 100 lived to reach 600 years of age within the simulated world. The value arrived at means that every plant in a simulation has a 0.75% chance of dying during each iteration of a simulation.

Growth Constraints: Lack of light

Death due to insufficient light caused by overtopping plants shading an individual was covered in the section discussing how G_T was calculated. To recapitulate, each species is defined by a minimum fraction of the projected canopy area that must receive sunlight. Currently, all species must have 20% or more of their projected canopy area exposed to full light to survive. This calculation is complicated by the fact that each

[7] It should be noted that individuals growing on the edges are in interesting situations where they encounter less competition due to shading, but that a percentage of their propagules are wasted by being dispersed off world. Therefore, growing along an edge is beneficial to the individual plant, but that individual's chances of reproducing successfully are reduced.

species also allows a certain fraction of light to pass through its canopy (default for each species is 2%). For example, if a plant is completely overtopped by a larger plant, the smaller plant's canopy would receive only 2% of the available solar energy. If a plant is shaded by only one other plant, the amount of shading is directly calculated using hemispherical canopy geometry. If, however, a plant is shaded by two or more plants, a Monte-Carlo method is employed to estimate total shading.

Growth Constraints: Senescence

Like stochastic mortality, there is little empirical data to model senescence. Vida's senescence routines assume that an organism's vigor decreases over time due to damage caused by external factors (physical damage, viruses, bacteria, etc.) that eventually exceeds the organism's rate of growth and repair. One can think of this as a Red Queen (not to be confused with Red Queen Hypothesis proposed by van Valen, 1973) model of senescence; when an organism is young, it can outgrow the rate of damage, but as its growth slows—and the rate of damage remains constant—death is inevitable.

To model senescence, a given tree's fastest growth in height ($G_{H_{\max}}$) is recorded. Its average increase in height (G_{H_μ}) over a moving 'growth memory' window is calculated every iteration a plant grows. These values are used to determine what the fraction of $G_{H_{\max}}$ the current G_{H_μ} is:

$$\theta_7 = \frac{G_{H_\mu}}{G_{H_{\max}}} \qquad \text{(Equation 2.27)}$$

If at any point θ_7 drops below a defined value (θ_8), the tree in question enters a second round of stochastic mortality. For example, if a given tree's growth has slowed to a point that $\theta_7 < \theta_8$, and if the stochastic mortality value is 0.75%, its overall chance of dying due to senescence and stochastic death doubles to 1.5%.

To determine reasonable values of θ_8, test simulations were run using different

values. Simulations were initialized with 100 trees and various values of θ_8 were used until all trees survived to be at least $H_{S_{\max}}$, but died before exceeding $H_{S_{\max}} + 5$.

A Real Species: Abies alba

Out of the six hundred and seventy-five entries in the Cannell database (Cannell, 1982), one entry proved particularly well documented. A single managed *Abies alba* population, with an initial population density of 25,000 plants/hectare, was observed and documented every five years, starting at the tenth year. The dataset concluded on the 95th year, providing 18 data points for most variables.

Abies alba, commonly called the silver fir, is native to Europe and primarily grows in mountainous regions stretching from Spain in the west, to Bulgaria and Greece in the east (Figure 2.10). In general, as one moves into lower latitudes, *A. alba* is found in higher elevations, forming belts 500 to 600 m wide within the areas with the densest growth (Wolf, 2003). *A. alba* is shade tolerant, and young trees are able to survive under tree canopies for decades (Niinemets 2006). When a gap appears in the canopy, these understory trees rapidly fill the space.

The silver fir is the tallest species of the genus. Trees can reach ages of 500 to 600 years old (Wolf 2003), reach heights of 45 to 55 m (Goudwaard, 2006) and have trunk diameters (DBH) of 150 to 200 cm (Wolf, 2003). Trees become reproductively mature between 25 and 35 years when isolated, whereas trees in a forest normally reach reproductive maturity between 60 and 70 (Wolf, 2003), with each seed weighing an average of 0.027 g (Goudwaard, 2006).

Using the Cannell dataset, standardized major axis (SMA, also known as reduced major axis) regression analyses of bivariate plots of plant height (H_S), trunk diameter (D_S), canopy, stem and total tree mass (M_L, M_S, and M_T), and annual canopy and stem growth rates (G_L and G_S) were determined. The numerical values of these regressions

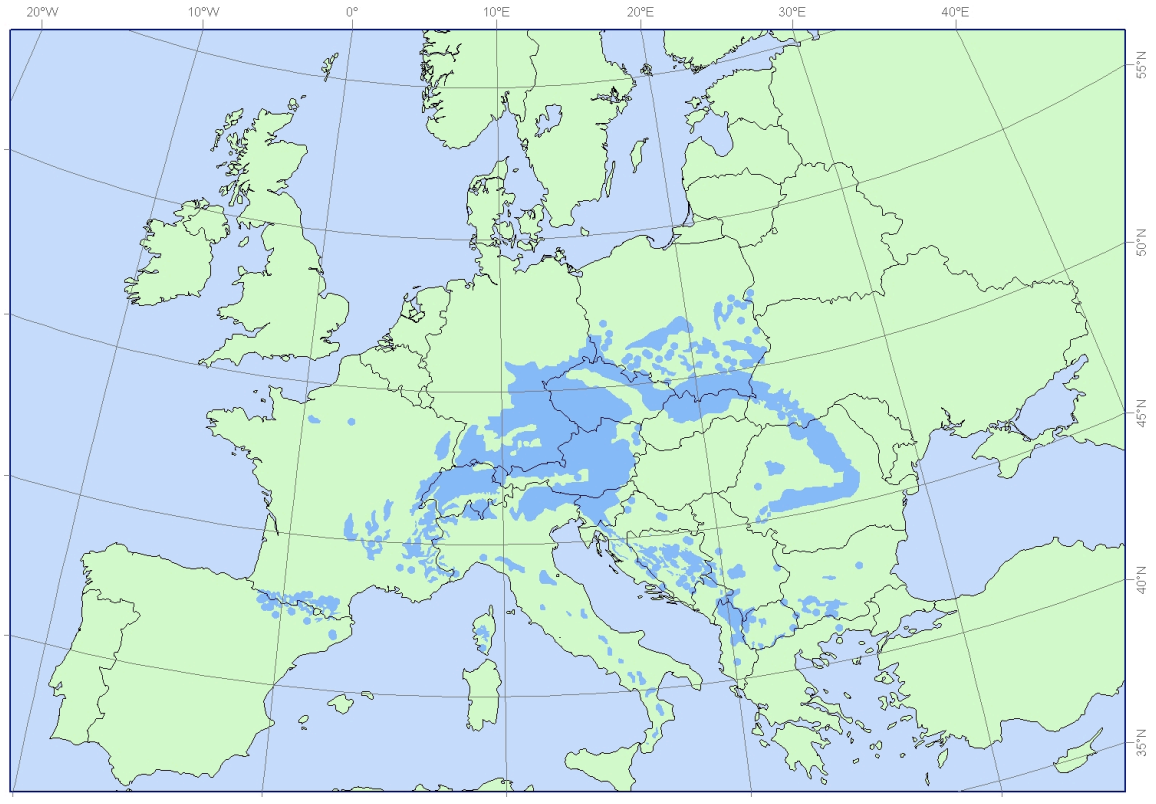


Figure 2.10: Distribution of *A. alba*, denoted in blue, in Europe as of 2004 (Wolf, 2004).

were used as input values in the generalized allometric formula $\log Y_2 = \log \beta \pm \alpha \log Y_1$, where Y_1 and Y_2 are interdependent variables (Figures 2.11-2.14) (Niklas, 1994; Warton, 2002, 2006).

As previously noted, close examination of the data indicated that the transition described by Equations 2.13 and 2.14 happens at the same time (within the time frames available in the dataset), i.e. after approximately 25 years of growth. As mentioned, Wolf (2003) reports that *A. alba* becomes reproductively mature after 25 to 35 years of growth in unshaded environments. Having both the onset of reproductive maturity and the transition happen at the same time as a result of some event is reasonable, given their empirical overlapping timeframes. For a simulation, one could simply define the age at which plants reach reproductive maturity and use that age as the trigger for the transition in carbon allocation to the canopy. However, doing so would not allow one to easily model subtleties regarding the onset of reproductive maturity. While *A. alba* grown in unshaded environments reaches reproductive maturity between 25 and 35 years of age, shaded trees growing in forests become reproductively mature as late as 60 to 70 years of age. Clearly, providing a fixed age at which plants become reproductively mature is not an ideal solution.

An examination of the relationship between stem height relative to diameter, as defined by Equation 2.13, offers a better solution. Table 2.1 shows values for H_S using both Equation 2.13a and Equation 2.13b in comparison to empirically reported values of H_S for *A. alba* from Cannell (1982). The point at which Equation 2.13b results in values for H_S that exceed those from Equation 2.13a represents a permanent change in how H_S is calculated in that equation 2.13a is replaced by 2.13b. The timing of this transition empirically corresponds with the transition in carbon allocation to the canopy as defined by Equation 2.14. For this reason, a basic model was constructed such that

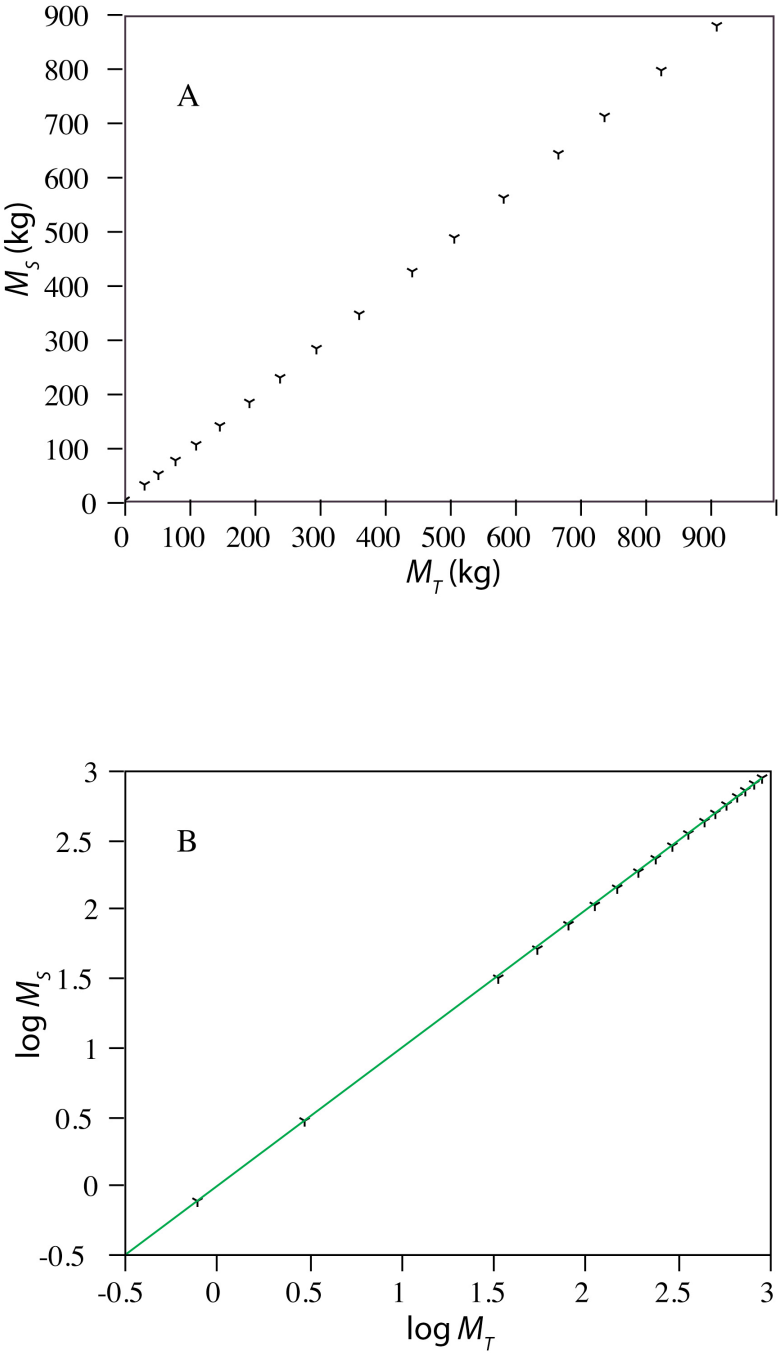


Figure 2.11: Average mass of all above ground woody growth (M_S) verses the total above ground mass (M_T) of a managed forest of *Abies alba.* A: simple plot of M_S versus M_T. B: log-log plot of M_S versus M_T, where the fit line is described by $M_S = 0.941 M_T^{1.0013}$

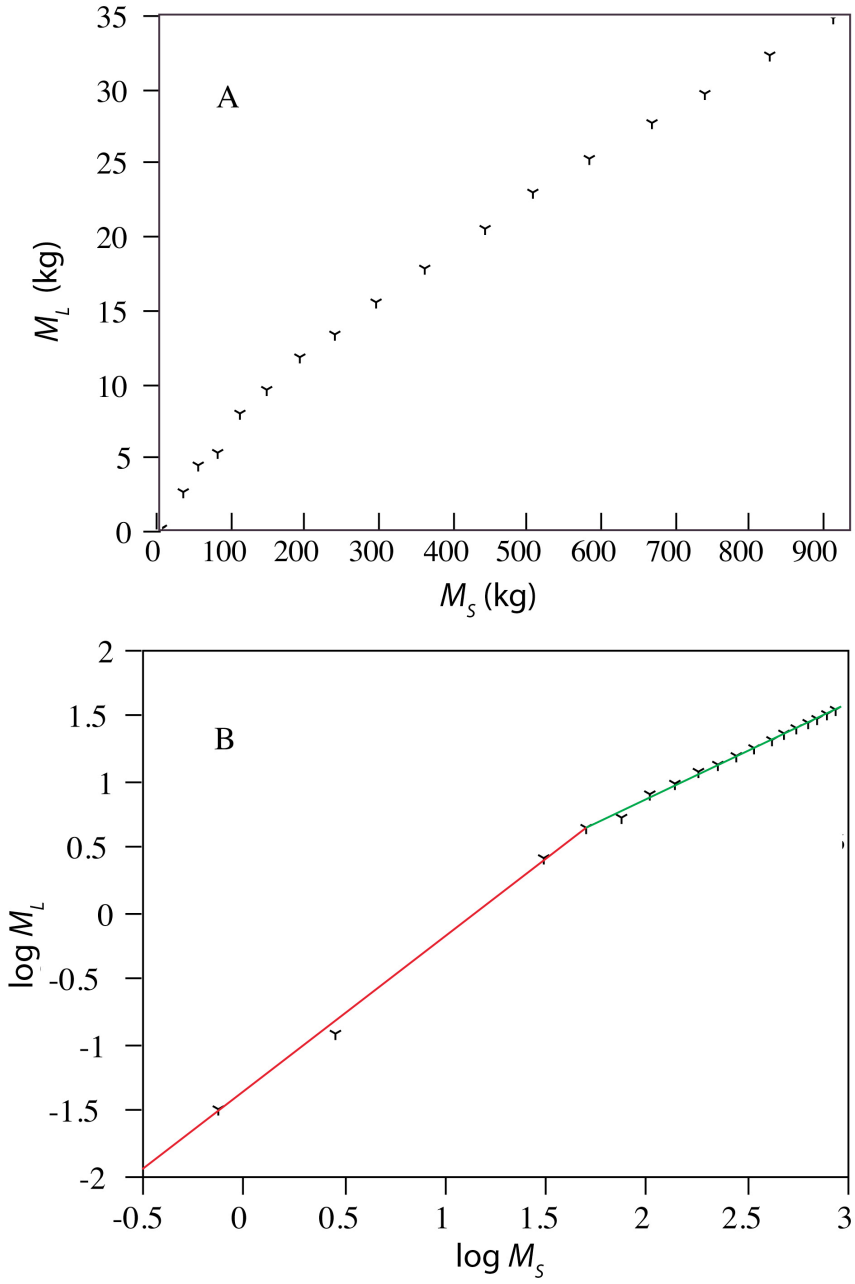


Figure 2.12: Average mass of the canopy (M_L) verses the all above ground woody growth (M_S) of a managed forest of *Abies alba.* A: simple plot of M_L versus M_S. B: log-log plot of M_L versus M_S, showing two distinct, where the fit lines are described by $M_{L_{young}} = 0.0397 M_S^{1.1982}$ and $M_{L_{mature}} = 0.2483 M_S^{0.7306}$

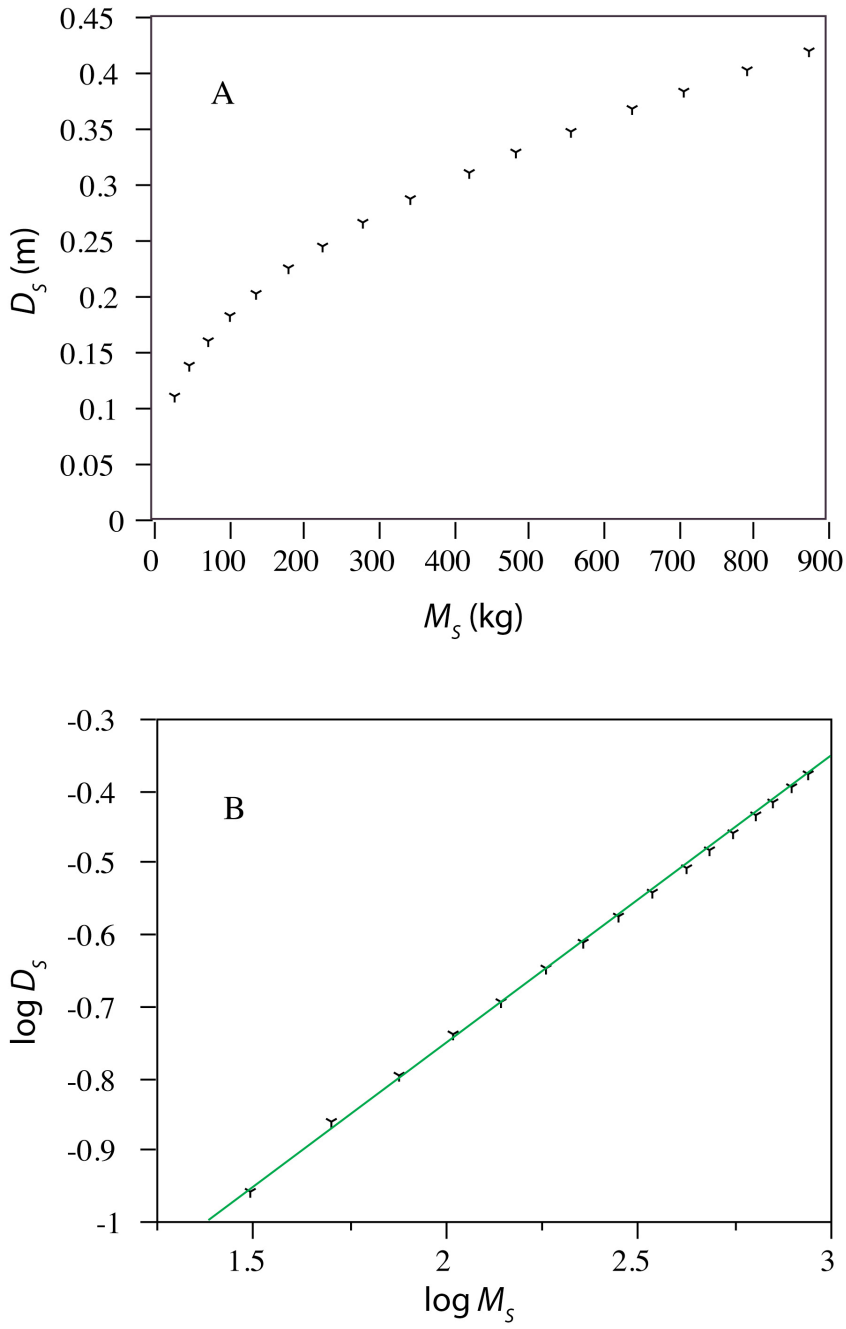


Figure 2.13: Average Diameter at Breast Height (DBH) (D_S) verses mass of all above ground woody growth (M_S) of a managed forest of *Abies alba.* A: simple plot of D_S versus M_S. B: log-log plot of D_S versus M_S, where the fit line is described by $D_S = 0.029 M_S^{0.3942}$

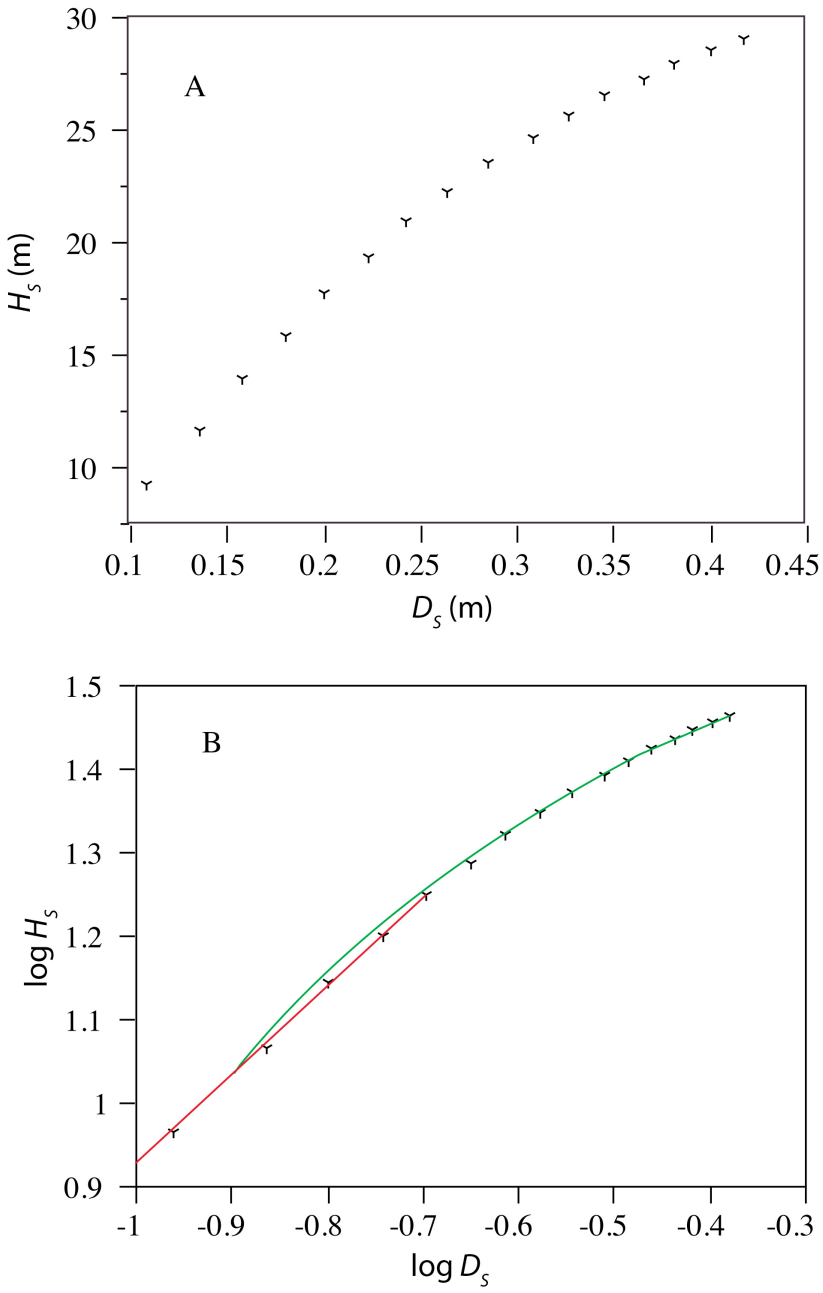


Figure 2.14: Average height of a tree (H_S) verses the Diameter at Breast Height (DBH) (D_S) of a managed forest of *Abies alba.* A: simple plot of H_S versus D_S. B: log-log plot of H_S versus D_S, showing two distinct, where the fit lines are described by $H_{S_{young}} = 100.0642 D_S^{1.0809}$ and $H_{S_{mature}} = 42.6982 + 15.5080 \ln D_S$

Table 2.1 Values of H_S generated through the use of $H_S = \beta_7 D_S^{\alpha_7} - \beta_8 \rightarrow H_S = \beta_9 + \beta_{10} \ln D_S$ (Equation 2.13), where the left hand side of the equation (Equation 2.13a) is H_S for *A. alba* younger than the age of sexual maturity, and the right side of the equation (Equation 2.13b) is H_S for *A. alba* at the age of maturity, or older. Values corresponding with the onset of sexual maturity are in bold. Units for variables: H_S (m).

Cycle	Actual H_S	Predicted $H_{S\ young}$	Predicted $H_{S\ mature}$
10	ND	1.92244	-14.005
15	ND	3.40142	-5.8187
20	9.2	9.54321	8.98244
25	**11.6**	**11.7526**	**11.9702**
30	13.9	13.8716	14.3485
35	15.8	15.9587	16.3594
40	17.7	17.9816	18.0717
45	19.3	20.1346	19.6942
50	20.9	22.0832	21.0196
55	22.2	24.1292	22.2908
60	23.5	26.27	23.5104
65	24.6	28.6324	24.7459
70	25.6	30.339	25.5765
75	26.5	32.1886	26.4256
80	27.2	34.0848	27.2468
85	27.9	35.5802	27.8628
90	28.5	37.3091	28.5436
95	29	38.9018	29.1433

Table 2.2: Values generated by a simple Microsoft Excel spreadsheet parameterized using empirical allometric relationships derived from data reported in Cannell, 1982. Values in red indicate when a transition in formula occurs. Units for variables: M_T (kg), M_S (kg), M_L (kg), D_S (m) and H_S (m).

Cycle	Actual MT	Predicted M_T	Actual M_S	Predicted M_S	Actual M_L	Predicted M_L	Predicted M_L	Actual DS	Predicted D_S	Actual H_S	Predicted $H_{S\ worst}$	Predicted $H_{S\ mature}$
10	0.792	**0.773**	0.760	**0.745**	0.032	**0.028**	0.202	ND	**0.026**	ND	1.922	-14.005
15	3.017	**2.983**	2.897	**2.844**	0.120	**0.138**	0.535	ND	**0.044**	ND	3.401	-5.819
20	33.870	**34.556**	31.279	**32.028**	2.590	**2.499**	3.119	0.110	**0.114**	9.200	9.543	8.982
25	55.183	**56.758**	50.780	**52.217**	4.404	4.480	4.453	0.137	**0.138**	11.600	11.753	11.970
30	81.388	**82.990**	76.130	**77.052**	5.258	7.131	**5.912**	0.159	**0.161**	13.900	13.872	**14.349**
35	113.043	**114.619**	105.122	**107.069**	7.921	10.565	**7.513**	0.182	**0.183**	15.800	15.959	**16.359**
40	149.533	**150.948**	140.000	**141.683**	9.533	14.764	**9.214**	0.202	**0.204**	17.700	17.982	**18.072**
45	194.924	**196.004**	183.182	**184.756**	11.742	20.274	**11.179**	0.225	**0.227**	19.300	20.135	**19.694**
50	242.052	**242.671**	228.764	**229.491**	13.288	26.268	**13.091**	0.244	**0.247**	20.900	22.083	**21.020**
55	297.925	**297.885**	282.453	**282.543**	15.472	33.677	**15.232**	0.265	**0.268**	22.200	24.129	**22.291**
60	363.617	**362.685**	345.851	**344.935**	17.766	42.742	**17.615**	0.287	**0.290**	23.500	26.270	**23.510**
65	444.945	**442.773**	424.482	**422.198**	20.463	54.415	**20.409**	0.310	**0.314**	24.600	28.632	**24.746**
70	509.618	**506.375**	486.693	**483.652**	22.925	64.007	**22.532**	0.328	**0.332**	25.600	30.339	**25.577**
75	585.447	**580.870**	560.231	**555.720**	25.216	75.560	**24.931**	0.347	**0.350**	26.500	32.189	**26.426**
80	669.510	**663.373**	641.864	**635.629**	27.646	88.715	**27.494**	0.367	**0.369**	27.200	34.085	**27.247**
85	740.404	**732.894**	710.774	**703.030**	29.630	100.066	**29.588**	0.383	**0.384**	27.900	35.580	**27.863**
90	827.505	**818.245**	795.264	**785.850**	32.240	114.306	**32.088**	0.402	**0.401**	28.500	37.309	**28.544**
95	912.695	**901.669**	877.930	**866.866**	34.766	128.522	**34.466**	0.419	**0.417**	29.000	38.902	**29.143**

the transition from using Equation 2.13a to 2.13b also served as a trigger to transition from Equation 2.14a to 2.14b.

Using these formulae and variables, it was possible to model the growth of a single *A. alba* tree using a Microsoft Excel spreadsheet (Table 2.2). Initial tests of the basic formulae (Equations 2.9, 2.11, 2.13, 2.14, 2.19 and 2.21 using the constants and exponents from Figures 2.7, 2.8, 2.11-2.14, and making transitions in Equations 2.13 and 2.14 as described above) showed an excellent correspondence with empirical data for the *A. alba* population reported in Cannell (1982). The initial tests only looked at time periods for which Cannell reported data, but did not allow for reiterative growth from year to year. Later tests included basic photosynthetic calculations for single plants. The final, fully functional version of Vida, which is able to simulate the growth of a single tree, groups of trees, and growth of mixed species, is discussed within chapters 3 and 4.

REFERENCES

CANNELL, M. G. R. 1982. *World Forest Biomass and Primary Production Data.* Academic Press, London, UK.

CANTIANA, M. 1974. Experimental research into dendrometry and auxometry. Part II. Initial enquires concerning the biomass of the white fir. [in Italian] *Bulletin 5. Instit. Assestamento Forestale*. Univ. of Firenze, Italy.

Centers for Disease Control and Prevention, National Center for Health Statistics. 30 May, 2000. CDC growth charts: United States. http://www.cdc.gov/growthcharts/.

DEUSSEN, O., P. HANRAHAN, B. LINTERMANN, R. MECH, M. PHARR, AND P. PRUSINKIEWICZ. 1998. Realistic Modeling and Rendering of Plant Ecosystems. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 275-286.

ENQUIST, B. J. AND K. J. NIKLAS. 2001a. Invariant Scaling Relations Across Tree-Dominated Communities. *Nature*. 410: 655-660.

ENQUIST, B. J. AND K. J. NIKLAS. 2001b. Supplementary Information for "Invariant Scaling Relations Across Tree-Dominated Communities." *Nature*. 410: 655-660.

GOUDWAARD, L. 2006. Tree factsheet: *Abies alba*. Wageningen University Forest Ecology and Forest Management Group.

GREENHILL, A. G. 1881. Determination of the greatest height consistent with stability that a vertical pole or mast can be made, and of the greatest height to which a tree or green proportions can grow. *Proceedings of the Cambridge Philosophical Society.* 4: 65-73.

HASEGAWA, Y., Y. YAMADA, M. TAMURA, AND Y. NOMURA. 1991. Monte Carlo simulation of light transmission through living tissues. *Applied Optics.* 30(31):4515-4520.

HAWKES, C. 2000. Woody Plant Mortality Algorithms: Description, Problems and Progress. *Ecological Modelling*. 126: 225-248.

HUXLEY, J. S. 1932. *Problems of relative growth.* The Dial Press, New York.

JENKINS, J. C, D. C. CHOJNACKY, L. S. HEATH, AND R. A. BIRDSEV. *Comprehensive Database of Diameter-based Biomass Regressions for North American Tree Species.* USDA Forest Service.

LINTERMANN, B. AND O. DEUSSEN. 1998. A Modelling Method and User Interface for Creating Plants. *Computer Graphics Forum*. 17(1): 73-82.

MANDELBROT, B. B. 1982. *The Fractal Geometry of Nature.* W.H. Freeman and Company.

MARTELLI, A., A. M. RAVENSCROFT, AND D. ASCHER. 2005. *Python Cookbook, Second Edition*. O'Reilly Media, Inc.

NAVAR, J. 2009. Allometric equations for tree species and carbon stocks for forests of northwestern Mexico. *Forest Ecology and Management*. 257: 427-434.

NEUBURG, M. 1999. *RealBasic: The Definitive Guide*. O' Reilly and Associates, Inc.

NIINEMETS, Ü. AND F. VALADARES. 2006. Tolerance to shade, drought, and waterlogging of temperate Northern Hemisphere trees and shrubs. *Ecological Monographs.* 76: 521-547.

NIKLAS, K. J. 1994. *Plant Allometry: The Scaling of Form and Process.* University of Chicago Press, Chicago.

NIKLAS, K. J. 2004. Plant allometry: is there a grand unifying theory? *Biological Reviews*.79: 871-889.

PRAHL, S. A., M. KEIJZER, S. L. JACQUES, AND A. J. WELCH. 1989. A Monte Carlo Model of Light Propagation in Tissue. *SPIE Institue Series* IS 5: 102-111.

PRINGLE, J., AND A. LANDCASTER. 2001. Unpublished code for PLANT.

RICHTER, J. P. (ed) 1939. The Literary Works of Leonardo da Vinci, Volume 1. Oxford University Press.

SHINOZAKI, K., K. YODA, K. HOZUMI, AND T. KIRA. 1964a. A Quantitative Analysis of Plant Form: The Pipe Model Theory. I. Basic Analysis. *Japanese Journal of Ecology.* 14(3): 97-105.

SHINOZAKI, K., K. YODA, K. HOZUMI, AND T. KIRA 1964b. A Quantitative Analysis of Plant Form: The Pipe Model Theory. II. Further Evidence of the Theory and its Application in Forest Ecology. *Japanese Journal of Ecology*. 14(3): 133-139.

TANENBAUM, A. S. 1992. *Modern Operating Systems*. Prentice-Hall, Inc.

VAN VALEN, L. 1973. A New Evolutionary Law. *Evolutionary. Theory*. 1: 1-30.

WALL, L., T. CHRISTIANSEN, AND J. ORWANT. 2000. *Programming Perl: Third Editon*. O'Reilly and Associates, Inc.

WARTON, D. I., AND N. C. WEBER. 2002. Common slope tests for bivariate errors-in-variables. *Biometry Journal.* 44: 161 – 174.

WARTON, D. I., I. J. WRIGHT, D. S. FALSTER, AND M. WESTOBY. 2006. Bivariate line-fitting methods for allometry. *Biological Reviews.* 81: 259 – 291.

WILSON, B. C. AND G. ADAM. 1983. A Monte Carlo model for the absorption and flux distributions of light in tissue. *Medical Physics.* 10(6):824-830.

WOLF, H. 2003. EUFORGEN Technical Guidelines for Genetic Conservation and Use for Silver Fir (*Abies alba*). International Plant Genetic Resource Institute, Rome, Italy.

ZIANIZ, D., P. MUUKKONEN, R. MÄKIPÄÄ, AND M. MENCUCCINI. 2005. Biomass and Stem Volume Equations for Tree Species in Europe. *Silva Fennica Monographs 4*

Chapter Three

Emergent properties of plants competing *in silico* for space and light: Seeing the Tree from the Forest[8]

*A spatially explicit, reiterative algorithm named Vida[9] is presented and used to predict multiple aspects of plant population and community dynamics. Using simple physical principles and empirically derived relationships, Vida provides an analytical venue to test alternative hypotheses about individual functional traits governing ecological or evolutionary processes at the population or community level of complexity. Analyses show that, as a result of competition for light and space, individual-level features scale up to produce species ensemble properties such as the scaling of self-thinning, size-dependent mortality, realistic size-frequency distributions, and a broad spectrum of empirically observed relationships for the species examined (*Abies alba*). Vida also predicts the competitive exclusion of conifers by angiosperms and the age at which reproductive maturity is achieved by different species. Vida serves as a null hypothesis by demonstrating that biologically complex phenomena, including widely observed species ensemble-level scaling relationships, can emerge from the operation of simple and transparent "rules" governing competition for space and light.*

The growth and development of an individual plant, and the changes that occur

[8] Originally appeared in the August, 2009 edition of *American Journal of Botany*: Hammond, S. T., Niklas, K. J. 2009. Emergent properties of plants competing *in silico* for space and light: Seeing the tree from the forest. *American Journal of Botany*. 96: 1430-1444.

[9] In the original publication, this software was named SERA, an acronym for Spatially Explicit Reiterative Algorithm.

within a plant population or community (a species ensemble), are influenced by similar ecological factors, such as nutrient availability and habitat stability, but they differ in their spatial and time scale (Harper, 1982; Tilman, 1982, 1990; Bolker et al., 2003; Busing et al., 2004). An individual plant may grow under relatively stable local environmental conditions and live for decades, whereas the ensemble of which it is a part may experience very different environmental conditions during its hundreds or thousands of years of existence. Nevertheless, a long sought after goal in plant ecology has been to find ways to predict the behavior of plant ensembles based on the observed properties of a relatively few representative individual plants (e.g., Smith and Smith, 1983; Lavorel and Garnier, 2002; Jenkins et al., 2004). Because body size affects the performance of virtually every biological function, one approach has been to study and quantify allometric phenomena to predict the behavior or attributes of plant or animal species ensembles (Blum, 1977; Economos, 1979; McMahon, 1973, 1980; Niklas, 1994, 2004; West et al., 1997; Enquist et al., 1998, 2007; Ernest et al., 2003). This approach has met with some success in that recent studies have shown that plant populations and communities exhibit many of the same size- and age-dependent trends, despite differences in species composition or growing conditions. For example, across ecologically diverse communities, the mass of tree canopies is reported to scale as the 2.0 power of basal stem diameter (Enquist and Niklas, 2001; Niklas, 2004). Likewise, a great variety of plant ensembles are observed to "self-thin" in accord with the same or very similar scaling exponents (Enquist et al., 1998), whereas tree size-frequency distributes appear to be governed by a –2.0 scaling relationship (Niklas et al., 2003).

However, there are grounds for confusion (and therefore obfuscation) between what is meant by a scaling relationship and what is perceived as an ecological process or pattern. In a strictly mathematical sense, a scaling relationship is any relationship between two variables of interest (Y_2 and Y_1) that can be mathematically described by a

slope (scaling exponent, a) that quantifies the change in one variable of interest (Y_2) with respect to the change in another variable of interest (Y_1) provided that the slope (a = Y_2/ Y_1) is statistically (and biologically) meaningful. By this definition, most quantifiable ecological processes and patterns involving two or more variables of interest can be expressed as one or more scaling relationships. Thus, ecological trends and patterns such as numbers of individual species per area ("species-area" relationships), numbers of individuals per area ("species abundance" curves), individual plant size versus number of conspecifics of equivalent size ("size frequency" distributions) as well as a host of other allometric phenomena are scaling relationships. This perspective may not be familiar because it crosses two, as of yet, very different disciplinary traditions. However, in the context of ecological observation and modeling, it has considerable utilitarian worth.

One of the most widely cited theories to explain the origin of many scaling relationships is that of West, Brown, and Enquist (West et al., 1997), denoted here as the WEB theory. This theory focuses on the individual organism and posits the existence of an internal, fractal-like transport and delivery system that has evolved to optimize the time and energy required to distribute materials throughout the organism (West et al., 1997). This theory has made numerous predictions, spanning animal and plant communities, and has been elaborated in numerous ways to provide metabolic optimization scenarios for plant communities (Enquist et al., 1998, 2007, 2009; Ernest et al., 2003). Despite its many successes, this theory and all of its more recent variants are based on a number of biological and physical assumptions that have been criticized on theoretical as well as empirical grounds (Dodds et al., 2001; Kozlowski and Konarzewski, 2004; Makarieva et al., 2008). To the best of our knowledge, no one has unequivocally shown that simple biophysical principles, acting at the level of an individual plant, scale up *a priori* to establish the properties typically observed for real

plant populations or communities. Nor has anyone shown that the properties of real plant populations or communities are the direct result of the size-dependent characteristics of their representative individual plants or species.

The absence of any such demonstration cannot be taken as proof that the WEB theory is fatally flawed. Attempts to demonstrate that it is the attributes of the individual ultimately lead to community-level relationships, even for very simple ecological systems, are confounded by the tremendous spatiotemporal heterogeneity that typically characterizes real environments and by the often dramatic physiological, morphological, and reproductive differences that set one species apart from another. The fact that the WEB theory has successfully predicted aspects of community dynamics argues that it cannot be easily dismissed, and that hierarchical population and community interactions may be the direct consequence of the properties of the individual.

In this context, a variety of empirical-field and theoretical-modeling approaches have been used to resolve whether "canonical" scaling relationships do in fact exist (West et al., 1997; Ernest et al., 2003; Enquist et al., 2007) in the face of well documented species-specific variation in response to different environmental conditions (Bolker et al., 2003; Diez and Pulliam, 2007; Lichstein et al., 2007). The empirical-field approach uses detailed measurements taken over ecological time frames to evaluate how species grow, interact, and respond to abiotic and biotic factors operating at multiple spatial and temporal scales. The theoretical-modeling approach uses mathematical and computational tools to predict relationships when abiotic and biotic factors are too complex to dissect experimentally, or when relationships operate in time frames that preclude direct experimentation. Each of these approaches has its strengths and drawbacks. Empirical-field studies are the most direct, but they can require impractically large expenditures of time and effort. Theoretical approaches can provide

detailed predictions about complex phenomena that may comply reassuringly well with observations, although sometimes for the wrong reasons (for example, Ptolemy of Alexandria's *Almagest* accurately predicted the movements of the stars and the known planets despite being a geocentric model). A balanced juxtaposition of both the empirical and theoretical approaches is advisable if long-standing debates are to be resolved about the height-structured competition for light, the relative roles of niche-based versus stochastic processes affecting species distributions, community composition and stability, and other ecologically and evolutionarily important aspects of plant ensemble dynamics (Tilman, 1982; Purves and Pacala, 2008).

Here, we use a new computer model called Vida to explore whether population or community attributes, such as self-thinning and competitive displacement of one species by another, are the direct consequence of the characteristics of their individual plant components. Vida was designed to evaluate how different species compete for light and space in a world-space whose physical attributes, such as the direction and intensity of incident sunlight, are clearly defined. Because each variable required to parameterize a species can be varied and manipulated individually, Vida provides a venue for running controlled experiments (individual computer simulations) to assess the influence of each variable on the dynamical behavior of a population or community. Vida therefore allows us to explore the consequences of competition for light and space as a population or community grows *in silico*, under rigidly specified conditions in an environment that can be made homogeneous or heterogeneous depending on the type of computer simulations required.

Clearly, any software attempting to accurately model the growth of individual plants must successfully mimic the detailed behavior of real populations and communities, show that its outputs are consistent and reproducible for each starting condition, and must demonstrate that any purported "emergent property" is not an artifact of circular

(computational) reasoning. These challenges dictated the organization of this paper, wherein we analyze the extent to which Vida successfully emulates the detailed dynamic and allometric behavior of real and generalized angiosperm and conifer populations and communities, assess the ability of our algorithm to give reproducible results for each set of input parameters, and explore whether canonical scaling relationships (*sensu* West et al., 1997; Ernest et al., 2003) emerge when single- or mixed-species ensembles compete for light and space. Throughout our presentation of Vida, we fully acknowledge that real plant populations and communities differ in their species composition and that ecosystems function in spatiotemporally heterogeneous environments whereas the world-spaces simulated by Vida for this paper are homogeneous. In this sense, Vida provides a null hypothesis about plant ensemble dynamics in the absence of a heterogeneous environment. The results of our computer simulations are therefore discussed exclusively in the context of competition for light and space *in silico* as a null hypothesis for assessing competing theories purporting to explain the behavior of the real world of plants.

MATERIALS AND METHODS

The model—

Vida contains approximately 2400 lines of code written in Python. A full version of Vida used in this study is provided at http://www.botany.org/downloads/HammondNiklas.zip; the complete source code is available upon request. Each Vida simulation involves arguments entered via a command line. Species and world-space specification files are loaded at the start of each computer run (Figure 3.1). Vida operates for a specified number of iterations, or until a target population or community size is reached. As output, it provides a number of specified file types to quantify the behavior of a population or community, including

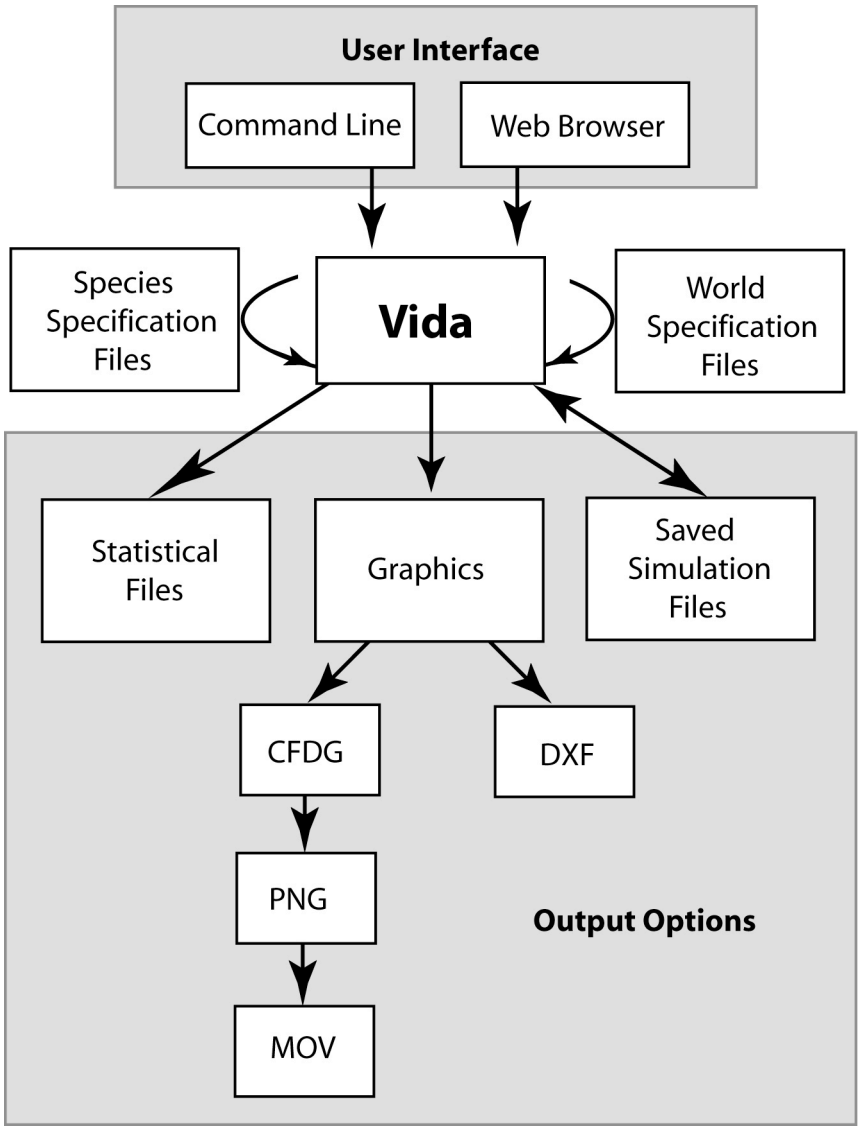


Figure 3.1: User interface and output options for Vida. World specification files define the size and boundaries of the area that can be occupied by a species ensemble. Species specification files numerically parameterize specific attributes (see Table 3.1).

CSV files giving the size, plant proportions, and condition of each object (propagule or plant). The state of any iteration can be saved and used to resume a computer run with the same or different species-specific parameters. Data stream outputs are also provided for subsequent statistical analyses (e.g., mean values and regression analyses of the variables of interest). Graphical options include saving files as an AutoCAD DXF to visualize the 3-D appearance of a simulated community in any iteration, or as a Context Free Design Grammar file, which can then be converted to a PNG image using the CFDG binary application, showing polar and lateral simulated world views. The generation of videos from PNG images is currently restricted to OSX environments as Vida embeds Applescript code to automate Quicktime Player's assembly of movies from sequential files (Figure 3.1).

Vida is conceptually similar to other computer models, notably those of Chave (1999) and Niklas (2000; Enquist and Niklas 2001), in that each plant is intentionally simplified to consist of a single photosynthetic surface (canopy) elevated by a single stem. However, Vida differs from previously published models in at least three important ways (Figure 3.2). First, in contrast to the Niklas (2000; Enquist and Niklas 2001) model, Vida requires fewer input variables among which only six scaling exponents are required to initialize a computer run (Table 3.1). Of these six exponents, one is necessary only if a species undergoes a change in the allocation of biomass upon reaching sexual maturity. Second, Vida is also an individual-based algorithm in that species identification files define only the properties of individual plants belonging to that species. The resulting mathematical transparency allows a user to quickly identify whether an output is the direct *a priori* result of using specific numerical values to characterize individual plants, or a consequence to individual-individual interactions. Third, where some models represent the canopy as a flat disc (e.g., Chave, 1999; Enquist et al., 2001), Vida treats each canopy as a hemisphere of uniform thickness.

Table 3.1: Species-specific parameters and representative values used to initialize Vida computations. Numerical values of all six scaling exponents shown in bold. Units for each parameter indicated at right of numerical values; o/o indicates a dimensionless parameter. Units for variables in each formula: G_T (kg/yr), M_T (kg), G_S (kg/yr), G_L (kg/yr), M_L (kg), M_S (kg), D_S (m), and H_S (m). World-space = 100 m x 100 m.

Species-specific parameter (formula in which it appears)	*Abies alba*	Generalized Conifer	Generalized Angiosperm	
1. Number of seeds (n, initial seed number)	25000	25000	25000	o/o
2. Canopy transmittance (T_C, fraction of light through canopy)	0.02	0.02	0.02	o/o
3. Survival transmittance (T_S, fraction of light needed for survival)	0.2	0.2	0.2	o/o
4. Maximum leaf thickness (H_C)	0.0003	0.0003	0.0003	m
5. Photosynthesis growth exponent (α_9 in $G_T = E\beta_{12} A_L M_L^{\alpha_9}$)	**-0.46**	**-0.46**	**-0.45**	o/o
6. Photosynthesis growth constant (β_{12} in $G_T = E\beta_{12} A_L M_L^{\alpha_9}$)	1.53	0.58	1.50	m^{-2} yr^{-1}
7. Growth memory (m, years prior growth rates are remembered)	2.0	2.0	2.0	yr
8. Young canopy mass exponent (α_5 in $M_L = \beta_5 M_S^{\alpha_5}$)	**1.2**	**0.97**	**0.88**	o/o
9. Young canopy mass constant (β_5 in $M_L = \beta_5 M_S^{\alpha_5}$)	0.04	0.29	0.12	o/o
10. Mature canopy mass exponent (α_8 in $M_L = \beta_{11} M_S^{\alpha_8}$)	**0.73**	**0.71**	**0.77**	o/o
11. Mature canopy mass constant (β_{11} in $M_L = \beta_{11} M_S^{\alpha_8}$)	0.25	0.39	0.09	o/o
12. Stem mass exponent (α_4 in $M_S = \beta_4 M_T^{\alpha_4}$)	**1.0**	**1.0**	**1.0**	o/o
13. Stem mass constant (β_4 in $M_S = \beta_4 M_T^{\alpha_4}$)	0.94	0.72	0.86	o/o
14. Stem diameter exponent (α_6 in $D_S = \beta_6 M_S^{\alpha_6}$)	**0.39**	**0.40**	**0.38**	o/o
15. Stem diameter constant (β_6 in $D_S = \beta_6 M_S^{\alpha_6}$)	0.03	0.03	0.03	m/kg
16. Young stem height exponent (α_7 in $H_S = \beta_7 D_S^{\alpha_7} - \beta_8$)	**1.1**	**0.93**	**1.1**	o/o
17. Young stem height constant (β_7 in $H_S = \beta_7 D_S^{\alpha_7} - \beta_8$)	100	80.3	161	o/o
18. Young stem height constant (β_8 in $H_S = \beta_7 D_S^{\alpha_7} - \beta_8$)	0.0	0.0	0.0	m
19. Mature stem height exponent (β_9 in $H_S = \beta_9 + \beta_{10} \ln D_S$)	42.7	30.1	31.5	m
20. Mature stem height exponent (β_{10} in $H_S = \beta_9 + \beta_{10} \ln D_S$)	15.5	8.49	7.71	m

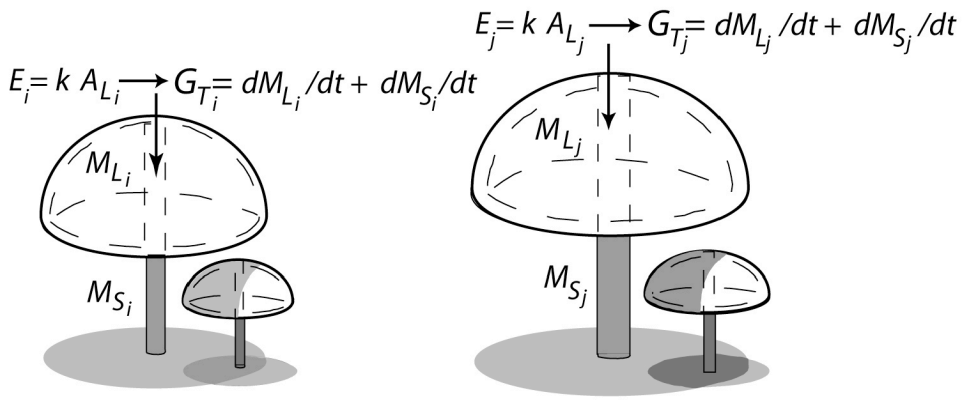
This difference is very important because a canopy described as a flat disk will cast a significantly larger projected area than a hemispherical canopy with an equivalent mass and thickness. A flat disk canopy model, therefore, over –estimates the ability of a simulated plant to capture light and shade its neighbors. It also over-estimates the space that must be occupied to capture a fixed quantity of light. While contributing to more realistic measures of a plant's ability to harvest light and to shade individuals below its canopy, we also chose hemispherical canopies for our prototype because they can be relatively easily modified in future versions of Vida to describe prolate or oblate spheroid canopy geometries, or even cones.

As in the Chave (1999) and Niklas (2000) models, the angle of solar incidence is time-averaged to be 0º (i.e., light comes from directly above each plant) such that the photosynthetic area available for each plant is the projected area of the canopy (A_L). This stipulation can be relaxed to cope with daily changes in the solar angle or differences in latitude. However, one of our objectives was to simplify the real world by reducing the number of variables that can influence the outcome of our computer runs. By eliminating latitudinal and diurnal differences in the solar angle, the results of computer runs can be assumed to be due to plant-plant interactions and not due to variations in ecologically important variables such as total solar energy.

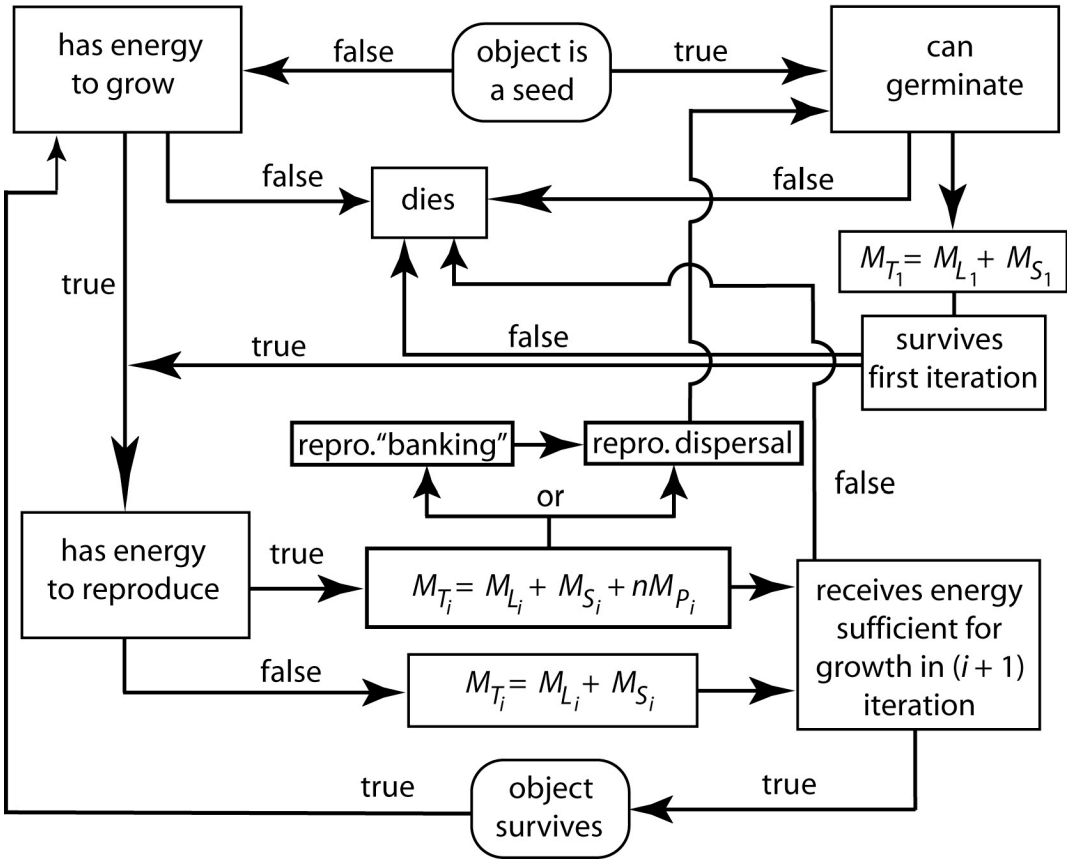
The size of the canopy and the extent to which it is shaded by neighboring plants dictate the ability of the individual to harvest light (E) and thus grow annually (Figure 3.2A). The light energy harvested defines net biomass accumulation ("annual growth"), which is allocated to the construction of new canopy, stem biomass and propagules. Total annual growth (G_T) equals the sum of the change in leaf mass per year (dM_L/dt) and the change in stem mass per year (dM_S/dt). The conversion of E into G_T uses the formula $G_T = E\beta_{12} A_L M_L^{\alpha_9}$, where β_{12} and α_9 denote a species-specific normalization (allometric) constant and scaling exponent, respectively, and A_L is the projected canopy

Figure 3.2: Plant schematics and flow chart for Vida computational logic. A. Plant canopy mass (M_L) and projected area (A_L) determine the energy (E) captured and converted into annual growth (G_T) that is subsequently allotted to new M_L and stem mass (M_S). Partial shading reduces the effective A_L and thus G_T. B. An object's status as a propagule or non- propagule (plant) is assessed at each iteration. If a propagule successfully germinates, its mass is allocated to the construction of seedling M_L and M_S. If the object is an established plant that has sufficient energy to continue growing, it is evaluated in the next iteration for vegetative and reproductive status (total reproductive mass equals propagule number n times individual propagule mass M_P).

A


$E_i = k\ A_{L_i} \rightarrow G_{T_i} = dM_{L_i}/dt + dM_{S_i}/dt$
M_{L_i}
M_{S_i}
$E_j = k\ A_{L_j} \rightarrow G_{T_j} = dM_{L_j}/dt + dM_{S_j}/dt$
M_{L_j}
M_{S_j}

B


has energy to grow
false
object is a seed
true
can germinate
false
dies
false
$M_{T_1} = M_{L_1} + M_{S_1}$
true
true
false
survives first iteration
repro."banking"
repro. dispersal
or
false
has energy to reproduce
true
$M_{T_i} = M_{L_i} + M_{S_i} + nM_{P_i}$
receives energy sufficient for growth in $(i+1)$ iteration
false
$M_{T_i} = M_{L_i} + M_{S_i}$
true
object survives
true

area (Figure 3.2B). In turn, the allocation of dM_S/dt to stem diameter and height (D_S and H_S, respectively) is governed by relationships that permit (but do not demand) a transition from stem geometric self-similarity to geometric non-similarity (i.e., $H_S = \beta_7 D_S^{\alpha_7} - \beta_8 \rightarrow H_S = \beta_9 + \beta_{10} \ln D_S$) depending on the species (input variables 16 – 20, see Table 3.1 and Appendix 1). This optional transition is predicated on the empirical observation that the stems of juvenile woody plants contain little or no secondary tissues, whereas wood steadily accumulates in stems as plants age (see Niklas, 1994, 2004). Depending on growth in previous years, input variables provide the option of using some portion of the new biomass gained per year to construct propagules (differing in number n and individual mass M_P among species), or of "banking" biomass for reproduction in one or more future growth cycles, i.e., $G_T = \beta_3 nM_P$ or $\Sigma(G_T/\beta_4) = nM_P$. Propagules can be randomly or non-randomly distributed in the world space depending on the characteristics of the species being modeled. The distribution of M_T to M_L and M_S immediately upon "germination" is species-specific. Mortality can result from one or more processes or events (see Figure 3.2B).

The default setting for the world-space in Vida is 100 m x 100 m; all biological units conform to SIU standards (see Tables 3.1 – 3.3). The numerical values that determine propagule production and dispersal, plant size, growth, and biomass allocation patterns however are species-specific (Table 3.1). Each value can be assigned *a priori* to evaluate the effects of different input variables on ensemble dynamics, or it can be determined empirically using data reported for real species to diagnose the performance and biological fidelity of Vida's computational logic. Parameterization of computer runs that violate physical laws results in the death of plants (e.g., plants cannot exceed their critical buckling heights or canopy loads). Mortality also results from light deprivation (which is a function of the attenuation of light through over-topping canopies), or stochastic/age-dependent processes (Figure

Table 3.2: Comparisons between observed and predicted exponents (α and 95% CIs) for observed and predicted scaling relationships of *A. alba* and generalized conifer and angiosperm populations. Units: G_T (kg/yr), M_T (kg), G_S (kg/yr), G_L (kg/yr), M_L (kg), M_S (kg), D_S (m), H_S (m).

	Abies alba		Generalized Conifer		Generalized Angiosperm	
Scaling relationship	Observed	Predicted	Observed	Predicted	Observed	Predicted
G_T vs. M_T	0.67 (0.65, 0.68)	0.66 (0.65, 0.67)	0.80 (0.75, 0.86)	0.80 (0.78, 0.82)	0.76 (0.72, 0.81)	0.75 (0.73, 0.77)
G_S vs. D	1.18 (1.10, 1.27)	1.18 (1.08, 1.28)	2.04 (1.98, 2.11)	2.05 (2.04, 2.06)	2.04 (1.93, 2.15)	2.02 (2.00, 2.04)
G_L vs. D	0.72 (0.58, 0.87)	0.64 (0.19, 1.09)	1.95 (1.87, 1.96)	1.95 (1.93, 1.97)	1.79 (1.75, 1.80)	1.80 (1.79, 1.81)
G_T vs. M_L	0.60 (0.58, 0.61)	0.57 (0.56, 0.58)	1.00 (0.98, 1.03)	1.00 (0.99, 1.01)	1.15 (1.07, 1.23)	1.15 (1.13, 1.18)
M_L vs. M_S	0.99 (0.93, 1.05)	0.95 (0.94, 0.96)	0.78 (0.74, 0.81)	0.78 (0.77, 0.79)	0.73 (0.71, 0.74)	0.74 (0.73, 0.76)
M_L vs. D	1.91 (1.89, 1.94)	2.03 (1.86, 2.21)	1.93 (1.85, 2.02)	1.98 (1.96, 2.00)	1.86 (1.72, 2.00)	1.98 (1.96, 2.00)
M_S vs. D	2.54 (2.53, 2.55)	2.54 (2.53, 2.55)	2.48 (2.39, 2.56)	2.44 (2.42, 2.47)	2.63 (2.55, 2.70)	2.66 (2.62, 2.67)
M_S vs. H	2.98 (2.90, 3.06)	2.84 (2.81, 2.88)	2.59 (2.47, 2.71)	2.60 (2.59, 2.61)	3.12 (2.88, 3.35)	3.10 (3.08, 3.13)
H vs. D	0.85 (0.83, 0.88)	0.89 (0.79, 0.99)	0.93 (0.88, 0.97)	0.94 (0.93, 0.97)	0.83 (0.75, 0.91)	0.86 (0.80, 0.87)

Table 3.3: Comparisons of scaling exponents predicted for a *Abies alba* population (see Table 3.2) and a single plant (r^2 reported). With the exception of M_S vs. D_S, all scaling exponents differ statistically at $P > 0.05$ based on comparisons of 95% confidence intervals. Units: G_T (kg/yr), M_T (kg), G_S (kg/yr), G_L (kg/yr), M_L (in kg), M_S (in kg), D_S (in m), and H_S (in m). World-space = 100 m x 100 m.

	Scaling exponent (α)		
	Population	Single Plant	r^2
	Scaling relationship		
G_T vs. M_T	0.66	0.70	0.998
G_S vs. D_S	1.18	1.80	0.998
G_L vs. D_S	0.64	1.83	0.995
G_T vs. M_L	0.57	0.65	0.999
M_L vs. M_S	0.95	1.07	0.998
M_L vs. D_S	2.03	2.71	0.998
M_S vs. D_S	2.54	2.54	1.000
M_S vs. H_S	2.84	2.44	0.998
H vs. D_S	0.89	1.04	0.998

3.2B). Simulations are initiated with one or more "seeds" that grow into one or more populations (Figure 3.3).

Field Data used to parameterize computer runs —

The biological fidelity of Vida was assessed by quantitatively comparing predicted species ensemble behavior against that observed for species for which sufficient data could be gathered to initialize computer runs. The world-wide compendium for forestry data compiled by Cannell (1982) was used to survey the primary literature published before 1982 to identify long-term studies of monospecific populations reporting data for N (dimensionless, o/o), H_S (in m), D_S (in m), M_L (in kg), M_S (in kg), G_L (kg/yr), G_S (kg/yr), and nM_P (in kg). The most useful data set for any species was for a single *A. alba* population (with an initial density of 25000 plants/ha), which was documented every five years over a 95 year period, starting ten years after the initial planting (Cantiana, 1974; Hellrigl, 1974). No data were given for reproductive biomass (although trees reached reproductive maturity), nor were the cause(s) of mortality reported (other than the culling of trees for biometric measurements). Equivalently large data sets for other species could not be identified using this compendium, nor could we find any similarly useful or detailed data in the primary literature appearing after 1982.

The *A. alba* data set used in this paper was particularly attractive because it provided information for a population growing in a single habitat at a well defined latitude and elevation that was monitored for an exceptionally long time. Data for this species (or other species of interest) were available from other sources, but these additional data sets were drawn from different populations growing in very different locations and under different soil conditions, ambient light intensities, and many other important ecological variables known to affect plant survival, growth, and lifespan.

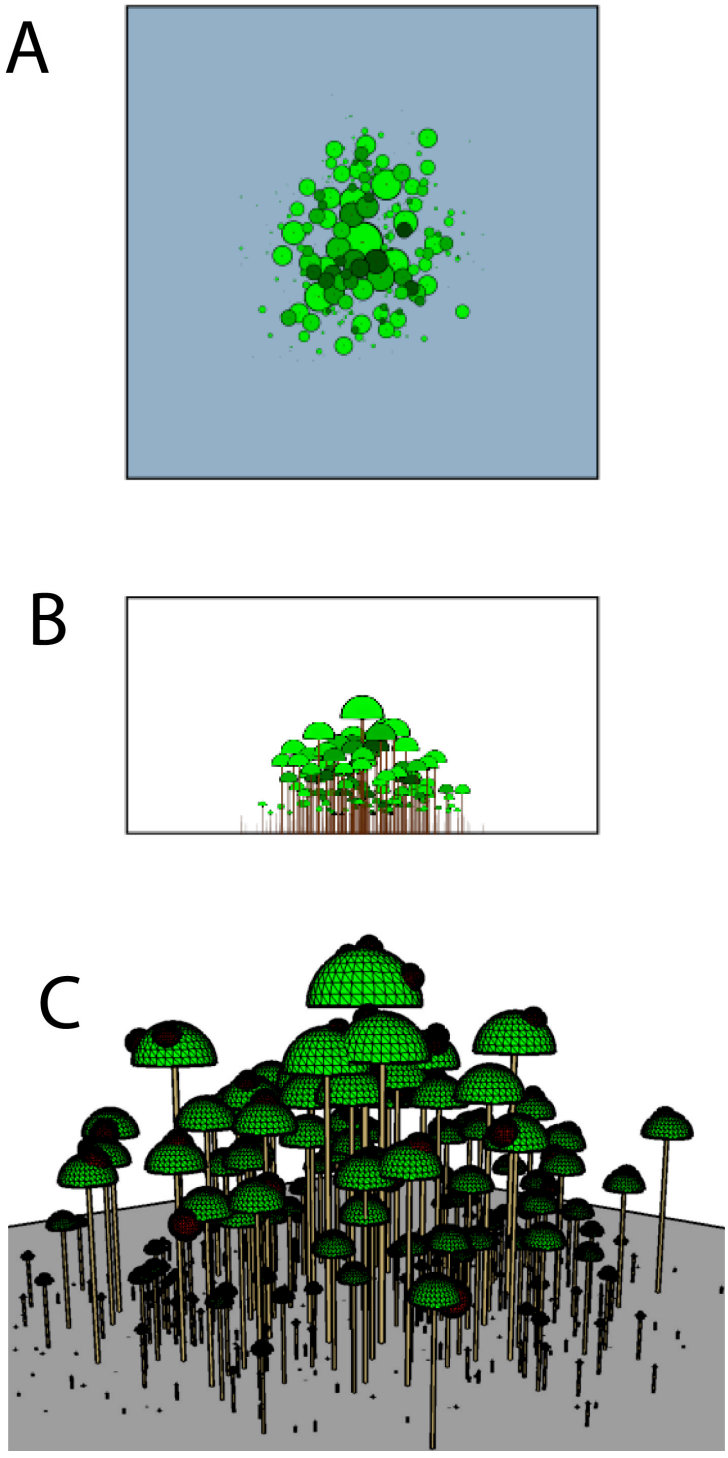


Figure 3.3: Polar, lateral, and inclined views (A, B, and C, respectively) of a Vida generated plant growing from one propagule. World-space = 100 m x 100 m.

Because most data for the *A. alba* population of choice were recorded at five year intervals starting at 10 years after the initial planting of the population, statistical comparisons between observed and predicted trends were confined to these discrete time intervals. Based on the performance of these computer runs, the compendium (Cannell, 1982) was used to statistically characterize 332 angiosperm and 343 conifer (tree and shrub) dominated communities to emulate the dynamics of "generalized" angiosperm and "generalized" conifer populations (Table 3.1). With the aid of these two generalized species, we explored the effects of competition for space and light on competitive exclusion.

Data analyses —

Standardized major axis (SMA, also known as reduced major axis) regression analyses of bivariate plots of plant height (H_S), trunk diameter (D_S), canopy and stem mass (M_L and M_S), and annual canopy and stem growth rates (G_L and G_S) were used to determine the numerical values of input variables using the generalized allometric formula $\log Y_2 = \log \beta \pm \alpha \log Y_1$, where Y_1 and Y_2 are interdependent variables (Niklas, 1994; Warton and Weber, 2002; Warton et al., 2006). SMA analyses were also used to assess whether empirical and predicted scaling exponents were statistically equivalent based on their 95% confidence intervals. The numerical values of input variables for which no data were available (e.g., probabilities of random mortality and canopy cross sectional areas) were estimated based on exploratory computer runs designed to assess how different values alter the survival and growth of hypothetical populations (differing from those evaluated in the present study). As will be shown, sensitivity analyses of simulation outputs indicated that critically important scaling exponents were numerically insensitive to the numerical values used to simulate mortality and canopy cross sectional areas (e.g., the scaling exponents for total growth vs. total mass and of canopy mass vs. stem diameter).

RESULTS

The extent to which Vida was successful at emulating the behavior of a real *A. alba* population or a hypothetical "generalized" angiosperm-conifer community was evaluated on the basis of the numerical agreement between observed and predicted scaling exponents (and their 95% confidence intervals; Tables 3.2 – 3.3). Predicted and observed data were also regressed against one another, and the extent to which they agreed was evaluated on the basis of ordinary least squares regression values for r^2. Sets of three to five computer runs were performed for each set of parameterizations. For example, the parameter values determined using data from the real *A. alba* population (Table 3.1) were used in three computer runs. The computer runs in each set of simulations were parameterized in exactly the same way. The only differences among the computer runs in each set of simulations resulted from the fact that each computer run was "seeded" randomly (but with the same number of propagules). Computer runs in which modeled the growth of individual plants were seeded with a single propagule placed in the center of the world-space, and runs modeling the *Abies alba* population used to test Vida's biological fidelity were seeded with 25,000 propagules, arranged in an hexagonal pattern.

Despite differences in the initial spatial distribution of propagules in each computer run, statistical analyses of Vida outputs from each set of three or more simulations indicated they were statistically indistinguishable. Nevertheless, the numerical values of outputs presented here are averages of each set of three or more simulations (Tables 3.2 – 3.4). Finally, it is important to note that, despite eight different ways plants can die in Vida simulations, the most prevalent cause of death in each computer run was light deprivation. Across all of the computer runs we performed, light deprivation accounted for 85% of all deaths, while 11% of all deaths resulted from propagules

produced by parent plants that were randomly dispersed outside the world-space. The remaining 4% of all deaths is the result of random (stochastic) or age-dependent deaths. Thus, the Vida outputs reported here are largely insensitive to how random mortality was parameterized, with the exception of the set of angiosperm vs. conifer competition simulations in which we intentionally increased the intensity of mortality to mimic cold-induced vascular embolisms in angiosperm stems (see below).

Vida predicts the behavior of real populations —

Based on tests for slope heterogeneity, 95% confidence intervals of scaling exponents and *y*-intercepts, and regression analyses of predicted versus observed data points, Vida successfully reproduced all observed age- and size-dependent trends reported for the real *A. alba* population, and for the generalized angiosperm and conifer species based on three or more computer runs parameterized in exactly the same way (Figures 3.4 -3.5; Tables 3.2-3.3). These trends included the scaling of canopy and stem growth rates (G_L and G_S vs. age; regression of predicted versus observed values gave $r^2 = 0.927$ and 0.989, respectively), age- and size-dependent changes in their respective mass (M_L and M_S vs. age; regression of predicted versus observed values gave $r^2 = 0.988$ and 0.978, respectively), and changes in stem diameter and height (D_S and H_S vs. age; regression of predicted against observed values gave $r^2 = 0.945$ and 0.998, respectively). Three computer runs imposing greater reproductive effort (i.e., a larger reproductive biomass commitment) predicted smaller values of leaf and stem growth (G_L and G_S) as well as smaller canopy and stem mass (M_L and M_S), a phenomenology that closely mimicked the behavior of the real *A. alba* population (Figure 3.4A – B).

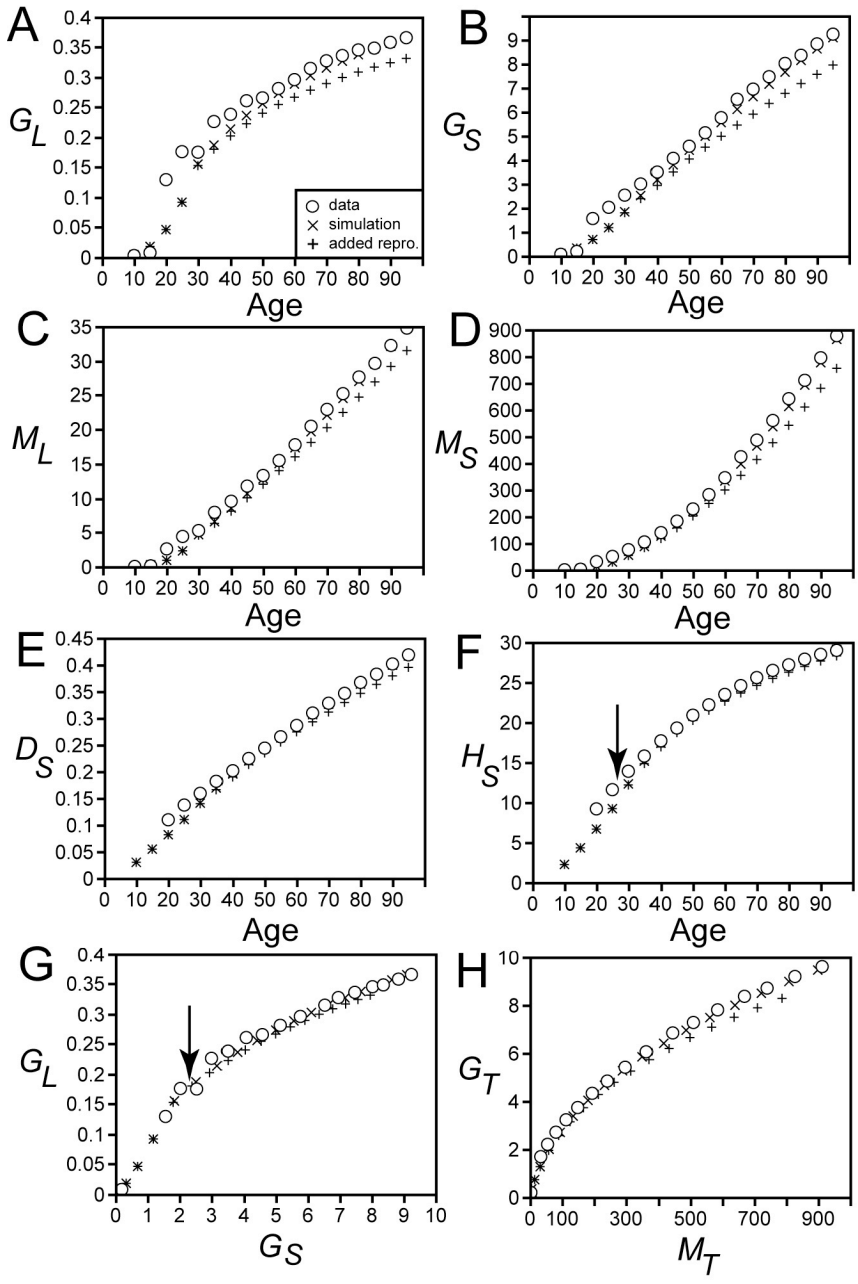


Figure 3.4: Bivariate comparisons among observed age-dependent trends (in 5 yr intervals) in a real *A. alba* population (denoted by o) and those predicted by Vida emulating normal reproductive effort and three times normal effort (denoted by x and +, respectively). Arrows in F and G show the predicted transition to reproductive maturity.

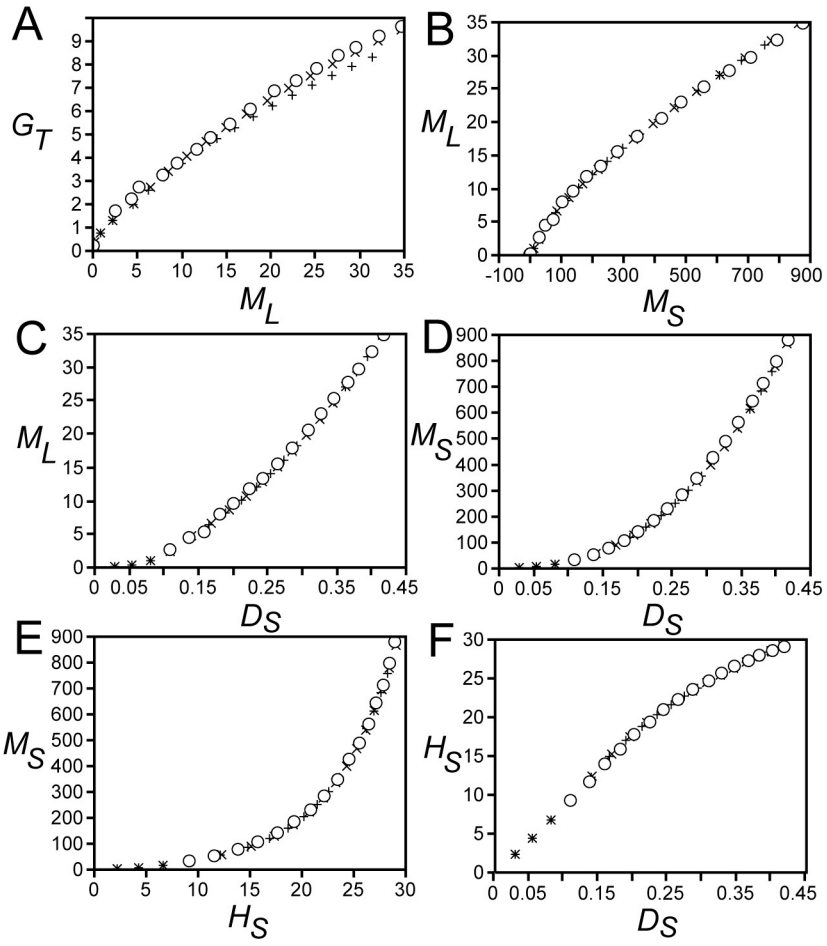


Figure 3.5: Bivariate comparisons among observed size-dependent trends (in 5 yr intervals) in an *A. alba* population (denoted by o) and those predicted by Vida emulating normal reproductive effort and three times normal effort (denoted by x and +, respectively).

Though not explicitly programmed to do so, computer runs simulating a single *A. alba* plant successfully predicted the age at which *A. alba* achieves sexual maturity based on an empirically observed juvenile-to-mature shift in the allometry of plant height H_S versus stem diameter D_S (i.e., $H_S = \beta_7 D_S^{\alpha_7} - \beta_8 \rightarrow H_S = \beta_9 + \beta_{10} \ln D_S$; see Table 3.1). Specifically, *A. alba* is reported to reach sexual maturity at 25 under ideal growing conditions (Wolf, 2003), whereas Vida predicted that individual plants would reach sexual maturity at 24 years of age (Table 3.3). In Vida's computational logic, the change in the scaling of plant height vs. stem diameter triggers a change in the allometry of leaf growth with respect to stem growth (G_L vs. G_S; Figure 3.4 E–G) that alters the subsequent ability of a plant to harvest light. Despite changes in the allometry height with respect to stem diameter, total vegetative growth ($G_T = G_L + G_S$) with respect to total aboveground body mass ($M_T = M_L + M_S$) showed no simultaneous perturbation either for individual plants or for the entire simulated or real *A. alba* population (Figure 3.4 H; regression of predicted versus observed values gave $r^2 = 0.971$). We concluded therefore that the shift in the allometry of H_S vs. D_S is an example of a global set of allometric shifts both for real and simulated *A. alba* plants.

Extensive testing of Vida's ability to accurately predict the onset of sexual maturity for other diverse species has not been performed. In part this is due to a lack of sufficient data for paired measurements of actual plant height versus stem diameter among very young individual trees, particularly of plants in their first and second year of growth during the shift from primary to secondary growth. However, preliminary results for *A. alba* and additional species show that Vida's ability to accurately predict the age of reproductive maturity increases in proportion to the number of data points available for plant height vs. stem diameter (Table 3.4).

Table 3.4: Predicted and observed age to reproductive maturity for eight species for which data for Vida computations were available and for a generalized angiosperm and conifer species (data from Cannell, 1982). Observed ages for reproductive maturity are from a variety of sources. The number of data points that were available to calculate the ontogenetic shift in plant height vs. stem diameter is indicated by *n*.

Taxon (*n*)	Predicted Age (yr)	Observed Age (yr)
Abies alba (18)	~24	25 – 35
Betula verrucosa (9)	~32	15
Cryptomeria japonica (7)	~10	15 – 20
Hevea braziliensis (11)	~2	5 – 6
Pinus radiata (8)	~5	5 – 10
Pinus sylvestris (12)	~9	10 – 15
Shorea robusta (12)	~7	15
Tectona grandis (6)	~8	8 – 10

Sensitivity analyses involving three to five separate computer runs, each using different numerical values for parameterization, showed that some population-level outputs are inextricably dependent on the numerical values used to initialize them, but that other key outputs are invariant regardless of how Vida was parameterized. For example, the scaling exponents for three input variables in the computer runs for the real *A. alba* population (i.e., $M_{L_{young}} \propto M_S^{\alpha_5=1.2}$, $M_{L_{mature}} \propto M_S^{\alpha_8=0.73}$, and $D_S \propto M_S^{\alpha_6=0.39}$; see input variables 8, 10, and 14 in Table 3.1, respectively) *a priori* predicted $M_L \propto D_S^{1.87}$, which fell well within the 95% confidence intervals of the exponent observed for this relationship (see Table 3.2). However, very different a-values for the same three relationships (e.g., $M_{L_{young}} \propto M_S^{\alpha_5=0.5}$, $M_{L_{mature}} \propto M_S^{\alpha_8=2.0}$, and $D_S \propto M_S^{\alpha_6=0.89}$) gave outputs that still predicted a $M_L \propto D_S^{a\approx 2.0}$ scaling relationship (Table 3.2), which has been reported for an ecologically diverse array of real populations (Enquist and Niklas, 2001; Niklas, 2004).

Likewise, the scaling exponents governing the relationship between average plant mass and standing plant density ("self-thinning"), size-frequency distributions, and lifespan versus body mass were successfully predicted by Vida despite the absence of input variables that affect these biological relationships in Vida's computational logic. Specifically, the average predicted exponent for *A. alba* self-thinning in three computer runs was – 1.79 (the observed value for the real population was $\alpha = -1.77$). When computer runs achieved plant density equilibrium ($N \approx$ constant), the exponents for log-transformed data for plant density (N) versus stem diameter averaged – 2.0, which is numerically indistinguishable from the exponent reported for real plant populations (Enquist et al., 1998, 2001; Niklas et al., 2003). Yet, there is nothing in the Vida algorithm that dictates size-frequency distributions other than the effects of competition for light on plant survival and size. Finally, all computer runs predicted that plant life-spans scale as the 1/4 power of body mass, a scaling relationship that has reported for

diverse plant species (Marbà et al., 2007).

Vida predicts the competitive displacement of conifers by angiosperms —

A classic observation in field ecology is that arborescent angiosperms tend to displace the majority of tree-sized conifer species at low elevations or low latitudes. In contrast, conifers tend to dominate forested communities at high elevations and latitudes, presumably because their xylem is less prone to embolisms caused by subfreezing temperatures (Harper, 1982; Carlquist, 2001). Therefore, in addition to testing whether Vida can mimic a "generalized" conifer population and a "generalized" angiosperm population, another test was to see if Vida predicts competitive displacement in a community initially consisting of equal numbers of a generalized conifer and angiosperm propagules. A third test was to see if conifers could become the dominant species if the mortality of angiosperms was increased to mimic vascular embolisms caused by freezing. The combination of data drawn from 332 angiosperm- and 343 conifer-dominated communities used to construct these two generalized species (Table 3.2) therefore spans a huge range of individual species attributes. It is important to recognize that each of these two "generalized" species reflects an amalgam of species, some of which are pioneer evergreen or deciduous shrub or tree species while others are shade tolerant species that are either rare or commonplace.

With this caveat in mind, Vida successfully simulated all scaling relationships observed for monospecific "generalized" populations (Table 3.2). These computer runs also manifested specific dynamical properties as did *A. alba* computer runs. For example, three computer runs of a generalized angiosperm population showed that self-thinning (i.e., $\overline{M_T}$ vs. N) was governed by $\alpha - 1.42$ scaling exponent, while the observed exponent across 332 angiosperm-dominated communities was – 1.43; the scaling exponent predicted for the conifer self-thinning was – 1.56, while the numerical value observed across the 343 conifer-dominated communities was – 1.57. Once

populations reach equilibrium, the predicted exponent for angiosperm total growth vs. body mass (i.e., G_T vs. M_T) was 0.75 (across species, the observed value was 0.74); the exponent predicted for conifer G_T vs. M_T was 0.79 (the observed value was 0.80). Regardless of how Vida was parameterized, canopy mass was predicted to scale roughly as the 2.0 power of D_S for both generalized species, as in the case of the real and simulated *A. alba* population. Regression of predicted versus observed numerical values for all of the aforementioned variables of interest gave $r^2 > 0.924$.

Three additional computer runs were performed in which equal numbers of "generalized" angiosperm and "generalized" conifer propagules were allowed to grow to reproductive maturity and compete for light and space. In each case, as plants increased in size and canopies progressively overtopped one another, conifers were relegated to smaller "nested" groupings that were eventually extinguished (Figure 3.6). The reduction in conifer density N was attributable in large part to angiosperm reproductive precocity and fecundity. Vida predicted that angiosperms produced a greater number of propagules and reached reproductive maturity approximately 5 years earlier than their conifer counterparts. However, conifers were displaced even when both generalized species were assigned equivalent reproductive effort and age at maturity. Despite this "level playing field", conifers were displaced by angiosperms due to light deprivation and reduced growth rates; stochastic death of conifers played a minimal role in influencing the fate of conspecifics (Figure 3.7). Conifers maintained a presence in simulated communities (and even gained dominance) as angiosperm (stochastic or age-dependent) mortality was increased to mimic the susceptibility of angiosperms to the formation of vascular tissue embolisms, or when the fraction of light required for conifer seedling survival was reduced (i.e., to mimic increased shade tolerance).

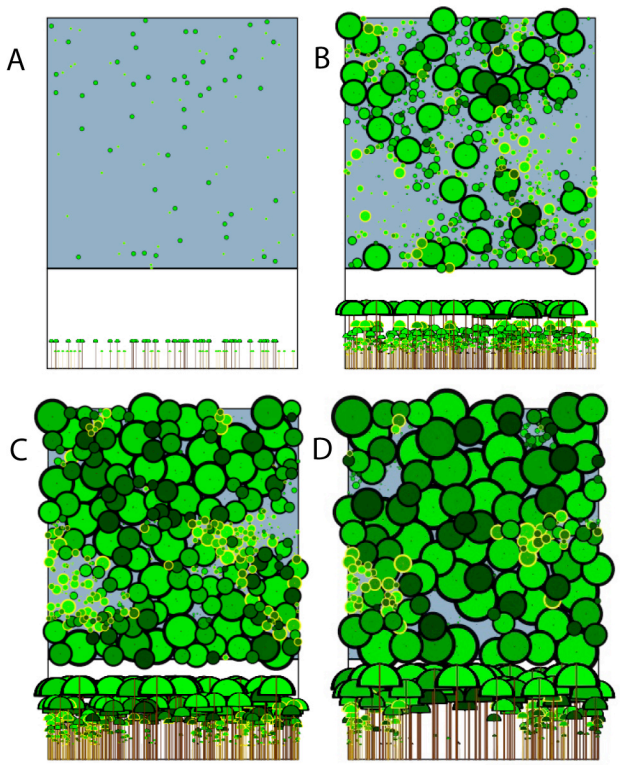


Figure 3.6: Four stages in the maturation of an angiosperm-conifer mixed community (initialized with equal numbers of propagules. Conifers indicated by canopies with yellow halos. A. Community at 15 years. B. Community at 50 years. C. Community at 75 years. D. Community at 100 years. World-space = 100 m x 100 m.

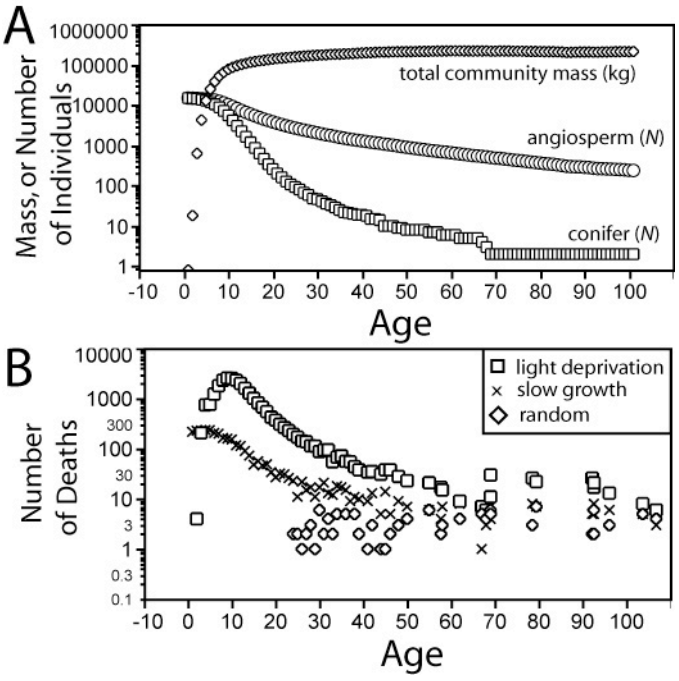


Figure 3.7: Angiosperm vs. conifer competitive displacement beginning with equal numbers of propagules (n = 50). A. Total community mass (in kg) and plant density (N) plotted against community age. B. Principal causes of mortality. World-space = 100 m x 100 m.

Simple rules obtain "emergent" (hierarchical) properties —

The most direct method to determine whether the scaling exponents observed for Vida computer outputs are the result of ensemble plant-plant dynamics was to compare the scaling exponents observed for an individual plant to those observed for a simulated species ensemble. This method was applied to Vida simulations of *A. alba* because they were based on an empirically robust data set (Table 3.4). Based on 95% confidence intervals, comparisons between the exponents observed for three computer runs of a single isolated plant and three computer runs of a population revealed only one case in which an exponent was numerically the result of an input variable, viz., the scaling exponent for stem mass versus stem diameter (Table 3.4). This single exception is the result of the physical constraints placed on stem proportions resulting from the Euler-Greenhill mechanical relationship, which Vida applies equally to all simulated plants. From this, we concluded that the scaling exponents predicted by Vida for the *A. alba* population and for hypothetical confer-angiosperm communities (Tables 3.2 – 3.3) are properties that emerge from hierarchical interactions among plants differing in size and not from the attributes of individual plants.

DISCUSSION

The goal of this paper is to see whether a simple algorithm (Vida) is capable of mimicking biological scaling relationships observed for real plant populations and communities directly from the constraints imposed by simple physical laws and principles that govern how functionally similar organisms compete for limited resources. Vida served as our null hypothesis by virtue of its simplification and redaction of the biological attributes of real plants, populations, or communities. Our analyses of Vida computer runs indicate that a number of naturally occurring scaling relationships emerge as a direct result of competition for light and space, even in a

simple homogeneous, abiotic world-space. Using only the attributes of an individual plant, Vida successfully replicated the numerical values of all the scaling exponents reported for a real *A. alba* population and for an angiosperm/conifer mix-species community, including those governing annual growth versus body mass, canopy mass versus stem mass, age at mortality versus body mass, size-frequency distributions, and the size-dependent self-thinning of real populations and a two-species community. We believe that population and community-level scaling exponents can be viewed as emergent properties and that they are the result of a propensity toward self-organization reminiscent of self-organized criticality, which characterizes many dynamical systems. If true, mathematically complex theories based on numerous assumptions that attempt to explain why a broad range of allometric phenomena exists and why they are indifferent to species compositions or a host of physical environmental variables, may be unwarranted. Before examining this conclusion in greater detail, however, it is necessary to define what is meant by "emergent properties," how they relate to self-organization, and why Vida simulations converge on specific "canonical" scaling exponents.

Like all mathematical models, Vida reveals nothing more than how it is numerically parameterized. A model that fails to do so is suspect, but it can also be accused of being trivial. In this respect, there are two polarized perspectives regarding the circularity of a model's computational logic. One perspective sees all outputs as the reducible outcome of parameterization and the mathematical structure of the algorithm driving its manipulation. This perspective *a priori* denies any model of having emergent properties in the sense that they are unexplained outcomes. The logic of this *emergere ex machina* perspective is undeniable, and, in this sense, emergent properties *sensu stricto* cannot exist. Yet, when dealing with algorithms such as Vida (as well as with real ecosystems) the phrase "emergent properties" has a different meaning, one that

rests on the distinction between the properties of an individual and those that characterize the behavior of a population or community of individuals (*in vitro* or *in vivo*). We believe that this *emergere ex civitas* perspective is defensible in the case of Vida because the dynamics we observe for artificial populations and communities cannot be deduced directly from (or reduced to) the properties of the individual organisms used to parameterize Vida computer runs.

The input variables in all Vida computer runs exclusively define the properties of an individual (plant or species) and the physics of the world-space in which the individual competes for light and space. The only factor driving plant-plant interactions (and thus population or community dynamics) is the availability of light and space, which affects the individual's vegetative growth, fecundity, and ultimately its death. Thus, the emergence of population- and community-wide scaling exponents that numerically agree with those observed for real populations and communities must involve some form of self-organization, viz. the emergence of robust and reproducible complexity that does not depend on how details of the system are finely-tuned. When and how this happens in Vida appears to be governed by plant growth in size rather than changes in plant density *per se*. When a population or community is randomly "seeded" to initialize a simulation (the seedling establishment phase), individuals rarely interact because they are widely spaced apart. When physically isolated from one another, individual plants establish allometric trends that differ dramatically from those observed at the level of a maturing population or community (see Table 3.3). However, as plants increase in size, interactions among neighboring plants increase in regularity and intensity because of shading by overlapping canopies. As these interactions intensify, they reach a critical point and dynamical properties — such as self-thinning and extensive mortality due to light deprivation — emerge as a result of cascades of death and birth differing in size and scale. This phenomenology, which is a hallmark of what

Bak et al. (1988) call self-organized criticality (for a thoughtful critique, see Levin, 1999), indicates that simple rules govern how Vida populations and communities behave. It is in this sense of meaning that the phrase "emergent properties" applies to Vida simulations.

Two important emergent properties in our simulations are the scaling of canopy mass with respect to stem diameter (M_L vs. D_S) and the scaling of self-thinning ($\overline{M_T}$ vs. N). Vida consistently predicts that the scaling exponent for M_L vs. D_S will converge on or numerically equal the value of 2.0, regardless of how computer runs are parameterized. This value has been reported for a variety of unrelated species growing under different conditions as well as collections of species growing as communities (Niklas, 2004). Vida also predicts that self-thinning will be governed by a –3/4 scaling exponent, which has been reported by a variety of workers. Yet, the only parameter directly relating to plant number is the number of propagules used to initialize a computer run (Table 3.1).

Another intriguing emergent property is the ability to predict the age at which reproductive maturity is achieved (Table 3.3) based on a shift in the allometry of height with respect to stem diameter. This shift has a cascade affect, because it results in a significant change in canopy growth with respect to stem growth that in turn affects the ability of a plant to harvest sunlight that in turn alters total growth and thus the amount of biomass that can be used to construct propagules. Importantly, the shift in plant height vs. stem diameter emulated by Vida has been predicted and empirically demonstrated using a zero-order biophysical theory (Niklas and Spatz, 2006). However, this biophysical theory is not part of Vida's computational logic (see Figure 3.1). It is a direct consequence of empirically determined input variables based exclusively on how plant height scales with respect to stem diameter among real plants. It is noteworthy that the ability to predict when a species reaches reproductive age based

on vegetative growth is consistent with prior allometric theory as well as empirical observation (Niklas and Enquist, 2003).

As noted, we believe that these and other emergent properties are the result of dynamical self-organization. But an attribution to a general phenomenon does not explain the particular numerical values for the scaling relationships predicted by Vida (nor those that have been repeatedly reported by other workers for real plant populations or communities). We believe that these particular numerical values may be the result of how a few very basic physical laws influence how sedentary organisms are able to fill available space as they grow in size and compete for light.

The land plants are undeniably an exceedingly diverse group of organisms that manifest a broad range of functional traits (Bolker et al., 2003; Diez and Pulliam, 2007; Lichstein et al., 2007). Yet, all land plants share a key set of morphological and physiological attributes as a result of having evolved from a common ancestor (Gifford and Foster, 1989; Taiz and Zeiger, 2002). Our analyses suggest that these shared attributes place limits on how space can be occupied, how light energy is harvested, and how annual growth in biomass can be allocated to construct new vegetative and reproductive organs (see, for example, Olsen et al., 2009). In turn, the resulting "workable" range of allocation patterns is quantitatively expressed (and thus observed empirically) in the form of a limited range of numerical values for the exponents governing important scaling relationships such as annual growth versus body mass (G_T vs. M_T) and canopy mass versus stem diameter (M_L vs. D_S). For example, both empirical observation and Vida show that the exponent for G_T vs. M_T differs among species or species-groupings (see Table 3.2). Yet, the numerical range of this exponent is comparatively circumspect (i.e., between 2/3 and 3/4), which can be explained using basic engineering and hydraulic theory. Stems must increase in girth as they increase in length or as they support an increasing mass of leaves (Niklas, 1992; Niklas and Spatz,

2006). In turn, successful competition for light requires taller and broader canopies among conspecifics or species sharing similar functional traits, whereas the display of reproductive organs and their abiotic long-distance dispersal are facilitated by increases in canopy area or plant stature (Harper, 1982). The allocation of biomass to stems, leaves, and propagules (seed or fruits), therefore, is critical to vegetative and reproductive success just as it is inexorably linked to the operation of physical laws that broadly define tractable solutions regardless of whether an individual competes for light and space with its own offspring or different species. For these reasons, exponents such as 2/3 and 3/4, repeatedly emerge in engineering theories dealing with the elastic stability of columns (trunks) and cantilevered beams (branches) supporting variously placed loads (leaves) (e.g., McMahon, 1973) as well as in ecological treatments of population density and body size relationships (e.g., Damuth, 1981; Norberg, 1988).

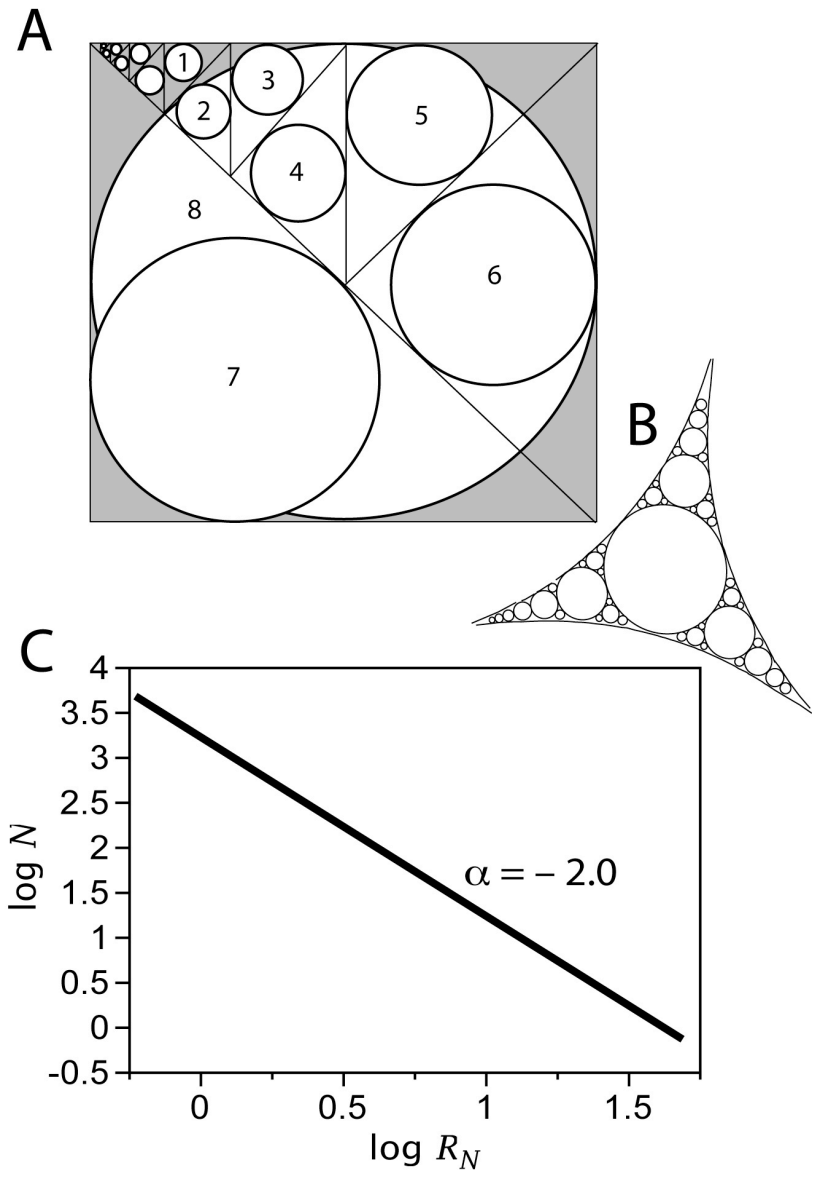
Many of the scaling relationships predicted by Vida also emerge from the West, Enquist, and Brown (WEB) theory and its recent conceptual variants and elaborations (West et al., 1997; Ernest et al., 2003; Enquist et al., 2007). This convergence suggests to us that Vida and erudite mathematical/theoretic explanations for observed real-world scaling relationships such as the WEB theory have conceptual common ground.

Certainly, the WEB theory depends on basic physical principles, much like Vida does. But the WEB theory and Vida share another very important feature: they both deal with the consequences of packaging objects in real space. The WEB theory deals with a fractal-like network packaged inside the organism. Vida deals with packaging circular stems and hemispherical canopies into three dimensional space. This commonality requires fractal-like rules if the tilling of space is to be optimized. For example, space can be tiled iteratively with different sized circles tangentially touching three, four, or more other circles. This kind of tiling is called “Apollonian packing” (see Karner and Supnick, 1943) and, depending on the number of circles touching one

another, it requires scaling exponents for size-frequency distributions that converge onto very discrete numbers, such as –2.0, –4/3, and –3/4 (Figure 3.8), much like those predicted by Vida and the West, Brown, and Enquist theory (West et al., 1997). Clearly, Apollonian packing involves non-intersecting circles, whereas tree canopies are capable of growing into one another. However, many consequences of canopy intersection and overtopping are deleterious (stem and leaf abrasion and light-deprivation) and thus limit the extent to which plants differing in size can occupy two- or three-dimensional space. The extent to which space can be occupied by hypothetical circles and by real tree canopies (or stems) differing in size may help to explain why exponents such as –1.8, –1.24, and 0.71 (rather than –2.0, –4/3, and –3/4) more frequently occur in nature.

We have explored only a small part of Vida's potential. While not discussed in this paper, Vida is capable of predicting the fate of a species under varying degrees of spatial and temporal heterogeneity by altering the amounts of limiting nutrients in space or changing nutrient availability over different time intervals. It can also be used to evaluate the properties that contribute to the success or failure of an invasive species by examining the growth of populations characterized by different scaling exponents or allometric constants. Vida can also be used to test whether theoretically predicted "optimal" or "canonical" scaling exponents confer a competitive advantage (and are thus under positive selection), or whether different scaling exponents are equally amenable to species survival. Future exploration of Vida's intricacies may reveal more about self-organization in plant populations and communities and whether this phenomenon is really an expression of self-organized criticality. For now, we only suggest that the emergent properties demonstrated by Vida have heuristic value — they show that competition for limited resources can result in strikingly complex behavior at the level of populations and communities, even *in silico*.

Figure 3.8: Graphic representation of "Apollonian" packing of circles in a square world-space. A. Progressively larger circles (e.g., 1 to 8) are installed in the world-space (to mimic plant growth in size as gauged either by canopy area, or stem cross sectional area); each iteration doubles the size of the circles in the previous iteration until a single circle occupies the world- space (to mimic competition for space and mortality resulting from shading). B. Apollonian packing of circles ultimately results in curvilinear triangular areas in the world-space, which can be occupied by cadres of circles differing in size. C. The size-frequency distribution of circles differing in size (defined by canopy or stem radius, R_N) conforms to a log-log linear relation with a scaling exponent close to – 2.0.


A
1
2
3
4
5
6
7
8
B
C
log N
log R_N
$\alpha = -2.0$
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
0.5
1
1.5

REFERENCES

BAK, P., C. TANG, AND K. WIESENFIELD. 1988. Self-organized criticality. *Physical Review A.* 38: 364-374.

BLUM, J. J. 1977. On the geometry of four-dimensions and the relationship between metabolism and body mass. *Journal of Theoretical Biology*. 64: 599-601.

BOLKER, B. M., S. W. PACALA, AND C. NEUHAUSER. 2003. Spatial dynamics in model plant communities: What do we really know? *American Naturalist.* 162:135 – 148.

BUSING, R. T., AND D. MAILLY. 2004. Advances in spatial, individual-based modelling of forest dynamics. *Journal of Vegetation Science.* 15: 831-824.

CANNELL, M. G. R. 1982. World Forest Biomass and Primary Production Data. Academic Press, London, UK.

CANTIANA, M. 1974. Experimental research into dendrometry and auxometry. Part II. Initial enquires concerning the biomass of the white fir. [in Italian] *Bulletin 5. Instit. Assestamento Forestale*. Univ. of Firenze, Italy.

CHAVE, J. 1999. Study of structural, successional and spatial patterns in tropical rain forests using TROLL, a spatially explicit forest model. *Ecological Modelling.* 124: 233-254.

CARLQUIST, S. 2001. Comparative wood anatomy, 2nd Edition. Springer-Verlag, Berlin.

DAMUTH, J. 1981. Population density and body size mammals. *Nature.* 290: 699 – 700.

DIEZ, J. M., AND H. R. PULLIAM. 2007. Hierarchical analysis of species distributions and abundance across environmental gradients. *Ecology.* 88: 3144 – 3152.

DODDS, P. S., D. H. ROTHMAN, AND J. S. WEITZ. 2001. Re-examination of the "3/4-law" of metabolism. *Journal of Theoretical Biology.* 209: 9 – 27.

ECONOMOS, A. C. 1979. Gravity, metabolic rate and body size of mammals.

Physiologist. 22: S71-S72.

ENQUIST, B. J., J. H. BROWN, AND G. B. WEST. 1998. Allometric scaling of plant energetics and population density. *Nature.* 395: 163 – 165.

ENQUIST, B. J., G. B. WEST, AND J. H. BROWN. 2009. Extensions and evaluations of a general quantitative theory of forest structure and dynamics. *Proceedings of the National Academy of Science.* 109: 7046 – 7051.

ENQUIST, B. J., AND K. J. NIKLAS. 2001. Invariant scaling relations across tree-dominated communities. *Nature.* 410: 655 – 660.

ENQUIST, B. J., A. J. KERKHOFF, S. C. STARK, N. G SWENSON, M. C. MCCARTHY, AND C. A. PRICE. 2007. A general integrative model for scaling plant growth, carbon flux, and functional trait spectra. *Nature.* 449: 218 – 222.

ERNEST, S. K. M., et al. 2003. Thermodynamic and metabolic effects on the scaling of production and population energy use. *Ecology Letters.* 6: 990 – 995.

GIFFORD, E. M., AND FOSTER, A. S. 1989. Morphology and evolution of vascular plants. Freeman, New York, New York, USA.

HARPER, J. L. 1982. Population biology of plants. Academic Press, London, UK.

HELLRIGL, B. 1974. Experimental research into dendrometry and auxometry. Part I. Tabulation of the productivity of tree biomass. [in Italian] *Bull. 5. Instit. Assestamento Forestale*. Univ. of Firenze, Italy.

JENKINS, J. J., D. C. CHOJNACKY, L. S. HEATH, AND R. A. BIRDSEY. 2004. Comparative database of diameter-based biomass regressions for North American tree species. USDA General Technical Report NE–319. Northeastern Research Station. USA Forest Service, Newtown Square, Pennsylvania, USA.

KARNER, E., AND F. SUPNICK. 1943. The Apollonian packing of circles. *Proceedings of the National Academy of Sciences (USA).* 29: 378 – 384.

KOZLOWSKI, J. AND M. KONARZEWSKI. 2004. Is West, Brown and Enquist's model of

allometric scaling mathematically correct and biologically relevant? Functional Ecology 18:283 – 289.

LAVOREL, S., AND E. GARNIER. 2002. Predicting changes in community composition and ecosystem functioning from plant traits; revisiting the Holy Grail. *Functional Ecology.* 16: 545 – 556.

LICHSTEIN, J. W., J. DUSHOFF, S. A. LEVIN, AND S. W. PACALA. 2007. Intraspecific variation and species coexistence. *American Naturalist.* 170: 807 – 818.

LEVIN, S. 1999. Fragile dominion. Perseus Books, Reading, Massachusetts, MA, USA.

MAKARIEVA, A.M., V.G. GORSHKOV, B.-L. LI, S. L. CHOWN, P.B. REICH, AND V.M. GAVRILOVE. 2008. Mean mass-specific metabolic rates are strikingly similar across life's major domains: Evidence for life's metabolic optimum. *Proceedings of the National Academy of Sciences (USA).* 105:16994 – 16999.

MARBÀ, N., C. M. DUARTE, AND S. AGUSTÆ. 2007. Allometric scaling of plant life history. *Proceedings of the National Academy of Sciences (USA).* 104: 15777-15780.

MCMAHON, T. A. 1973. Size and shape in biology. *Science.* 179: 1201 – 1204.

MCMAHON, T. A. 1980. Food habits, energetics, and the population biology of mammals. *American Naturalist.* 109: 547 – 563.

NIKLAS, K. J. 1992. Plant biomechanics. University of Chicago Press, Chicago, Illinois, USA.

NIKLAS, K. J. 1994. Plant allometry. University of Chicago Press, Chicago, Illinois, USA.

NIKLAS, K. J. 2000. Modeling fossil plant form-function relationships: a critique. *Paleobiology.* 26: 289– 304.

NIKLAS, K. J. 2004. Plant allometry: is there a grand unifying theory? *Biological Reviews.* 79: 871-889.

NIKLAS, K. J., AND B. J. ENQUIST. 2003. An allometric model for seed plant reproduction. *Evolutionary Ecological Research.* 5: 79 – 88.

NIKLAS, K. J., MIDGLEY, J. J., AND RAND, R. H. 2003. Tree size frequency distributions, plant density, age and community disturbance. *Ecology Letters.* 6: 405-411.

NIKLAS, K. J., AND SPATZ, H.-C. 2006. Allometric theory and the mechanical stability of large trees: proof and conjecture. *American Journal of Botany.* 93: 824-828.

NORBERG, R. A. 1988. Theory of growth allometry of plants and self-thinning of plant populations: geometric similarity, elastic similarity, and different growth modes of plant parts. *American Naturalist.* 131: 220 – 256.

OLSEN, M. E., R. AGUIRRE-HERNÁNDEZ, AND J. A. ROSELL. 2008. Universal foliage-stem scaling across environments and species in dicot trees: plasticity, biomechanics and Corner's rules. *Ecology Letters.* 12: 210 – 219.

PURVES, D., AND S. PACALA. 2008. Predictive models of forest dynamics. *Science.* 320: 1452 – 1453.

SMITH, W. B., AND G. J. SMITH. 1983. Allometric biomass equations for 98 species of herbs, shrubs, and small trees. Forest Service (USDA) Research Note NC 299.

TAIZ, L., AND E. ZEIGER. 2002. Plant Physiology. [3rd Edition] Sinauer, Sunderland, Massachusetts, USA.

TILMAN, D. 1982. Resource competition and community structure. Princeton University Press, Princeton, New Jersey, USA.

TILMAN, D. 1990. Constraints and tradeoffs: Toward a predictive theory of competition and succession. *Oikos.* 58: 3-15.

WARTON, D. I., AND N. C. WEBER. 2002. Common slope tests for bivariate errors-in variables. *Biometry Journal.* 44: 161 – 174.

WARTON, D. I., I. J. WRIGHT, D. S. FALSTER, AND M. WESTOBY. 2006. Bivariate line

fitting methods for allometry. *Biological Reviews.* 81: 259 – 291.

WEST, G. B., J. H. BROWN, J. H. AND B. J. ENQUIST. 1997. A General Model for the Origin of Allometric Scaling Laws in Biology. *Science.* 276: 122-126.

WOLF, H. 2003. EUFORGEN Technical Guidelines for Genetic Conservation and Use for Silver Fir (*Abies alba*). International Plant Genetic Resource Institute, Rome, Italy.

Chapter Four

Computer Simulations Identify Factors that favor Species Co-Existence in Homogeneous and Heterogeneous Environments

Understanding how species co-exist while competing for similar resources remains a central objective of ecological theory. A spatially explicit, reiterative algorithm (Vida) to simulate spatially homogeneous and heterogeneous environments differing in the availability of a growth-limiting nutrient. Four species were parameterized to simulate niche differences and introduced into these environments to compete for the limiting nutrient, space, and light. The objective was to evaluate the effects of habitat and niche differences on species co-existence and survival. These simulations make six predictions: (1) small niche differences can result in strong competitor advantages, (2) spatially homogeneous environments do not favor species co-existence, (3) spatially heterogeneous and predictable environments favor species co-existence, (4) spatially heterogeneous and disturbed environments foster greater species co-existence but in unpredictable ways, (5) although the same species repeatedly emerge as dominant species in spatially predictable and unpredictable environments, the identities of subdominant and rare species differ, and (6) the canopies of old plants and the boundaries between regions differing in nutrient availability provide refugia *for rare species. These predictions are consistent with contemporary ecological theory, but they emerge from a more mathematically transparent model that relies on fewer assumptions.*

In 1959, G. E. Hutchinson sought to resolve one of the major theoretical ecological dilemmas of his time by asking how is it that numerous species are capable of co-existing in the same habitat. Most aquatic and terrestrial habitats are occupied by

numerous and diverse species regardless of whether "diversity" is measured by a phyletic, physiological, or a morphological yardstick (May, 1994). Yet, many of the ecological models available to Hutchinson predicted that habitats should be dominated by a minimal number of equally fit species. Hutchinson dealt with this obvious paradox by pointing out that models predicting low species richness make a number of simplifying assumptions that are biologically unreasonable (e.g., species-species interactions are at equilibrium, food-webs consist of only two trophic levels, limiting physical factors do not exist, and the environment is spatially and temporally homogeneous) and by arguing that the violation of any one of these assumptions permits diverse species to co-exist. Thus, the paradox of species co-existence posed by Hutchinson is a paradox of theory rather than reality, because the simplest models of his time dealt with spatially homogeneous habitats, at equilibrium, occupied by biologically simple (and unrealistic) hypothetical organisms.

The subsequent development of the ecological theory for the maintenance of habitat species richness has been far more sophisticated and has included a variety of conceptual advances (reviewed by Tilman, 1982, 1990; Tilman et al., 1997; Chesson, 2000). For example, most current theories either implicitly or explicitly assume that resources are limited and that species reduce resources, at equilibrium, at different rates because they occupy different niches. Eliminating the case of functionally identical species (i.e., species that reduce the concentrations of resources at the same rates to identical levels), many models predict that the number of co-existing species will be less than or equal to the number of limiting resources (Tilman et al., 1997; Tilman, 2004). As the number of limiting resources increases, models predict a corresponding increase in the number of co-existing species. A critical parameter (denoted by R^*) in this kind of model is the resource level at which the resource-dependent growth rate of a species exactly equals the rate of total biomass loss resulting from herbivory, mortality,

and all other sources of biomass loss (Tilman et al., 1997). When two or more species compete for the same limiting resource, the species with the lowest *R** is predicted to displace all other species. It follows, therefore, that the co-existence of many species requires that competitors for limited resources must differ substantively in their niches. In turn, this suggests that niche differences can cause conspecifics to compete more among themselves than among the members of other species. If true, niche differences help to stabilize competitor dynamics by giving species higher per capita population growth rates when rare than when common.

Although most plant species require the same resources for growth and reproduction (Taiz and Zeiger, 2002), niche differences can take on a variety of potentially subtle morphological or physiological forms, i.e., differences in shoot architecture or rooting depth, water use efficiency, specific leaf area, shade tolerance, hydraulic conductivity, nitrogen and phosphorus stoichiometry, or seed dispersal and germination requirements (reviewed by Harper, 1982; e.g., Groves and Williams, 1975). Niche differences can also be demographic, e.g., differences in the age at which plants reach sexual maturity (Takahashi and Lechowicz, 2008). These and many other niche differences help to explain how many plant species can inhabit the same environment (Levine and HilleRisLambers, 2009), or why height-structure in forested communities can result in an *R** for light (Adams et al., 2007). Yet, the importance of niche differences for stabilizing biodiversity remains poorly understood despite numerous experimental and field studies examining phenotypic or demographic differences among co-occurring species (Eriksson, 1996; Debinski and Holt, 2000; Ries et al., 2004). Additionally, the theoretical perspective emerging from the role of resource limitation on habitat species-richness has been disputed by the Neutral Biodiversity Theory, which proposes that community-level patterns are primarily determined by the effects of stochastic processes governing birth and death rates and that a detailed knowledge of the traits of

(and interactions among) the individuals or species occupying a habitat is largely irrelevant (Hubbell, 2001, 2005).

In a strictly formal sense, the perspectives emerging from the Neutral Biodiversity Theory and other ecological theories emphasizing phenomena such as the effects of resource limitation on species richness bracket a spectrum of hypotheses about the importance (and nature) of niche differences and other ecologically important factors such as habitat spatiotemporal heterogeneity, resource-dependent growth rates, the nature of species-species interactions, and the effects of birth and death rates and size-dependent (allometric) variations on species richness (Chesson, 2000). When viewed in their entirety, these manifold factors present a huge number of theoretically viable hypotheses. Empirically evaluating each of these hypotheses is a daunting, if not impractical task because even comparatively simple habitats can differ abiotically in a myriad of un-quantified ways and because the functional traits and niche differences of radically different kinds of organisms, such as plants and animals, can result in dramatically different empirical patterns of species richness and persistence. Yet another concern is the time-span that a particular habitat should be observed to assess species co-existence. Are tens, hundreds, or even thousands of years sufficient? For these reasons, ecologists continue to explore and further refine ecological theories for species co-existence, while at the same time suggesting how to test their theoretical expectations (O'Dwyer et al., 2009).

The goal of this paper is to test the hypothesis that a few simple assumptions generate species-richness trends observed for real communities using a spatially explicit reiterative algorithm[10] named Vida to model plant community dynamics. Vida was designed to specifically evaluate how different plant species compete for light and space

[10] In the original publication, this software was named SERA, an acronym of Spatially Explicit Reiterative Algorithm

in a world-space whose physical attributes, such as the direction and intensity of incident sunlight, are clearly defined (Hammond and Niklas, 2009). Since each variable that stipulates the world-space and individual species can be manipulated, Vida provides a venue for evaluating the effects of very specific niche and habitat differences on the co-existence of different species under rigidly stipulated environmental conditions. This algorithm has an additional attractive feature — it can simulate stochastic processes such as random seed dispersal and mortality and changes in the location of a limiting resource in a spatially heterogeneous and unpredictable world-space, factors that figure prominently in recent elaborations of ecological theory (e.g., Nathan and Muller-Landau, 2000; O'Dwyer et al., 2009). Vida therefore allows the consequences of competition for space, light, and growth-limiting nutrients to be evaluated quantitatively as a community grows *in silico* within a homogeneous or heterogeneous environment, depending on the type of computer simulations required. Vida can also track and report the history of an individual species competing with other species, or an individual plant competing with its neighbors so that changes in species abundance or spatial distribution, or changes in individual body size and fecundity can be evaluated on a case-by-case level.

In this paper, we report the results of 108 Vida simulations that were designed to examine the behavior of a community consisting of four hypothetical tree species, each differing in its ability to compete for (and utilize) space, light, and a single growth-limiting soil nutrient (such as nitrogen or phosphorous; e.g., Wilson and Tilman, 1991). The niche differences distinguishing these four species were based on allometric differences observed for real tree species, such as differences in biomass allocation to leaves and stems and size-dependent changes in trunk diameter with respect to tree height. Throughout all these simulations, light was uniformly distributed in a world-space that did not differ in geographic size. However, different sets of Vida simulations

were run in which the location and concentration of a growth-limiting nutrient were changed to evaluate the effects of resource patchiness and resource disturbance on the ability of the four species to successfully reproduce and co-exist. To establish baselines with which to compare the effects of resource heterogeneity or temporal variation on species co-existence, simulations were performed in which all four species successfully colonized a habitat in which the nutrient was inexhaustible and homogeneously distributed. The predictions of these simulations were then summarized and compared both to current ecological theoretical expectations and to empirically observed patterns of plant co-existence in spatiotemporally heterogeneous habitats.

Given the tremendous complexity and diversity of real plant communities and the long-standing debates in ecological theory regarding species co-existence and persistence (e.g., Hubbell, 2005; Alonso et al., 2006; Westoby and Wright, 2006), the discussion of our results and conclusions is confined to evaluating Vida as a heuristic tool. We believe that one of the best uses for computer models such as Vida lies in their ability to reveal the logical (mathematical) consequences of assumptions. If a model employing these assumptions produces a phenomenology that mimics the behavior of the real system being modeled, such as a simple plant community, it is possible that these assumptions are necessary and sufficient for a theoretic explication of the phenomenology. However, we are equally cognizant of an important caveat — a model can give the correct answer for the wrong reason. This philosophy and caveat dictated this paper.

MATERIALS AND METHODS

The model —

Vida's assumptions and default settings, along with a downloadable version of the application are provided as Supplemental Material in a previous publication (Hammond

and Niklas, 2009). The complete source code for Vida is available upon request.

Vida is conceptually similar to other computer models, notably those of Chave (1999) and Niklas (2000; Enquist and Niklas 2001), in that each plant is simplified to consist of a single photosynthetic surface (canopy) elevated by a single stem. Vida differs from previously published models in at least three important ways: (1) it requires fewer input variables among which only six scaling exponents are required to initialize a simulation (Table 4.1), (2) it is an individual-based algorithm (i.e., species identification files define only the properties of plants belonging to a particular species), and (3), where some models represent the canopy as a flat disc (e.g., Chave, 1999; Enquist and Niklas, 2001), Vida's default setting creates hemispherical canopies of uniform thickness. This last difference is important because a disk-shaped canopy casts a significantly larger projected area than a hemispherical canopy (with equivalent mass and thickness) and thus over-estimates the space that must be occupied to capture a fixed quantity of light.

As in other models (e.g., Chave, 1999; Niklas, 2000), the angle of solar incidence is time-averaged, i.e., light comes from directly above each plant. Therefore, the photosynthetic area available for each plant is the projected area of the canopy (A_L). This stipulation can be relaxed to cope with daily changes in the solar angle or differences in latitude. However, our objective here is to simplify the real world by reducing the number of variables that can influence the outcomes of simulations. By eliminating latitudinal and diurnal differences in the solar angle, the results of simulations can be assumed to be due to plant-plant interactions and not due to variations in ecologically important variables such as total solar energy.

The size of the canopy and the extent to which it is shaded by neighboring plants dictate the ability of the individual to harvest light and grow annually. The light energy (E)harvested defines net biomass accumulation ("annual growth", G_T), which is

allocated to the construction of new leaves (M_L), stem biomass (M_S), and propagules (M_P). The conversion of light energy into G_T uses the formula $G_T = E\beta_{12} A_L M_L^{\alpha_9}$, where β_{12} and α_9 are the normalization (allometric) constant and scaling exponent, respectively, and A_L is the projected canopy area. In turn, the allocation of biomass to stem diameter and height (D_S and H_S, respectively) is governed by relationships that permit (but do not demand) a transition from stem geometric self-similarity to geometric non-similarity, i.e., $H_S = \beta_7 D_S^{\alpha_7} - \beta_8 \rightarrow H_S = \beta_9 + \beta_{10} \ln D_S$ (see Table 4.1 footnotes). This mathematical transition is based on observation and biomechanical theory that relate to the steady accumulation of secondary tissues in tree trunks (Niklas, 1994, 2004).

Depending on previous growth, input variables provide the option of using some portion of net biomass gain to construct propagules (differing in number *n* and individual mass M_P among species). Propagules can be randomly or non-randomly distributed depending on the characteristics of the species being modeled. The distribution of M_T to M_L and M_S immediately upon germination is species-specific. Mortality can result from one or more processes or events, e.g., the violation of physical laws, excessive light deprivation (which is a function of the attenuation of light through over-topping canopies), or stochastic/age-dependent processes.

Species niche differences —

Four species (designated as A, B, C, and D) were parameterized to create niche differences. Species A mimics the characteristics of *Abies alba*, while species B, C, and D were parameterized to have niche differences based on how they grow vegetatively as defined by the formula $G_T = E\beta_{12} A_L M_L^{\alpha_9}$. This formula was selected because the numerical value of α_1 defines the proportional (scaling) relationship between G_T and A_L M_L, while the numerical value of β_9 dictates the absolute amount of biomass that is allocated to increase the size of a trunk or canopy each year (see Table 4.1, variables 5

Table 4.1: Parameters and numerical values used to simulate four species (formulas and units in footnote). Species A parameterized using data for *Abies alba*; species B is a generic conifer; species C is species B with the photosynthesis growth constant of species A; and species D is species B with the photosynthesis growth constant of *Cryptomeria japonica*. Differences among species indicated in bold.

Species-specific parameter	Species			
	A	**B**	**C**	**D**
No difference across all species				
1. Initial seed number (n)	63	63	63	63
2. Canopy transmittance (T_C, fraction of light through canopy)	0.02	0.02	0.02	0.02
3. Survival transmittance (T_S, fraction of light needed for survival)	0.20	0.20	0.20	0.20
4. Maximum leaf thickness (H_C)	0.0003	0.0003	0.0003	0.0003
5. Photosynthesis growth exponent (α_9)	-0.46	-0.46	-0.46	-0.46
6. Growth memory (m, years growth rates are remembered)	2.00	2.00	2.00	2.00
7. Stem mass exponent (α_4)	1.00	1.00	1.00	1.00
8. Stem diameter constant (β_6)	0.03	0.03	0.03	0.03
9. Young stem height constant (β_8)	0.00	0.00	0.00	0.00
10. Mature stem height constant (β_9)	30.1	30.1	30.1	30.1
Differences between species A and species B–C				
11. Mature stem height constant (β_{10})	**15.5**	8.49	8.49	8.49
12. Young canopy mass exponent (α_5)	**1.2**	0.97	0.97	0.97
13. Young canopy mass constant (β_5)	**0.04**	0.29	0.29	0.29
14. Mature canopy mass exponent (α_8)	**0.73**	0.71	0.71	0.71
15. Mature canopy mass constant (β_{11})	**0.25**	0.39	0.39	0.39
16. Stem mass constant (β_4)	**0.94**	0.72	0.72	0.72
17. Stem diameter exponent (α_6)	**0.39**	0.40	0.40	0.40
18. Young stem height exponent ($\alpha 7$)	**1.1**	0.93	0.93	0.93
19. Young stem height constant (β_7)	**100**	80.3	80.3	80.3
Differences among all species				
20. Photosynthesis growth constant (β_{12})	**1.53**	**0.58**	**1.53**	**0.91**

Formulas: $D_S = \beta_6 M_S^{\alpha_6}$, $G_T = E\beta_{12} A_L M_L^{\alpha_9}$, $H_S = \beta_7 D_S^{\alpha_7} - \beta_8 \rightarrow H_S = \beta_9 + \beta_{10} \ln D_S$, $M_L = \beta_5 M_S^{\alpha_5} \rightarrow M_L = \beta_{11} M_S^{\alpha_8}$, and $M_S = \beta_4 M_T^{\alpha_4}$. Units: A_L (m^2), D_S (m), G_T (kg/yr), H_S (m), M_T (kg), M_L (kg), and M_S (kg). World-space size = 100 m x 100 m.

and 20). Therefore, comparatively small differences in β_{12}-can result in large differences in body size and thus in the ability to capture sunlight and compete for space and soil nutrients.

The numerical values of β_{12} (and all other parameters that define the properties of Vida species) were identified on the basis of the data for real tree species summarized in the Cannell (1982) world-wide forest productivity and growth compendium. Using this resource, species A has been shown to mimic all of the characteristics reported for a population of *Abies alba* that was observed for 85 years (data taken from Cantiana, 1974; Hellrigl, 1974; see Hammond and Niklas, 2009). For this species, $\beta_{12} = 1.53$ (Table 4.1).

Species B was parameterized to represent a generalized (average, or generic) conifer species based on the data tabulated by Cannell (1982) for 343 coniferous shrub and tree species growing world-wide, either naturally or in managed stands for commercial exploitation. In addition to having a different photosynthesis growth constant from that of *A. alba* ($\beta_{12} = 0.58$ versus 1.53), species B differs from *A. alba* in the numerical values of nine other parameters that collectively influence the allometric relationship between stem height and diameter and the biomass allocation pattern to stem versus canopy (see Table 4.1, parameters 11 – 19). Because the generalized conifer species reflects the allometry and the growth patterns of shrub as well as tree species, species B mimics the niche occupied by a conifer with a shrubby-tree growth habit.

Species C and D differ from the generalized conifer species (species B) only in terms of the numerical values of β_{12} (see Table 4.1). Species C represents a generalized conifer species parameterized with the photosynthesis growth constant of *A. alba*. Likewise, species D was parameterized as a generalized conifer but with the photosynthesis growth constant of *Cryptomeria japonica* ($\beta_{12} = 0.91$) as determined, once again, from the data reported by Cannell (1982). Although long-term data sets for

specific conifer forest-types are available (e.g., Stalter and Kincaid, 2008; Stalter et al., 2009), *C. japonica* was selected because the Cannell (1982) compendium provides a large body of data for trees assigned to this genus owing to their commercial value as a source of timber and because sufficiently large detailed data sets comparable to that of *A. alba* and *C. japonica* for other conifers were not available.

None of the four species used in this study possesses the attributes of a specific or even a generalized angiosperm species. The reasons for this are twofold. First, prior simulations indicated that a generalized angiosperm tree species rapidly out-competes *A. alba* (species A) and the generalized conifer species (species B) for light and space (see Hammond and Niklas, 2009), and, second, a careful study of the Cannell (1982) data compendium failed to reveal sufficient data necessary to parameterize a specific (real) angiosperm species. Simulations involving a generalized angiosperm species, therefore, would not provide useful or specific information concerning the nature of niche differences that might permit the co-existence of angiosperm and conifer species, while simulations involving a species parameterized to mimic a specific angiosperm species are not currently possible owing to a paucity of reliable data.

Homogeneous world-space simulations —

The usefulness of each of the four conifer species (Table 4.1) for the purpose of this study was evaluated by simulating single populations of each species and allowing each to grow under the same ambient light conditions but under five different levels of a hypothetical soil nutrient affecting overall plant growth. The effect of the nutrient's concentration (availability) on the ability of each species to compete for space and light was modeled by scaling plant growth to the level of nutrient availability.

Mathematically, total growth (G_T) was adjusted linearly to the nutrient concentration. For example, for an un-shaded plant growing in a world-space with an unlimited supply of the nutrient, G_T depends exclusively on ambient light intensity and

the size of the plant's canopy as given by the formula $G_T = E\beta_{12} A_L M_L^{\alpha_9}$. However, in the case of an identical plant growing in a world-space with a nutrient concentration of 50%, G_T is reduced by one-half. The reliability of this protocol was gauged by comparing the maximum annual growth rate reported for *A. alba* plants growing in a managed population (data reported by Cantiana, 1974; Hellrigl, 1974) to the results of Vida simulations of an *A. alba* population growing in a world-space with 100% nutrient availability and successively reduced levels of nutrient availability. In each case, the allometry and population dynamics reported for the real population were mimicked precisely, but with successively decreasing annual accumulations of total biomass.

Although totals plant growth was scaled linearly to nutrient availability levels, preliminary simulations showed that propagules fail to germinate and survive beyond the first year at nutrient levels below 20%. This nonlinear growth response is the result of a subroutine in Vida's computational logic, which requires a minimum growth rate for the continued survival of an individual. This minimum can be changed, but doing so resulted in anomalous behavior in the community dynamics of *A. alba* populations, which was our test-species.

To evaluate the effects of how nutrient availability on species co-existence, simulations were randomly seeded with the same numbers of the propagules from each of the four species to initiate community assembly. Three community simulations were run for each of the five levels of nutrient availability, i.e., 25%, 50%, 62.5%, 75%, and 100% (n = 15 simulations). Changes in the number and average size of the individuals of each species were recorded throughout each simulation to monitor the effects of competition for space, light, and the nutrient. Log-log regression of plant density and plant size versus time was used to measure the competitive performance of each species; the slope of the log-log regression was taken as a measure of species

competitiveness. Data collected during the first 50 years of each simulation were not used in these regression analyses because plants began to experience size-asymmetric competition (*sensu* Begon, 1984; Weiner, 1990) after a minimum of 35 years.

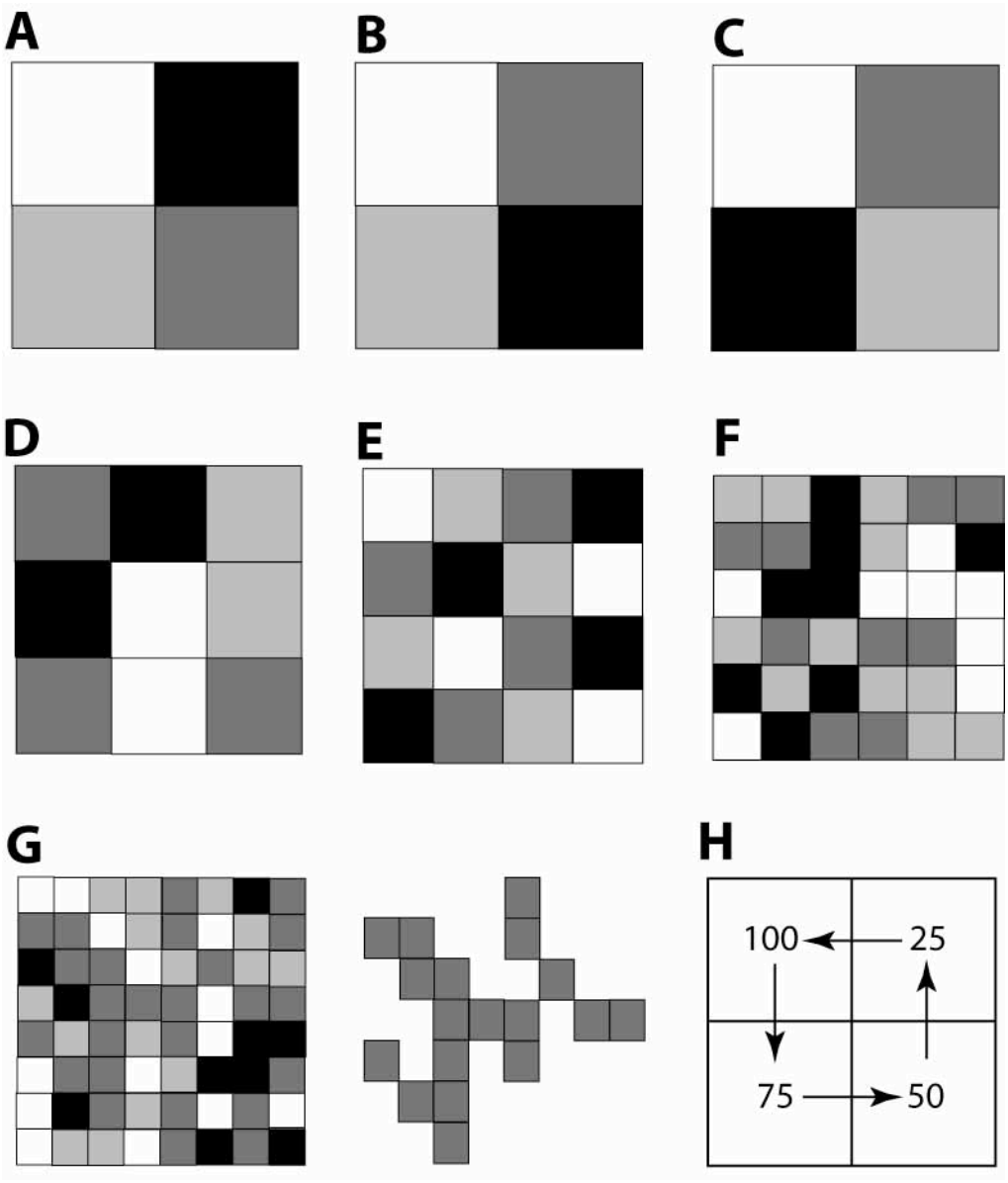
Spatially heterogeneous world-space simulations —

To determine the effects of environmental heterogeneity (patchiness) on species co-existence, the nutrient was distributed randomly at different levels of availability in a world-space (size = 100 m x 100 m) that was divided into five increasingly smaller patch-sizes, i.e., 1/4, 1/9, 1/16, 1/36, and 1/64 patch-sizes (Figure 4.1). Because the 1/4 patch-size has four different nutrient availabilities (25%, 50%, 75%, and 100%) and three different possible configurations (Figure 4.1 A – C), the overall nutrient availability was maintained at 62.5% in all spatially heterogeneous world-spaces (Figure 4.1D – G). Three simulations were run for each of the seven world-space configurations; each was seeded with equivalent numbers of propagules from each of the four species (n = 3 duplicates x 7 patch-size world-spaces = 21 simulations). As before, changes in plant densities and body sizes were monitored during the course of each simulation to evaluate the effects of environmental patchiness on species co-existence and sustainability.

Spatiotemporally heterogeneous simulations —

Simulations were also run to evaluate the effects of changing nutrient availability levels on species co-existence. Two kinds of simulations were performed — one in which the level of nutrient availability was reduced at different but predictable rates (to mimic rapid or slow rates of nutrient depletion), and another in which the level of nutrient availability was changed randomly at different rates (to mimic an unpredictable world-space). In the first kind of simulation, a 1/4 patch-size was used and the nutrient level was shifted from 100% to progressively lower availability levels (75%, 50%, and 25%) every 5, 10, 20, or 40 years (see Figure 4.1 H). This shift unavoidably resulted in

Figure 4.1: World-spaces differing in patchiness with respect to nutrient availabilities. White = 100% nutrient availability; light gray = 75%, dark gray = 50%; black = 25%. Total nutrient availability in each world-space = 62.5%. A – C. World-spaces with 1/4 patch-sizes. D. World-space with 1/9 patch-size. E. World-space with 1/16 patch-size. F. World-space with 1/36 patch-size. G. World-space with 1/64 patch-size (50% nutrient availability grid spaces highlighted to right of G illustrate an embedded macro-patch in a randomly generated pattern). H. World-space with 1/4 patch-size in which the nutrient availability periodically changed as indicated by directions of arrows.


A
B
C
D
E
F
G
H
100
25
75
50

an increase in nutrient availability in one of the four regions of the world-space, which mimicked the introduction of the nutrient in every cycle of change. Three simulations, each randomly seeded with equal numbers of propagules for each of the four species, were performed using each of the four rates of change, i.e., n = 3 x (5, 10, 20 and 40 years) = 12 simulations. Within each of the four quadrants, species interactions were monitored to assess the effects of nutrient depletion (and nutrient replenishment) on survival, co-existence, and reproductive success. In the second kind of simulation, nutrient availability levels were randomly changed every 5, 10, 20, or 40 years in world-spaces differing in patch-size, which resulted in a total of 60 simulations (n = 3 replicates x 5 patch-sizes x 4 rates of change).

RESULTS

Comparatively small niche differences (see Table 4.1) and even relatively minor changes in how the world-space was parameterized (see Figure 4.1) resulted in significant differences in species co-existence patterns observed in simulations run for 250 years. Across all 108 simulations, species C numerically dominated all of the other species; species B was either driven to extinction or became increasingly rare; and species A and D were suppressed as subdominants to species C, depending on nutrient availability or how the world-space was partitioned or disturbed (Figure 4.2 A – F). However, using the slope of log-transformed data for plant number (N) versus time, the response of each of the four species to nutrient availability and to environmental patchiness or disturbance was distinctly non-linear (Figure 4.3 A – D). Some species, like species B (the generic gymnosperm), achieved maximum population growth at intermediate levels of nutrient availability, or environmental patchiness or disturbance, while other species, like species C (the generic gymnosperm with the photosynthesis growth constant of *A. alba*), were indifferent to world-space parameterizations. These

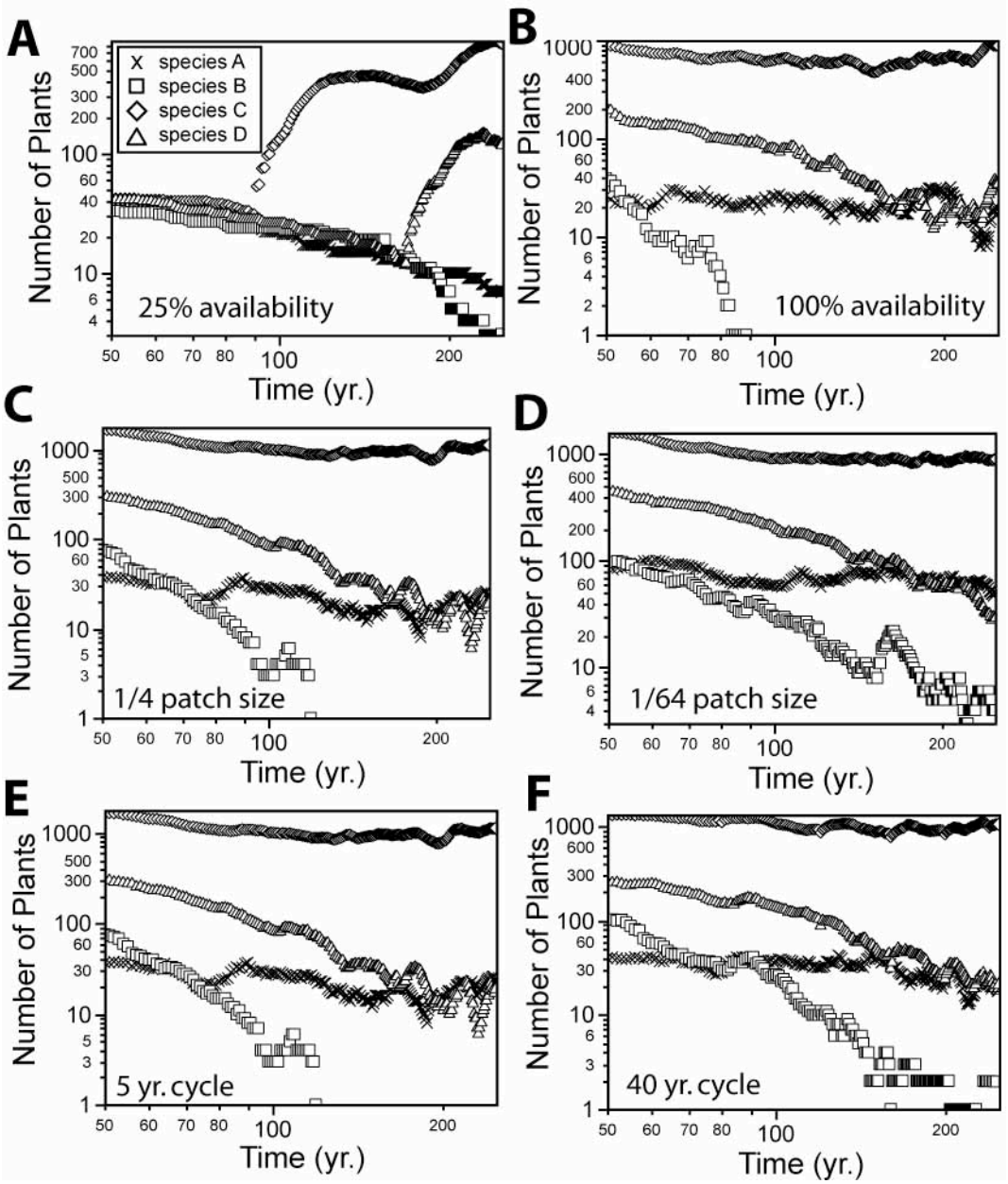


Figure 4.2: Graphic comparisons between the extremes in plant number plotted against time for each of four species (see insert for symbols) growing in world-spaces differing in nutrient availability levels (25% versus 100%, A – B), patch size (1/4 versus 1/64, C – D), and frequency of disturbance (5 yr. versus 40 yr. cycles, E – F). Each datum is the mean of three simulations. SE bars are hidden by symbols.

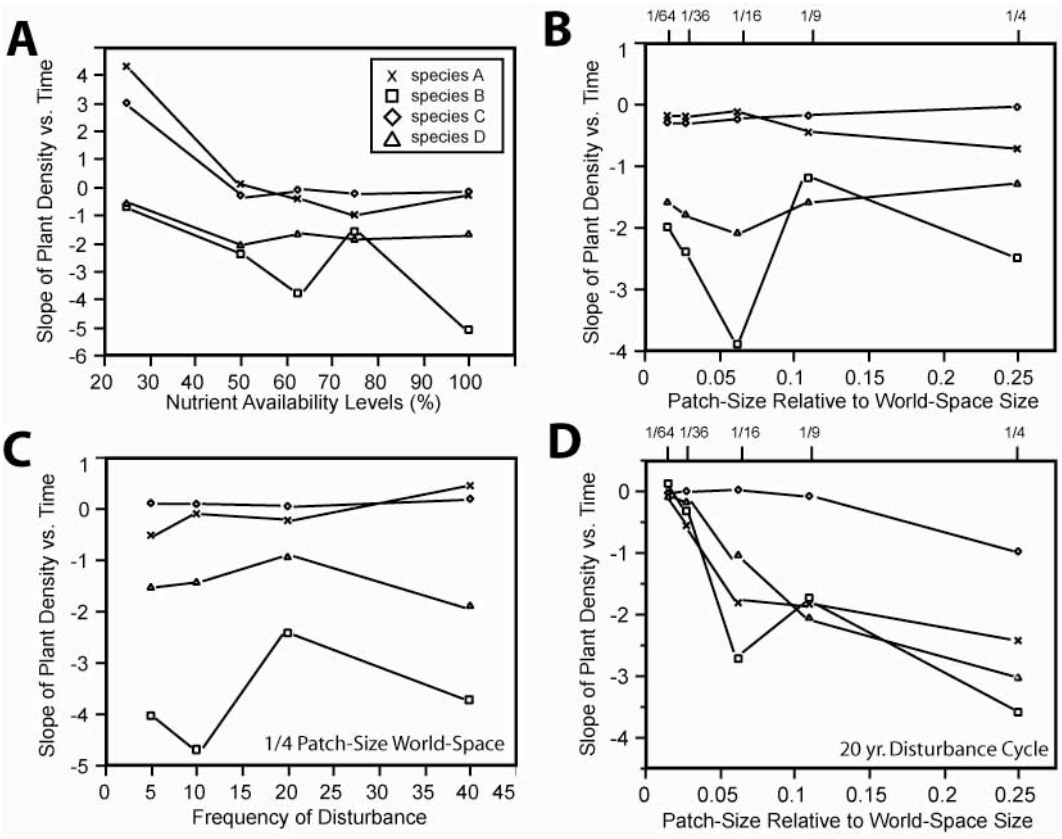


Figure 4.3: Slopes of the log-log regression curves for plant density versus time for each of four species (see insert for symbols) plotted against different nutrient availability levels in a spatially homogeneous world-space (A), world spaces-differing in their patch-sizes (B), a ¼ world-space experiencing different cycles of nutrient level disturbance (C), and world-space differing in patch sizes, all experiencing a 20 yr. disturbance cycle. The more positive the slope, the more a species is increasing in abundance. Each datum is the mean of three simulations; SE bars are hidden by symbols.

patterns were not due to differences in the cause of mortality. Light deprivation accounted for 95% of all plant deaths during the first 150 years of each community assembly; death resulting from senescence or slow-growth because of nutrient deprivation accounted for only 5% of all deaths.

Species co-existence increased as the world-space was made more spatially heterogeneous and as it was increasingly disturbed. For example, species B was driven to extinction in all spatially homogeneous simulations, but it survived, albeit as a rare species, in over 90% of all spatially heterogeneous and disturbed world-spaces. Likewise, although the number of species A and D plants were invariably low, these species achieved greater abundance in spatially heterogeneous and disturbed world-spaces (Figure 4.2). Species A, B, and D maintained a presence in environmentally patchy or disturbed simulations generally as a result of individuals growing along the boundaries between regions differing in nutrient availability, or as a result of light-gaps generated by the death of large, old plants (which were generally representatives of rare species).

Size-dependent (allometric) phenomena were also observed both for individual simulations and across all simulations (Figure 4.4). Of particular theoretical interest was the scaling relationship between plant growth (G_T) and body mass (M_T) and the relationship between body mass and plant density (N). Recent allometric theory predicts 3/4 and –4/3 scaling exponents for these two relationships (see Discussion). Across all 108 simulations, regression of log-transformed data for plant growth rates (G_T) versus total body mass (M_T) identified a scaling exponent of 0.77 (95% CI = 0.73, 0.81) (Figure 4.4A), whereas the scaling exponent for M_T versus plant density was –1.31 (95% CI = –1.29, –1.34) (Figure 4.4B). These exponents, which are statistically indistinguishable from 3/4 and – 4/3, differed significantly from those observed when populations of each species were simulated as monocultures, e.g., the scaling exponents

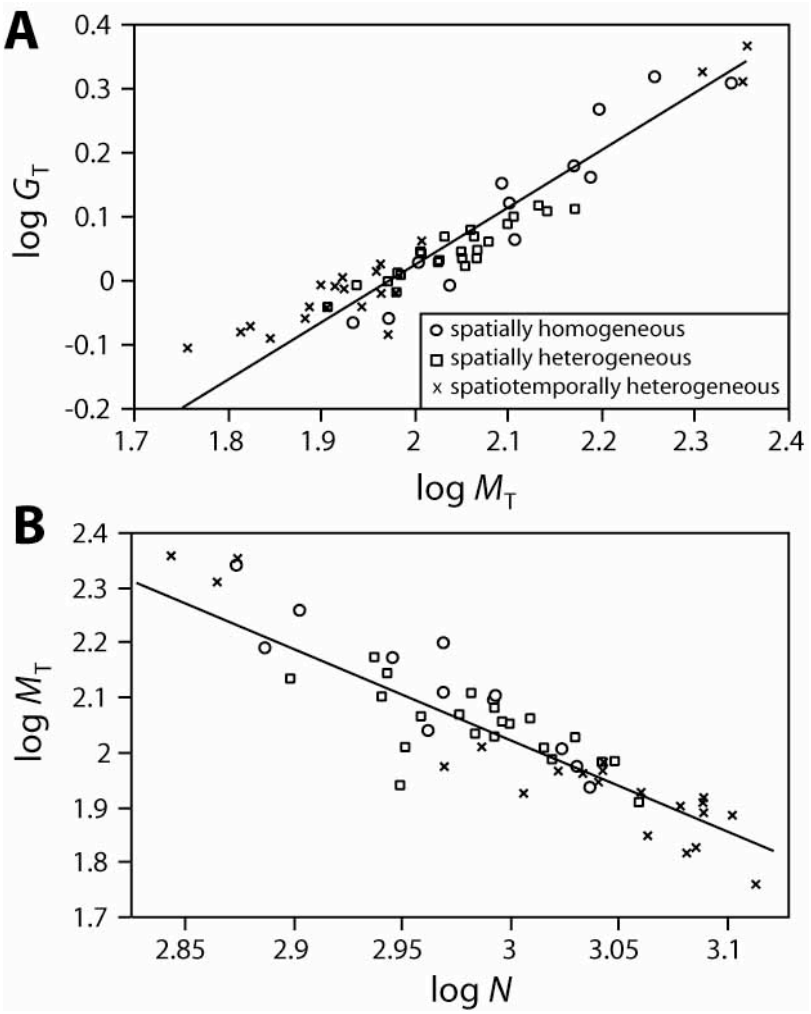


Figure 4.4: Bivariate plots showing the relationships among average plant growth (G_T), body mass (M_T), and plant number (N) at the end of all community assembly simulations. Lines are reduced major axis regression curves. A. G_T versus M_T. B. M_T versus N.

of G_T versus M_T for species A – D were 0.66 (r^2 = 0.998), 0.80 (r^2 = 0.989), 0.62 (r^2 = 0.988), and 0.70 (r^2 = 0.996), respectively, with an overall mean of 0.70 (95% CI = 0.66, 0.73).

The following sections provide more detailed information and other noteworthy results.

Nutrient levels control the timing of competition for light and space —

For each of the four species, nutrient availability levels below 20% resulted in plant death and thus species extinction. Above this threshold, plant growth rates (G_T) and total body mass (M_T) increased linearly with increasing nutrient levels (Figure 4.5A), i.e., the scaling exponent for G_T versus M_T was statistically indistinguishable from one (r^2 = 0.980; Figure 4.5B). Regression analyses showed that G_T and M_T each scaled as the 5/4 power of nutrient availability (r^2 = 0.938 and 0.981, respectively), which explains the one-to-one correspondence of G_T with M_T.

These phenomena were the result of how nutrient levels influence the timing of plant competition for space and light. At 25% nutrient availability levels, plants grow slowly, plant density increases slowly, and only a few plants "see their neighbors" and thus compete directly for space and light. As a result, species can co-exist for a comparatively long time. However, with increasing levels of nutrient availability, plant growth and body size increase more rapidly, which results leads to progressively more rapid size-asymmetric competition as communities assemble.

Species-specific optima for environmental patchiness favors co-existence —

G_T, M_T, and average plant age (A) decreased as the world-space was increasingly subdivided into regions differing in nutrient availability (Figure 4.6). However, this could not be entirely ascribed to the degree to which the environment was subdivided because statistically significant differences in G_T, M_T, and A were observed among two of the three 1/4 world-spaces. Therefore, the way in which a world-space is subdivided

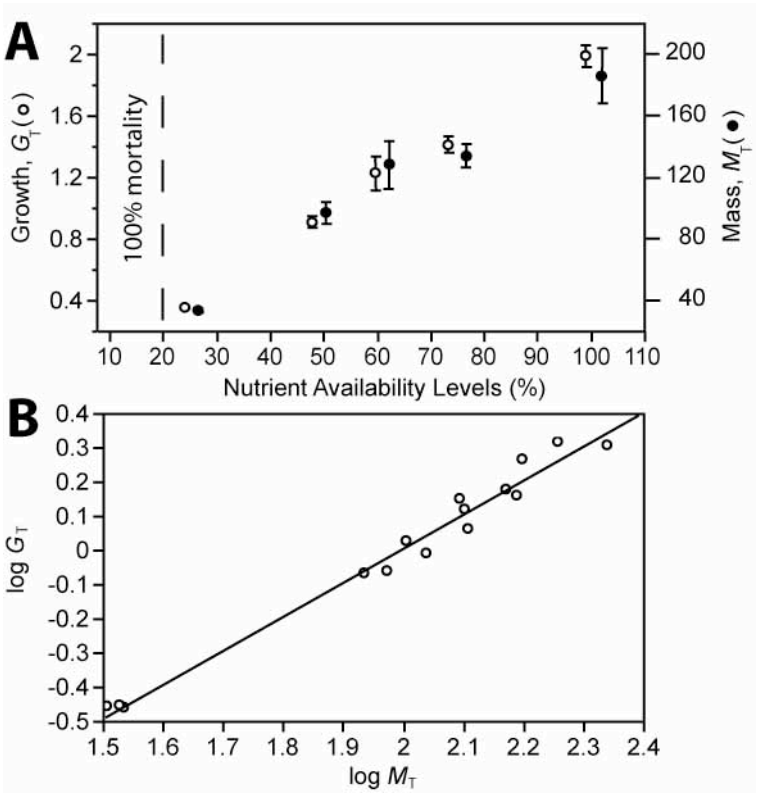


Figure 4.5: Mean (± SE) values for plant growth (G_T, in kg/plant/year) and total body mass (M_T, in kg) for three simulations of a homogeneous world-space with five different nutrient availability levels. The seeds of all four species fail to survive at nutrient levels less than 20% (indicated by dashed line). A. Mean (± SE) G_T and M_T for each nutrient level. B. Bivariate plot of log-transformed data for G_T vs. M_T.

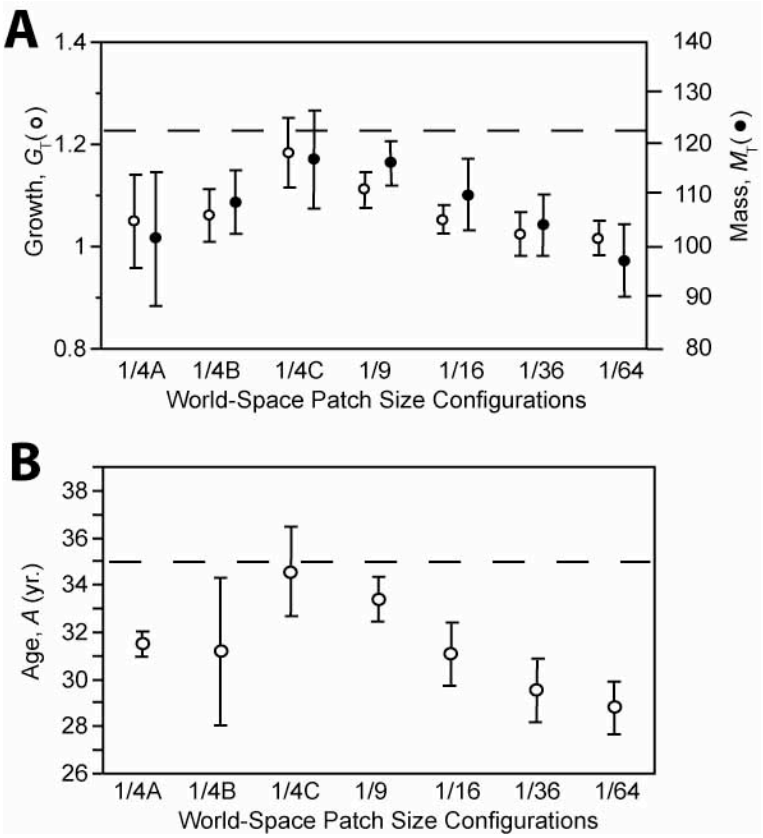


Figure 4.6: Mean (± SE) values for plant growth (G_T, in kg/plant/year), total (stem and canopy) body mass (M_T, in kg), and age (A, in years) based on three simulations each of a spatially heterogeneous world-space partitioned into different patch-sizes differing in nutrient availability levels (see Figure 1). Dashed lines indicate mean values for each parameter based on three sets of simulations in a spatially homogeneous world-space with a 62.5% nutrient availability level. A. G_T and M_T vs. patch-size configurations. B. A vs. patch-size configurations.

(that is, the arrangement of regions in relation to their neighbors) influences G_T, M_T, and A (and thus the time plants begin to competitively interact). Although species co-existence increased as the environment was made more patchy (see Figure 4.2C – D), the rate at which each species increased or decreased in abundance changed non-linearly with environmental patchiness. This was particularly evident for species B, which increased in abundance most rapidly in a 1/9 world-space compared to a 1/16 world-space (Figure 4.3B).

Vegetative niche differences linked to reproductive success and timing—

G_T, M_T, and A varied as the nutrient availability level steadily declined in the 1/4 world-space. G_T and M_T reached their maxima when the availability level changed every 20 years, while A reached its maximum when nutrient levels changed every 10 or 20 years. Minimum G_T was observed for simulations using 5-year cycles; minimum A occurred with 5- or 40-year cycles. Species co-existence also dependent on the disturbance cycle. Species C dominated communities regardless of the period of disturbance; species B survived only in 20-year disturbance cycles; while species A and D varied, sometimes dramatically, in their relative success (Figure 4.2E – F; Figure 4.3C).

These differences resulted only in part from the age at which each species reached reproductive maturity. Simulations of populations for each of the four species in a homogeneous world-space with a 100% nutrient availability level predicted that plants of species A, B, C, and D achieve sexual maturity at 25, 15, 7, and 9 years of age, respectively. Thus, in terms of their potential for reproductive advantage, the four species ranked as C > D > B > A. Yet, in terms of their competitive ability as reflected by all of the 1/4 patch-size world-space simulations, the species ranked as C >> (A + D) >> B. Accordingly, our simulations indicate that the ability of a species to persist in these simulations depends on its vegetative as well as its reproductive characteristics.

Optima for patch-size and disturbance frequencies result in different species co-existence patterns —

Plant growth rates, body size, age, and species co-existence became increasingly harder to predict as the environment became more spatiotemporally unpredictable. An increase in world-space patchiness from 1/4 to 1/16 doubled the average growth rate (from 1.1 kg/plant/yr to 2.2 kg/plant/yr), nearly tripled body size (82 kg to 225 kg), and increased average plant age by two years (from 26.5 to 28.5 years). However, increasing patchiness from 1/16 to 1/64 decreased all three variables of interest (Figure 4.7).

Species co-existence increased as the environment became more patchy, provided that the disturbance cycle was of sufficient duration to permit reproduction. As noted, the optimal disturbance cycle for the reproduction of all four species was 20 years. Using this disturbance cycle, simulations showed that species co-existence increased as the environment was increasingly sub-divided (Figure 4.3D). Individuals representing rare species persisted in these (and other simulations) in one of two ways — either as large plants that got established as seedlings during early community assembly, or as individuals that grew at or near the edge of two or more regions differing in nutrient availability. The former provided an "incumbent advantage"; the later resulted in reduced canopy obstruction because of poor growing conditions in a neighboring patch of space. Both of these phenomenologies are illustrated in Figure 4.8, which shows four aerial views of a community assembling in the 1/4 patch-size world-space disturbed every 40 years (see Figure 4.1H). Inspection shows that the more rare species are clustered at the edges of the four regions, or they are represented by a few very large adults whose canopy shadows are delineated by crescent shaped groupings of small representatives of the dominant species.

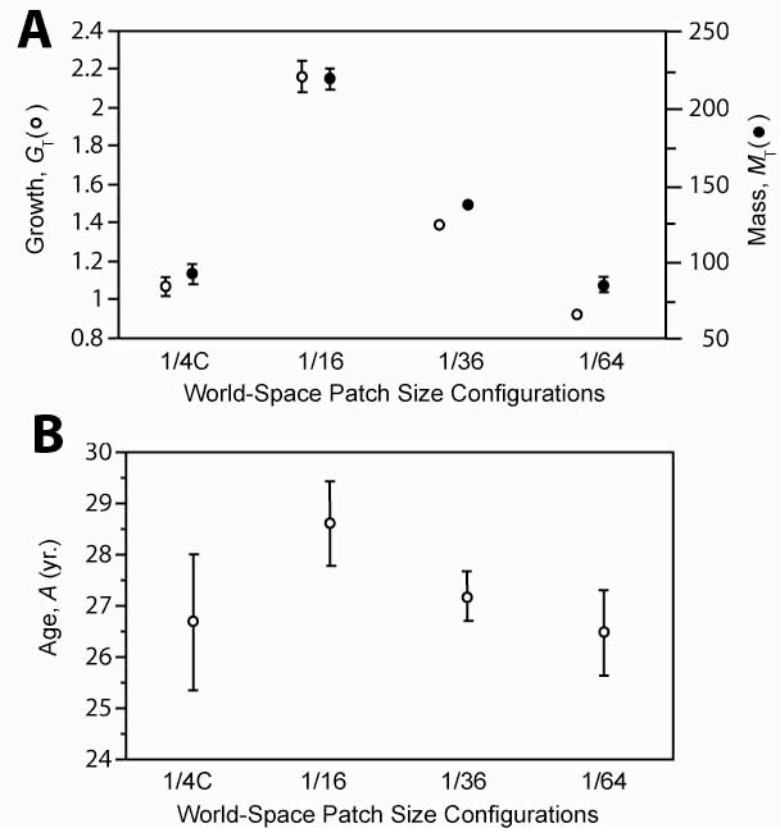


Figure 4.7: Mean (± SE) values for plant growth (G_T, in kg/plant/year), total (stem and canopy) body mass (M_T, in kg), and age (A, in years) based on three simulations each of a 1/4 patch-size world-space experiencing repeated periodic declines in nutrient availability (as depicted in Figure 1 H) every 5, 10, 20, and 40 years (n = 3 duplicates x 4 periods of disturbance = 12 simulations). A. G_T and M_T vs. world-space patch-sizes. B. A vs. world-space patch-sizes.

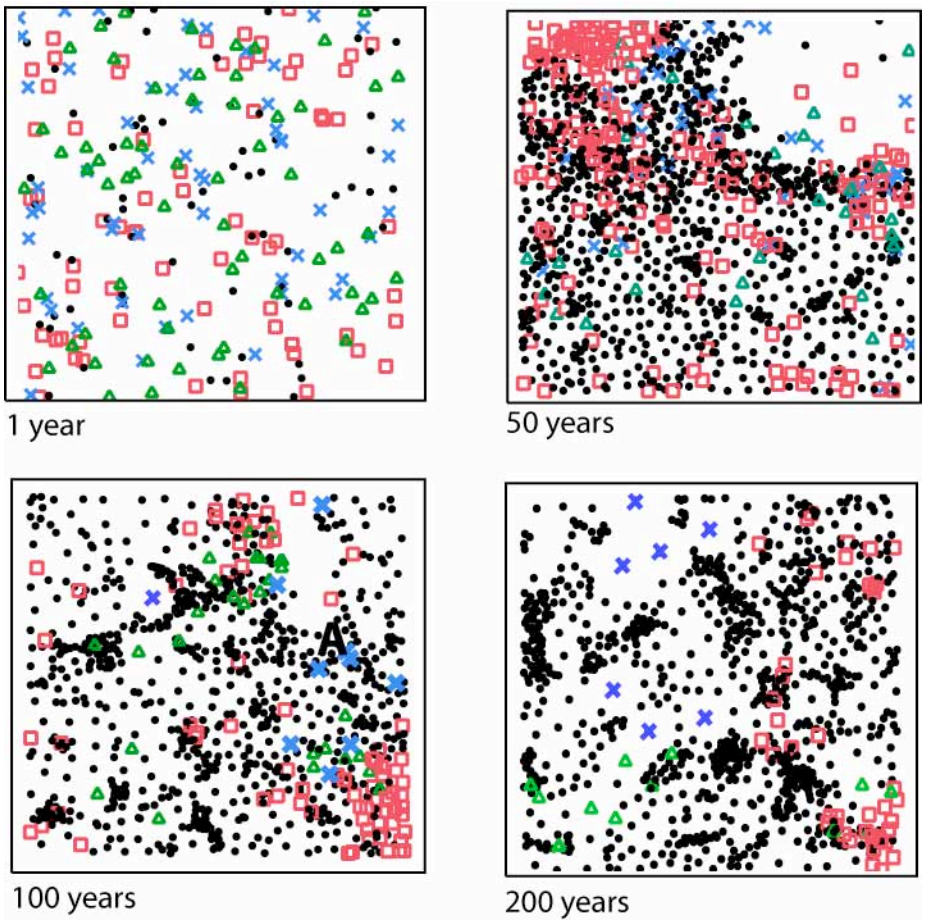


Figure 4.8: Polar views showing the locations of plants at for times during a simulation in a 1/4 world-space experiencing a 40 yr. disturbance cycle in nutrient availability. Dominant species (species C) dots; most rare species (species B) x's; co-subdominants (species A and D) triangles and squares.

DISCUSSION

Vida was designed to expose the mathematical (logical) consequences of how very specific, simplifying assumptions affect plant growth and competition for space and limited resources in a rigidly defined physical environment. The results of each simulation reveal how its algorithm manipulates the ways in which different species and world-spaces are parameterized. Prior work has shown that Vida accurately predicts the behavior of real plant species when sufficient (and reliable) data are used to model a particular species (Hammond and Niklas, 2009). This work also shows that Vida predicts empirically verifiable phenomena that do not result directly from how simulations are initialized. In the case of *A. alba*, we have shown that Vida successfully predicts the scaling relationship between plant number and body size as a population increases in size (self-thinning), the scaling relationship between annual growth and body mass, and the age at which *A. alba* plants reach sexual maturity, even though none of the data used to parameterize simulations relate directly to these phenomena (Hammond and Niklas, 2009). These and other features result from self-organization resulting from size-asymmetric competition among neighboring plants (*sensu* Begon, 1984; Weiner, 1990) as simulated populations or communities increase in plant density. Therefore, Vida has the potential to transcend the attributes of a simple opaque model in ways that can shed meaningful light on a number of important ecological and evolutionary questions.

The goal of the present study was strictly theoretical, i.e., to examine how mathematically well-defined niche differences affect species co-existence in a world-space manipulated to change the availability, distribution, and predictability of a nutrient affecting plant growth (such as nitrogen or phosphorus; e.g., Wilson and Tilman, 1991). To establish very small niche differences, we selected a single

biologically important variable, which distinguishes among three otherwise identical hypothetical species (i.e., the photosynthesis growth constant; see Table 4.1), and we manipulated three parameters to generate different world-spaces (i.e., the level of nutrient availability, the rate at which it changes, and the extent to which the world-space is subdivided into regions differing its availability). The resulting simulations therefore describe the dynamics of stringently defined, highly contrived, and very simplified plant communities in which species co-existence is dictated exclusively by the ability to compete for space, light, and a single nutrient. This simplicity results in significant mathematical transparency — the effect of each variable on species co-existence is made immediately apparent and explicit. In addition, because Vida is a spatially explicit algorithm, the location, behavior, and history of any individual plant or group of plants can be monitored over the course of a simulation such that neighborhood effects can be investigated.

Despite this simplicity, Vida simulations make specific predictions that are strikingly consistent with far more mathematically complex ecological theories about plant allometry or species co-existence. In terms of plant allometry, our simulations identify a 3/4 scaling relationship between plant growth and body size and a –4/3 scaling relationship between body mass and plant density. Both of these exponents numerically agree well with allometric theory and empirical observations (reviewed by Niklas, 2004). Likewise, among ecological theories that assume that resource supply rates and physical factors are spatially homogeneous, that each organism is distributed more or less uniformly throughout the environment, that resources do not fluctuate substantially, that localized mortality is minimal or does not occur, and that higher tropic levels are unimportant, all predict the stable co-existence of no more species than there are limiting resources and physical factors affecting growth (reviewed by Tilman, 1982, 1990; Tilman et al., 1997). Similarly, among theories that assume some sort of

environmental complexity, that allocation-based trade-offs exist, and that the responses of species differ in response to these trade-offs, all predict the existence of more species than there are limiting resources and physical limitations (Tilman, 1982, 1990; Tilman et al., 1997; Maestre et al., 2009; see, however, Golubski et al., 2008; Lundholm, 2009). In our Vida simulations, there are three limiting factors (light, space, and a hypothetical growth-limiting nutrient), and, in simulations for spatially homogeneous and predictable world-spaces, three out of four species typically co-exist for hundreds of years, whereas, in simulations of heterogeneous and unpredictable worlds, all four species survive for hundreds of years, albeit to varying degrees of abundance depending on how and when the world-space is changed.

Given the consensus between the predictions of our simulations and those arising from more complex theories (e.g., Chesson, 2000; Haegeman and Etienne, 2008; O'Dwyer et al., 2009), two questions immediately surface: "Does Vida provide testable macro-ecological predictions?" and "Does it have any advantage over competing theories/models as an analytical or heuristic tool?" The following sections address these questions in light of the simulations presented here.

Niche differences and vegetative-reproductive linkages —

As in a previous study (Hammond and Niklas, 2009), the simulations designed to examine species co-existence manifest features that emerge from plant-plant interactions rather than from how individual species are parameterized. Among the most striking of these emergent features is the ability to predict the age at which the individual members of a particular species reach sexual maturity. This property is a consequence of Vida's computational logic, which links the mathematics describing vegetative growth to the ability to capture light and resources and thus reproduce. As a result, any niche difference defined on the basis of how vegetative growth occurs *a priori* results in a reproductive niche difference.

In this context, it is important to note that all four competing species were parameterized to produce propagules of the same number and size. In addition, all propagules, regardless of the species designation, were programmed to produce the same amount of stem and canopy biomass upon germination. Thus, none of the four species had a competitive advantage during the initial seedling establishment phase of community-assembly. Nevertheless, species C invariably dominated each plant community. The explanation for this success lies in how species are parameterized to grow vegetatively, which has an indirect affect on sexual reproduction.

Species A was modeled explicitly to mimic the vegetative growth of *A. alba*. Each of the other three species was parameterized to perform as a generic conifer species in all ways but one. Each was assigned a different photosynthesis growth constant (denoted by β_{12}; see Table 4.1, variable 20). Therefore, although species B – D had the same photosynthesis growth exponent (denoted by α_i; see Table 4.1, variable 5) and thus allocated the same fraction of new biomass to stem and canopy construction, the absolute amount of biomass allocated to stems and canopies differed in direct proportion to the numerical value of β_{12}. Because species C had the highest growth constant (β_{12}= 1.53), it vegetatively dominated all other species. In turn, its rapid vegetative growth allowed species C to garner disproportionately more energy and thus reach sexual maturity earlier compared to other species. This linkage between vegetative growth and reproductive success suggests that even very small vegetative niche differences can have a profound effect on the ability of a species to compete for space, light, and nutrients. In the worlds created by Vida, each plant is a fully integrated phenotype. Consequently, it is more biologically appropriate to think of vegetative and reproductive niche differences than of vegetative versus reproductive niche differences. In addition, *contra* the Neutral Biodiversity Theory (Hubble, 2001), Vida simulations predict that very small differences in functional traits can have a

profound affect on species co-existence, despite the obvious importance of demographic stochasticity (see O'Dwyer et al., 2009).

The time it takes to win or lose —

An average of 35 years was required before plant mortality resulted from light deprivation or nutrient depletion. Likewise, the number of individual plants representing each species fluctuated, often dramatically, over 10 to 50 year cycles even in spatially homogeneous and predictable world-spaces. Analyses of the data reveals that these cycles are of 17 yr. durations and the result of synchronized reproduction and mortality among equivalent age cohorts. Even longer time-spans for assessing species co-existence were required for communities growing in spatially heterogeneous and disturbed world-spaces. Analyses of the data from these simulations indicate that species survival depends on the frequency of disturbance in relation to the time required for an average plant to become reproductively mature. If these simulations shed light on the dynamics of real plant communities, they caution against assessing species co-existence (or invasive) patterns based on data collected over less than 10 years. They also suggest that the harmonics resulting from the interplay between reproductive and disturbance cycles can profoundly influence species co-existence and competitive exclusion.

Windows of opportunity, unpredictability, and co-existence —

Vida simulations also predict that species co-existence depends on the extent to which reproductive cycles and vegetative growth are in or out of phase with cycles of disturbance, particularly in very patchy world-spaces. That the success of a species depends in part on the frequency with which a world-space is disturbed and whether this frequency coincides with the time it takes a plant to reach sexual maturity is indicated by the fact that species co-existence achieved its highest level when world-spaces experienced 20-year disturbance cycles rather than 5-, 10-, or 40-year cycles. When

nutrient availability was randomly changed, rapidly growing species had an opportunity to reproduce (species C and D with 7 and 9 year reproductive cycles, respectively), whereas slower growing species were disadvantaged (species A and B with 25 and 15 year reproductive cycles). Five-year disturbance cycles disrupted the reproductive cycles of all four species; 10- and 20-year disturbance cycles permitted some adults of all four species to reproduce and thus increased average plant growth, size, and age. Species co-existence in world-spaces experiencing 40-year disturbance cycles mirrored the extent to which the four species co-existed in homogeneous and predictable world-spaces.

Two additional features emerged from Vida simulations of randomly changing world-spaces. With increasing patchiness, plant growth, size, and age decreased and the species that become sub-dominant or rare were increasingly difficult to predict. Our analyses indicate that the partitioning of the world-space into progressively smaller regions differing in nutrient availability intensifies plant-plant interactions because the spatial scale separating different nutrient levels contracts the spatial scale in which competition occurs. As the intensity of competition increases, mortality resulting from light deprivation or inadequate nutrient availability increases, and thus plant growth, size, and age decrease (although at different rates among the four species used in this study). Because propagules are ballistically distributed from parent plants during each reproductive cycle, the competitors in progressively smaller world-space regions become increasingly more difficult to predict and stochastic processes result in different outcomes during plant-plant competition. Species C always "won", albeit to lesser or greater degree, because of its incumbent advantage, i.e., this species establishes mature and reproductively viable individuals during the early phases of a community simulation, and some of these individuals survive and persist over many years to reseed spaces vacated by less competitive species. These simulations may shed some light on

a paradox in plant competition/co-existence theory. If, as inferred by theory, large adult size is the principal trait that confers competitive ability, why is it that, even in habitats where competition appears to be intense, the majority of resident species are relatively small? Our simulations indicate that the preponderance of small plant species may reflect the consequences of habitat disturbance or heterogeneity, which is one of a number of alternative explanations advanced by Aarssen et al. (2006)

Edge effects and buffer species —

Another feature emerging in simulations of unpredictable and heterogeneous world-spaces is the clustering of subdominant and rare species at the edges of the world-space, or at the borders between regions differing in nutrient availability, or around the canopies of large and old trees (for a recent review, of neighborhood or edge effects, see Ries et al., 2004). Our analyses of simulations show that the first plants to establish themselves at the edge of the world-space tend to persist regardless of their species affiliation because their canopies are, on average, less shaded than their conspecifics competing with other plants further from the edge. Likewise, plants with canopies that grow and extend into interior regions of the world-space that have low plant densities tend to experience less competition for light than plants growing in more densely occupied regions, again regardless of species affiliation. In either case, rare or subdominant species are repeatedly predicted to survive in small isolated clusters.

Another phenomenon favoring the survival of rare species is the death of old (and generally rare) plants, which creates light-gaps that are quickly occupied by the juveniles of these species. These old trees are typically established during the early phases of community growth when competition for resources is less intense. The death of such large plants is either stochastic or the result of Vida's plant-senescence sub-routine (see Hammond and Niklas, 2009). Regardless, their death can (and, in our simulations, often does) provide a buffer against species extinction by allowing a rare

species to regain (or retain) some of the territory it lost previously. These observations stand in opposition to conventional ecological theory, which predicts that larger individuals are likely to competitively exclude smaller individuals (for a critique of this conventional wisdom, see Keating and Aarssen, 2009).

Concluding remarks —

Vida simulations mimic stochastic, size-structured community dynamics and predicts its consequences on the likelihood of species co-existence. These simulations indicate that species co-existence is encouraged in mosaic and unpredictable world-spaces and that communities become increasing dominated by small and medium sized individuals as either habitat disturbance or spatial heterogeneity increases. These predictions resonate with the conclusions reached by G. E. Hutchinson toward the end of his essay on the factors favoring (and limiting) biodiversity when he wrote that "A final aspect of the limitation of possible diversity, and one that perhaps is of greatest importance, concerns what may be called the mosaic nature of the environment" and that "there are likely, above a certain limit of size, to be more species of small or medium sized organisms than of large organisms" (Hutchinson, 1959 p. 154).

Given the consistency between the predictions of other ecological models and those of Vida, an important question remains. What advantages can Vida offer? We believe there are at least three. First, unlike many ecological models, Vida predicts the behavior of a community using information derived from individual plants representative of their species. The stochasticity of simulated communities emerges as a direct result of the stochasticity of dispersal, death, and disturbance, while the scaling relationships among important variables (such as annual growth and body mass) emerge directly as a result of how neighboring plants interact in size-dependent ways. The second advantage is that unlike many individual-based simulators, Vida is capable of emulating communities composed of hundreds of real species over hundreds of years in

a relatively short time. For this reason, our results represent only a fraction of Vida's potential to reveal how and why plant species co-exist under different environmental conditions. The third advantage is that Vida relies on very few assumptions about how plants occupy space, or capture energy and resources. It also allows each of these assumptions to be relaxed or made more stringent depending on the questions being asked. In this way, Vida can be used to dissect the role and importance of each biological or abiotic variable. For all these reasons, we believe that Vida has a place in the classroom where students can construct real or artificial species and manipulate individual variables to discover the consequences on the dynamics of individual populations or entire communities of plants.

REFERENCES

AARSSEN, L. W., B. S. SCHAMP, AND J. PITHER. 2006. Why are there so many small plants? Implications for species coexistence. *Journal of Ecology* 94: 569 – 580.

ADAMS, T. P., D. W. PURVES, AND S. W. PACALA. 2007. Understanding height-structured competition in forests: Is there an R* for light? *Proceedings of the Royal Society of London, Ser. B* 274: 3039 – 3047 [and erratum (2008) 275: 591].

ALONSO, D., R. S. ETIENNE, AND A. J. MCKANE. 2006. The merits of neutral theory. *Trends in Ecology and Evolution* 21: 451 – 457.

BEGON, M. 1984. Density and individual fitness: asymmetric competition. *In* B. Shorrocks [ed.], *Evolutionary ecology*, pp 175 – 194. Blackwell Scientific Publications, Oxford, UK.

CANNELL, M. G. R. 1982. World Forest Biomass and Primary Production Data. Academic Press, London, UK.

CANTIANA, M. 1974. Experimental research into dendrometry and auxometry. Part II. Initial enquires concerning the biomass of the white fir. [in Italian] *Bulletin 5. Instituto di Assestamento Forestale.* University of Firenze, Florence, Italy.

CHAVE, J. 1999. Study of structural, successional and spatial patterns in tropical rain forests using TROLL, a spatially explicit forest model. *Ecological Modelling* 124: 233-254.

CHESSON, P. 2000. Mechanisms of maintenance of species diversity. *Annual Review of Ecology and Systematics* 31: 343 – 366.

DEBINSKI, D. M., AND R. D. HOLT. 2000. A survey and overview of habitat fragmentation experiments. *Conservation Biology* 14: 342 – 355.

ENQUIST, B. J., AND K. J. NIKLAS. 2001. Invariant scaling relations across tree-dominated communities. *Nature* 410: 655 – 660.

ERIKSSON, O. 1996. Regional dyanimcs of plants: A review of evidence for remnant source-sink and metapopulation. *Oikos* 77: 248 – 258.

GOLUBSKI, A. J., K. L. GROSS, AND G. G. MITTELBACH. 2008. Competition among plant species that interact with their environment at different spatial scales.*Proceedings of the Royal Society, Ser. B* 275: 1897 – 1906.

GROVES, R. H., AND J. D. WILLIAMS. 1975. Growth of skeleton weed (C*hondrilla juncea* L.) as affected by growth of subterranean clover (*Trifolium subterraneum* L.) and infection by *Puccinia chondrilla* Bubak and Syd. *Australian Journal of Agricultural Research* 26: 975 – 983.

HAEGEMAN, B, AND R. S. ETIENNE. 2008. Relaxing the zero-sum assumption in neutral biodiversity theory. *Journal of Theoretical Biology* 252: 288 – 294.

HAMMOND, S. T, AND K. J. NIKLAS. Emergent properties of plants competing *in silico* for space and light: Seeing the Tree from the Forest. *American Journal of Botany* 96: 1430 – 1444.

HARPER, J. L. 1982. Population biology of plants. Academic Press, London, UK.

HELLRIGL, B. 1974. Experimental research into dendrometry and auxometry. Part I. Tabulation of the productivity of tree biomass. [in Italian] *Bulletin 5.Instituto di Assestamento Forestale*. University of Firenze, Florence, Italy.

HUBBLE, S. P. 2001. The unified neutral theory of biodiversity and biogeography. Princeton University Press, Princeton, New Jersey, USA.

HUBBLE, S. P. 2005. Neutral theory in community ecology and the hypothesis of functional equivalence. *Functional Ecology* 19: 166 – 172.

HUTCHINSON, G. E. 1959. Homage to Santa Rosalia or why are there so any kinds of animals? *American Naturalist* 93: 145 – 159.

KEATING, L. M., AND L. W. AARSSEN. 2009. Big plants — do they limit species coexistence? *Journal of Plant Ecology* (UK) 2: 119 – 124.

LEVINE, J. M., AND J. HILLERISLAMBERS. 2009. The importance of niches for the maintenance of species diversity. *Nature* 461: 254 – 257.

LUNDHOLM, J. T. 2009. Plant species diversity and environmental heterogeneity:Spatial scale and competing hypotheses. *Journal of Vegetation Science* 20: 377 – 391.

NATHAN, R., AND H. MULLER-LANDAU. 2000. Spatial patterns of seed dispersal, their determinants and consequences for recruitment. *Tree* 15: 278 – 284.

NIKLAS, K. J. 1994. *Plant allometry*. University of Chicago Press, Chicago, Illinois, USA.

NIKLAS, K. J. 2000. Modeling fossil plant form-function relationships: a critique. *Paleobiology* 26: 289– 304.

NIKLAS, K. J. 2004. Plant allometry: is there a grand unifying theory? *Biological Reviews* 79: 871-889.

MAESTRE, F. T., R. M. CALLAWAY, F. VALLADARES, AND C. J. LORTIE. 2009. Refining the stress-gradient hypothesis for competition and facilitation in plant communities. *Journal of Ecology* 97: 199 – 205.

MAY, R. M. 1994. Conceptual aspects of the quantification of the extent of biological diversity. *Philosophical Transactions of the Royal Society of London, B*. 345: 13-20.

O'DWYER, J. P., J. K. LAKE, A. OSTLING, V. M. SAVAGE, AND J. L. GREEN.

2009. An integrative framework for stochastic size-structured community assembly. *Proceedings of the national Academy of Sciences, USA* 106: 6170 – 6175.

RIES, L., R. J. FLETCHER JR., J. BATTIN, AND T. D. SISK. 2004. Ecological responses of habitat edges: Mechanisms, models, and variability explained. *Annual Review of Ecology, Evolution and Systematics* 35: 491 – 522.

STALTER, R., S. DIAL, AND D. KINCAID. 2009. Ecological observations on a shortleaf pine (*Pinus echinata*) stand in the Piedmont of North Carolina. *Bartonia* 64: 37 –44.

STALTER, R., AND D. KINCAID. 2008. A 70-year history of arborescent vegetation of

Inwood Park, Manhattan, New York, U.S. *Arboriculture & Urban Forestry* 34: 245 – 251.

TAIZ, L., AND E. ZEIGER. 2002. Plant physiology. [3rd Edition] Sinauer, Sunderland, Massachusetts, USA.

TAKAHASHI, K., AND M. J. LECHOWICZ. 2008. Do interspecific differences in sapling growth traits contribute to the co-dominance of *Acer saccharum* and *Fagus grandifolia*? *Annals of Botany* 101: 103 – 109.

TILMAN, D. 1982. Resource competition and community structure. Princeton University Press, Princeton, New Jersey, USA.

TILMAN, D. 1990. Constraints and tradeoffs: Toward a predictive theory of competition and succession. *Oikos* 58: 3 – 15.

TILMAN, D. 2004. Niche tradeoffs, neutrality, and community structure: a stochastic theory of resource competition, invasion, and community assembly. *Proceedings of the National Academy of Sciences, USA* 101: 10854 – 10861.

TILMAN, D., C. L. LEHMAN, AND K. T. THOMSON. 1997. Plant diversity and ecosystem productivity: Theoretical considerations. *Proceedings of the National Academy of Sciences, USA* 94: 1857 – 1861.

WEINER, J. 1990. Asymmetric competition in plant populations. *Trends in Ecology and Evolution* 5: 360 – 364.

WESTOBY, M., AND I. J. WRIGHT. 2006. Land-plant ecology on the basis of functional traits. *Trends in Ecology and Evolution* 21: 261 – 268.

WILSON, S. D., AND D. TILMAN. 1991. Components of plant competition along an experimental gradient of nitrogen availability. *Ecology* 72: 1050 – 1065.

Chapter Five

At the onset of this project, Vida was conceived of as an opaque model that would attempt to answer whether simple biophysical principles and size dependant characteristics of individual plants result in properties typically observed for real plant populations or communities; in other words, do populations of simulated individual trees—parameterized using real-world data—competing for light and space, exhibit relationships seen within real-world communities.

As has been shown, using empirical data, simple geometric shapes, well established allometric relationships describing carbon allocation within individual trees, and simple rules related to an individual's ability to harvest light, Vida has been able to accurately simulate the growth of a real species, *Abies alba.* It has also been shown that, when these individuals begin to interact within simulations (i.e. when their canopies begin to overlap and direct competition for light begins), the simulated populations and exhibit characteristics typical of real-world tree populations and communities. Vida has also been used to test the effects of homogeneous, spatially heterogeneous and temporally heterogeneous environments, again showing relationships mirror what has been observed by ecologists.

The simplicity of Vida's equations and the results speak well for whether Vida is accurately modeling populations, but I am very aware of the risks of reading too much into the results of any computer simulation. Because the parameterization of a simulation, the rules under which the simulated world works, and the underlying code are all conceived of myself, the results can be seen as a reflection of my biases and assumptions. It is also possible for a model to reasonable facsimile of reality, but for the wrong reasons.

The Physiology of in silico *Organisms*

With Vida able to simulate the growth of individual trees, it is reasonable to ask how well Vida models the physiology of trees. The routines used to model light interception, conversion of light energy into mass, and the allocation of mass into organs are all based on empirical data, and therefore encode very complex physiological information., However, Vida itself does not model any aspect of plant physiology directly. For example, there is no code which alters growth based on water stress, mineral nutrition, effects of temperature, pH, etc. Trees simulated by Vida are more akin to crystals than organic trees.

Each of the allometric equations used within Vida, and the values used to parameterize species, are empirically derived from real-world species. Since these species represent unique results of organisms evolving within a physical world, they all possess an array of enzymes, cell and organ types that allow the organism to survive. All of the physiological processes are encoded within, and made invisible by, the allometric relationships used.

Imagine one had access to a series of extraterrestrial, photosynthetic organisms that occupy the same niche as plants. These organisms would have physiologies unlike anything humans had ever seen, but would have evolved with the same physical laws and would need to perform the same tasks as plants: mainly they would compete for light and space.

Nevertheless, the allometry one might see in these organisms might be radically different from that seen in plants. However, if I am correct in saying that population-level relationships are due to Apollonian circle packing (Chapter 3), then the population dynamics of these organisms would be the same as those of terrestrial plants. In short, physiology is important to the individual, but becomes irrelevant when one examines populations.

One can imagine improvements to Vida that would introduce more physiology into the simulations of individuals. Such changes would certainly allow one to better simulate how an individual species responds to a particular environment, one with heterogeneous pH, water or temperature, for example. With different responses to environmental conditions in line with a real plant's physiology, one would undoubtedly see species filling simulated environments for which they were best adapted, but the overall relationships seen within a simulation would remain unchanged.

Would such changes to Vida make it a better model? Again, it depends on what one wishes to model. If one wants to better model an individual species' growth under certain environmental conditions, or how two different species will compete for space given a heterogeneous environment, then changes to Vida would be warranted. As Vida is currently written, a species can be rather complex, but it can also be very simple. For example, a given species can make transitions in how it allocates carbon to its canopy, but it does not need to. If a species is parameterized so that Equations 2.8a and 2.8b have the same values, no transition is made, and the species is simpler than those that do. One could stick with such a philosophy in extending Vida's species to have requirements for pH, water, temperature, etc.

Possible Extensions to Vida

Improved geometry

One aspect in which Vida has consistently failed to mimic real-world trees is its treatment of canopy radius. Because Vida assumes all canopies are hemispherical, Vida consistently overestimates the canopy radius of trees it attempts to model. For example, if a tree has 35kg of leaf mass (a value similar to what was reported for *Abies alba* after 95 years of growth), and one sets leaf density (ρ_L) to 923.43kg/L , and a leaf thickness (H_L) to 0.000296 m (roughly the thickness of a piece of paper), one gets a canopy

spread of 28.5m when using Equation 2.5…truly a prodigious tree. Of course, few canopies are hemispheres. One could more accurately model the geometry of a tree using oblate or prolate spheroids and achieve more realistic canopy spreads:

$$e = \left[1 - \left(\frac{b}{a}\right)^2\right]^{1/2} \quad \text{(Equation 5.1)}$$

Oblate spheroid:

$$A_{L_{oblate}} = \pi\left(2a^2 + \frac{b^2}{e}\ln\frac{1+e}{1-e}\right) \quad \text{(Equation 5.2)}$$

$$V_{L_{oblate}} = \frac{4}{3}\pi a^2 b \quad \text{(Equation 5.3)}$$

Prolate spheroid:

$$A_{L_{prolate}} = 2\pi b\left(b + \frac{a}{e}\arcsin e\right) \quad \text{(Equation 5.4)}$$

$$V_{L_{prolate}} = \frac{4}{3}\pi a b^2 \quad \text{(Equation 5.5)}$$

where e is the eccentricity of an ellipsoid, a and b are the major and minor axes, A_L is the area of the canopy and V_L is the volume of the canopy.

By introducing changes in canopy geometry between species, one could see differences in how species are able to cope with excessive canopy loads. The Euler-Greenhill equation (Equation 2.26) describes the upper height limit a tree can reach before buckling.

$$H_{S_{Critical}} = 0.79\left(\frac{E}{g\rho_S}\right)^{1/3}\left(D_S\right)^{2/3} \quad \text{(Equation 2.26)}$$

Given $H_{S_{Critical}}$, one can describe the critical buckling mass a given column can support as

$$M_{L_{Critical}} = 0.785(\rho_S)(H_{S_{Critical}})\left(D_S{}^2\right) \quad \text{(Equation 5.6)}$$

The maximum height of most terrestrial plants appear to have a safety factor of four in relation to their Euler-Greenhill critical buckling heights (Niklas, 1994). It is reasonable that their critical buckling masses also possess a safety since trees must be

able to cope with increases in canopy mass during heavy rainstorms and during snowfall. Changes to the geometry of a simulated tree's canopy would mean different species to shed excess mass in the form of snow or rain in different ways, therefore affecting whether a given species would experience catastrophic structural failure.

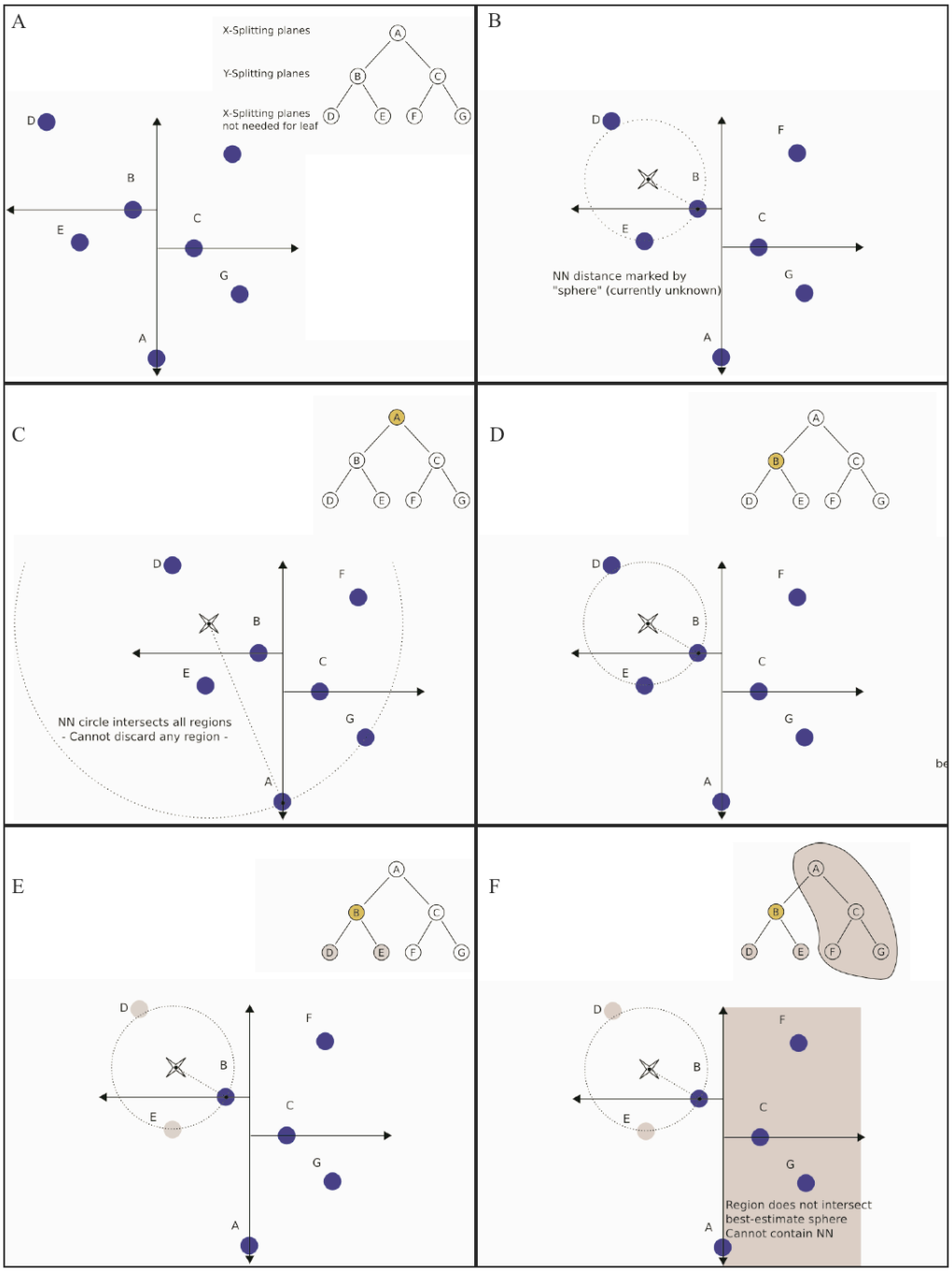
Improved Code

All of the proposed changes to Vida require additional calculations, and thus increase the time needed to run even simple simulations. The C programming language, being closer to assembly than an interpreted language like Python, allows one to write code which executes faster. However, not all of Vida would benefit from being rewritten in C. Parts of Vida, such as routines that write data to files on a disc, are limited by the read/write speeds of the media being used, and not necessarily by the language the code is written in. It is possible to rewrite specific subroutines in C and call them from Python using SWIG (Simplified Wrapper and Interface Generator) (Beazley, 1996, 2003). For example, one subroutine of Vida—the routine responsible for determining whether canopies are partially overlapping, wholly overlapping, or not touching—has been rewritten in C.

There are also a number of aspects of Vida that would increase in speed if rewritten. When Vida was first assembled, I had very little experience with Python and Vida's code clearly shows a learning curve, with newer code being much more efficiently written. There are also routines that could benefit from being rewritten. The routines responsible for determining a given object's nearest-neighbor, for example, is written so as to use a "naïve" methodology, in which each object tests each other object in the simulation space to determine whether overlap occurs. Such an approach can be described as

$$(N_{objects})^{N_{objects}-1} \qquad \text{(Equation 5.7)}$$

Figure 5.1: Nearest neighbor (NN) searching using a kd-tree. A: A 2D tree, rooted at point A. A line is drawn through point A. The dimensionality rotates to have lines drawn through points B and C. The dimensionality would rotate again for lines to be drawn through D, E, F and G. The resulting tree is shown in the upper right corner. B: A new point, represented by the cross, is introduced and a NN region described. C: Starting with point A, determine the distance between the new point, X, and A. D: Proceed to the nodes connected to A, and again determine the distance between X and the nodes in question. The shortest distance indicates the portion of the tree to traverse. E: Node B lies within the NN region of interest. The distance between X and the nodes connected to B (D and E) are examined, but B has the shortest distance. F: The majority of points were rapidly excluded and B identified as the NN (User_A1, 2008).


A
X-Splitting planes
Y-Splitting planes
X-Splitting planes
not needed for leaf
B
NN distance marked by
"sphere" (currently unknown)
C
NN circle intersects all regions
- Cannot discard any region -
D
E
F
Region does not intersect
best-estimate sphere
Cannot contain NN

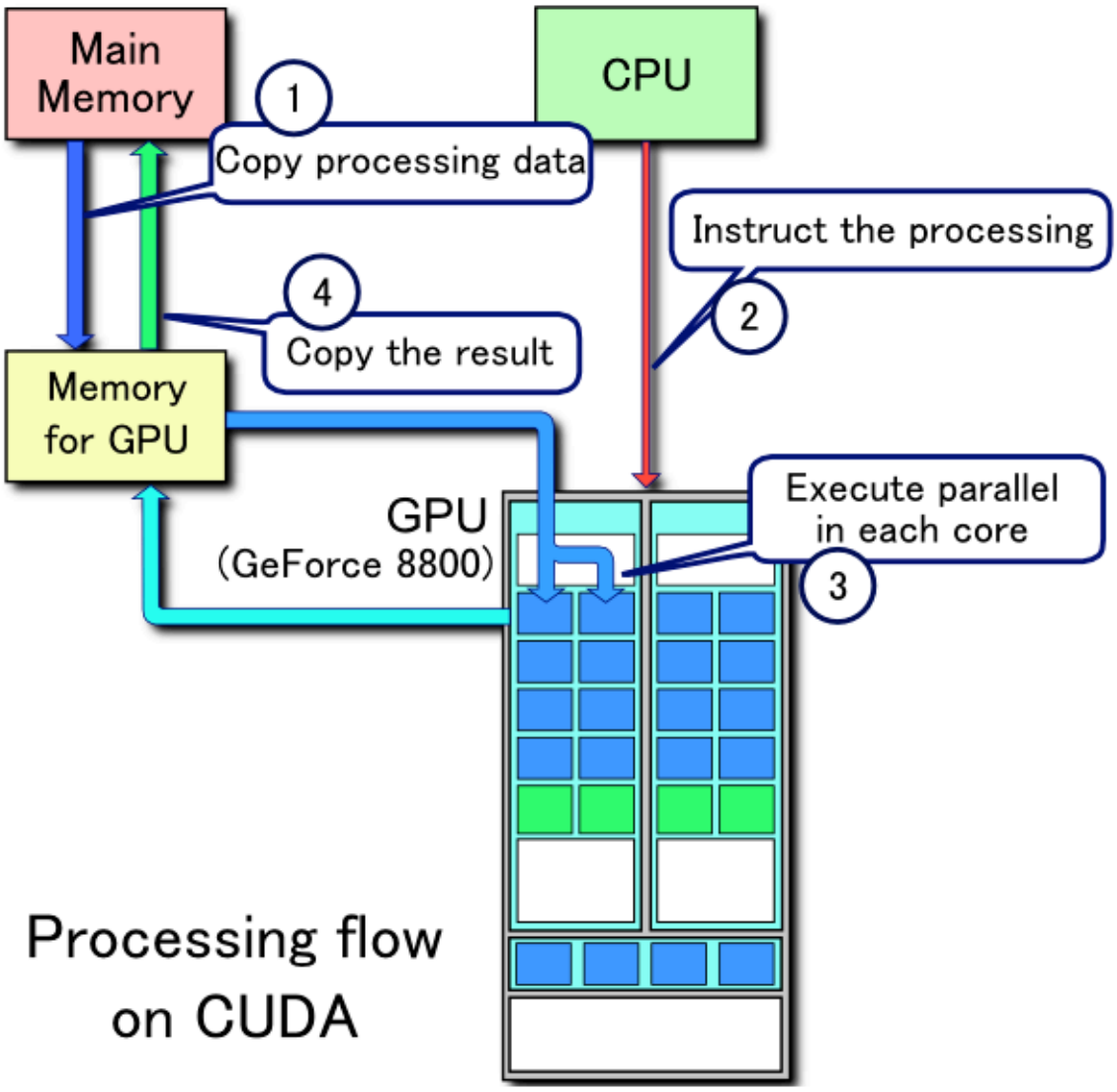


Figure 5.2: Schematic illustrating the relationship between a traditional CPU, main memory, a GPU, and parallel execution of code within a GPU (Tosaka, 2008).

As $N_{objects}$ increases, the amount of time necessary to test all objects increases exponentially. A better method would be to implement a *k* dimensional-tree (kd-tree), in which all points are linked together in a tree, allowing for rapid nearest neighbor determination (Lee, 1977; Berg, 2000) (Figure 5.1)

In addition to cleaning up code and improving routines, it is possible for Vida to be run on a supercomputing cluster, such as Cornell's V3 system. To do so, Vida would need to be rewritten in ways that would allow its calculations to work on a cluster of processors. Starting with Python 2.5, new multiprocessor methods have been introduced that allow an operator to run code on more than one processor in parallel. This can be used to run Vida on existing supercomputer clusters, set up a SETI@Home-type system, or utilize graphic processing units (GPUs) that use the Compute Unified Device Architecture (CUDA), such as implemented in many graphics cards manufactured by NVIDIA Corporation (Steele, 2009; Dematté, 2010) (Figure 5.2).

Even without alterations to Vida's code, the software is useful for future studies. One avenue currently being pursued is an examination of an ideal West, Enquist, Brown (WEB) type species to answer whether such a species performs equally well when compared to a "generic angiosperm" species (see Chapter 3). Another avenue of interest is to determine what happens when the scaling exponents used in allometric formula deviate more and more from what is observed in Nature. Is there a fitness corridor that real-world plants have evolved to fill? If so, hypothetical species with scaling exponents that are radically different from those found empirically should be less fit within Vida simulations, and thus ultimately be driven to extinction by species modeled after plants found in the real world. These, and other projects, are being pursued.

REFERENCES

BEAZLEY, D. M. 1996. SWIG: an easy to use tool for integrating scripting languages with C and C++. *Proceedings of the Fourth USENIX Tcl/Tk Workshop*.129–139.

BEAZLEY, D. M. 2003 Automated Scientific Software Scripting with SWIG. *Future Generation Computer Systems.* 19: 599-609.

DE BERG, M., M. VAN KREVELD, M. OVERMARS AND O. SCHWARZKOPF. 2000. *Computational Geometry: Algorithms and Applications, Second Edition*. Springer-Verlag

DEMATTÉ, L AND D. PRANDI. 2010. GPU Computing for Systems Biology. *Briefings in Bioinformatics*. Advance access. 1-11.

LEE, D. T. AND C. K. WONG. 1977. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica* 9 (1): 23–29.

NIKLAS, K. J. 1994. Allometries of Critical Buckling Height and Actual Plant Height. American Journal of Botany. 18(10): 1275-1279.

STEELE, J. E AND R. GEIST. 2009. Relighting Forest Ecosystems. *Lecture Notes in Computer Science*. 5875:55-66.

TOSAKA. 2008. CUDA_processing_flow_(En).PNG. This work is licensed under the Creative Commons Attribution 3.0 Unported License.

USER_A1. 2008. KDTree-animation.gif. This work is licensed under the Creative Commons Attribution-ShareAlike license, versions 3.0, 3.5, 2.0, and 1.0.

WEST, G. B, J. H. BROWN, AND B. J. ENQUIST. 1997. A General Model for the Origin of Allometric Scaling Laws in Biology. *Science*. 276: 122-126.

APPENDIX A

Directory and file structure overview

The software, Vida, requires various files be in specific locations. The following outlines the file structure necessary:

Vida/
- LICENSE.txt
- Vida.py
- Vita World Preferences.yml
- Read Me Files/
  - Vida HOWTO.txt
  - Vida README.txt
- Vida_Data/
  - Default_species.yml
  - geometry_utils.py
  - list_utils.py
  - progressBarClass.py
  - sdxf_utils.py
  - vdefaults.py
  - vdxfGraphics.py
  - vextract.py
  - vgraphics.py
  - vplantr.py
  - vworldr.py
- Species/
  - Abies alba.yml
  - Generic Angiosperm.yml
  - Generic Gymnosperm with Abies Photo constant.yml
  - Generic Gymnosperm with Cryptomeria Photo constant.yml
  - Generic Gymnosperm.yml
- Event Files/
  - events_examples.yml
- Placement Files/
  - ExamplePlacementFile.csv
  - TreePattern.csv
- Tools/
  - TreePattern.jpg
  - PlacementFileMaker.py

APPENDIX B

Files in top level (Vida) directory

Vida/LICENSE.txt


```
License:
Academic Software License Agreement
Sean T. Hammond (Developer) gives permission for you and your laboratory (Institution) to use
the Vida Version 0.5 software (Vida). The Developer allows researchers at your Institution to
copy and modify Vida, for internal, non-profit research purposes, on the following conditions:

1. The Vida software remains at your Institution and is not published, distributed, or
otherwise transferred or made available to other than Institution employees and students
involved in research under your supervision.

If you wish to obtain Vida for any commercial purposes, you will need to execute a separate
licensing agreement with the Developer and pay a fee. This includes, but is not limited to,
using Vida to provide services to outside parties for a fee. In that case please contact:

Sean T. Hammond
Care of:
Plant Biology Department
412 Mann Library Building
Cornell University
Ithaca, NY   14853
U.S.A
1.607.255.2131

2. You retain in Vida and any modifications to Vida, the copyright, trademark, or other notices
pertaining to Vida as provided by the Developer.

3. You provide the Developer with feedback on the use of the Vida software in your research,
and that the Developer is permitted to use any information you provide in making changes to the
Vida software. All bug reports and technical questions shall be sent to Sean T. Hammond, email:
sth24@gmail.com.

4. You acknowledge that the Developer and its licensees may develop modifications to Vida that
may be substantially similar to your modifications of Vida, and that the Developer and its
licensees shall not be constrained in any way by you in the Developer's or its licensees' use
or management of such modifications. You acknowledge the right of the Developer to prepare and
publish modifications to Vida that may be substantially similar or functionally equivalent to
your modifications and improvements, and if you obtain patent protection for any modification
or improvement to Vida you agree not to allege or enjoin infringement of your patent by the
Developer or by any of the Developer's licensees obtaining modifications or improvements to
Vida from the Developer.

5. If utilization of the Vida software results in outcomes which will be published, please
specify the version of Vida you used and cite source [A] below.

[A] Hammond S. T. and Niklas K. J. (2009) "Emergent properties of plants competing in silico
for space and light: Seeing the tree from the forest." American Journal of Botany
96(8):1430-1444.

6. Any risk associated with using the Vida software at your institution is with you and your
Institution. Vida is experimental in nature and is made available as a research courtesy "AS
IS," without obligation by the Developer to provide accompanying services or support. The
Developer AND THE AUTHORS EXPRESSLY DISCLAIM ANY AND ALL WARRANTIES REGARDING THE SOFTWARE,
WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES PERTAINING TO
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
```

Vida/Vida.py

Page 1 of 16

```
#!/usr/bin/env python
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import random
import math
import os
import os.path
import glob
import sys
#import pp #08.10.12
#print sys.path

runningInNodeBox=False
if __name__=="__builtin__":
  ##probably running in NodeBox

sys.path.append("/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-packages")
  runningInNodeBox=True

import copy
import time
import pickle
###append the path to basic data files
sys.path.append("Vida_Data")
import vworldr as worldBasics
import vplantr as defaultSpecies
import vgraphics as outputGraphics
import list_utils as list_utils
import geometry_utils as geometry_utils

import progressBarClass
###append the path to where species are
sys.path.append("Species")

###import default settings
import vdefaults as thePrefs
simulationName= thePrefs.simulationName
theWorldSize= thePrefs.theWorldSize
startPopulationSize=thePrefs.startPopulationSize
maxPopulation=thePrefs.maxPopulation
maxCycles=thePrefs.maxCycles
outputDirectory=thePrefs.outputDirectory
webDirectory=thePrefs.webDirectory
produceGraphics=thePrefs.produceGraphics
produceDXFGraphics=thePrefs.produceDXFGraphics
produceVideo=thePrefs.produceVideo
framesPerSecond=thePrefs.framesPerSecond
webOutput=thePrefs.webOutput
percentTimeStamp=thePrefs.percentTimeStamp
theView=thePrefs.theView
resumeSim=thePrefs.resumeSim
reloadSpeciesData=thePrefs.reloadSpeciesData
archive=thePrefs.archive
saveData=thePrefs.saveData
deleteCfdgFiles=thePrefs.deleteCfdgFiles
deletePngFiles=thePrefs.deletePngFiles
showProgressBar= thePrefs.showProgressBar
debug=thePrefs.debug
```

Page 2 of 16

```
debug2=thePrefs.debug2
timesToRepeat=thePrefs.timesToRepeat
sList=[]
seedPlacement="random"
produceStats=0
simulationFile=""
eventFile="events.yml"

###code sample from myself and gib
#import pp
#server = pp.Server()
#if (server.get_ncpus() == 1):
#    server.destory()

class Species1(defaultSpecies.genericPlant):
  ###The routine in defaultSpecies.genericPlant reads in default values from .yml file
  def __init__(self):
    ##super(type, obj) -> bound super object; requires isinstance(obj, type)
    super(Species1, self).__init__()


def saveSimulationPoint(theDirectory, theFileName, theGarden):
  #turn this off until I can understand why it isn't working
  if not runningInNodeBox:
    simulationStateFile =open(theDirectory + theFileName, 'wb')
    pickle.dump(theGarden, simulationStateFile)
    simulationStateFile.close()
    #print "simulation state saved to file"

def saveDataPoint (theDirectory, theFileName, theGarden):
  basicHeaders="Cycle #, Plant Name, Species, Mother Plant Name, X Location, Y Location, is a
seed, is mature, cycles until germination, Age, Mass of Stem, Mass of Canopy, # of Seeds, Mass
of all Seeds, Mass Stem+Mass Canopy, Mass Total, Diameter Stem, Radius Canopy, Height Stem,
Maximum Thickness of a Leaf, Height of Plant, Yearly Growth Stem (kg), Yearly Growth Canopy
(kg), Yearly Growth Stem+Canopy (kg), Yearly Growth Stem diameter (m), Yearly Growth Stem
Height (m), Average Growth Stem (kg), Average Growth Canopy (kg), Average Growth Stem+Canopy
(kg), Average Growth Stem diameter (m), Average Growth Stem Height (m), Cause of Death \n"
  photosynthesisHeaders="leaf is a hemisphere?, area canopy available for photosynthesis, area
canopy covered, fraction canopy 100% shaded, canopy transmittance, canopy transmittance impacts
photosynthesis?, faction of canopy that needs to be 0% shaded for survival"

  reproductionHeaders="make seeds?, age to start making seeds, height to start making seeds,
fraction of mass used for seeds, Maximum Seed Mass, Where seeds grow, Seed dispersal method,
fraction of seeds which fail to germinate, delay in germination, fraction of maximum seed mass
needed to germinate, fraction of seed mass converted to plant mass"

  allometryHeaders="B1 in Ms=B1*(Mt^a1), a1 in Ms=B1*(Mt^a1), B2 in Mlyoung=B2*(Ms^a2), a2 in
Mlyoung=B2*(Ms^a2), B3 in Mlmature=B3*(Ms^a3), a3 in Mlmature=B3*(Ms^a3), B4 in Ds=B4*(Ms^a4),
a4 in Ds=B4*(Ms^a4), B5 in Hs=[B5*(Ds^a5)]-B6, a5 in [B5*(Ds^a5)]-B6, B6 in Hs=[B5*(Ds^a5)]-B6,
B7 in Mg=[B7*(Ml^a7)]/ area canopy 100% uncovered,  a7 in Mg=[B7*(Ml^a7)]/ area canopy 100%
uncovered, B8 in Mpt=B8*(Al^a8), a8 in Mpt=B8*(Al^a8)"


  #theHeader="Plant Name, Mother Plant, Species, X Location, Y Location, Mass Stem, Mass Leaf,
# Seeds, Mass all Seeds, Radius Stem, Radius Leaf, Height Plant, Cause of Death \n"
  theHeader= basicHeaders
  thePlantList=[]
  theSeedList=[]
  theCorpseList=[]
  thePlantList.append(theHeader)
  theSeedList.append(theHeader)
  theCorpseList.append(theHeader)
  for plant in theGarden.soil:
    if plant.age>0:

theData="%i,%s,%s,%s,%f,%f,%s,%s,%i,%i,%f,%f,%i,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,
%f,%f,%s \n" %
(theGarden.cycleNumber,plant.name,plant.nameSpecies,plant.motherPlant,plant.x,plant.y,plant.isSe
```

Page 3 of 16

```
ed,plant.isMature,plant.countToGerm,plant.age,plant.massStem,plant.massLeaf,len(plant.seedList),
plant.massSeedsTotal,plant.massStem+plant.massLeaf,plant.massTotal,plant.radiusStem*2,plant.radi
usLeaf,plant.heightStem,plant.heightLeafMax,plant.z,plant.GMs,plant.GMl,plant.GMs+plant.GMl,2.0*
plant.GRs,plant.GHs,plant.massStem/plant.age,plant.massLeaf/plant.age,(plant.massStem+plant.mass
Leaf)/plant.age,(plant.radiusStem*2)/plant.age,plant.heightStem/plant.age,"na")
    else:

theData="%i,%s,%s,%s,%f,%f,%s,%s,%i,%i,%f,%f,%i,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,
%f,%f,%s \n" %
(theGarden.cycleNumber,plant.name,plant.nameSpecies,plant.motherPlant,plant.x,plant.y,plant.isSe
ed,plant.isMature,plant.countToGerm,plant.age,plant.massStem,plant.massLeaf,len(plant.seedList),
plant.massSeedsTotal,plant.massStem+plant.massLeaf,plant.massTotal,plant.radiusStem*2,plant.radi
usLeaf,plant.heightStem,plant.heightLeafMax,plant.z,plant.GMs,plant.GMl,plant.GMs+plant.GMl,2.0*
plant.GRs,plant.GHs,0,0,0,0,0,"na")
    if plant.isSeed:
      theSeedList.append(theData)
    else:
      thePlantList.append(theData)
  for plant in theGarden.deathNote:
    if plant.age>0:

theData="%i,%s,%s,%s,%f,%f,%s,%s,%i,%i,%f,%f,%i,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,
%f,%f,%s \n" %
(theGarden.cycleNumber,plant.name,plant.nameSpecies,plant.motherPlant,plant.x,plant.y,plant.isSe
ed,plant.isMature,plant.countToGerm,plant.age,plant.massStem,plant.massLeaf,len(plant.seedList),
plant.massSeedsTotal,plant.massStem+plant.massLeaf,plant.massTotal,plant.radiusStem*2,plant.radi
usLeaf,plant.heightStem,plant.heightLeafMax,plant.z,plant.GMs,plant.GMl,plant.GMs+plant.GMl,2.0*
plant.GRs,plant.GHs,plant.massStem/plant.age,plant.massLeaf/plant.age,(plant.massStem+plant.mass
Leaf)/plant.age,(plant.radiusStem*2)/plant.age,plant.heightStem/plant.age,plant.causeOfDeath)
    else:

theData="%i,%s,%s,%s,%f,%f,%s,%s,%i,%i,%f,%f,%i,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,
%f,%f,%s \n" %
(theGarden.cycleNumber,plant.name,plant.nameSpecies,plant.motherPlant,plant.x,plant.y,plant.isSe
ed,plant.isMature,plant.countToGerm,plant.age,plant.massStem,plant.massLeaf,len(plant.seedList),
plant.massSeedsTotal,plant.massStem+plant.massLeaf,plant.massTotal,plant.radiusStem*2,plant.radi
usLeaf,plant.heightStem,plant.heightLeafMax,plant.z,plant.GMs,plant.GMl,plant.GMs+plant.GMl,2.0*
plant.GRs,plant.GHs,0,0,0,0,0,plant.causeOfDeath)
    theCorpseList.append(theData)
  if len(thePlantList)>1:
    saveDataFile =open(theDirectory+"Plants/"+ theFileName, 'wb')
    saveDataFile.writelines(thePlantList)
    saveDataFile.close()
  if len(theSeedList)>1:
    saveDataFile =open(theDirectory+"Seeds/"+ theFileName, 'wb')
    saveDataFile.writelines(theSeedList)
    saveDataFile.close()
  if len(theCorpseList)>1:
    saveDataFile =open(theDirectory+"Corpses/"+ theFileName, 'wb')
    saveDataFile.writelines(theCorpseList)
    saveDataFile.close()
  thePlantList=[]
  theSeedList=[]
  theCorpseList=[]
  #print "simulation data saved to file



def makeDirectory(theDirectory):
  if not os.path.exists(theDirectory):
    os.mkdir(theDirectory)
  else:
    ###all this adds an increasing numberic suffix to the folders so
    ###one doesn't overwrite things
    while os.path.exists(theDirectory):
      basicPath=os.path.split(theDirectory)[0]
      i=0
      for n in basicPath:
```

Page 4 of 16

```
        i=i-1
        if n=="/": break
        if not basicPath[i].isdigit():break
      theSuffix=""
      if i<-1:
        theSuffix=int(basicPath[i+1:])+1
        basicPath=basicPath[0:i+1]
      else:
        theSuffix=1
      basicPath=basicPath+str(theSuffix)
      theDirectory=basicPath+"/"
    os.mkdir(theDirectory)
  return theDirectory

def correctType(theItem):
  returnValue="na"
  if theItem.isdigit():
      returnValue=int(theItem)
  else:
    try:
      returnValue=float(theItem)
    except ValueError:
      try:
        returnValue=str(theItem)
      except ValueError:
        returnValue="na"
  return returnValue

def checkSeedPlacementList(seedPlacementList):
  #remove blank lines that are just \n
  i=0
  for aLine in seedPlacementList:
    if aLine=="\n":seedPlacementList[i]=""
    i=i+1
  i=0
  for aLine in seedPlacementList:
    if not aLine=="":
      aLine=aLine.rstrip("\n")
      aLine=aLine.split(",")
      for j in range(len(aLine)):
        if not j==0: aLine[j]=aLine[j].strip()
      theLength=len(aLine)
      if theLength>5: del aLine[5:theLength] #if it is too long, just chop it
      #if too short, make it longer
      if theLength<5:
        toAdd=5-theLength
        for k in range(toAdd):
          aLine.append("0.0")
      aLine= map(correctType, aLine)
      seedPlacementList[i]=aLine
    i=i+1
  i=0
  printErrorMessage=0
  for aLine in seedPlacementList:
    if aLine=="":
      del seedPlacementList[i]
      i=i+1
    elif not type(aLine[0])==str or not type(aLine[1])==float or not type(aLine[2])==float or
not type(aLine[3])==int or not type(aLine[4])==float:
      printErrorMessage=1
      del seedPlacementList[i]
      i=i+1
  if printErrorMessage:
    print "***Improper seeding file format..."
    print "     Questionable lines will be ignored."
  return seedPlacementList
```

```
def correctSList(sList):
  startPopulationSize=thePrefs.startPopulationSize
  ###this is probably a list###
  sList=sList[1:-1]
  sList=sList.split(",")
  sList=map(correctType, sList)
  i=0
  showSAlert1=0
  showSAlert2=0
  showSAlert3=0
  for x in sList:
    if i==0 and type(x)==int:
      #the first item is a number. This is not correct
      #insert the default species here
      showSAlert1=1
      showSAlert2=1
      sList.insert(0, "_random_")
    if i+1<=len(sList) and type(sList[i])==str:
      #the last item in the list is a string
      #put a int after it
      showSAlert1=1
      showSAlert3=1
      sList.insert(i+1, thePrefs.startPopulationSize)
    elif type(sList[i])==str and type(sList[i+1])==str:
      #this is probably a two species names next to one another
      #insert a int between them
      showSAlert1=1
      showSAlert3=1
      sList.insert(i+1, thePrefs.startPopulationSize)
    elif type(sList[i])==int and type(sList[i-1])==int:
      showSAlert1=1
      showSAlert2=1
      sList.insert(i, "_random_")
    i=i+1
  if showSAlert1: print "***Improper seeding (list) format..."
  if showSAlert2: print "     Inserting random species..."
  if showSAlert3: print "     Inserting default starting population size..."
  if showSAlert1: print "         input set to: %s." % (sList)
  return sList

def main():
  #why do only these need to be reminded they are global?
  global simulationFile
  global theWorldSize
  global startPopulationSize
  global seedPlacement
  global sList

  #Import Psyco if possible
  try:
    import psyco
    psyco.log()
    psyco.full()
  except ImportError:
    pass

  CFDGtext=""

  ##################################
  ###experiments in importing events
  import yaml
  if os.path.exists(eventFile):
    print "***Loading event file: %s***" % (eventFile)
    theFile=open(eventFile)
    eventData=yaml.load(theFile)
    theFile.close
    eventTimes=eventData.keys()
  else:
```



```
    eventTimes=[]
  ##################################

  if debug==1: print "***debug is on***"

  theGarden= worldBasics.garden()
  theGarden.platonicSeeds={}
  theGarden.theRegions=[]


  ####

  ########Check for multiple species. If none, use default
  fileList=os.listdir("Species")
  ymlList=[]
  pythonList=[]
  useDefaultYml=True
  print "***Checking for species...***"
  for file in fileList:
    theExtension=os.path.splitext(file)[1]
    if theExtension==".yml":
      #add this file to the list of yaml files
      ymlList.append(file)
      useDefaultYml=False
    elif theExtension==".py":
      #this might be override info for a species
      #add this file to the list of species python files
      #this isn't implemented
      pythonList.append(file)
  fileList=[]
  #########

  if resumeSim==1 and not simulationFile=="":
    simulationFile=open(simulationFile, 'r')
    theGarden=pickle.load(simulationFile)
    simulationFile.close()
    theWorldSize=theGarden.theWorldSize
    print "***Resuming Simulation: %s as %s***" % (theGarden.name, simulationName)
    theGarden.name=simulationName
    startPopulationSize=theGarden.numbPlants
    ##this should reload species data.
    ###Important if you want to compare runs
    if reloadSpeciesData==1:
      ###fileLoc will be different for each species eventually
      fileLoc="Vida_Data/default_species.txt"
      for item in theGarden.soil:
        item.importPrefs(fileLoc)
  else:
    theGarden.makePlatonicSeedDict(ymlList, SpeciesL)
    print "***Species loaded.***"
    theGarden.name=simulationName
    theGarden.theWorldSize=theWorldSize
    print "***Beginning Simulation: %s***" % (simulationName)

  theGarden.showProgressBar=showProgressBar

  print "     World size: %ix%i" % (theWorldSize, theWorldSize)
  print "     Maximum population size will be: %i" % (maxPopulation)
  print "                  and"
  print "     Running simulation for %i cycles" % (maxCycles)
  print "                  (whichever comes first)"
  print "     Starting population size: %i" % (startPopulationSize)
  if theGarden.carbonAllocationMethod==0:
    print "     Plants will allocate carbon to stem and leaf using methods defined by the
species."
  else:
    print "     All plants will allocate carbon to stem and leaf using method %i" %
(theGarden.carbonAllocationMethod)
```



```

  print ""
  if produceGraphics==1:
    print "     Graphical output will be produced."
    if theView==1:
      print "          Graphical output will be a bottom-up view."
    elif theView==2:
      print "          Graphical output will be a top-down view."
    elif theView==3:
      print "          Graphical output will be a side-view."
    elif theView==12:
      print "          Graphical output will be a combination bottom-up and top-down view."
    elif theView==21:
      print "          Graphical output will be a combination top-down and bottom-up view."
    elif theView==23:
      print "          Graphical output will be a combined top-down and side view."
    elif theView==13:
      print "          Graphical output will be a combined bottom-up and side view."
    elif theView==123:
      print "          Graphical output will be a combination bottom-up, top-down and side view."
    if produceVideo==1:
      print "          Graphical output will include a %s frame/second video." % (framesPerSecond)
  ###I think this is where to start the times to repeat bit
  for x in range(timesToRepeat):
    ###make necessary directories
    outputDirectory="Output-"+simulationName+"/"
    outputDirectory=makeDirectory(outputDirectory)
    if produceGraphics==1 or produceDXFGraphics==1:
      outputGraphicsDirectory = outputDirectory +"Graphics/"
      makeDirectory(outputGraphicsDirectory)
      if produceDXFGraphics==1:
        outputDXFGraphicsDirectory = outputGraphicsDirectory +"DXF/"
        makeDirectory(outputDXFGraphicsDirectory)
      ###SUPER IMPORTANT!!!!
      ####DON'T USE SPACES IN DIR NAMES!
      ####COMMAND LINES HATE THAT SHIT
      if theView==1:
        outputGraphicsDirectory = outputGraphicsDirectory +'bottom-up/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==2:
        outputGraphicsDirectory = outputGraphicsDirectory +'top-down/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==3:
        outputGraphicsDirectory = outputGraphicsDirectory +'side/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==12:
        outputGraphicsDirectory = outputGraphicsDirectory +'combined-bottom-top/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==21:
        outputGraphicsDirectory = outputGraphicsDirectory +'combined-top-bottom/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==13:
        outputGraphicsDirectory = outputGraphicsDirectory +'combined-bottom-side/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==23:
        outputGraphicsDirectory = outputGraphicsDirectory +'combined-top-side/'
        makeDirectory(outputGraphicsDirectory)
      elif theView==123:
        outputGraphicsDirectory = outputGraphicsDirectory +'combined-bottom-top-side/'
        makeDirectory(outputGraphicsDirectory)
    if not archive=="n":
      saveDirectory = outputDirectory+"Save_points/"
      makeDirectory(saveDirectory)
    if not saveData=="n":
      dataDirectory = outputDirectory+"Simulation_data/"
      makeDirectory(dataDirectory)
      makeDirectory(dataDirectory+"Seeds/")
      makeDirectory(dataDirectory+"Plants/")
```



```
      makeDirectory(dataDirectory+"Corpses/")


    if resumeSim==0:
      #2008.11.06 Moved a huge block of code related to placing seeds to vworldr.py
      theGarden.placeSeed(seedPlacement, sList, startPopulationSize, useDefaultYml, ymlList)
    if produceGraphics and CFDGtext=="":
      CFDGtext=outputGraphics.initCFDGText(theGarden, theView, percentTimeStamp, 50.0)
    #######
    cycleNumber=0
    print "\n***Running simulation.***"
    if not showProgressBar and not runningInNodeBox:
      theProgressBar= progressBarClass.progressbarClass(maxCycles,"^") #why -1? because index
0. So if total=100, 0-99.
    #print "Cycle, plant age, Mass Stem, Mass Leaf, # Seeds, Mass all Seeds, Radius Stem,
Radius Leaf, Height Plant, areaPhoto, new mass"
    while (theGarden.numbPlants<=maxPopulation and cycleNumber<=maxCycles) and
(theGarden.numbPlants+theGarden.numbSeeds)>0:
      ##########################################################################################
      ####Experimental scripting event stuff                                                  #
      if cycleNumber in eventTimes:                                                           #
        for aItem in eventData[cycleNumber]:                                                 #
          for aKey in aItem.keys():                                                         #
            if aKey=='Garden':                                                             #
              if debug==1: print "debug: A garden related event has been triggered."   #
              theDict=aItem[aKey][0]                                                       #
              gardenAttrs=theDict.keys()                                                   #
              for theGardenAttr in gardenAttrs:                                            #
                setattr(theGarden, theGardenAttr, theDict[theGardenAttr])     #
              gardenAttrs=""
            elif aKey=="Killzone" or aKey=="Safezone":
              if debug==1: print "debug: generation of a zone event has been triggered."
              theDict=aItem[aKey][0]                                                       #
              zoneAttrs=theDict.keys()
              zoneX=float(theDict['x'])
              zoneY=float(theDict['y'])
              zoneSize=float(theDict['size'])
              zoneShape=theDict['shape']
              if zoneShape not in ['circle','square']:
                print "***WARNING: improper zone shape defined. Defaulting to square.***"
                zoneShape='square'
              zoneTarget=theDict['target']
              if zoneTarget not in ['all','plants','seeds']:
                print "***WARNING: improper zone target defined. Defaulting to all.***"
                zoneTarget='all'
              if zoneShape=='circle':
                killThese=[]
                for theObject in theGarden.soil:
                  if theObject.isSeed:
                    r=theObject.radiusSeed
                  else:
                    r=theObject.radiusStem
                  theResult=geometry_utils.checkOverlap(theObject.x, theObject.y, r, zoneX,
zoneY, zoneSize)
                  if theResult>0 and aKey=='Killzone':
                    if zoneTarget=='all' or (theObject.isSeed and zoneTarget=='seeds') or (not
theObject.isSeed and zoneTarget=='plants'):
                      killThese.append(theObject)
                  elif aKey=='Safezone':
                    if theResult==0:
                      killThese.append(theObject)
                    elif theResult>0:
                      if (theObject.isSeed and zoneTarget=='plants') or (not theObject.isSeed
and zoneTarget=='seeds'):
                        killThese.append(theObject)

                for theObject in killThese:
```

```
                    theObject.causeOfDeath="zone"
                    theGarden.kill(theObject)
            elif aKey=="Seed":
                if debug>=1: print "debug: A seeding related event has been triggered."   #
                theDict=aItem[aKey][0]                                                   #
                seedingInfo=theDict.keys()                                               #
                for infoItem in seedingInfo:
                  if infoItem=="number" and not seedPlacement=="fromFile":
startPopulationSize=theDict[infoItem]
                  if infoItem=="species":
                    sList=theDict[infoItem]
                    if sList=="random": sList=[]
                  if infoItem=="placement": seedPlacement=theDict[infoItem]
                  if seedPlacement=="hexagon": seedPlacement="hex" #just make sure it is
consistant
                  if os.path.isfile(seedPlacement):
                    theFile=open(seedPlacement)
                    try:
                      sList=theFile.readlines()
                    finally:
                      theFile.close()
                    sList=checkSeedPlacementList(sList)
                    startPopulationSize=len(sList)
                    seedPlacement="fromFile"
                if not seedPlacement=="fromFile" and not sList==[]:
                  newList=[]
                  for j in range(startPopulationSize):
                    newList.append(sList)
                  #print sList
                  sList=newList
                  newList=[]
                  #print sList
                theGarden.placeSeed(seedPlacement, sList, startPopulationSize, useDefaultYml,
ymlList)
            elif aKey=="Region":
                if debug: print "debug: Region event detected..."
                theDict=aItem[aKey][0]
                regionAttrs=theDict.keys()
                theRegionName=str(theDict['name'])
                if debug: print "debug: Region %s event detected." % (theRegionName)
                regionNames=[]
                for i in theGarden.theRegions:
                  regionNames.append(i.name)
                if theRegionName in regionNames:
                  for j in theGarden.theRegions:
                    if j.name==theRegionName:
                      theRegion=j
                      break
                  for aAttr in regionAttrs:
                    if not getattr(theRegion,aAttr,"does not exist")==theDict[aAttr]:
                      if debug: print "debug: Region %s has had a change in one or more
attributes." % (theRegionName)
                        updatePlants=True
                        break
                  #if (not theRegion.size==theDict['size']) or (not theRegion.x==theDict['x']) or
(not theRegion.y==theDict['y']) or (not theRegion.shape==theDict['shape']):
                    # if debug:print "debug: a region has changed shape, size or location"
                    # updatePlants=True
                    ##now just read in the values#
                    if debug: print "debug: Updating attributes for region %s." % (theRegionName)
                    for theRegionAttr in regionAttrs:                                    #
                      setattr(theRegion, theRegionAttr, theDict[theRegionAttr])      #
                    if updatePlants:
                      if debug:print "debug: updating plants with changed region info"
                      for aPlant in theGarden.soil:
                        plantX=aPlant.x
                        plantY=aPlant.y
                        if theRegion.shape=='square':
```



```
                          inSubregion=geometry_utils.pointInsideSquare(theRegion.x, theRegion.y,
theRegion.size, plantX, plantY)
                        elif theRegion.shape=='circle':
                          #size needs to be radius but region defines diameter
                          inSubregion=geometry_utils.pointInsideCircle(theRegion.x, theRegion.y,
theRegion.size/2.0, plantX, plantY)
                        if inSubregion:
                          if not theRegion in aPlant.subregion:
                            aPlant.subregion.append(theRegion)
                            #print "\nX: %f  Y: %f  In region: %s" % (plantX, plantY, newRegion)

                    #print theRegion.size
                  else:
                    newRegion=worldBasics.garden()
                    newRegion.name=theRegionName
                    ###these are default values###
                    newRegion.x=0.0                #
                    newRegion.y=0.0                #
                    newRegion.worldSize=1.0        #
                    newRegion.shape='square'       #
                    ############################
                    ##now just read in the values#
                    if debug: print "debug: Making attributes for region"
                    for theRegionAttr in regionAttrs:                                    #
                      setattr(newRegion, theRegionAttr, theDict[theRegionAttr])      #
                    theGarden.theRegions.append(newRegion)
                    for aPlant in theGarden.soil:
                      plantX=aPlant.x
                      plantY=aPlant.y
                      if newRegion.shape=='square':
                        inSubregion=geometry_utils.pointInsideSquare(newRegion.x, newRegion.y,
newRegion.size, plantX, plantY)
                      elif newRegion.shape=='circle':
                        #size needs to be radius but region defines diameter
                        inSubregion=geometry_utils.pointInsideCircle(newRegion.x, newRegion.y,
newRegion.size/2.0, plantX, plantY)
                      if inSubregion:
                        if not newRegion in aPlant.subregion:
                          aPlant.subregion.append(newRegion)
                          #print "\nX: %f  Y: %f  In region: %s" % (plantX, plantY, newRegion)

                    newRegion=""


            theDict=[]#just clear this to free up the memory
        ##############################################################################
        theGarden.cycleNumber=cycleNumber
        if not showProgressBar and not runningInNodeBox:
          theProgressBar.update(cycleNumber)
          #print "%i, %i, %f, %f, %i, %f, %f, %f, %f, %f, %f" % (cycleNumber-1,
theGarden.soil[0].age, theGarden.soil[0].massStem, theGarden.soil[0].massLeaf,
len(theGarden.soil[0].seedList), theGarden.soil[0].massSeedsTotal,
theGarden.soil[0].radiusStem, theGarden.soil[0].radiusLeaf,
theGarden.soil[0].heightStem+theGarden.soil[0].heightLeafMax,
theGarden.soil[0].areaPhotosynthesis, theGarden.soil[0].massFixed)

        ###START OF SEEING CHANGES TO SPECIES FOLDER
        ########Check for multiple species. If none, use default
        fileList=os.listdir("Species")
        #print fileList
        #ymlList=[]
        #print '***Checking for species...***'
        #for file in fileList:
        # theExtension=os.path.splitext(file)[1]
        # if theExtension=='.yml':
        #   #add this file to the list of yaml files
        #   ymlList.append(file)
        #   useDefaultYml=False
```



```
    #fileList=[]
    ##########









    if debug>=1: print "number of plants: "+str(theGarden.numbPlants)
    if debug>=1: print "number of seeds: "+str(theGarden.numbSeeds)

    #generate graphics if requested
    if produceDXFGraphics:
      theData=vdxfGraphics.makeDXF(theGarden)
      theFileName= simulationName+str(cycleNumber)
      vdxfGraphics.writeDXF(outputDXFGraphicsDirectory, theFileName, theData)
    if produceGraphics:
      theData=outputGraphics.makeCFDG(theView, CFDGtext, theGarden, cycleNumber)
      if webOutput==0:
        if theView==1:
          cfdgFileName= simulationName +'-bottom-'+str(cycleNumber)
        elif theView==2:
          cfdgFileName= simulationName +'-top-'+str(cycleNumber)
        elif theView==3:
          cfdgFileName= simulationName +'-side-'+str(cycleNumber)
        elif theView==12:
          cfdgFileName= simulationName +'-bottom-top-'+str(cycleNumber)
        elif theView==21:
          cfdgFileName= simulationName +'-top-bottom-'+str(cycleNumber)
        elif theView==13:
          cfdgFileName= simulationName +'-bottom-side-'+str(cycleNumber)
        elif theView==23:
          cfdgFileName= simulationName +'-top-side-'+str(cycleNumber)
        elif theView==123:
          cfdgFileName= simulationName +'-bottom-top-side'+str(cycleNumber)
        outputGraphics.writeCFDG(outputGraphicsDirectory, cfdgFileName, theData)
      else:
        cfdgFileName= simulationName
        outputGraphics.writeCFDG(outputGraphicsDirectory, cfdgFileName, theData)
        outputGraphics.outputPNGs(outputGraphicsDirectory, webDirectory)
        #outputGraphics.deleteCFDGFiles(outputGraphicsDirectory)

    ###go through the soil list and germinate seed or grow plant
    if theGarden.showProgressBar:
      print"***Allowing plants a turn to grow***"
      theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
      theBar=0
    for obj in theGarden.soil[:]:
      if obj.isSeed:
        obj.germinate(theGarden)
      else:
        obj.growPlant(theGarden)
      if theGarden.showProgressBar:
        theBar=theBar+1
        theProgressBar.update(theBar)


    ###deal with violaters of basic physics
    theGarden.causeRandomDeath()
    theGarden.checkSenescence()
    theGarden.removeOffWorldViolaters()
    theGarden.removeEulerGreenhillViolaters()
    theGarden.removeOverlaps()


    ###sort the garden.soil by height of the plants.Ordered shortest to tallest
    theGarden.soil= list_utils.sort_by_attr(theGarden.soil, "heightStem")
    ###flip the list so it's ordered tallest to shortest
    theGarden.soil.reverse()
```



```
    ###work out shading
    worldBasics.determineShade(theGarden)

    ###Calculate the amount of carbon each plant will have to start the next turn
    if theGarden.showProgressBar:
      print "***Calculating new mass from photosynthesis***"
      theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
      i=0
    for plant in theGarden.soil[:]:
      if plant.isSeed==False:
        plant.massFixed=plant.calcNewMassFromLeaf(theGarden)
        if plant.massFixed<-1.0:
          plant.causeOfDeath="lack of light"
          theGarden.kill(plant)
        else:
          plant.massFixedRecord.append(plant.massFixed)
          while len(plant.massFixedRecord)>plant.numYearsGrowthMemory:
            plant.massFixedRecord.pop(0)
      if theGarden.showProgressBar:
        i=i+1
        theProgressBar.update(i)




    if archive=="a":
      fileName=simulationName+'-'+str(cycleNumber)+'.pickle'
      saveSimulationPoint(saveDirectory, fileName, theGarden)

    if saveData=="a":
      fileName=simulationName+'-'+str(cycleNumber)+'.csv'
      saveDataPoint(dataDirectory, fileName, theGarden)
    #print theGarden.deathNote
    theGarden.deathNote=[]

    cycleNumber= cycleNumber+1

    ###pause if you're making web graphic
    #if webOutput==1:
    # time.sleep(5)

  if archive=="e":
    fileName=simulationName+'-'+str(cycleNumber)+'.pickle'
    saveSimulationPoint(saveDirectory, fileName, theGarden)

  if saveData=="e":
    fileName=simulationName+'-'+str(cycleNumber)+'.csv'
    saveDataPoint(dataDirectory, fileName, theGarden)


  if produceStats:
    #print dataDirectory
    theArgument="-n '%s' -fs" % (dataDirectory+"Seeds/")
    print "***sending to Extract: %s" % (theArgument)
    os.system("python Vida_Data/vextract.py %s" % (theArgument))
    theArgument="-n '%s' -fs" % (dataDirectory+"Plants/")
    print "***sending to Extract: %s" % (theArgument)
    os.system("python Vida_Data/vextract.py %s" % (theArgument))
    theArgument="-n '%s' -fs" % (dataDirectory+"Corpses/")
    print "***sending to Extract: %s" % (theArgument)
    os.system("python Vida_Data/vextract.py %s" % (theArgument))


  ###final graphics calls
  if produceGraphics==1 and webOutput==0:
    print "Producing PNG files..."
    outputGraphics.outputPNGs(outputGraphicsDirectory, outputGraphicsDirectory)
  if produceGraphics==1 and deleteCfdgFiles==1 and webOutput==0:
```

```
    print "Deleting .cfdg files..."
    outputGraphics.deleteCFDGFiles(outputGraphicsDirectory)

  ###only try and make a video if it is wanted and if pngs were made
  if produceVideo and produceGraphics and webOutput==0:
    print "Producing MOV file..."
    outputGraphics.outputMOV(outputGraphicsDirectory, simulationName, framesPerSecond)
  print "*****Simulation Complete*****"
  #print theGarden.deathNote
  #clear the values
  theGarden.soil=[]
  theGarden.deathNote=[]
  theGarden.cycleNumber=0
  for aRegion in theGarden.theRegions:
    aRegion.size=0.0
  #print theGarden.theRegions[0].size
###And this would be the end of the loop bit


if __name__ == '__main__':
  #This section could use some serious fixing. The parsing is very kludgy

  ######Read in any run time arguments provided
  print ""
  theArguments=sys.argv
  specifiedSeedPlaces=0
  seedPlacement="random"
  ###Turn on debug output text
  if "-d" in theArguments:
    debug=1
  if "-dd" in theArguments:
    debug2=1
    debug=0
  if "-n" in theArguments:
    loc= theArguments.index("-n")
    simulationName=theArguments[loc+1]
  ##########################
  if "-g" in theArguments:
    produceGraphics=1
    theView=1
  if "-gb" in theArguments:
    produceGraphics=1
    theView=1
  if "-gt" in theArguments:
    produceGraphics=1
    theView=2
  if "-gs" in theArguments:
    produceGraphics=1
    theView=3
  if "-gts" in theArguments or "-gst" in theArguments:
    produceGraphics=1
    theView=23
  if "-gbs" in theArguments or "-gsb" in theArguments:
    produceGraphics=1
    theView=13
  if "-gbt" in theArguments:
    produceGraphics=1
    theView=12
  if "-gtb" in theArguments:
    produceGraphics=1
    theView=21
  if "-gbts" in theArguments:
    produceGraphics=1
    theView=123
  if "-gld" in theArguments:
    import vdxfGraphics
    produceDXFGraphics=1
```



```
  ##########################
  if "-o" in theArguments:
    produceGraphics=1
    webOutput=1
  if "-v" in theArguments:
    produceVideo=1
    loc= theArguments.index("-v")
    if (len(theArguments)-1)>=(loc+1):
      if theArguments[loc+1].isdigit():
        framesPerSecond=int(theArguments[loc+1])
  if "-c" in theArguments:
    deleteCfdgFiles=0
  if "-p" in theArguments:
    deletePngFiles=1
  if "-w" in theArguments:
    loc= theArguments.index("-w")
    theWorldSize=int(theArguments[loc+1])
  if "-rl" in theArguments:
    loc= theArguments.index("-rl")
    simulationFile=theArguments[loc+1]
    resumeSim=1
    reloadSpeciesData=1
  if "-r" in theArguments:
    loc= theArguments.index("-r")
    simulationFile=theArguments[loc+1]
    resumeSim=1
  ###load in the events file###
  if "-e" in theArguments:
    loc= theArguments.index("-e")
    eventFile=theArguments[loc+1]

  ###Seed placement###
  if "-s" in theArguments or "-sh" in theArguments or "-ss" in theArguments:
    if "-s" in theArguments:
      seedPlacement="random"
      loc=theArguments.index("-s")
    elif "-sh" in theArguments:
      seedPlacement="hex"
      loc=theArguments.index("-sh")
    elif "-ss" in theArguments:
      seedPlacement="square"
      loc=theArguments.index("-ss")
    sList=theArguments[loc+1]
    #print sList
    if (len(theArguments)-1)>=(loc+1):
      if sList.isdigit():
        startPopulationSize=int(sList)
        sList=[]
      elif sList[0]=="[" and sList[-1]=="]":
        #print "here"
        ###it is probably a list
        sList=correctSList(sList)
        ###figure out what the total population count is
        for x in sList:
          if type(x) is int:
            startPopulationSize=startPopulationSize+x
        if not startPopulationSize==thePrefs.startPopulationSize:
          startPopulationSize=startPopulationSize-thePrefs.startPopulationSize
        ###expand the list so you have correct numbers of each species
        ###this makes the list compatable with existing seeding routines
        k=1
        newList=[]
        for i in range(len(sList)/2):
          for j in range(sList[k]):
            newList.append(sList[k-1])
          k=k+2
        sList=newList
        newList=[]
```



```
        ###mix up the list if you have multiple species
        if not sList.count(sList[0])==len(sList):
          random.shuffle(sList)



  #if "-sl" in theArguments:
  # ###Be nice to have a -sf to load x,y from a file
  # ###Make it a tupple with x,y,species so you can seed multiple species
  # loc= theArguments.index("-sl")
  # theInput=theArguments[loc+1]
  # if theInput[0]=="[" and theInput[-1]=="]":
  #   ###this is probably a list###
  #   theInput=theInput[1:-1]
  # theInput=theInput.split(",")
  # seedPlacementList= map(correctType, theInput)
  # print seedPlacementList
  # #print seedPlacementList
  # seedPlacement="argumentList"

  if "-sf" in theArguments:
    print "***Attempting to load plant placement file...***"
    loc= theArguments.index("-sf")
    seedPlacementFile=theArguments[loc+1]
    if os.path.isfile(seedPlacementFile):
      theFile=open(seedPlacementFile)
      try:
        sList=theFile.readlines()
      finally:
        theFile.close()
      ##send the file off to make sure it's in the correct format
      sList=checkSeedPlacementList(sList)
      startPopulationSize=len(sList)
      seedPlacement="fromFile"
      print "***Plant placement file loaded.***"
      #print sList
    else:
        seedPlacement="random"
        startPopulationSize=int(sList)
        sList=[]


  ################
  if "-m" in theArguments:
    loc= theArguments.index("-m")
    maxPopulation=abs(int(theArguments[loc+1]))
  if "-t" in theArguments:
    loc= theArguments.index("-t")
    maxCycles=abs(int(theArguments[loc+1]))
  if "-i" in theArguments:
    loc= theArguments.index("-i")
    percentTimeStamp=abs(int(theArguments[loc+1]))
  if "-aa" in theArguments:
    archive="a"##archive all
  if "-ae" in theArguments:
    archive="e"##archive end only
  if "-an" in theArguments:
    archive="n"##archive nothing
  if "-a" in theArguments:
    loc= theArguments.index("-a")
    archive=int(theArguments[loc+1])
  if "-fa" in theArguments:
    saveData="a"##archive all
  if "-fe" in theArguments:
    saveData ="e"##archive end only
  if "-fn" in theArguments:
    saveData ="n"##archive nothing
  if "-f" in theArguments:
```



```
    loc= theArguments.index("-f")
    saveData =int(theArguments[loc+1])
  if "-fa" in theArguments:
    produceStats=1
  if "-b" in theArguments:
    if runningInNodeBox:
      print"***Vida running within NodeBox. Progress bar is disabled.***"
      showProgressBar=0
    else:
      showProgressBar=1
  if "-x" in theArguments:
    loc= theArguments.index("-x")
    timesToRepeat=int(theArguments[loc+1])
  if "-?" in theArguments:
    print """
Usage:
scriptName [options]


***These options need to be updated***
Options:
    -d        turn basic debugging text on.
    -dd       turn on debugging for accessory subroutines and basic debugging.
    -n str    simulation name.Default simulation name is 'default'.
    -g        graphical output
    -c        delete .cfdg files after .pngs files made.
    -p        NOT IMPLEMENTED.delete .png files after .mov made.
    -v        produce a quicktime movie of the simulation.
    -i        how large to make the time stamp text.This is a size in %.Default is 2.5%.
    -s int    number of seeds to begin the simulation with.
              Default is 1.
    -m int    Maximum population.
    -t int    Time to run simulation for(number of cycles).
              Note: -m and -t can be combined to end the simulation after a given time or when
a population is reached.
    -w int    world size in meters.Default is 200.
              A value of 200 results in a world 200x200.
    -?        show this message.
              NOTE IT SHOULD SHOW THIS AND THEN EXIT, BUT DOESN'T
"""
  main()
else:
  main()
```

Vida/Vida World Preferences.yml


```
debug: FALSE #moved
gravity: 9.81
atmosphere: 1.0 #unused in v0.8
lightIntensity: 1.0
maxSeedsPerPlant: 5
ignoreGermDeathAtStart: TRUE
allowRandomDeath: FALSE
randomDeath: 0.0075
allowSlowGrowthDeath: FALSE
randomSlowGrowth: 0.05
allowOverlaps: FALSE
allowOffWorld: FALSE
allowEulerGreenhillViolations: FALSE
carbonAllocationMethod: 0 #discontinued
canopyTransmittanceImpactsConversion: SPECIES #discontinued
fractionWorldGraphicEdgeSpacer: 0.3 #to be moved
fractionWorldBetweenLRGraphicsSpacer: 0.0 #to be moved
fractionWorldBetweenTBGraphicsSpacer: 0.5 #to be moved
```

APPENDIX C

Event Files

Vida/Event Files/Events_example.yml


```
2:
   - Region:
     - name: test1
       size: 100
       shape: circle #circle or square
       x: 0.0
       y: 0.0
       light: 14
       gravity: 15
       atmosphere: 16
3:
   - Region:
     - name: test1
       size: 1
       shape: square #circle or square
       x: 22
       y: 23
       light: 24
       gravity: 25
       atmosphere: 26

10:
   - Seed:
     - species: Abies alba.yml
       placement: hex
       number: 100
20:
   - Seed:
     - species: Abies alba.yml
       placement: random
       number: 1000
   - Safezone:
     - x: 0.0
       y: 0.0
       shape: circle
       size: 5
       target: seeds
50:
   - Seed:
     - species: random
       placement: random
       number: 100
100:
   - Garden:
     - gravity: 9.9
       atmosphere: 1.1
       light: 0.5
```

APPENDIX D

Placement Files

Vida/Placement Files/ExamplePlacementFile.csv


```
random, 0.0, 0.0, 0
random, 1.0, 0.0, 0
random, -1.0, 0.0, 0
random, 0.0, 1.0, 0
random, 0.0, -1.0, 0
random, 1.0, 1.0, 0
random, 1.0, -1.0, 0
random, -1.0, 1.0, 0
random, -1.0, -1.0, 0
```

APPENDIX E

Read Me Files

Vida/Read Me Files/Vida HOWTO.txt


```
NAME
----
  VIDA - tree growth simulation software


SYNOPSIS
--------
  VIDA.py [OPTIONS]


CONTENTS
--------
  Description
  Option summary
  Basic Usage (with examples)
  Advanced Usage
    Default values for VIDA
    World constants
    Species files
    Seed placement files
    Scripting events


DESCRIPTION
-----------
  VIDA is a software suite that attempts to model the growth of individual trees using
empiricly derived--or randomly chosen--values for use with allometric relationships. By
modeling the behavior of an individual tree, it is possible to model population dynamics in a
spatially explicit simulationspace.

Trees are intentionally represented in highly stylized, simplified forms, specificly as an
untapered trunk which supports a hemispherical canopy. The simulationspace represents a
Platonic, square world where there is no wind,   there is no light scattering due to materials
in the atmosphere, there is no reflection of light and light comes from directly above each
tree. As a consequence of the source of light and the lack of scattering or reflection, the
available photosynthetic area for each tree is the projected area of the canopy.

Some built in features include:
  * ability to define square or circular regions in the simulation space that have different
light intensities, kill all plants, seeds or both, and to have these regions change according
over time as defined in a human-readable event file
  * ability to plant defined or randomly chosen species in random locations, in rows, in an
hexigonal arrangement, or as defined in a file specifing x,y locations for each seed
  * ability to save and resume simulations
  * ability to save statistics on each individual in the simulation, as well as population-wide
statistics in CSV files for later analysis
  * ability to generate various graphical views of the simulationspace, including autoCAD dxf
files and Quicktime movies(Macintosh OSX only)

SETUP
-----
  See the file README for installation instructions.

OPTIONS SUMMARY
---------------
  -d       turn on basic debugging
  -dd      turn on module debugging
  -n <string>   simulation name
  -w <integer>     world size in meters
  -s <mixed>    starting population size, a list of species, locations
           and germination delays, or a file defining species,
           locations and delays in germination
  -m <integer>    maximum population size
  -t <integer>    maximum number of cycles
  -g[<characters>]  produce graphics (see Graphic Options)
  -g3d      produce a dxf file
  -c       keep CFDG files
  -p       delete PNG files
```



```
  -v [<integer>]    produce a Quicktime video, with an optional frames/second
           value(Macintosh only)
  -r <string>   reload a previously saved simulation
  -e <string>   load an event file named <string>
  -a<characters>    save a simulation state
  -f<characters>    save statistical data
  -x <integer>    number of times to rereun the simulation

Graphic Options:
  VIDA can generate graphics showing views of the simulation from various angles, using the
following options:
  -gs     show the simulation from the side without any depth
  -gt     show the simulation from the top-down
  -gb     show the simulation from the bottom-up
  -gbs      show the simulation from the bottom-up and from the side
  -gts      show the simulation from the top-down and from the side
  -gtb      show the simulation from the top-down and from the
           bottom-up(images are oriented left to right)
  -gbt      show the simulation from the bottom-up and from the
           top-down(images are oriented left to right)

  NOTE: As of version 0.9 -gbts and -gtbs are not implemented.


BASIC USAGE
-----------
  At the bare minimum, one can run a simulation with VIDA by simply typing:

    python VIDA.py

  Running VIDA without any arguments will make use of default values in VIDA_Data/vdefaults.py.
One can modify how VIDA's simulations are run by either altering the default values in the
vdefaults.py file, or by entering options on the command line. The order of the arguments does
not matter. For example:
    python VIDA.py -n Terra -W 75
  is functionally the same as:
    python VIDA.py -w 75 -n Terra

  The best way to illustrate the options is to provide some examples:

    python VIDA.py -n 1example -w 100 -s 10 -t 200 -m 550 -gsb -v

  This example would create a simulation called "1example". All files would be placed in the
VIDA directory in a folder named "Output-1example". The simulationspace would be 100x100 meters
in size and be seeded with 10 seeds of species randomly chosen from those present in the
"Species" folder and placed in random locations within the simulation space. The simulation
will run for 200 iterations, or until the maximum population reaches 550. Graphics will be made
showing a bottom-up view and a side-on view. In addition, a video will be generated from the
graphics. <include example graphical views>

  VIDA uses the ContextFree application (http://www.contextfreeart.org/) to generate graphics.
To do so, VIDA first generates cfdg files which ContextFree can read. By default, VIDA deletes
the cfdg files after png files are generated. If one does not wish to have the cfdg file
deleted, one includes the -c option in the command line. For example:

    python VIDA.py -n 2example -w 100 -s 10 -t 200 -m 550 -gsb -c -v

  This example would create a simulation called "2example". All files would be placed in the
VIDA directory in a folder named "Output-2example". The simulationspace would be 100x100 meters
in size and be seeded with 10 seeds of species randomly chosen from those present in the
"Species" folder and placed in random locations within the simulation space. The simulation
will run for 200 iterations, or until the maximum population reaches 550. Graphics will be made
showing a bottom-up view and a side-on view, and the cfdg files will not be deleted at the end
of the simulation.


    python VIDA.py -n 3example -w 100 -s 10 -t 200 -m 550 -gsb -v 10
```



```
  This is obviously very similar to the first and second examples, but in this case the 10
randomly chosen seeds are randomly placed on a 100x100m world. The addition of an integer (10)
after the -v option results in a video being made with 10 frames/second.

  If one is interested in examining numerical values and relationships for the plants and seeds
for a given iteration of a simulation, one can save information pertaining to an object's
location, mass, size, and parentage using the -f<character> options.

    -fe     save the state of the last iteration of a simulation
    -fa     save the state of all iterations of a simulation
    -fn     do not save any states

  For example:

    python VIDA.py -n 4example -w 100 -s 10 -t 200 -m 550 -gsb -v -fa

  is identical to example 2 with the exception that files containing information on each object
in the simulation will be saved to a folder named "Simulation_data" within the
"Output-1example" folder. Since each file can be very large, objects are divided into seeds,
plants and corpses, and are saved to appropriate subfolders.<make figure showing screenshot of
folder structure> When one uses the -fa command, VIDA takes the additional step to generate
statistical summary files from the individual data files.

  Vida saves the data for the final iteration a simulation by default. This is the equivilant
to using the command -fe. If one didn't want to save any data files, one would use the -fn
command.

  In addition to saving graphical files, data files, and statistical files, VIDA can save the
state of iterations so that one could resume a simulation from a known point. The default
setting for VIDA is to save the final iteration of a simulation. This is the equivilant to
saying:

    python VIDA.py -n 5example -w 100 -s 10 -t 200 -m 550 -gsb -v 10 -ae

  The options available for saving files are:
    -ae     save the state of the last iteration of a simulation
    -aa     save the state of all iterations of a simulation
    -an     do not save any states

  For example:

    python VIDA.py -n 6example -w 100 -ss 10 -t 200 -m 550 -gsb -v 10 -fa

  As in example 3, 10 randomly chosen seeds are randomly placed on a 100x100m world. The
addition of an integer (10) after the -v option results in a video being made with 10
frames/second. For every iteration statistical data is saved for each and every plant and seed
in the simulationspace, and overall statistics will also be produced.

  It is possible to reload a saved simulation state and run a simulation from that point.

    python VIDA.py -n 7example -r <file name> -s 10 -t 200 -m 550 -gsb

  In example 7, a previously saved simulation is loaded and will be run for 200 cycles or until
a maximum population of 550 is reached. It is important to point out that when resuming a
simulation, the -t command is time added onto whatever the saved simulation has. For example,
if a reloaded simulation was from the 100th iteration of a simulation and the command -t 200 is
used, the newly running simulation will run for 200 iterations, starting at the saved point.
Note that -s can be used to introduce new seeds to an established world. Because the simulation
world has already been created, the -w command will not do anything when resuming a simulation;
once a simulation's world space is defined, it is fixed at that size.

  At times it might be necessary to repeat a simulation multiple times so as to get meaningful
statistics. One can accomplish this using the -x <integer> command:

    python VIDA.py -n 8example -w 100 -s 10 -t 200 -m 550 -fa -x 3

  As should be obvious, this simulation is essentially the same as example 1 with the exception
that data files will be generated for each iteration. In addition to data files, a statistical
```



```
summary file will be made. This simulation will run a total of 3 times, with the name of the
output folder having a numerical suffix appended to it. <have an example screen shot of this>


ADVANCED OPTIONS
----------------

  VIDA has a number of default values which it uses. This is why simply typing 'python Vida.py'
works. Default values, and the information for a default species (which is used when there are
no species files in the 'Species' folder) and stored in three different files: Vita World
Preferences.yml, vdefaults.py, and Default_species.yml. Please note that that there should be
revision as to what values are in Vita World Preferences.yml versus vdefaults.py.
  By editing the values within these files, one can alter simulations in ways not possible
using the command line. One can also provide default setting such that one can omit certain
command line arguments.

  Default values for VIDA
  -----------------------
    See table for default values used by VIDA and saved in the Vida_Data/vdefaults.py
file.<include a screen shot showing folder path>

  World constants
  ---------------
    Found in Vita World Preferences.yml<include a screen shot showing folder path>
    Gravity:
      This is is the force of gravity in the simulation world. This value is used to determine
aspects of seed dispersal and is used in calculations determining whether a plant will buckle
due to excessive height in relation to diameter. The default value is 9.81m/s^2.
    Atmosphere:
      Currently unused.
    LightIntensity:
      This value is the amount of light present in the world, in arbitrary units. The default
value of 1.0 can be seen as meaning 100%.
    MaxSeedsPerPlant:
      There are situations where a given species will produce an impractically large number of
seeds. Because each object in the world must be evaluated, excessive numbers of seeds can slow
down simulation speeds. This variable allows one to cap the maximum number of seeds that any
individual plant can make. The default value is 5.
    IgnoreGermDeathAtStart:
      This variable determines whether the simulation should ignore germination failure defined
by a species for the starting iteration. The default value is TRUE.
    AllowRandomDeath:
      It is possible to turn various methods of death on and off. In this case, this variable
controls whether plants will die due to pseudorandom (stochastic) events. The default value is
TRUE.
    RandomDeath:
      If AllowRandomDeath is set to True, the value that RandomDeath is set to is used to
determine the chances that any plant will die during any given iteration. The default value is
0.0075, which translates as a 0.75% chance of death for each tree, each iteration.
    AllowSlowGrowthDeath:
      Depending on the species in question, as a species ages, it's growth in height will slow.
When this setting is true, a tree risks dying due to growing too slow, simulating senescence.
The default value is true.
    RandomSlowGrowth:
      When AllowSlowGrowthDeath is set to true, this variable is used to help determine whether
the plant in question enters a second round of stochastic death. If the average growth of the
plant (as a fraction) is less than this value, the plant enters another round of potential
stochastic death. The default value is 0.05, which can be read as, if a plant's average height
growth rate is less than 5% of it's maximum height growth rate, it will enter another round of
random death.
    AllowOverlaps:
      When true, the simulation will allow objects to occupy the same space. Default is false.
    AllowOffWorld:
      When true, the simulation will allow stem objects to grow so that their edges are off the
world space. The default is false.
    AllowEulerGreenhillViolations:
      When true, the simulation will ignore the possibility that trees might grow too tall
relative to their diameter. The default is false.
```

```
    FractionWorldGraphicEdgeSpacer:
      This value is a fractional number specifing the amount of graphical space around the edge
of the world. This helps stop images from resizing as large canopies grow beyond the edge of
the world.
The default is 0.3.
    FractionWorldBetweenLRGraphicsSpacer:
    When generating graphics, this value is a fractional value defining the amount of space
between top-down and bottom-up graphics as requested using the -gbt, -gtb, or -gtbs commands.
Default is 0.0.

    FractionWorldBetweenTBGraphicsSpacer:
    When generating graphics, this value is a fractional value defining the amount of space
between top-down/bottom-up graphics and the side-view graphic as requested using the -gbs,
-gts, or -gtbs commands. Default is 0.5.

  Species files
  -------------
    The default species file, Default_species.yml, is found within the Vida_Data folder and is
used when there are no species filed present within the Species folder.
    Species files contain all the physical, allometric and behavioriel information for how a
species behaves in a simulation. Values range from wood density to where seeds form on the
canopy, to how it responds to shading. For a complete list of each attribute and what it is
for, please see table <n. this will be a reworked version of SpeciesFileHelpData.yml>.

  Seed placement files
  --------------------
    Seed placement files are simple csv files that have information in the following format:
      species name, x coordinate, y coordinate, delay in germination

    For example, the following line:
      Generic Gymnosperm.yml, 0.0, 0.0, 0
    would place a species called Generic Gymnosperm at 0.0, 0.0 when the world was first made.
This plant would attempt to germinate during the first iteration (a delay of 0).

    One can place multiple plants by defining multiple lines, such as:
      Generic Gymnosperm.yml, 0.0, 0.0, 0
      Generic Gymnosperm.yml, 1.0, 0.0, 5
      Generic Gymnosperm.yml, -1.0, 0.0, 10

    In that example, three Generic Gymnosperm plants are placed at [0.0,0.0], [1.0,0.0] and
[-1.0, 0.0] on the world. The first plant will germinate in the first cycle (a delay of 0), the
second will germinate after waiting five iterations, and the third will germinate after ten
iterations.

    In general, one can keep seed placement files in any directory, but it is advised one puts
them within the "Placement Files" folder withing the VIDA directory. VIDA comes with a number
of example placement files in the Placement Files folder. One, Metroid.csv, will place the
species Abies alba in a pattern that resembles a Metroid from the classic NES game<image of
metroid placement>. To load a placement file, one uses the following commandline syntax:

      python Vida.py -sf <path to file>

    To load the example Metroid placement file, one would enter:

      python Vida.py -sf 'Placement Files/Metroid.csv'

    All other commandline arguments (except additional -s commands) are valid. For example, one
could use the following commands:

      python Vida.py -n metroidExample -sf 'Placement Files/Metroid.csv' -t 100 -m 10,000 -gsb
-v 12 -fa -aa -x 3

  Scripting events
  ----------------
    It is sometimes useful to be able to script various events to occur during a simulation.
One might want to change light intensities in various locations as a simulation runs, or
introduce a new species, or cause a mass die-off in a specific region. To do any of these, one
can use event files.
```



```
    Event files are yml files with the basic structure:
      Iteration#:
        -Object or action:
          -property1: value
          -property2: value

    For example, if one wanted to change the light intensity of the entire simulation world at
iteration 10 of a simulation, one would add the following to an event file:
        10:
          - Region:
              lightIntensity: 0.75

    For example, if one wanted to make the upper left quadrant of a 100x100m world have 50%
light starting at iteration 10, one would write:
        10:
          - Garden:
              - lightIntensity: 0.5

    One can edit any property of the simulation world in an event file, so one can say
something like:
        10:
          - Garden:
              - lightIntensity: 0.5
                atmosphere: 1.1
                                  gravity: 10.81

    It is possible to create subregions in the simulation using event files by defining a
region and its properties:
        10:
          - Region:
              - name: reg1
                size: 50.0
                shape: square #circle or square
                x: -25.0
                y: 25.0
                lightIntensity: 0.5

    Here, a new region named 'reg1' is made. It is centered on -25.0, 25.0, has a total size of
50m, is a square, and has a light intensity of 0.5. After a region is defined in a event file,
one can change any of the properties that it has:
        10:
          - Region:
              - name: reg1
                size: 50.0
                shape: square #circle or square
                x: -25.0
                y: 25.0
                lightIntensity: 0.5
        20:
          - Region:
              - name: reg1
                lightIntensity: 0.75
        25:
          - Region:
              - name: reg1
                gravity: 10.0

    All regions have the same properties as the world in general (gravity, atmospheric
pressure, light intensity), and can be modified using event files. An example event file
showing how to change the light intensity in four different regions every five years is the
file '2x2changeEvery05yearsClockwise.yml'

    If one wishes to introduce seeds using an event file, one uses the format:
        Iteration#:
          - Seed:
              - species: <name or 'random'>
                placement: <placement file, 'square', 'hex' or 'random'>
                number: <number of seeds>
```



```
    For example, if one wished to place seeds using a placement file at iteration 21, one would
say:
        21:
          - Seed:
            - placement: 'Placement Files/Metroid.csv'

    If one wanted to place 27 Abies alba seeds in a square pattern on iteration 100, one would
say:
        100:
          - Seed:
            - species: 'Abies alba.yml'
              placement: square
              number: 27

    Finally, if one wanted to place 45 random species randomly in the world on iteration 51,
one would say:
        51:
          - Seed:
            - species: random
              placement: random
              number: 45

    There are times when one would like to kill plants, seeds or both in a given location. One
can accomplish this using a syntax similar to that used for making regions:
        Iteration#:
          - <'Killzone' or 'Safezone'>:
            - x: <x location zone is centered on>
              y: <y location zone is centered on>
              shape: <circle or square>
              size: <in meters>
              target: <'plants' or 'seeds' or 'all'>

    If one wanted to kill all plants and seeds in a 10m circle centered on 0.0, 0.0 on
iteration 87, one would use the following:
        87:
          - Killzone:
            - x: 0.0
              y: 0.0
              shape: circle
              size: 10
              target: all

    Alternately, if one had wanted to kill only seeds in the above example:
        87:
          - Killzone:
            - x: 0.0
              y: 0.0
              shape: circle
              size: 10
              target: seeds

    Each of these aspects of event files can be combined into a single file such as:
        0:
          - Garden:
            - lightIntensity: 0.5
              atmosphere: 1.1
                                  gravity: 10.81
        10:
          - Region:
            - name: reg1
              size: 50.0
              shape: square #circle or square
              x: -25.0
              y: 25.0
              lightIntensity: 0.5
        20:
          - Region:
            - name: reg1
```



```
              lightIntensity: 0.75
        87:
          - Killzone:
            - x: 0.0
              y: 0.0
              shape: circle
              size: 10
              target: all
        100:
          - Seed:
            - species: 'Abies alba.yml'
              placement: square
              number: 27

    Of all the aspects of Vida, event files are by far the most powerful in that multiple
regions and events in a single file can be combined. Examination of included event files is
encouraged.

    To make use of an event file, one uses the syntax:
      python Vida.py -e <path to file>

    To load an example event file, one would enter:

      python Vida.py -e 'Event Files/events_examples.yml'

    All other commandline arguments (except additional -s commands) are valid. For example, one
could use the following commands:

      python Vida.py -n eventExample -s 'Placement Files/Metroid.csv' -e 'Event
Files/events_examples.yml' -t 100 -m 10,000 -gsb -v 12 -fa -aa -x 3
```

Vida/Read Me Files/Vida ReadME.txt


```
======================================================================
            VIDA - Tree and Forest Growth Simulation Software
======================================================================

Version 0.9: 2009.05.01
-----------

VIDA is a software suite that attempts to model the growth of individual trees using empiriclly
derived--or randomly chosen--values for use with allometric relationships. By modeling the
behavior of an individual tree, it is possible to model population dynamics in a spatially
explicit simulationspace.

This is a development release.

email: seanth@gmail.com

DEPENDENCIES/REQUIREMENTS
_________________________
  *Python v2.5 (http://python.org/download/)
  *pyYAML v3.06 (http://pyyaml.org/wiki/PyYAML)
  *Psyco v1.6 (optional. Psyco improves the speed at which some calculations are made. This can
only be used on 32-bit, 386-compatible processors, i.e. it will not work on PowerPC Macs.
http://psyco.sourceforge.net/download.html)
  *ContextFree v2.2 (optional. Needed to generate graphics.
http://www.contextfreeart.org/download/ContextFreeSource2.2.tgz)
  *AppleScript v1.83 (optional. Needed to automatically generate videos from graphics. This can
only be used with Apple's OS X operating system.)
  *Quicktime v7.6 (optional. Needed to automatically generate videos from graphics. Because
VIDA uses AppleScript is to automate Quicktime, this will only work computer's running Apple's
OS X. http://www.apple.com/quicktime/download/)

HOW TO USE
----------
After installing all the dependencies necessary, simply cd to the VIDA folder and, in the
simplest form, type:
  >python VIDA.py
For more information, including command line options and ways to make species, event files and
define planting locations, please see VIDA HOWTO.txt

EXAMPLES
--------
For more examples, including command line options and ways to make species, event files and
define planting locations, please see VIDA HOWTO.txt
```

APPENDIX F

Files in Species directory

Vida/Species/Abies alba.yml


```
nameSpecies: Abies alba
creator: seanth@gmail.com
comments: Species generated using Cannell data set. Revised 08.09.11. Minor change to seed mass
and fraction to plant on 8.9.17. Considered final.

###Basic
densityStem: 718.90 #Lavers 1967 Wood properties
densityLeaf: 923.43
densitySeed: 359.45 #assume density is half as stem
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.2
heightLeafMax: 0.000296
heightStemMax: 42.69821
youngsModulusStem: 7.4 #GPa-Lavers 1967 Wood properties

###Reproduction
makeSeeds: TRUE
fractionSelfishness: 0.5
reproductionConstant: 0.09974943
reproductionExponent: 1.09777536
numYearsGrowthMemory: 2
massSeedMax:  0.60
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 1
borderImagePercent: 25
colourSpecies: [0.0, 0.0, 0.0] #black
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 1.531
photoExponent: -0.4602
canopyTransmittanceImpactsConversion: FALSE

###Allometry
fractionCarbonToSeeds: 1.0
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.000045 #seedling will avg 0.000027kg
fractionCarbonToStem: 0.99
speciesConstant1: 0.94119204
speciesExponent1: 1.00133354
speciesConstant2: 0.03970297
speciesExponent2: 1.19822771
speciesConstant3: 0.24834698
speciesExponent3: 0.73063124
speciesConstant20: 0.029 #0.03599987
speciesExponent20: 0.39415386
speciesConstant6: 0.0 #0.15
speciesConstant7: 100.064153
speciesExponent7: 1.08090778
speciesConstant8: 15.508049
```

Vida/Species/Generic Angiosperm.yml


```
nameSpecies: Generic Angiosperm
creator: seanth@gmail.com
comments: Species generated using Cannell data set. Revised 08.10.02 based on cannell data set
with any questionable data removed.

###Basic
densityStem: 995.76 #Lavers 1967 Wood properties
densityLeaf: 500.0
densitySeed: 497.88 #assume density is half as stem
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.20
heightLeafMax: 0.000296
heightStemMax: 31.464273
youngsModulusStem: 9.6543478 #GPa-Lavers 1967 Wood properties

###Reproduction
makeSeeds: TRUE
fractionSelfishness: 0.5
reproductionConstant: 0.00700713
reproductionExponent: 1.7088611
numYearsGrowthMemory: 2

massSeedMax:  0.008
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 10
borderImagePercent: 25
colourSpecies: [0.0, 0.0, 0.3] #black
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 1.5056
photoExponent: -0.44834
canopyTransmittanceImpactsConversion: FALSE


###Allometry
fractionCarbonToSeeds: 1.0
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.003375 #seedling should ave 0.000027kg
fractionCarbonToStem: 0.99
speciesConstant1: 0.86366077
speciesExponent1: 1.02212979
speciesConstant2: 0.11637722
speciesExponent2: 0.88193255
speciesConstant3: 0.08536206
speciesExponent3: 0.77192318
speciesConstant20: 0.02649732
speciesExponent20: 0.38252183
speciesConstant6: 0.0
speciesConstant7: 160.81198
speciesExponent7: 1.0964662
speciesConstant8: 7.7053399
```

Vida/Species/Generic Gymnosperm with Abies Photo constant.yml


```
nameSpecies: Generic Gymnosperm with Abies photo constant
creator: seanth@gmail.com
comments: Species generated using Cannell data set. Revised 08.08.23

###Basic
densityStem: 764.30 #Lavers 1967 Wood properties
densityLeaf: 923.43
densitySeed: 479.0 #assume density is half as stem
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.2
heightLeafMax: 0.000296
heightStemMax: 30.112785
youngsModulusStem: 7.3263158 #GPa- Lavers 1967 Wood properties

###Reproduction
makeSeeds: TRUE
fractionCarbonToSeeds: 1.0
fractionSelfishness: 0.5
reproductionConstant: 0.08543487
reproductionExponent: 1.09777536
numYearsGrowthMemory: 2

massSeedMax:  0.21 #0.085
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 10
borderImagePercent: 25
colourSpecies: [0.0, 0.0, 0.5] #grey
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 1.531 #0.58408
photoExponent: -0.45767
canopyTransmittanceImpactsConversion: FALSE


###Allometry
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.000128571428571 #seedling should avg 0.000027kg
fractionCarbonToStem: 0.99
speciesConstant1: 0.72400752
speciesExponent1: 1.04357385
speciesConstant2: 0.29023803
speciesExponent2: 0.97153097
speciesConstant3: 0.38547077
speciesExponent3: 0.7092191
speciesConstant20: 0.03034632
speciesExponent20: 0.4030135
speciesConstant6: 0.0
speciesConstant7: 80.2588695
speciesExponent7: 0.92930453
speciesConstant8: 8.4941704
```

Vida/Species/Generic Gymnosperm with Cryptomeria Photo constant.yml


```
nameSpecies: Generic Gymnosperm with Cryptomeria photo constant
creator: seanth@gmail.com
comments: Species generated using Cannell data set. Revised 08.08.23

###Basic
densityStem: 764.30 #Lavers 1967 Wood properties
densityLeaf: 923.43
densitySeed: 479.0 #assume density is half as stem
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.2
heightLeafMax: 0.000296
heightStemMax: 30.112785
youngsModulusStem: 7.3263158 #GPa- Lavers 1967 Wood properties

###Reproduction
makeSeeds: TRUE
fractionCarbonToSeeds: 1.0
fractionSelfishness: 0.5
reproductionConstant: 0.08543487
reproductionExponent: 1.09777536
numYearsGrowthMemory: 2

massSeedMax:  0.13 #0.085
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 1
borderImagePercent: 25
colourSpecies: [240.0, 1.0, 1.0] #blue
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 0.91272 #0.58408
photoExponent: -0.45767
canopyTransmittanceImpactsConversion: FALSE


###Allometry
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.000207692307692 #seedling should avg 0.000027kg
fractionCarbonToStem: 0.99
speciesConstant1: 0.72400752
speciesExponent1: 1.04357385
speciesConstant2: 0.29023803
speciesExponent2: 0.97153097
speciesConstant3: 0.38547077
speciesExponent3: 0.7092191
speciesConstant20: 0.03034632
speciesExponent20: 0.4030135
speciesConstant6: 0.0
speciesConstant7: 80.2588695
speciesExponent7: 0.92930453
speciesConstant8: 8.4941704
```

Vida/Species/Generic Gymnosperm.yml


```
nameSpecies: Generic Gymnosperm
creator: seanth@gmail.com
comments: Species generated using Cannell data set. Revised 08.08.23

###Basic
densityStem: 764.30 #Lavers 1967 Wood properties
densityLeaf: 923.43
densitySeed: 479.0 #assume density is half as stem
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.2
heightLeafMax: 0.000296
heightStemMax: 30.112785
youngsModulusStem: 7.3263158 #GPa- Lavers 1967 Wood properties

###Reproduction
makeSeeds: TRUE
fractionCarbonToSeeds: 1.0
fractionSelfishness: 0.5
reproductionConstant: 0.08543487
reproductionExponent: 1.09777536
numYearsGrowthMemory: 2

massSeedMax:  0.085
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 10
borderImagePercent: 25
colourSpecies: [58.0, 1.0, 1.0] #yellow
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 0.58408
photoExponent: -0.45767
canopyTransmittanceImpactsConversion: FALSE


###Allometry
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.000317647 #seedling should avg 0.000027kg
fractionCarbonToStem: 0.99
speciesConstant1: 0.72400752
speciesExponent1: 1.04357385
speciesConstant2: 0.29023803
speciesExponent2: 0.97153097
speciesConstant3: 0.38547077
speciesExponent3: 0.7092191
speciesConstant20: 0.03034632
speciesExponent20: 0.4030135
speciesConstant6: 0.0
speciesConstant7: 80.2588695
speciesExponent7: 0.92930453
speciesConstant8: 8.4941704
```

Vida/Species/WEB species.yml


```
nameSpecies: WEB species
creator: seanth@gmail.com
comments: Species generated partly using Cannell data set for Generic Angiosperm. See Generic
Angiosperm file for more info. WEB Species makes use of alpha values in keeping with the WEB
theory

###Basic
densityStem: 995.76 #Lavers 1967 Wood properties
densityLeaf: 500.0
densitySeed: 497.88 #assume density is half as stem
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.20
heightLeafMax: 0.000296
heightStemMax: 31.464273
youngsModulusStem: 9.6543478 #GPa-Lavers 1967 Wood properties

###Reproduction
makeSeeds: TRUE
fractionSelfishness: 0.5
reproductionConstant: 0.00700713
reproductionExponent: 1.7088611
numYearsGrowthMemory: 2

massSeedMax:  0.008
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 10
borderImagePercent: 25
colourSpecies: [285.0, 1.0, 0.7] #Purple
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 1.5056
photoExponent: -0.44834
canopyTransmittanceImpactsConversion: FALSE


###Allometry
fractionCarbonToSeeds: 1.0
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.003375 #seedling should ave 0.000027kg
fractionCarbonToStem: 0.99
speciesConstant1: 0.86366077
speciesExponent1: 1.0 #web value
speciesConstant2: 0.11637722
speciesExponent2: 0.75 #web value
speciesConstant3: 0.08536206
speciesExponent3: 0.75 #web value
speciesConstant20: 0.02649732
speciesExponent20: 0.375 #web value
speciesConstant6: 0.0
speciesConstant7: 1.0 #35.0 #160.81198
speciesExponent7: 0.666 #web value
speciesConstant8: 7.7053399
```

APPENDIX G

Files in Tools directory

Vida/Tools/PlacementFileMaker.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

#This file can be used to convert an image into a placement file for use with vida.py
from PIL import Image

input_file = "TreePattern.jpg"
output_file = "TreePattern.csv"

im = Image.open(input_file)
out = open(output_file,'w')
width, height = im.size
for x in xrange(width):
  for y in xrange(height):
    if im.getpixel((x,y))<=50:
      #print >> out, "%s, %s, %s" % (x-(width/2.0),y-(height/2.0),im.getpixel((x,y)))
      print >> out, "random, %f, %f, 0" % (x-(width/2.0),y-(height/2.0))
out.close()
```

Vida/Tools/TreePattern.jpg

APPENDIX H

Files in Vida_Data directory

Vida/Vida_Data/Default_species.yml


```
nameSpecies: Default species
creator: seanth@gmail.com
comments: Based on Abies alba

###Basic
densityStem: 700.0 #958.0
densityLeaf: 923.43
densitySeed: 958.0
canopyTransmittance: 0.02
fractionMinimumSurvival: 0.2
heightLeafMax: 0.000296
heightStemMax: 42.69821
youngsModulusStem: 7.4 #GPa-Lavers 1967 Wood properties

###Reproduction
makeSeeds: FALSE
fractionSelfishness: 0.5
startMakingSeedsAge: 100 #to be discontinued
startMakingSeedsHeight: 1000.0 #to be discontinued
reproductionConstant: 0.09974943
reproductionExponent: 1.09777536
numYearsGrowthMemory: 2
massSeedMax:  0.6 #0.000749 #0.6 #1.0 #0.00001
locSeedFormation: [1.0, 0.0]
seedDispersalMethod: [4, 45, 5]
delayInGermination: 0
randomSlowGrowth: 0.0
minimumLightForGermination: 0.0 #unused
fractionFailGerminate: 0.0

###Graphics
leafIsHemisphere: TRUE
radiusStemMultiplier: 1
radiusLeafMultiplier: 1
radiusSeedMultiplier: 5
borderImagePercent: 10
colourSpecies: [0.0, 0.0, 0.0]
colourLeaf: [116.6, 1.0, 1.0]
colourStem: [20.0, 0.9 , 0.5]
colourSeedDispersed: [240.0, 0.343, 0.5]
colourSeedAttached: [300.0, 0.9, 0.5]

###Photosynthesis
photoConstant: 1.531 #1.1885 #1.5
photoExponent: -0.4602 #-0.50462 #-0.465
canopyTransmittanceImpactsConversion: FALSE

###Allometry
fractionCarbonToSeeds: 1.0
fractMassSeedMaxToGerm: 0.8
fractionSeedMassToPlant: 0.000045 #0.00001 #1.0
fractionCarbonToStem: 0.99
speciesConstant1: 0.94119204
speciesExponent1: 1.00133354
speciesConstant2: 0.03970297
speciesExponent2: 1.19822771
speciesConstant3: 0.24834698
speciesExponent3: 0.73063124
speciesConstant20: 0.029 #0.03599987
speciesExponent20: 0.39415386
speciesConstant6: 0.0 #0.15
speciesConstant7: 100.064153 #65.5589222  #100.064199
speciesExponent7: 1.08090778 #0.8518291 #1.08090778
speciesConstant8: 15.508049
```

Vida/Vida_Data/geometry_utils.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import math

#Import Psyco if possible
try:
  import psyco
  psyco.log()
  psyco.full()
except ImportError:
  pass

def checkOverlap(x, y, r, xx, yy, rr):
  return circleOverlap(x,y,r,xx,yy,rr)

def python_circle_circle_overlap(x,y,r,xx,yy,rr):
  theDistance =distBetweenPoints(x, y, xx, yy)
  ###now look at the distance in relation to the radii
  if theDistance>(r+rr):
    ###no overlap
    return 0
  elif theDistance<(math.fabs(r-rr)):
    ###complete overlap
    return 1
  else:
    ###partial overlap
    return 2

#####look and see if there is a special C version
try:
  import circtest
  circleOverlap = circtest.circle_circle_overlap
  print "*********Will use gib's code**************"
except ImportError:
  circleOverlap = python_circle_circle_overlap
  print "*********Will use straight python*********"
  pass

def placePointsInGrid(numbPoints, lengthSquareSide):
  ###assumes the area is a square
  distance= lengthSquareSide/(1.0*math.sqrt(numbPoints))
  return distance

def distBetweenPoints(x0, y0, x1, y1):
  dx=x0-x1
  dy=y0-y1
  theDistance=math.hypot(dx,dy)
  return theDistance

def areaCircle(radius):
  #area=math.pi*math.pow(radius, 2)
  area=3.14*radius*radius
  return area

def radiusCircle(area):
  #radius=math.sqrt(area/math.pi)
  radius=math.sqrt(area/3.14)
  return radius
```



```
def areaTriangle(distToRadical, lengthRadicalLine):
  area=2.0*0.5* distToRadical* lengthRadicalLine
  return area

def areaRectangle(rectSides):
  dx=math.fabs(rectSides[1]-rectSides[0])
  dy=math.fabs(rectSides[3]-rectSides[2])
  area=dx*dy
  return area

def areaSector(radius, radians):
  area=0.5*math.pow(radius, 2)*radians
  return area

def boundCircle(x, y, r):
  ###return the [xMin, xMax, yMin, yMax] of a box bounding a circle
  xMin=x-r
  xMax=x+r
  yMin=y-r
  yMax=y+r
  return [xMin, xMax, yMin, yMax]

#def checkOverlap(x, y, r, xx, yy, rr):
# if useCforCheckingOverlap==False:
#   theDistance =distBetweenPoints(x, y, xx, yy)
#   ###now look at the distance in relation to the radii
#   if theDistance>(r+rr):
#     ###no overlap
#     return 0
#   elif theDistance<(math.fabs(r-rr)):
#     ###complete overlap
#     return 1
#   else:
#     ###partial overlap
#     return 2
# else:
#   theReturn=circtest.circle_circle_overlap(x,y,r,xx,yy,rr)
#   return theReturn

def checkOverlapSquare(x, y, r, xx, yy, size):
  #x,y,r are the plant
  #xx, yy, size are the square
  halfSize=size/2.0
  if x<=xx+halfSize and x>=xx-halfSize:
    return 1

def pointInsideCircle(circX, circY, circR, pointX, pointY):
  ###accepts the x,y and r of a circle and an x,y point
  ###return whether the x,y point is in the circle
  theDistance=distBetweenPoints(circX, circY, pointX, pointY)
  if theDistance<=circR:
    return 1
  else:
    return 0

def pointInsideSquare(squareX, squareY, squareSize,pointX, pointY):
    ##returns 1 if out of bounds
    ##returns 0 if in bounds
    maxSquareX=squareX+(squareSize/2.0)
    minSquareX=squareX-(squareSize/2.0)
    maxSquareY=squareY+(squareSize/2.0)
    minSquareY=squareY-(squareSize/2.0)
    if pointX<=maxSquareX and pointX>=minSquareX and pointY<=maxSquareY and pointY>=minSquareY:
      return 1
    else:
      return 0
```



```


def distToRadical(r0, r1, distance):
  ###Send it radius of circle 1, radius circle 2, distance between the two circles
  ###returns a list in format:[distance from circle 1 to radical, distance from circle 2 to
radical]
  d0=(math.pow(r0,2)-math.pow(r1,2)+math.pow(distance, 2))/(2*distance)
  d1=(math.pow(r1,2)-math.pow(r0,2)+math.pow(distance, 2))/(2*distance)
  radicalDistances=[]
  radicalDistances.append(d0)
  radicalDistances.append(d1)
  return radicalDistances

def lengthRadicalLine(r0, distToRadical):
  halfRadicalLine=math.pow((math.pow(r0,2)-math.pow(distToRadical,2)),0.5)
  return halfRadicalLine*2

def radiansSector(distance, radius):
  radians=math.acos(distance/radius)*2.0
  return radians



def areaThumbnail(areaSector, areaTriangle):
  area=areaSector-areaTriangle
  return area

def areaOverlappingCircles(x, y, r, xx, yy, rr):
        ###get the distance between the two plants
        #if debug2==1:print "          Plant one radius x y: %f, %f, %f" % (r, x,  y)
        #if debug2==1:print "          Plant two radius x y: %f, %f, %f" % (rr, xx, yy)
        theDistance =distBetweenPoints(x, y, xx, yy)
        #if debug2==1:print "               Distance is: %f" % (theDistance)
        ###now look at the distance in relation to the radii
        if theDistance>(r+rr):
          ###no overlap
          #if debug2==1:print "               No overlap"
          areaOverlap=0.0
        elif theDistance<(math.fabs(r-rr)):
          ###one plant is completely covered
          #if debug2==1:print "          %s(%f) completely covered by %s(%f)." % ("circle 1",
r, "circle 2", rr)
          #areaOverlap=math.pi*math.pow(r,2.0)
          areaOverlap=3.14*r*r
        else:
          ###there is partial overlap
          #if debug2==1:print "          %s(%f) partly covered by %s(%f)." % ("circle 1", r,
"circle 2", rr)
          theDistanceToRadical= distToRadical(r, rr, theDistance)
          circleOneDistToRadical= theDistanceToRadical[0]
          circleTwoDistToRadical= theDistanceToRadical[1]
          #if debug2==1:print "          plant one distance to radical: %f" %
(circleOneDistToRadical)
          #if debug2==1:print "          plant two distance to radical: %f" %
(circleTwoDistToRadical)
          theLengthRadicalLine=lengthRadicalLine(r, circleOneDistToRadical)
          #if debug2==1:print "          length radical line: %f" % (theLengthRadicalLine)
          radiansCircleOneSector= radiansSector(circleOneDistToRadical, r)
          radiansCircleTwoSector= radiansSector(circleTwoDistToRadical, rr)
          #if debug2==1:print "     radians for circle one: %f" % (radiansCircleOneSector)
          #if debug2==1:print "     radians for circle two: %f" % (radiansCircleTwoSector)
          areaCircleOneSector=areaSector(r, radiansCircleOneSector)
          areaCircleTwoSector=areaSector(rr, radiansCircleTwoSector)
          #if debug2==1:print "     area for plant one sector: %f" % (areaCircleOneSector)
          #if debug2==1:print "     area for plant two sector: %f" % (areaCircleTwoSector)
          areaCircleOneTriangle=areaTriangle(circleOneDistToRadical, theLengthRadicalLine/2.0)
          areaCircleTwoTriangle=areaTriangle(circleTwoDistToRadical, theLengthRadicalLine/2.0)
          #if debug2==1:print "     area for plant one triangle: %f" % (areaCircleOneTriangle)
```



```
          #if debug2==1:print "     area for plant two triangle: %f" % (areaCircleTwoTriangle)
          areaCircleOneThumbnail=areaThumbnail(areaCircleOneSector, areaCircleOneTriangle)
          areaCircleTwoThumbnail=areaThumbnail(areaCircleTwoSector, areaCircleTwoTriangle)
          #if debug2==1:print "     area for circle one thumbnail: %f" %
(areaCircleOneThumbnail)
          #if debug2==1:print "     area for circle two thumbnail: %f" %
(areaCircleTwoThumbnail)
          areaOverlap= areaCircleOneThumbnail+ areaCircleTwoThumbnail
          #if debug2==1:print "     area of thumbnail: %f" % (areaOverlap)
        return areaOverlap
```

Vida/Vida_Data/list_utils.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

#import operator#python 2.4 only
def sort_by_attr(seq, attr):
  #python 2.3
  intermed=[(getattr(x, attr),i, x) for i, x in enumerate(seq)]
  intermed.sort()
  return [x[-1]for x in intermed]
  #python 2.4
  #return sorted(seq, key=operator.attrgetter(attr))

def sort_by_attr_inplace(lst, attr):
  #python 2.3
  lst[:]=sort_by_attr(lst,attr)
  #python 2.4
  #lst.sort(key=operator.attrgetter(attr))
```

Vida/Vida_Data/progressBarClass.py


```
# CLASS NAME: DLLInterface
#
# Author: Larry Bates (lbates@syscononline.com)
#
# Written: 12/09/2002
#
# Released under: GNU GENERAL PUBLIC LICENSE
#
#Minor changes by Sean T. Hammond 2008.03.12
class progressbarClass:
    def __init__(self, finalcount, progresschar=None):
        import sys
        self.finalcount=finalcount
        self.blockcount=0
        #
        # See if caller passed me a character to use on the
        # progress bar (like "*").  If not use the block
        # character that makes it look like a real progress
        # bar.
        #
        if not progresschar:
    self.block=chr(178)
        else:
    self.block=progresschar
        # Get pointer to sys.stdout so I can use the write/flush
        # methods to display the progress bar.
        self.f=sys.stdout
        # If the final count is zero, don't start the progress gauge
        if not self.finalcount :
    return
        self.f.write('[----------------- % Progress ----------------------]\n')
  self.f.write('[')
        #self.f.write('    10   20   30   40   50   60   70   80   90   100\n')
        #self.f.write('----|----|----|----|----|----|----|----|----|----|\n')
        return

    def update(self, count):
        # Make sure I don't try to go off the end (e.g. >100%)
        count=min(count, self.finalcount)
        # If finalcount is zero, I'm done
        if self.finalcount:
            percentcomplete=int(round(100*count/self.finalcount))
            if percentcomplete < 1:
    percentcomplete=1
        else:
            percentcomplete=100
        blockcount=int(percentcomplete/2)
        if blockcount > self.blockcount:
            for i in range(self.blockcount,blockcount):
                self.f.write(self.block)
                self.f.flush()

        if percentcomplete == 100:
    self.f.write("]\n\n")
        self.blockcount=blockcount
        return

#this is so you can test it alone
if __name__ == "__main__":
    maxValue=25000
    pb=progressbarClass(maxValue,"*")
    count=0
    while count< maxValue:
        count+=1
        pb.progress(count)
```

Vida/Vida_Data/sdxf_utils.py

This file is based on version 1.1 of Stani's DXF Python library to generate dxf drawings (http://www.stani.be/python/sdxf). The version used by Vida was extended to include various primitive DXF shapes, such as a sphere, hemisphere and cylinders. The sphere and hemisphere definition consist of a series of numbers defining the location and angles of triangles, resulting in a file which is several hundred pages long if printed. The file is not necessary for the core functionality of Vida, since it only allows one to generate dxf files. Due to its excessive length, it has been omitted. It can be viewed at http://www.ice-nine.org/seant/Vida/ .

Vida/Vida_Data/vdefaults.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

###default values for simulations
simulationName="default"
theWorldSize=100
startPopulationSize=1
maxPopulation=10
maxCycles=100
outputDirectory=""
webDirectory="../../public_html/Bahai/"
produceGraphics=0
produceDXFGraphics=0
produceVideo=0
framesPerSecond=15
webOutput=0
percentTimeStamp=2.5
theView=1
resumeSim=0
reloadSpeciesData=0
archive="e"
deleteCfdgFiles=1
deletePngFiles=1
saveData="e"
debug=0
debug2=0
showProgressBar=0
timesToRepeat=1
```

Vida/Vida_Data/vdxfGraphics.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import glob
import os
import sdxf_utils as sdxf

def makeDXF(theGarden):
  theData=sdxf.Drawing()
  #Blocks
  b=sdxf.Block('world')
  b.append(sdxf.Solid(points=[(0,0,0),(1,0,0),(1,1,0),(0,1,0)]))
  theData.blocks.append(b)
  b=sdxf.Block('stem')
  b.append(sdxf.Circle(center=(0,0,0),radius=1,thickness=1))
  theData.blocks.append(b)
  b=sdxf.Block('seed')
  b.append(sdxf.Sphere())
  theData.blocks.append(b)
  b=sdxf.Block('canopy')
  b.append(sdxf.Hemisphere())
  theData.blocks.append(b)
  theWorldSize=theGarden.theWorldSize

theData.append(sdxf.Insert('world',point=(0-(theWorldSize/2.0),0-(theWorldSize/2.0),0),xscale=th
eWorldSize,yscale=theWorldSize,zscale=0,color=0,rotation=0))
  for obj in theGarden.soil:
    x=obj.x
    y=obj.y
    z=obj.z
    if obj.isSeed:
      theSeedRadius=obj.radiusSeed*obj.radiusSeedMultiplier

theData.append(sdxf.Insert('seed',point=(x,y,0+theSeedRadius),xscale=theSeedRadius,yscale=theSee
dRadius,zscale=theSeedRadius,color=45,rotation=0))
    else:
      theStemRadius=obj.radiusStem*obj.radiusStemMultiplier
      theLeafRadius=obj.radiusLeaf*obj.radiusLeafMultiplier

theData.append(sdxf.Insert('canopy',point=(x,y,obj.heightStem-theLeafRadius),xscale=theLeafRadiu
s,yscale=theLeafRadius,zscale=theLeafRadius,color=90,rotation=0))

theData.append(sdxf.Insert('stem',point=(x,y,0),xscale=theStemRadius,yscale=theStemRadius,zscale
=obj.heightStem,color=45,rotation=0))
      for attachedSeed in obj.seedList:
        x= attachedSeed.x
        y= attachedSeed.y
        z= attachedSeed.z
        theSeedRadius= attachedSeed.radiusSeed* attachedSeed.radiusSeedMultiplier

theData.append(sdxf.Insert('seed',point=(x,y,z),xscale=theSeedRadius,yscale=theSeedRadius,zscale
=theSeedRadius,color=1,rotation=0))
  return theData

def writeDXF(outputDirectory, fileName, theData):
  ###writes the files to a destination folder
  theData.saveas(outputDirectory + fileName+".dxf")
```

Vida/Vida_Data/vextract.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import sys
import os
import shutil
import csv
import glob
import linecache
import pickle
###append the path to basic data files
sys.path.append("Vida_Data")
import vgraphics as outputGraphics
import vplantr as defaultSpecies

###import default settings
import vdefaults as thePrefs
produceGraphics=thePrefs.produceGraphics
produceVideo=thePrefs.produceVideo
percentTimeStamp=thePrefs.percentTimeStamp
framesPerSecond=thePrefs.framesPerSecond
produceSummary=0

class Species1(defaultSpecies.genericPlant):
  ###The routine in defaultSpecies.genericPlant reads in default values from .yml file
  def __init__(self):
    ##super(type, obj) -> bound super object; requires isinstance(obj, type)
    super(Species1, self).__init__()

def makeDirectory(theDirectory):
  if not os.path.exists(theDirectory):
    os.mkdir(theDirectory)

def saveDataToCSVFile (theDirectory, theFileName, theData):
  makeDirectory(theDirectory)
  if not theFileName.endswith('.csv'):
    theFileName=theFileName+".csv"
  saveDataFile =csv.writer(file(theDirectory + theFileName,
'w'),delimiter=',',lineterminator='\n')
  for l in theData:
    saveDataFile.writerow(l)

if __name__ == '__main__':
  ######Read in any run time arguments provided
  theArguments=sys.argv

  if "-n" in theArguments:
    loc= theArguments.index("-n")
    fileOrFolder=theArguments[loc+1]
    fileOrFolder=fileOrFolder.split(",")
    #print fileOrFolder

  ######################
  if "-g" in theArguments:
    produceGraphics=1
    theView=13
  if "-gb" in theArguments:
    produceGraphics=1
    theView=1
```



```
  if "-gt" in theArguments:
    produceGraphics=1
    theView=2
  if "-gs" in theArguments:
    produceGraphics=1
    theView=3
  if "-gts" in theArguments or "-gst" in theArguments:
    produceGraphics=1
    theView=23
  if "-gbs" in theArguments or "-gsb" in theArguments:
    produceGraphics=1
    theView=13
  if "-gbt" in theArguments:
    produceGraphics=1
    theView=12
  if "-gtb" in theArguments:
    produceGraphics=1
    theView=21
  if "-gbts" in theArguments:
    produceGraphics=1
    theView=123
  if "-v" in theArguments:
    produceVideo=1
    allFiles=0
  if "-va" in theArguments:
    produceVideo=1
    allFiles=1
  ##########################
  if "-fs" in theArguments:
    produceSummary=1
    allFiles=0
  if "-fsa" in theArguments:
    produceSummary=1
    allFiles=1

  #need to parse the -n to see if it's a file or folder
  theLastChar= fileOrFolder[0][len(fileOrFolder[0])-1]
  theOutputFolder=os.path.dirname(fileOrFolder[0])+"/"
  #print theOutputFolder
  if theLastChar=="/":
    theOutputFolder=os.path.abspath(fileOrFolder[0])+"/"
    fileOrFolder=os.listdir(theOutputFolder)
    fileOrFolder=[i for i in fileOrFolder if not i.startswith('.')]
  #print theOutputFolder
  if len(fileOrFolder)>0:
    fileName=fileOrFolder[0].split("/")[-1]
    #what is the file suffix?
    #fileSuffix=fileOrFolder[0].split("/")[-1]
    #fileSuffix=fileSuffix.split(".")[-1]
    fileSuffix=fileName.split(".")[-1]
    #print fileSuffix
    #try and get a simulation name
    #simulationName= fileOrFolder[0]
    #simulationName= simulationName.split("/")
    #simulationName=simulationName[-1]
    #simulationName= simulationName.split("-")[0]
    simulationName=fileName.split("-")[0]

  if produceGraphics==1:
    print "Tasked to produce graphics"
    if fileSuffix=="pickle":
      ###make the necessary directories, if needed
      outputGraphicsDirectory = theOutputFolder+"../Graphics/"
      makeDirectory(outputGraphicsDirectory)
      if theView==1:
        outputGraphicsDirectory = outputGraphicsDirectory +"bottom-up/"
      elif theView==2:
        outputGraphicsDirectory = outputGraphicsDirectory +"top-down/"
```



```
      elif theView==3:
        outputGraphicsDirectory = outputGraphicsDirectory +"side/"
      elif theView==12:
        outputGraphicsDirectory = outputGraphicsDirectory +"combined-bottom-top/"
      elif theView==21:
        outputGraphicsDirectory = outputGraphicsDirectory +"combined-top-bottom/"
      elif theView==13:
        outputGraphicsDirectory = outputGraphicsDirectory +"combined-bottom-side/"
      elif theView==23:
        outputGraphicsDirectory = outputGraphicsDirectory +"combined-top-side/"
      elif theView==123:
        outputGraphicsDirectory = outputGraphicsDirectory +"combined-bottom-top-side/"
      makeDirectory(outputGraphicsDirectory)
      CFDGtext=""

      ###this is a saved simulation state
      print "***Loading Simulation Data and making CFDG...***"
      for aFile in fileOrFolder:
        ###load in the pickle
        simulationFile=open(aFile, 'r')
        theGarden=pickle.load(simulationFile)
        simulationFile.close()
        ###set variables from garden data
        simulationName=theGarden.name
        cycleNumber=theGarden.cycleNumber
        ###make the cfdg
        if CFDGtext=="":
          CFDGtext=outputGraphics.initCFDGText(theGarden, theView, percentTimeStamp, 50.0)
        theData=outputGraphics.makeCFDG(theView, CFDGtext, theGarden, cycleNumber)
        cfdgFileName= simulationName+"-"+str(cycleNumber)
        outputGraphics.writeCFDG(outputGraphicsDirectory, cfdgFileName, theData)
    simulationData=""
    theGarden=""
    CFDGText=""
    theData=""
    cfdgFileName=""
    ###make the png
    #print "Producing PNG files..."
    #print outputGraphicsDirectory
    #print "###"
    #outputGraphics.outputPNGs(outputGraphicsDirectory, outputGraphicsDirectory)
    if fileSuffix =="cfdg":
      print "###Producing PNG files...###"
      outputGraphics.outputPNGs(fileOrFolder, outputGraphicsDirectory)

  if produceVideo==1:
    if fileSuffix =="png":
      if allFiles==0:
        theTempFolder=theOutputFolder+"_temp/"
        if not os.path.exists(theTempFolder):
          os.mkdir(theTempFolder)
        for aFile in fileOrFolder:
          shutil.copy(aFile, theTempFolder)
        outputGraphicsDirectory = theTempFolder
        produceGraphics=1
      else:
        outputGraphicsDirectory= theOutputFolder
        produceGraphics=1

  ##only try and make a video if it is wanted and if pngs were made
  if produceVideo and produceGraphics==1:
    print "Producing MOV file..."
    #print outputGraphicsDirectory
    outputGraphics.outputMOV(outputGraphicsDirectory, simulationName)
    if allFiles==0:
      ###delete the temp folder and files
      shutil.rmtree(outputGraphicsDirectory)
```



```
if produceSummary==1 and len(fileOrFolder)>0:
  #print 'yes'
  theSummaryOutput=[]
  statDictList=[]
  theSummaryOutputHeader=""
  for theFile in fileOrFolder:
    theFileName=os.path.basename(theFile)
    theOutput=[[],[]]
    if not os.path.exists(theFile):
      theFile=theOutputFolder+theFile
    if os.path.isfile(theFile):
      #what's the sequence # of the file?
      theSimName=theFileName.split("-")[0]
      theSequenceNum=theFileName.split("-")[-1]
      theSequenceNumb=int(theSequenceNumb.split(".csv")[0])
      theOutput[0].append("Cycle #")
      theOutput[1].append(theSequenceNumb)

      fileObject=open(theFile)
      try:
        allText=fileObject.readlines()
      finally:
        fileObject.close()

      ###get rid of the '\n' from end lines
      tempList=[aLine.rstrip('\n') for aLine in allText]
      allText=tempList
      ###get rid of any weird trailing of leading spaces
      tempList=[aLine.strip() for aLine in allText]
      allText=tempList
      ###turn the read file into a l list/line (2d array)
      tempList=[aLine.split(",") for aLine in allText]
      allText=tempList
      #print '######'
      #print allText

      #this makes sure the files are in the correct format
      #if not, updates them
      #allows for backward compatability
      if not tempList[0][0]=="Cycle #":
        for line in tempList:
          #print line
          line=line.insert(0, str(theSequenceNumb))
        tempList[0][0]="Cycle #"
        saveDataToCSVFile(theOutputFolder, theFileName, tempList)
      tempList=[]

      ##get average, max and min of data that are numbers
      ##In general, most of the things that are numbers should be used
      ##some should be ignored
      theIndex=0
      while theIndex<len(allText[0]):
        theColumn=[row[theIndex] for row in allText]
        ignoreTheseColumns=["Cycle #", " X Location", " Y Location"]
        totalCommunitySums=["Mass of Stem", "Mass of Canopy", "Mass Stem+Mass Canopy", "Mass
of all Seeds", "Mass Total", "Growth Stem (kg)", "Growth Canopy (kg)", "Growth Stem+Canopy
(kg)"]
        if not theColumn[0] in ignoreTheseColumns:
          try:
            #print theColumn[0]
            if theColumn[0]==" X Location":
              theIndex=theIndex
```

```
            else:
              float(theColumn[1])
          except:
            #do nothing
            #print "it's string"
            theIndex=theIndex
          else:
            theColumnTitle=theColumn.pop(0).strip()
            theColumn=[float(theRow) for theRow in theColumn]
            theColumnMin=min(theColumn)
            theColumnMax=max(theColumn)
            theColumnSum=sum(theColumn)
            theColumnAvg=theColumnSum/float(len(theColumn))
            theOutput[0].extend(["min "+theColumnTitle, "max "+theColumnTitle, "ave
"+theColumnTitle])
            theOutput[1].extend([theColumnMin, theColumnMax, theColumnAvg])
            if theColumnTitle in totalCommunitySums:
              theOutput[0].append("sum community "+theColumnTitle)
              theOutput[1].append(theColumnSum)
        theIndex=theIndex+1

      #How many seeds on the ground?
      try:
        theIndex=allText[0].index(" is a seed")
      except:
        theIndex="na"
      if not theIndex=="na":
        theColumn=[row[theIndex] for row in allText]
        theColumn.pop(0)
        theCount=0
        for i in range(len(theColumn)):
          theValue=theColumn[i]
          if theValue=="True":
            theCount=theCount+1
        theOutput[0].append("# of seeds in soil")
        theOutput[1].append(theCount)

      #print allText[0]
      #how many alive?
      try:
        theIndex=allText[0].index(" Cause of Death")
      except:
        theIndex="na"
      if not theIndex=="na":
        theColumn=[row[theIndex] for row in allText]
        thePopulationAlive=theColumn.count("na")
        thePopulationDead=len(theColumn)-1-thePopulationAlive
        theOutput[0].extend(["# Alive", "# Dead"])
        theOutput[1].extend([thePopulationAlive, thePopulationDead])
        #print theColumn
        theDeathNames=[]
        theDeathCount=[]
        for i in range(len(theColumn)):
          theName=theColumn[i]
          if not theName in theDeathNames:
            theDeathNames.append(theName)
            theDeathCount.append(theColumn.count(theColumn[i]))
      deathHeaders=["failed to germinate(immaturity)", "failed to germinate(other)",
"random death", "crushed", "overlap violation", "stem off world", "lack of light", "growth too
slow", "violated Euler-Greenhill"]
        for i in deathHeaders:
          theOutput[0].append("#%s" % (i))
          theValue=0
          if i in theDeathNames:
            theIndex=theDeathNames.index(i)
            theValue=theDeathCount[theIndex]
          theOutput[1].append(theValue)
```



```
      #How many Species
      try:
        theIndex=allText[0].index(" Species")
      except:
        theIndex="na"
      if not theIndex=="na":
        theColumn=[row[theIndex] for row in allText]
        theColumn.pop(0)
        theSpeciesNames=[]
        theSpeciesCount=[]
        for i in range(len(theColumn)):
          theName=theColumn[i]
          if not theName in theSpeciesNames:
            theSpeciesNames.append(theName)
            theSpeciesNames.sort()
            #theSpeciesCount.append(theColumn.count(theColumn[i]))
        theSpeciesNumb=len(theSpeciesNames)
        theOutput[0].append("# of species")
        theOutput[1].append(theSpeciesNumb)
        for name in theSpeciesNames:
          theOutput[0].append(name)
          theOutput[1].append(theColumn.count(name))

      ###make sure the header for the summary file is the one having the most data
      if len(theSummaryOutputHeader)<len(theOutput[0]):
        theSummaryOutputHeader=theOutput[0]

      ####this starts building a series of dictionaries for building an output file###
      statDict=dict(zip(theOutput[0], theOutput[1]))
      statDictList.append(statDict)
      ####Write the individual file
      theStatsFolder=theOutputFolder+"statistics/"
      saveDataToCSVFile(theStatsFolder, "stat_"+theFileName, theOutput)

  ###this code making the summary file is not so good. there are probably better solutions
  for statDict in statDictList:
    aLine=[]
    for key in theSummaryOutputHeader:
      theValue=statDict.get(key,0)
      aLine.append(theValue)
    theSummaryOutput.append(aLine)

  ###this sorting is so the cycle numbers are in order
  ###The real solution would be to list the files in the selection dialog in order
  theSummaryOutput=sorted(theSummaryOutput)
  theSummaryOutput.insert(0, theSummaryOutputHeader)

  ###save the data to a summary file
  theStatsFolder=theOutputFolder+"statistics/"
  saveDataToCSVFile(theStatsFolder, "summary_"+theSimName, theSummaryOutput)



  ###Go through and turn the individual .csv files into one big one
  ###Add cycle number if you need to. Assumes cycle # is the word between '_' and '.csv'
  #this is the simple way to cat the files. It assumes no changes need to be made.
  #Just concatenate them.
  concatFileName="merged_"+theSimName
  theOutput=open(theStatsFolder+concatFileName+".csv",'w')
  print "***Merging files...."
  fileList=glob.glob(theOutputFolder+"*.csv")
  theHeader=""
  for aFile in fileList:
    if theHeader=="":
      theHeader=linecache.getline(aFile,1)
      #print "this is the header: %s" % (theHeader)
    theFile=open(aFile)
```



```
    try:
      fileData=theFile.read()
      theOutput.write(fileData)
    finally:
      theFile.close()
  theOutput.close()
  print "***Finished merging. Removing extra headers..."
  import fileinput
  #print theStatsFolder+concatFileName+".csv"
  theFile=fileinput.input(theStatsFolder+concatFileName+".csv", inplace=1)
  i=0
  for line in theFile:
    if not line==theHeader or i==0:
      line=line.strip('\n')
      print line
    i=i+1
  print "***Finished***"
```

Vida/Vida_Data/vgraphics.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import glob
import os
import sys

CFDGtext="\
"""
startshape garden

//include ../../../Vida_Data/i_pix.cfdg
include %(thePathToiPix)s

rule garden{
  %(theWorldType)s
  %(theTimeStamp)s
  %%(thePopulationData)s
}

%(theWorldTypeRule)s

rule seed{CIRCLE{size 1 1}}
rule seedAttached{CIRCLE{size 1 1}}

rule leafCircle{CIRCLE{size 1 1}}
rule stemCircle{CIRCLE{size 1 1}}

rule leafRect{SQUARE{size 1 1}}
rule stemRect{SQUARE{size 1 1}}

rule leafHemi{
  91*[x 0.0086 r -1]SQUARE{size 0.1 1.0}
  91*[x -0.0086 r 1]SQUARE{size 0.1 1.0}
}

%(theTimeStampRule)s
"""

CFDGTopWorldRule=\
"""
rule worldTop{
  //provides border
  SQUARE{x 0 y 0 z -1000001 size %i %i b 1}
  SQUARE{x 0 y 0 z -1000000 size %i %i }
  //for 'blue sky':
  SQUARE{x 0 y 0 z -1000000 size %i %i hue 40.2 sat 0.4205 b 0.3837}
}
"""

CFDGBottomWorldRule=\
"""
rule worldBottom{
  //provides border
  SQUARE{x 0 y 0 z -1000001 size %i %i b 1}
  SQUARE{x 0 y 0 z -1000000 size %i %i }
  //for 'blue sky':
  SQUARE{x 0 y 0 z -1000000 size %i %i hue 206.6 sat 0.3037 b 0.74}
}
```



```
"""

CFDGSideWorldRule=\
"""
rule worldSide{
  //provides border
  SQUARE{x 0 y 0 z -1000001 size %i %i b 1}
  SQUARE{x 0 y 0 z -1000000 size %i %f}
  SQUARE{x 0 y 0 z -1000000 size %i %f b 1}
}
"""

CFDGTimeStamp="timeStamp{x %f y %f size %f %f}"

CFDGTimeStampRule=\
"""
rule timeStamp{
  C_5by5{x 0}
  Y_5by5{x 1.2}
  C_5by5{x 2.4}
  L_5by5{x 3.6}
  E_5by5{x 4.8}
      %(theCycleNumber)s
}
"""

############
def initCFDGText(theGarden, displayType, percentTimeStamp, maxHeightPlant):
  ###display type: 1=bottom up; 2=side; 3=bottom up and side
  global CFDGtext
  global CFDGTopWorldRule
  global CFDGSideWorldRule
  global CFDGTimeStamp
  global CFDGTimeStampRule
  theWorldSize=theGarden.theWorldSize
  fractWorldGraphicEdgeSpacer=theGarden.fractionWorldGraphicEdgeSpacer
  fractWorldBetweenLRGraphicsSpacer=theGarden.fractionWorldBetweenLRGraphicsSpacer
  fractWorldBetweenTBGraphicsSpacer=theGarden.fractionWorldBetweenTBGraphicsSpacer
  worldX=theWorldSize
  if displayType==1:
    ###bottom Up
    worldY1=theWorldSize
    worldY2=0
    worldY=worldY1+worldY2
    worldTypeCode="worldBottom{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0)
    worldTypeRule=CFDGBottomWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX*fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
  elif displayType==2:
    ###top down
    worldY1=theWorldSize
    worldY2=0
    worldY=worldY1+worldY2
    worldTypeCode="worldTop{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0)
    worldTypeRule=CFDGTopWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX*fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
  elif displayType==3:
    ###side view
    worldY1=0
    worldY2=maxHeightPlant
    worldY=worldY1+worldY2
    worldTypeCode="worldSide{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, worldY/2.0)
    worldTypeRule =CFDGSideWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX*fractWorldGraphicEdgeSpacer),
```



```
worldY2+(worldY1* fractWorldGraphicEdgeSpacer),worldX, worldY2, worldX-1, worldY2-1)
  elif displayType==12:
    ###bottom up and top down
    worldY1=theWorldSize
    worldY2=theWorldSize
    worldY= theWorldSize
    worldTypeCode="worldBottom{x %f y %f}\n        worldTop{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0, worldY1+(worldY1*
fractWorldBetweenLRGraphicsSpacer), 0.0)
    worldTypeRule=CFDGBottomWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
    worldTypeRule=worldTypeRule+CFDGTopWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
  elif displayType==13:
    ###bottom up and side view
    worldY1=theWorldSize
    worldY2=maxHeightPlant
    worldY=worldY1+worldY2
    worldTypeCode="worldBottom{x %f y %f}\n        worldSide{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0, 0.0,
((worldY+(theWorldSize*fractWorldBetweenTBGraphicsSpacer))/-2))
    worldTypeRule= CFDGBottomWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
    worldTypeRule=worldTypeRule+CFDGSideWorldRule
    #worldTypeRule=worldTypeRule % (worldX, worldY2, worldX-1, worldY2-1)
    worldTypeRule=worldTypeRule % (worldX+(worldX*fractWorldGraphicEdgeSpacer),
worldY2+(worldY1* fractWorldGraphicEdgeSpacer),worldX, worldY2, worldX-1, worldY2-1)
  elif displayType==21:
    ###bottom up and top down
    worldY1=theWorldSize
    worldY2=theWorldSize
    worldY= theWorldSize
    worldTypeCode="worldTop{x %f y %f}\n        worldBottom{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0, worldY1+(worldY1*
fractWorldBetweenLRGraphicsSpacer), 0.0)
    worldTypeRule=CFDGTopWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
    worldTypeRule=worldTypeRule+CFDGBottomWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
  elif displayType==23:
    ###Bottom up and side view
    worldY1=theWorldSize
    worldY2=maxHeightPlant
    worldY=worldY1+worldY2
    worldTypeCode="worldTop{x %f y %f}\n        worldSide{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0, 0.0,
((worldY+(theWorldSize*fractWorldBetweenTBGraphicsSpacer))/-2))
    worldTypeRule=CFDGTopWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY1+(worldY1* fractWorldGraphicEdgeSpacer), worldX, worldY1, worldX-1, worldY1-1)
    worldTypeRule=worldTypeRule+CFDGSideWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX* fractWorldGraphicEdgeSpacer),
worldY2+(worldY2* fractWorldGraphicEdgeSpacer),worldX, worldY2, worldX-1, worldY2-1)
  elif displayType==123:
    ###bottom up and top down and side
    worldY1=theWorldSize
    worldY2=maxHeightPlant
    worldY=worldY1+worldY2
    worldTypeCode="worldBottom{x %f y %f}\n        worldTop{x %f y %f}\n        worldSide{x %f
y %f}\n        worldSide{x %f y %f}"
    worldTypeCode=worldTypeCode % (0.0, 0.0, worldY1+(worldY1*0.3), 0.0, 0.0,
-(worldY2+(worldY1/4)), worldY1+(worldY1*0.3), -(worldY2+(worldY1/4)))
    worldTypeRule=CFDGBottomWorldRule
```



```
    worldTypeRule=worldTypeRule % (worldX+(worldX*0.3), worldY1+(worldY1*0.3), worldX, worldY1,
worldX-1, worldY1-1)
    worldTypeRule=worldTypeRule+CFDGTopWorldRule
    worldTypeRule=worldTypeRule % (worldX+(worldX*0.3), worldY1+(worldY1*0.3), worldX, worldY1,
worldX-1, worldY1-1)
    worldTypeRule=worldTypeRule+CFDGSideWorldRule
    worldTypeRule=worldTypeRule % (worldX, worldY2, worldX-1, worldY2-1)

  if percentTimeStamp>0.0:
    timeStampCode= CFDGTimeStamp
    ###this puts the time stamp in the lower left corner
    i=(percentTimeStamp*theWorldSize)/100
    x= (worldX/-2)+(i/2)
    if displayType==1 or displayType==2 or displayType==12 or displayType==21:
      ###bottom up or top down
      y= (worldY/-2)-(i/2)-1
    if displayType==3:
      y= 0-(i/2)-1
    if displayType==13 or displayType==23 or displayType==123:
      y=(worldY1/-2)-worldY2-(theWorldSize* fractWorldBetweenTBGraphicsSpacer/2)-(i/2)-1
    timeStampCode = timeStampCode % (x, y, i, i)
  if sys.platform=="win32":
    #why is it ../../ on windows? how weird
    CFDGtext=CFDGtext % {"thePathToiPix": "\"../../Vida_Data/i_pix.cfdg\"", "theWorldType":
worldTypeCode, "theTimeStamp": timeStampCode, "theWorldTypeRule": worldTypeRule,
'theTimeStampRule': CFDGTimeStampRule}
  else:
    CFDGtext=CFDGtext % {"thePathToiPix": "../../../Vida_Data/i_pix.cfdg", "theWorldType":
worldTypeCode, "theTimeStamp": timeStampCode, "theWorldTypeRule": worldTypeRule,
'theTimeStampRule': CFDGTimeStampRule}
  return CFDGtext


def makeCFDG(theView, CFDGtext, theGarden, cycleNumber):
  thePlantData=""
  theWorldSize=theGarden.theWorldSize
  fractWorldGraphicEdgeSpacer=theGarden.fractionWorldGraphicEdgeSpacer
  fractWorldBetweenLRGraphicsSpacer=theGarden.fractionWorldBetweenLRGraphicsSpacer
  fractWorldBetweenTBGraphicsSpacer=theGarden.fractionWorldBetweenTBGraphicsSpacer
  theData= CFDGtext
  seedCode="          seed{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f size %2f %2f}\n"
  attachedSeedCode="          seedAttached{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f
size %2f %2f}\n"
  leafCode1="          leafCircle{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f size %2f
%2f}\n"
  stemCode1="          stemCircle{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f size %2f
%2f}\n"
  leafCode2="          leafRect{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f size %2f
%2f}\n"
  stemCode2="          stemRect{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f size %2f
%2f}\n"
  leafCode3="          leafHemi{x %2f y %2f z %2f hue %2f sat %2f b %2f alpha -%f size %2f}\n"
  for obj in theGarden.soil:
    x=obj.x
    y=obj.y
    color1=obj.colourSpecies[0]
    color2=obj.colourSpecies[1]
    color3=obj.colourSpecies[2]
    alpha=obj.canopyTransmittance
    if obj.isSeed:
      theSeedDiameter=obj.radiusSeed*2*obj.radiusSeedMultiplier
      theSeedBorder=theSeedDiameter-(theSeedDiameter*obj.borderImagePercent/100)
      color4=obj.colourSeedDispersed[0]
      color5=obj.colourSeedDispersed[1]
      color6=obj.colourSeedDispersed[2]
      if theView==1:
        ###bottom up
        z1=10000.0
```

```
        thePlantData=thePlantData+seedCode %  (x, y, z1, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y, z1, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
      elif theView==2:
        ###top down
        z2=0.0
        thePlantData=thePlantData+seedCode %  (x, y, z2, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y, z2, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
      elif theView==3:
        ###side view
        y3=0.0
        z3=0.0
        thePlantData=thePlantData+seedCode %  (x, y3, z3, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y3, z3, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
      elif theView==12:
        ###bottom up
        z1=10000.0
        thePlantData=thePlantData+seedCode %  (x, y, z1, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y, z1, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
        ###top down
        z2=0.0
        thePlantData=thePlantData+seedCode %  ((theWorldSize)+(theWorldSize*
fractWorldBetweenLRGraphicsSpacer)-x, y, z2, color1, color2, color3, 0.0, theSeedDiameter,
theSeedDiameter)
        thePlantData=thePlantData+seedCode %  ((theWorldSize)+(theWorldSize*
fractWorldBetweenLRGraphicsSpacer)-x, y, z2, color4, color5, color6, 0.0, theSeedBorder,
theSeedBorder)
      elif theView==21:
        ###top down
        z2=0.0
        thePlantData=thePlantData+seedCode %  (x, y, z2, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y, z2, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
        ###bottom up and side view
        z1=10000.0
        thePlantData=thePlantData+seedCode %  ((theWorldSize)+(theWorldSize*
fractWorldBetweenLRGraphicsSpacer)-x, y, z1, color1, color2, color3, 0.0, theSeedDiameter,
theSeedDiameter)
        thePlantData=thePlantData+seedCode %  ((theWorldSize)+(theWorldSize*
fractWorldBetweenLRGraphicsSpacer)-x, y, z1, color4, color5, color6, 0.0, theSeedBorder,
theSeedBorder)
      elif theView==13 or theView==23:
        if theView==13: z1=10000.0
        if theView==23: z1=0.0
        #40 is the size of the side view world.
        #set up in init of graphics
        y3=(theWorldSize/-2)-50.0-(theWorldSize* fractWorldBetweenTBGraphicsSpacer/2)
        z3=0.0
        thePlantData=thePlantData+seedCode %  (x, y, z1, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y, z1, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
        thePlantData=thePlantData+seedCode %  (x, y3, z3, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y3, z3, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
      elif theView==123:
        z1=10000.0
        y3=(theWorldSize/-2)-35
        z3=0.0
```



```
        ###bottom up
        thePlantData=thePlantData+seedCode %  (x, y, z1, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y, z1, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
        ###top down
        thePlantData=thePlantData+seedCode %  (x+10, y, z1, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x+10, y, z1, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
        ###bottom up side
        thePlantData=thePlantData+seedCode %  (x, y3, z3, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x, y3, z3, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
        ###top down side
        thePlantData=thePlantData+seedCode %  (x+10, y3, z3, color1, color2, color3, 0.0,
theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+seedCode %  (x+10, y3, z3, color4, color5, color6, 0.0,
theSeedBorder, theSeedBorder)
      else:
        theStemDiameter=obj.radiusStem*2*obj.radiusStemMultiplier
        theStemBorderlxy= theStemDiameter-(theStemDiameter*obj.borderImagePercent/100)
        theLeafDiameter=obj.radiusLeaf*2*obj.radiusLeafMultiplier
        theLeafBorderlxy= theLeafDiameter-(theLeafDiameter*obj.borderImagePercent/100)
        if theView==1 or theView==13 or theView==23:
          if obj.heightLeafMax<=0.1:
            leafHeight=0.1##this is so a very thin leaf is visable at all
          else:
            leafHeight=obj.heightLeafMax
          theStemBorderAdj= theStemDiameter*obj.borderImagePercent/100
          theLeafBorderAdj= theLeafDiameter*obj.borderImagePercent/100
          theStemBorder2x=theStemDiameter-theStemBorderAdj
          theStemBorder2y=obj.heightStem-theStemBorderAdj
          theLeafBorder2x=theLeafDiameter-theLeafBorderAdj
          theLeafBorder2y=leafHeight-theLeafBorderAdj
        color4L=obj.colourLeaf[0]
        color5L=obj.colourLeaf[1]
        color6L=obj.colourLeaf[2]
        color4S=obj.colourStem[0]
        color5S=obj.colourStem[1]
        color6S=obj.colourStem[2]
        if theView==1:
          z1Leaf=obj.heightStem+obj.heightLeafMax
          z1Stem=obj.heightStem
          thePlantData= thePlantData+leafCode1 %  (x, y, -z1Leaf, color1, color2, color3, alpha,
theLeafDiameter, theLeafDiameter)
          thePlantData=thePlantData+leafCode1 %  (x, y, -z1Leaf, color4L, color5L, color6L,
alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
          thePlantData=thePlantData+stemCode1 %  (x, y, -z1Stem, color1, color2, color3, 0.0,
theStemDiameter, theStemDiameter)
          thePlantData=thePlantData+stemCode1 %  (x, y, -z1Stem, color4S, color5S, color6S, 0.0,
theStemBorderlxy, theStemBorderlxy)
        elif theView==2:
          z2Leaf=obj.heightStem+obj.heightLeafMax
          z2Stem=obj.heightStem
          thePlantData= thePlantData+leafCode1 %  (x, y, z2Leaf, color1, color2, color3, alpha,
theLeafDiameter, theLeafDiameter)
          thePlantData=thePlantData+leafCode1 %  (x, y, z2Leaf, color4L, color5L, color6L,
alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
          thePlantData=thePlantData+stemCode1 %  (x, y, z2Stem, color1, color2, color3, 0.0,
theStemDiameter, theStemDiameter)
          thePlantData=thePlantData+stemCode1 %  (x, y, z2Stem, color4S, color5S, color6S, 0.0,
theStemBorderlxy, theStemBorderlxy)
        elif theView==3:
          y2=obj.heightStem+(obj.heightLeafMax/2.0)
          z3=-y
          thePlantData=thePlantData+ attachedSeedCode %  (x, y3, z3, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
          thePlantData=thePlantData+ attachedSeedCode %  (x, y3, z3, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
        elif theView==12:
          z1=10000.0
          z1=obj.heightStem+obj.heightLeafMax
          thePlantData=thePlantData+ attachedSeedCode %  (x, y, -z1, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
          thePlantData=thePlantData+ attachedSeedCode %  (x, y, -z1, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
          z2=10000.0
          z2=obj.heightStem+obj.heightLeafMax
          thePlantData=thePlantData+ attachedSeedCode %  ((theWorldSize)+(theWorldSize*0.3)-x,
y, z2, color1, color2, color3, 0.0, theSeedDiameter, theSeedDiameter)
          thePlantData=thePlantData+ attachedSeedCode %  ((theWorldSize)+(theWorldSize*0.3)-x,
y, z2, color4, color5, color6, 0.0, theSeedBorder, theSeedBorder)
        elif theView==21:
          z2=10000.0
```



```
        if obj.leafIsHemisphere:
          thePlantData=thePlantData+leafCode3 %  (x, (y2-(theLeafDiameter/4.0)), z3, color1,
color2, color3, alpha, theLeafDiameter/2.0)
          thePlantData=thePlantData+leafCode3 %  (x, (y2-(theLeafDiameter/4.0)), z3, color4L,
color5L, color6L, alpha+alpha, theLeafBorder2x/2.0)
        else:
          thePlantData= thePlantData+leafCode2 %  (x, y2, z3, color1, color2, color3, alpha,
theLeafDiameter, leafHeight)
          thePlantData=thePlantData+leafCode2 %  (x, y2, z3, color4L, color5L, color6L,
alpha+alpha, theLeafBorder2x, theLeafBorder2y)
        thePlantData=thePlantData+stemCode2 %  (x, y3, z3-0.00003, color1, color2, color3, 0.0,
theStemDiameter, obj.heightStem)
        thePlantData=thePlantData+stemCode2 %  (x, y3, z3-0.00003, color4S, color5S, color6S,
0.0, theStemBorder2x, theStemBorder2y)
      elif theView==12:
        z1Leaf=obj.heightStem+obj.heightLeafMax
        z1Stem=obj.heightStem
        thePlantData= thePlantData+leafCode1 %  (x, y, -z1Leaf, color1, color2, color3, alpha,
theLeafDiameter, theLeafDiameter)
        thePlantData=thePlantData+leafCode1 %  (x, y, -z1Leaf, color4L, color5L, color6L,
alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
        thePlantData=thePlantData+stemCode1 %  (x, y, -z1Stem, color1, color2, color3, 0.0,
theStemDiameter, theStemDiameter)
        thePlantData=thePlantData+stemCode1 %  (x, y, -z1Stem, color4S, color5S, color6S, 0.0,
theStemBorderlxy, theStemBorderlxy)
        z2Leaf=obj.heightStem+obj.heightLeafMax
        z2Stem=obj.heightStem
        thePlantData= thePlantData+leafCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y,
z2Leaf, color1, color2, color3, alpha, theLeafDiameter, theLeafDiameter)
        thePlantData=thePlantData+leafCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y, z2Leaf,
color4L, color5L, color6L, alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
        thePlantData=thePlantData+stemCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y, z2Stem,
color1, color2, color3, 0.0, theStemDiameter, theStemDiameter)
        thePlantData=thePlantData+stemCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y, z2Stem,
color4S, color5S, color6S, 0.0, theStemBorderlxy, theStemBorderlxy)
      elif theView==21:
        z2Leaf=obj.heightStem+obj.heightLeafMax
        z2Stem=obj.heightStem
        thePlantData= thePlantData+leafCode1 %  (x, y, z2Leaf, color1, color2, color3, alpha,
theLeafDiameter, theLeafDiameter)
        thePlantData=thePlantData+leafCode1 %  (x, y, z2Leaf, color4L, color5L, color6L,
alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
        thePlantData=thePlantData+stemCode1 %  (x, y, z2Stem, color1, color2, color3, 0.0,
theStemDiameter, theStemDiameter)
        thePlantData=thePlantData+stemCode1 %  (x, y, z2Stem, color4S, color5S, color6S, 0.0,
theStemBorderlxy, theStemBorderlxy)
        z1Leaf=obj.heightStem+obj.heightLeafMax
        z1Stem=obj.heightStem
        thePlantData= thePlantData+leafCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y,
-z1Leaf, color1, color2, color3, alpha, theLeafDiameter, theLeafDiameter)
        thePlantData=thePlantData+leafCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y,
-z1Leaf, color4L, color5L, color6L, alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
        thePlantData=thePlantData+stemCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y,
-z1Stem, color1, color2, color3, 0.0, theStemDiameter, theStemDiameter)
        thePlantData=thePlantData+stemCode1 %  ((theWorldSize)+(theWorldSize*0.3)-x, y,
-z1Stem, color4S, color5S, color6S, 0.0, theStemBorderlxy, theStemBorderlxy)
      elif theView==13 or theView==23:
        z1Leaf=obj.heightStem+obj.heightLeafMax
        z1Stem=obj.heightStem
        if theView==13:
          zLeaf=0-z1Leaf
          zStem=0-z1Stem
        y2=(theGarden.theWorldSize/-2)-50.0+obj.heightStem-(theWorldSize*
fractWorldBetweenTBGraphicsSpacer/2)
        y3=(theGarden.theWorldSize/-2)-50.0+(obj.heightStem/2.0)-(theWorldSize*
fractWorldBetweenTBGraphicsSpacer/2)
        z3=-y
        thePlantData= thePlantData+leafCode1 %  (x, y, z1Leaf, color1, color2, color3, alpha,
```



```
theLeafDiameter, theLeafDiameter)
        thePlantData=thePlantData+leafCode1 %  (x, y, z1Leaf, color4L, color5L, color6L,
alpha+alpha, theLeafBorderlxy, theLeafBorderlxy)
        thePlantData=thePlantData+stemCode1 %  (x, y, z1Stem, color1, color2, color3, 0.0,
theStemDiameter, theStemDiameter)
        thePlantData=thePlantData+stemCode1 %  (x, y, z1Stem, color4S, color5S, color6S, 0.0,
theStemBorderlxy, theStemBorderlxy)

        if obj.leafIsHemisphere:
          thePlantData=thePlantData+leafCode3 %  (x, (y2-(theLeafDiameter/4.0)), z3, color1,
color2, color3, alpha, theLeafDiameter/2.0)
          thePlantData=thePlantData+leafCode3 %  (x, (y2-(theLeafDiameter/4.0)), z3, color4L,
color5L, color6L, alpha+alpha, theLeafBorder2x/2.0)
        else:
          thePlantData= thePlantData+leafCode2 %  (x, y2, z3, color1, color2, color3, alpha,
theLeafDiameter, leafHeight)
          thePlantData=thePlantData+leafCode2 %  (x, y2, z3, color4L, color5L, color6L,
alpha+alpha, theLeafBorder2x, theLeafBorder2y)
          thePlantData=thePlantData+stemCode2 %  (x, y3, z3-0.00003, color1, color2, color3, 0.0,
theStemDiameter, obj.heightStem)
          thePlantData=thePlantData+stemCode2 %  (x, y3, z3-0.00003, color4S, color5S, color6S,
0.0, theStemBorder2x, theStemBorder2y)

        for attachedSeed in obj.seedList:
          x= attachedSeed.x
          y= attachedSeed.y
          theSeedDiameter= attachedSeed.radiusSeed*2* attachedSeed.radiusSeedMultiplier
          theSeedBorder=theSeedDiameter-(theSeedDiameter* attachedSeed.borderImagePercent/100)
          color4= attachedSeed.colourSeedAttached[0]
          color5= attachedSeed.colourSeedAttached[1]
          color6= attachedSeed.colourSeedAttached[2]
          if theView==1:
            #z1=10000.0
            z1=obj.heightStem+obj.heightLeafMax
            thePlantData=thePlantData+ attachedSeedCode %  (x, y, -z1, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
            thePlantData=thePlantData+ attachedSeedCode %  (x, y, -z1, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
          elif theView==2:
            #z2=10000.0
            z2=obj.heightStem+obj.heightLeafMax
            thePlantData=thePlantData+ attachedSeedCode %  (x, y, z2, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
            thePlantData=thePlantData+ attachedSeedCode %  (x, y, z2, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
          elif theView==3:
            y2=obj.heightStem+(obj.heightLeafMax/2.0)
            z3=-y
            thePlantData=thePlantData+ attachedSeedCode %  (x, y3, z3, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
            thePlantData=thePlantData+ attachedSeedCode %  (x, y3, z3, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
          elif theView==12:
            #z1=10000.0
            z1=obj.heightStem+obj.heightLeafMax
            thePlantData=thePlantData+ attachedSeedCode %  (x, y, -z1, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
            thePlantData=thePlantData+ attachedSeedCode %  (x, y, -z1, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
            #z2=10000.0
            z2=obj.heightStem+obj.heightLeafMax
            thePlantData=thePlantData+ attachedSeedCode %  ((theWorldSize)+(theWorldSize*0.3)-x,
y, z2, color1, color2, color3, 0.0, theSeedDiameter, theSeedDiameter)
            thePlantData=thePlantData+ attachedSeedCode %  ((theWorldSize)+(theWorldSize*0.3)-x,
y, z2, color4, color5, color6, 0.0, theSeedBorder, theSeedBorder)
          elif theView==21:
            #z2=10000.0
```

```
        z2=obj.heightStem+obj.heightLeafMax
        thePlantData=thePlantData+ attachedSeedCode %  (x, y, z2, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+ attachedSeedCode %  (x, y, z2, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
        #z1=10000.0
        z1=obj.heightStem+obj.heightLeafMax
        thePlantData=thePlantData+ attachedSeedCode %  ((theWorldSize)+(theWorldSize*0.3)-x,
y, -z1, color1, color2, color3, 0.0, theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+ attachedSeedCode %  ((theWorldSize)+(theWorldSize*0.3)-x,
y, -z1, color4, color5, color6, 0.0, theSeedBorder, theSeedBorder)
      elif theView==13 or theView==23:
        z1=obj.heightStem+obj.heightLeafMax
        if theView==13: z1=0-z1
        #y3=(theGarden.theWorldSize/-2)-50.0+obj.heightStem+(obj.heightLeafMax/2.0)
        y3=(theGarden.theWorldSize/-2)-50.0+attachedSeed.z-(theWorldSize*
fractWorldBetweenTBGraphicsSpacer/2)
        z3=-y
        thePlantData=thePlantData+ attachedSeedCode %  (x, y, z1, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+ attachedSeedCode %  (x, y, z1, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)
        thePlantData=thePlantData+ attachedSeedCode %  (x, y3, z3, color1, color2, color3,
0.0, theSeedDiameter, theSeedDiameter)
        thePlantData=thePlantData+ attachedSeedCode %  (x, y3, z3, color4, color5, color6,
0.0, theSeedBorder, theSeedBorder)

  ###insert the time stamp into graphics
  timeString=str(cycleNumber)
  loc=7.2
  theTime=""
  for k in range(len(timeString)):
    theTime= theTime+"     NUM_%i_5by5{x %2f}\n" % (int(timeString[k]), loc)
    loc=loc+1.2

  theData=CFDGtext % {"thePopulationData":thePlantData, "theCycleNumber":theTime}

  return theData

def writeCFDG(outputDirectory, fileName, theData):
  ###writes the files to a destination folder
  cfdg = open(outputDirectory + fileName+".cfdg", 'w')
  cfdg.write(theData)
  cfdg.close()

def outputPNGs(inputDirectory, outputDirectory):
  #can supply a directory or list of files
  #theInput should either be a list of files or a directory
  if type(inputDirectory)==list:
    allTargetFiles=inputDirectory
  else:
    allTargetFiles=glob.glob(inputDirectory +"*.cfdg")
  inputDirectory=os.path.dirname(allTargetFiles[0])+"/"
  for fileItem in allTargetFiles:
    pngFileName =fileItem.replace(outputDirectory, "")
    pngFileName = pngFileName.replace(".cfdg", "")
    pngFileName= pngFileName +".png"
    ###this should get the cfdg app to do its thing
    ###windows and linux seem to handle path names differently
    if sys.platform=="win32":
      theArg="ContextFreeCLI.exe /c /b 0 /s 500 %s %s"
      theArg=theArg % (fileItem, pngFileName)
    else:
      theArg="cfdg -c -b 0 -s 500 %s %s > /dev/null"
      #theArg="cfdg -c -b 0 -s 500 %s %s"
      theArg=theArg % (fileItem, outputDirectory + pngFileName)
    os.system(theArg)
```



```
def deleteCFDGFiles(outputDirectory):
  allTargetFiles=glob.glob(outputDirectory+"*.cfdg")
  for fileItem in allTargetFiles:
    #print "Deleting .cfdg file %s." % (fileItem)
    os.remove(fileItem)

def outputMOV(outputDirectory, simulationName, framesPerSec):
  if sys.platform=="darwin":
    ###Absolute path to output png files necessary
    ###for the embedded applescript to work correctly
    absolutePath=os.path.abspath(outputDirectory)
    #print "################"
    #print simulationName
    #print outputDirectory
    #print glob.glob(outputDirectory+"*.png")
    firstFile=glob.glob(outputDirectory+"*.png")[0]
    firstFile=firstFile.replace(outputDirectory,"")
    ###turn "/" into ":" so applescript understand the path
    absolutePath= absolutePath.replace("/", ":")
    if not absolutePath.endswith(":"):
      absolutePath=absolutePath+":"
    ###get quicktime to make the video###
    appleScriptCmd="""osascript<<END
    set folderPath to path to me as string
    set the clipboard to folderPath as text
    tell application "QuickTime Player"
    activate
    --need absolute path of file here--
    open image sequence "%s%s" frames per second %i
    tell movie 1
      save self contained in "%s:%s.mov"
      --optional close?
      --close
    end tell
    end tell
    """
    appleScriptCmd=appleScriptCmd % (absolutePath, firstFile, framesPerSec, absolutePath,
simulationName)
    os.system(appleScriptCmd)
    print "      Quicktime video made"
    ###Delete png files if requested
    #if deletePngFiles:
    # allTargetFiles =glob.glob(outputDirectory+"*.png")
    # for fileItem in allTargetFiles:
    #   print "Deleting .png file %s." % (fileItem)
    #   os.remove(fileItem)
  else:
    print """A video was not made because this feature currently only works on Macintosh OS
X"""
```

Vida/Vida_Data/vplantr.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import sys
import math
import time
import copy
import random
###append the path to basic data files
sys.path.append("Vida_Data")
import geometry_utils
import yaml

debug=0

#class genericPlant(object):
class genericPlant(object):
  ###define the props on this object
  def __init__(self):
    ### if the object lacks a certain property, it has not imported the basic prefs
    if not hasattr(self, "massSeedMax"):
      fileLoc="Vida_Data/Default_species.yml"
      self.importPrefs(fileLoc)
    self.zeroSeedValues()


  def zeroSeedValues(self):
    ###these values are set during the simulation
    self.name=""
    self.isSeed=True
    self.isMature=False
    self.timeCreation=0
    self.timeGermination=0
    self.timePlanted=0
    self.subregion=[]
    self.massStem=0.0 #kg
    self.massLeaf=0.0 #kg
    self.massSeed=0.0 #kg
    self.massTotal=0.0 #kg

    self.massFixed=-0.0 #kg
    self.massFixedRecord=[]#list of kg

    self.massSeedsTotal=0.0 #kg
    self.heightStem=0.0
    self.radiusStem=0.0 #meters
    self.radiusLeaf=0.0 #meters
    self.radiusSeed=0.0 #radius in meters
    self.areaPhotosynthesis=0.0#meters^2
    self.areaCovered=0.0#meters^2
    self.fractionAreaCovered=0.0
    self.seedList=[]
    self.motherPlant=0
    self.countToGerm=0.0
    ###experimental. allows a plant to change how it does photosythesis based on how shaded it
is
    self.photoConstantShade=self.photoConstant#
    ############################
```



```
    self.prevHeightGrowthRate=0.0
    self.heightGrowthRate=[]
    self.avgHeightGrowthRate=0.0
    self.maxAvgHeightGrowthRate=0.0

    self.GHs=0.0 #change in Height. In meters.
    self.GRs=0.0 #change in radius of stem. In meters.
    self.GMs=0.0 #change in mass of stem. In kg.
    self.GMl=0.0 #change in mass of leaves. In kg.

    self.causeOfDeath=""
    ###World data
    self.x=0.0
    self.y=0.0
    self.z=0.0
    self.r=0.00
    self.age=0.0
    self.overlapList=[]
    self.colourLeaf[2]=1.0
    #fileLoc="Vida_Data/Default_species.yml"
    #self.importPrefs(fileLoc)

  def importPrefs(self, fileLoc):
    theFile=open(fileLoc)
    theData=yaml.load(theFile)
    theFile.close
    for key in theData:
      setattr(self, key, theData[key])

  def dieNow(self, thePlant, theGarden):
    #die!
    #print len(thePlant.seedList)
    #print "hmmm"
    if len(thePlant.seedList)>0:
      for theSeed in thePlant.seedList:
        ###The plant has seeds. Drop them prior to killing plant###
        ###just drop the seed straight down###
        #print "********seed %s be being placed at %f, %f" % (theSeed.name, newX, newY)
        theSeed.countToGerm=self.delayInGermination
        theGarden.plantSeed(theSeed)
        thePlant.seedList.remove(theSeed)
    thePlant.seedList=[]
    theGarden.soil.remove(self)
    if self.isSeed:
      theGarden.numbSeeds= theGarden.numbSeeds-1
    else:
      theGarden.numbPlants=theGarden.numbPlants-1


  def growPlant(self, theGarden):
    ##if you have attached seeds, allow them to grow
    tempList=copy.copy(self.seedList)
    if self.makeSeeds:
      numbSeeds=float(len(tempList))
      self.massSeedsTotal=0
      if numbSeeds>0.0:
        maxCarbonToSeed=(self.massFixed*self.fractionCarbonToSeeds)
        maxCarbonToSeed=maxCarbonToSeed/numbSeeds
        for attachedSeed in tempList:
          if self.massFixed<0.0:
            break
          ###I don't like this much
          #it gives a base amount of mass to the seed
          #+/- a random amount
          sign=random.random()
          theVarient=(random.random()*(maxCarbonToSeed-0)+0)
          if sign>0.5:
            ###Don't give the seed more than you have
```



```
            carbonToSeed = maxCarbonToSeed +theVarient
            if carbonToSeed>self.massFixed:
              carbonToSeed=self.massFixed
          else:
            carbonToSeed = maxCarbonToSeed -theVarient
          attachedSeed.growSeedOnPlant(carbonToSeed)
          ###if the seed is the max size, disperse it
          if attachedSeed.massSeed>=attachedSeed.massSeedMax:
            attachedSeed.disperseSeed(attachedSeed, self, theGarden)
          else:
            self.massSeedsTotal=self.massSeedsTotal+attachedSeed.massSeed
            #print "plant %f" % (self.massSeedsTotal)

      ##if you lack seeds, make them
      #if (self.heightStem>=self.startMakingSeedsHeight) or
(self.age>=self.startMakingSeedsAge) or self.isMature:
      if self.isMature:
        self.makeSomeSeeds(theGarden.maxSeedsPerPlant)

    self.calcMassStemFromMassNew()

    self.calcRadiusStemFromMassStem()

    self.calcHeightStemFromRadiusStem(theGarden)

    self.calcMassLeafFromMassStem()

    self.massTotal=self.massLeaf+self.massStem+self.massSeedsTotal


    ###convert mass to get radius of leaf
    self.calcRadiusLeafFromMassLeaf()

    self.z=self.heightStem+self.heightLeafMax

    if self.radiusLeaf>=self.radiusStem:
      self.r=self.radiusLeaf
    else:
      self.r=self.radiusStem
    #save the changes in height over time
    self.heightGrowthRate.append(self.heightStem-self.prevHeightGrowthRate)
    self.prevHeightGrowthRate=self.heightStem
    while len(self.heightGrowthRate)>self.numYearsGrowthMemory:
      self.heightGrowthRate.pop(0)
    self.avgHeightGrowthRate=sum(self.heightGrowthRate)/float(len(self.heightGrowthRate))
    if self.avgHeightGrowthRate>self.maxAvgHeightGrowthRate:
      self.maxAvgHeightGrowthRate=self.avgHeightGrowthRate
    self.age=self.age+1


  def makeSeed(self, theSeed):
    theSeed=copy.deepcopy(self)
    theSeed.zeroSeedValues()
    theNameList= theSeed.name.split()
    if len(theNameList)<2:
      idNumb=str(random.random())
    else:
      idNumb=theNameList[1]
    theMax=self.locSeedFormation[0]
    theMin=self.locSeedFormation[1]
    ###Check these values and fix them if they have weird values
    if theMax>1.0: self.locSeedFormation[0]=1.0
    if theMax<0.0: self.locSeedFormation[0]=0.0
    if theMin>1.0: self.locSeedFormation[1]=1.0
    if theMin<0.0: self.locSeedFormation[1]=0.0
    ###
    theRadius=self.radiusLeaf
    r=theRadius*theMax
    delta=theRadius*theMin
```



```
    r=(random.random()*(r-delta))+ delta
    twoPi=3.14*2
    theAngle=(random.random()*(twoPi-0)+0)
    seedX=r*math.cos(theAngle)
    seedY=r*math.sin(theAngle)
    theSeed.x= seedX +self.x
    theSeed.y= seedY +self.y
    if self.leafHemisphere:
      dist=geometry_utils.distBetweenPoints(seedX+self.x, seedY+self.y, self.x, self.y)
      if dist==self.radiusLeaf:
        deltaY=self.radiusLeaf
      elif dist==0.0:
        deltaY=0.0
      else:
        #deltaY=self.radiusLeaf-math.sqrt(abs((self.radiusLeaf*self.radiusLeaf)-(dist*dist)))
        deltaY=self.radiusLeaf-((abs((self.radiusLeaf*self.radiusLeaf)-(dist*dist)))**0.5)
      seedZ=self.heightStem+(self.heightLeafMax/2.0)-deltaY
    else:
      seedZ=self.heightStem+(self.heightLeafMax/2.0)

    theSeed.name="growingSeed %s" % (idNumb)
    theSeed.timeCreation=time.time()
    theSeed.motherPlant=self
    theSeed.x= seedX +self.x
    theSeed.y= seedY +self.y
    theSeed.z= seedZ
    theSeed.r= theSeed.radiusSeed
    self.seedList.append(theSeed)

  def growSeedOnPlant(self, maxCarbonForSeed):
    if maxCarbonForSeed > self.motherPlant.massSeedMax:
      maxCarbonForSeed =self.motherPlant.massSeedMax
    if not self.motherPlant.massFixed==-0.0:
      if maxCarbonForSeed>self.motherPlant.massFixed:
        maxCarbonForSeed=self.motherPlant.massFixed
    self.motherPlant.massFixed=self.motherPlant.massFixed-maxCarbonForSeed
    self.massSeed= self. massSeed + maxCarbonForSeed
    #use that value to calculate the radius of the seed
    #convert mass to volume and the get the radius
    i= self.massSeed/self.densitySeed #this is volume in m^3
    #i=i/(4.0/3.0)
    i=i/1.3333
    #i=i/(math.pi)
    i=i/3.14
    #i=math.pow(i, 1.0/3.0)
    #i=math.pow(i, 0.3333)
    i=i**0.3333
    self.radiusSeed=i

    ###need to move the seed as the plant grows###
    if self.motherPlant.leafIsHemisphere:
      dist=geometry_utils.distBetweenPoints(self.x, self.y, self.motherPlant.x,
self.motherPlant.y)
      if dist==self.motherPlant.radiusLeaf:
        deltaY=self.motherPlant.radiusLeaf
      elif dist==0.0:
        deltaY=0.0
      else:
deltaY=self.motherPlant.radiusLeaf-((abs((self.motherPlant.radiusLeaf*self.motherPlant.radiusLea
f)-(dist*dist)))**0.5)
        seedZ=self.motherPlant.heightStem+(self.motherPlant.heightLeafMax/2.0)-deltaY
    else:
      seedZ=self.motherPlant.heightStem+(self.motherPlant.heightLeafMax/2.0)
    self.z= seedZ
    self.r= self.radiusSeed

  def disperseSeed(self, theSeed, motherPlant, theGarden):
```

```
    ###this dispersal method needs to be better
    if motherPlant.seedDispersalMethod[0]==0:
      ###This is just random anywhere in world###
      newX
=random.randrange(-(theGarden.theWorldSize/2),(theGarden.theWorldSize/2))+random.random()
      newY
=random.randrange(-(theGarden.theWorldSize/2),(theGarden.theWorldSize/2))+random.random()
    elif motherPlant.seedDispersalMethod[0]==1:
      ###just drop the seed straight down###
      newX=theSeed.x
      newY=theSeed.y
    elif motherPlant.seedDispersalMethod[0]==2:
      ###drop the seed in a circle with a defined max radius###
      theRadius=motherPlant.seedDispersalMethod[1]
      theMax=1
      theMin=0
      r=theRadius*theMax
      delta=theRadius*theMin
      r=(random.random()*(r-delta))+ delta
      #twoPi=math.pi*2
      twoPi=3.14*2
      theAngle=(random.random()*(twoPi-0)+0)
      newX =r*math.cos(theAngle)
      newY =r*math.sin(theAngle)
      newX = newX + theSeed.x
      newY = newY + theSeed.y
    elif motherPlant.seedDispersalMethod[0]==3:
      ###eject the seed straight out from leaf, traveling a defined max distance###
      maxDistance=motherPlant.seedDispersalMethod[1]
      theMax=1
      theMin=0
      theDistance= maxDistance*theMax
      delta= maxDistance*theMin
      theDistance =(random.random()*(theDistance-delta))+ delta
      theHypot=geometry_utils.distBetweenPoints(motherPlant.x, motherPlant.y, theSeed.x,
theSeed.y)
      theRise=theSeed.y-motherPlant.y
      theRun=theSeed.x-motherPlant.x
      theAngle=math.asin(theRise/theHypot)
      #theAngle=math.degrees(theAngle)
      newX=math.cos(theAngle)*theDistance
      if theRun<0.0:
        newX=0.0-newX
      newY=math.sin(theAngle)*theDistance
      newX=newX+theSeed.x
      newY=newY+theSeed.y
    elif motherPlant.seedDispersalMethod[0]==4:
      ###get a random angle based on input###
      angle=motherPlant.seedDispersalMethod[1]
      theVariance=(random.random()*((angle*0.5)-0.0))+ 0.0
      if random.random()>0.5:
        theVariance=0-theVariance
      angle=angle+theVariance
      angle=math.radians(angle)

      ###get a random velocity based on input###
      ####it would be nicer to use a force so seed size influences things too
      v=motherPlant.seedDispersalMethod[2]
      theVariance=(random.random()*((v*0.5)-0.0))+ 0.0
      if random.random()>0.5:
        theVariance=0-theVariance
      v=v+theVariance

      g=theGarden.gravity
      h=motherPlant.heightStem+motherPlant.heightLeafMax

      tempVar1= (v*math.cos(angle))/g
      tempVar2= (v*math.sin(angle))
```



```
      #tempVar3= math.sqrt((tempVar2*tempVar2)+(2*g*h))
      tempVar3= ((tempVar2*tempVar2)+(2*g*h))**0.5
      tempVar3= tempVar2+tempVar3
      theDistance=tempVar1*tempVar3
      theHypot=geometry_utils.distBetweenPoints(motherPlant.x, motherPlant.y, theSeed.x,
theSeed.y)
      theRise=theSeed.y-motherPlant.y
      theRun=theSeed.x-motherPlant.x
      theAngle=math.asin(theRise/theHypot)
      newX=math.cos(theAngle)*theDistance
      if theRun<0.0:
        newX=0.0-newX
      newY=math.sin(theAngle)*theDistance
      newX=newX+theSeed.x
      newY=newY+theSeed.y

    #print "********seed %s be being placed at %f, %f" % (theSeed.name, newX, newY)
    theSeed.x=newX
    theSeed.y=newY
    theSeed.countToGerm=self.delayInGermination
    theGarden.plantSeed(theSeed)
    motherPlant.seedList.remove(theSeed)

  def germinate(self, theGarden):
    if self.countToGerm<1:
      ###death due to failure to germinate
      tooBad=random.random()
      if theGarden.cycleNumber==0 and theGarden.ignoreGermDeathAtStart:
        tooBad=1.0
      if tooBad<self.fractionFailGerminate:
        self.causeOfDeath="failed to germinate(random death)"
        theGarden.kill(self)
      else:
        massForGrowth=self.massSeed*self.fractionSeedMassToPlant
        if massForGrowth
<=self.massSeedMax*self.fractMassSeedMaxToGerm*self.fractionSeedMassToPlant or massForGrowth
<=0.0:
          self.causeOfDeath="failed to germinate(immaturity)"
          theGarden.kill(self)
        else:
          self.isSeed=False
          self.age=1
          self.timeGermination=time.time()
          theNameList=self.name.split()
          if len(theNameList)<2:
            idNumb=str(random.random())
          else:
            idNumb=theNameList[1]
          self.name="Plant %s" % (idNumb)
          ###update the garden
          theGarden.numbSeeds= theGarden.numbSeeds-1
          theGarden.numbPlants= theGarden.numbPlants+1

          ###For germination only, mass to
          ###stem and leaf is a straight fraction.
          self.massStem= massForGrowth*self.fractionCarbonToStem
          self.massLeaf= massForGrowth-self.massStem

          ###convert massStem to radiusStem and heightStem
          self.calcRadiusStemFromMassStem()
          self.calcHeightStemFromRadiusStem(theGarden)

          ###what's the radius of the leaf and area?
          self.calcRadiusLeafFromMassLeaf()

          ###need a special routine where a single plant is checked against the garden
          ###to determine its shadedness.
```



```
          ###will be in world_basics

          ###rename the seed so you know it's a plant
          self.z=self.heightStem+self.heightLeafMax
          ###why am I doing this, again?
          if self.radiusLeaf>=self.radiusStem:
            self.r=self.radiusLeaf
          else:
            self.r=self.radiusStem
      else:
        self.countToGerm=self.countToGerm-1

    def calcNewMassFromLeaf(self, theGarden):
      #print "mass stem: %s" % (self.massStem)
      #print "mass leaf: %s" % (self.massLeaf)
      areaAvailable=self.areaPhotosynthesis-self.areaCovered
      #print "Ap: %s" % (areaAvailable)
      if self.areaPhotosynthesis>0.0:
        fractionAvailable=areaAvailable/self.areaPhotosynthesis
      else:
        fractionAvailable=0.0
      if fractionAvailable>self.fractionMinimumSurvival:
        #lightConversion=self.photoConstant*(self.massLeaf**self.photoExponent)#in units of
kgGrowth/area leaf for photosynthesis
        #newMass= (areaAvailable*lightConversion)
        var1=(self.massLeaf**self.photoExponent)#in units of kgGrowth/area leaf
        var1=areaAvailable*var1#in units of kgGrowth/area leaf

var2=(self.photoConstant*fractionAvailable)+(self.photoConstantShade*(1-fractionAvailable))
        #print "%s: %s" % (self, var2)
        newMass=var2*var1
        ###decide whether canopy transmission impacts conversion
        alterMassWitTransmission=0
        if theGarden.canopyTransmittanceImpactsConversion==1:
          alterMassWitTransmission=1
        elif theGarden.canopyTransmittanceImpactsConversion==0:
          alterMassWitTransmission=0
        else:
          if self.canopyTransmittanceImpactsConversion:
            alterMassWitTransmission=1
          else:
            alterMassWitTransmission=0
        if alterMassWitTransmission==1:
          newMass=newMass*(1-self.canopyTransmittance)
        #print "mass new: %s" % (newMass)
        return newMass
      else:
        return -1.0

    def calcMassStemFromMassNew(self):
      massNew=self.massFixed
      #GMs=self.speciesConstant1*math.pow(massNew, self.speciesExponent1)
      GMs=self.speciesConstant1*(massNew**self.speciesExponent1)
      massStem=self.massStem+GMs
      self.GMs=GMs
      self.massStem=massStem

    def calcMassLeafFromMassStem(self):
      massLeaf=self.massLeaf
      massNew=self.massFixed
      massStem=self.massStem
      massTotal=self.massTotal+massNew
      if (self.age>=self.startMakingSeedsAge) or self.isMature:
        #massLeaf=self.speciesConstant3*(math.pow(massStem, self.speciesExponent3))
        massLeaf=self.speciesConstant3*(massStem**self.speciesExponent3)
        #print "mature at: %i" % (self.age)
      else:
        #massLeaf=self.speciesConstant2*(math.pow(massStem, self.speciesExponent2))
```



```
      massLeaf=self.speciesConstant2*(massStem**self.speciesExponent2)
      #print "young"
    #if not massStem+massLeaf==massTotal:
      ##something went wonky with calcs.
      #print "Age: %i  a bit off: %f" % (self.age, massTotal-massStem-massLeaf)
      #massLeaf=massTotal-massStem
    self.GM1=massLeaf-self.massLeaf
    self.massLeaf=massLeaf


  def calcRadiusLeafFromMassLeaf(self):
    leafVolume=self.massLeaf/self.densityLeaf
    leafAreaDisc=leafVolume/self.heightLeafMax
    leafRadius= geometry_utils.radiusCircle(leafAreaDisc)
    if self.leafIsHemisphere:
      leafRadius=leafRadius*0.7071067812
    self.radiusLeaf=leafRadius
    self.areaPhotosynthesis =3.14*leafRadius*leafRadius
    if debug==1:print "          calcRadiusLeafFromMassLeaf: Radius of Leaf is %f" %
(self.radiusLeaf)

  def calcRadiusStemFromMassStem(self):
    Ms=self.massStem
    #Ds=self.speciesConstant20*math.pow(Ms,self.speciesExponent20)
    Ds=self.speciesConstant20*(Ms**self.speciesExponent20)
    Rs=Ds/2.0
    self.GRs=Rs-self.radiusStem
    self.radiusStem=Rs
    if debug==1:print "          calcRadiusStemFromMassStem: Radius of stem is %f" %
(self.radiusStem)
    #print "          calcRadiusStemFromMassStem: Radius of stem is %f" % (self.radiusStem)

  def calcHeightStemFromRadiusStem(self, theGarden):
    Ds=self.radiusStem*2
    #mature allocation method
    Hs=(self.speciesConstant8*math.log(Ds))+self.heightStemMax
    if Hs<self.heightStem:
      #young method
      #Hs=(self.speciesConstant7*math.pow(Ds, self.speciesExponent7))-self.speciesConstant6
      Hs=(self.speciesConstant7*(Ds**self.speciesExponent7))-self.speciesConstant6
    elif not self.isMature:
      self.isMature=True
      #print "mature at: %i" % (self.age)
    GHs=Hs-self.heightStem
    if (Hs<0.0 or Hs<self.heightStem):
      print "oops. stem is shrinking. that's not right. Die"
      self.GHs=GHs
      self.heightStem=Hs
      #something is wrong. There's either a negative height or it is shrinking
      self.causeOfDeath="impossible height calculation"
      theGarden.kill(self)
    else:
      self.GHs=GHs
      self.heightStem=Hs


  def makeSomeSeeds(self, maxSeedsPerPlant):
    #make a seed on yourself if you don't have the max number of seeds
    theNum=float(sum(self.massFixedRecord))
    theDenom=float(len(self.massFixedRecord))
    ###this addresses a rare bug where theDenom==0.0
    if theDenom<=0:
      avgMassFixed=0
    else:
      avgMassFixed=theNum/theDenom
    ###see code from 2008.10.05 for attempts at having seed mass change as plant size increases.
    maxKgSeeds=self.reproductionConstant*(self.massFixed**self.reproductionExponent)
    #maxKgSeeds=self.reproductionConstant*(self.massFixed**self.reproductionExponent)
```

```
    adjMaxKgSeeds=maxKgSeeds*self.fractionCarbonToSeeds
    #the adjMaxKgSeeds makes use of how selfish a plant is (fractionCarbonToSeeds) to either
    ###use all of the possible seed carbon for seeds, or some fraction there of.
    if self.massFixed<avgMassFixed:
      ###if the plant is suddenly stressed, adjust how it is making seeds
      theFractDifference=self.massFixed/avgMassFixed
      if theFractDifference<self.fractionSelfishness and self.fractionCarbonToSeeds<1.0:
        #adjMaxKgSeeds=maxKgSeeds*1.0
        adjMaxKgSeeds=float(maxKgSeeds)#just make sure it's a float
    numbSeedsOnPlant=len(self.seedList)
    makeThisManySeeds=int(adjMaxKgSeeds/self.massSeedMax)-numbSeedsOnPlant
    if makeThisManySeeds>maxSeedsPerPlant:
      makeThisManySeeds=maxSeedsPerPlant
    for i in range(makeThisManySeeds):
      self.makeSeed(self)
```

Vida/Vida_Data/vworldr.py


```
"""This file is part of Vida.
--------------------------
Copyright 2009, Sean T. Hammond

Vida is experimental in nature and is made available as a research courtesy "AS IS," but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

You should have received a copy of academic software agreement along with Vida. If not, see
<http://iorek.ice-nine.org/seant/Vida/license.txt>.
"""

import sys
import math
import random
import time
import copy
###append the path to basic data files
sys.path.append("Vida_Data")
import geometry_utils
import list_utils
import yaml
import progressBarClass


#Import Psyco if possible
try:
  import psyco
  psyco.log()
  psyco.full()
except ImportError:
  pass

debug1=0
debug2=0

def determineShade(theGarden):
  if theGarden.showProgressBar:
    print "***Generating lists of overlapping plants. This could take a while...***"
    theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
    i=0
  ###populate the overlap list
  theIndex=0
  for plantOne in theGarden.soil:
    if plantOne.isSeed==0 or (plantOne.isSeed and plantOne.minimumLightForGermination>0.0):
      plantOne.overlapList=[]
      plantOne.areaCovered=0.0
      plantOne.colourLeaf[2]= 1.0
      ###the following might be able to be used to speed things up
      #for overlappingItem in plantOne.overlapList:
      # if not overlappingItem in theGarden.soil or not overlappingItem.z>plantOne.z:
      #   ###make sure the overlapping item is still alive
      #   ###and that overlapping item is still taller
      #   plantOne.overlapList.remove(overlappingItem)
      ###need to insert something so tallest plant looks at all the other plants of exactly the
same size
      ###if they are the same size and overlapping, they need to share shading
      for plantTwo in range(theIndex):
        plantTwo=theGarden.soil[plantTwo]
        ###is plant two overlapping you?
        overlapStatus=geometry_utils.checkOverlap(plantOne.x, plantOne.y, plantOne.r,
plantTwo.x, plantTwo.y, plantTwo.r)
        if overlapStatus>0:
          if not plantTwo in plantOne.overlapList:
            if not plantTwo==plantOne:
              plantOne.overlapList.append(plantTwo)
      ###sort the overlap list by height of the plants. Ordered shortest to tallest
      plantOne.overlapList = list_utils.sort_by_attr(plantOne.overlapList, "heightStem")
```



```
      ###flip the list so it's ordered tallest to shortest
      plantOne.overlapList.reverse()
      theIndex=theIndex+1
      if theGarden.showProgressBar:
        i=i+1
        theProgressBar.update(i)

  ###take the overlap list and start dropping photons onto it.
  if theGarden.showProgressBar:
    print '***Determining shading. This could take a while...***"
    theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
    i=0
  for plant in theGarden.soil:
    if plant.isSeed==0 or (plant.isSeed and plant.minimumLightForGermination>0.0):
      fractionExposed=1.0
      if len(plant.subregion)>0:
        theRegion=plant.subregion[-1]
      else:
        theRegion=theGarden
      if len(plant.overlapList)>0:
        thePlantAreaTotal=geometry_utils.areaCircle(plant.r)
        if thePlantAreaTotal==0.0:
          print 'name:%s r:%f isSeed:%i massSeed:%f massTotal:%f'%(plant.name, plant.r,
plant.isSeed, plant.massSeed, plant.massTotal)
        if len(plant.overlapList)==1: ###if you are covered by just 1 other, do a direct calc
          overPlant =plant.overlapList[0]
          areaCovered=geometry_utils.areaOverlappingCircles(plant.x, plant.y, plant.r,
overPlant.x, overPlant.y, overPlant.r)
          areaCovered=areaCovered-(areaCovered*plantTwo.canopyTransmittance)
          thePlantAreaExposed=thePlantAreaTotal-areaCovered
          #plant.areaCovered=areaCovered
          fractionExposed= thePlantAreaExposed/thePlantAreaTotal
          #fractionExposed=fractionExposed*theGarden.lightIntensity #try and take into account
overall world light intensity
          fractionExposed=fractionExposed*theRegion.lightIntensity #try and take into account
overall world light intensity
          thePlantAreaExposed= thePlantAreaTotal*fractionExposed
          plant.areaCovered=thePlantAreaTotal-thePlantAreaExposed
        elif len(plant.overlapList)>1: ###if you are covered by 2, use monte-carlo
          numbPhotons=int(thePlantAreaTotal)
          if numbPhotons==0:
            numbPhotons=1
          numbPhotons= numbPhotons*100
          if numbPhotons>750: #we don't need monster numbers
            numbPhotons=750
          hitCount=0
          for photon in range(numbPhotons):
            ####consider moving this to geometry_utils
            ###pick uniformly distributed point in a circle
            randr=(random.random()*(plant.r-0))+0 #random between 0 and the radius
            twoPi=math.pi*2
            randAngle=random.random()*twoPi
            #randr =math.sqrt(randr) #if you don't use sqrt, you get clustering in the center
            randr =randr**0.5 #if you don't use sqrt, you get clustering in the center
            photonX = (randr*math.cos(randAngle))+plant.x
            photonY = (randr*math.sin(randAngle))+plant.y
            #####
            for overPlant in plant.overlapList:
              if not photonX=="gone":
                if geometry_utils.pointInsideCircle(overPlant.x, overPlant.y, overPlant.r,
photonX, photonY):
                  randomValue=random.random()
                  if randomValue > overPlant.canopyTransmittance:
                    ###these points are where the overlap is
                    photonX="gone"
                  break
            if not photonX=="gone":
              hitCount=hitCount+1
```



```
          if numbPhotons ==0:
            fractionExposed=0.0
          else:
            fractionExposed=float(hitCount)/float(numbPhotons)
          #fractionExposed=fractionExposed*theGarden.lightIntensity #try and take into account
overall world light intensity
          fractionExposed=fractionExposed*theRegion.lightIntensity #try and take into account
overall world light intensity
          thePlantAreaExposed= thePlantAreaTotal*fractionExposed
          plant.areaCovered=thePlantAreaTotal-thePlantAreaExposed
        else: #if you are not covered at all
          thePlantAreaTotal=geometry_utils.areaCircle(plant.r)
          fractionExposed=1.0
          #fractionExposed=fractionExposed*theGarden.lightIntensity #try and take into account
overall world light intensity
          fractionExposed=fractionExposed*theRegion.lightIntensity #try and take into account
overall world light intensity
          thePlantAreaExposed= thePlantAreaTotal*fractionExposed
          plant.areaCovered=thePlantAreaTotal-thePlantAreaExposed
        ###now change the colour accordingly
        plant.colourLeaf[2]= fractionExposed
      if theGarden.showProgressBar:
        i=i+1
        theProgressBar.update(i)



class garden(object):
  def __init__(self):
    super(garden, self).__init__()
    self.name=''
    self.theWorldSize=0
    self.soil=[]
    self.numbSeeds=0
    self.numbPlants=0
    self.deathNote=[]
    self.cycleNumber=0
    self.lightIntensity=1.0
    fileLoc="Vida World Preferences.yml"
    self.importPrefs(fileLoc)
    if self.lightIntensity>1.0: self.lightIntensity=1.0

  def importPrefs(self, fileLoc):
    theFile=open(fileLoc)
    theData=yaml.load(theFile)
    theFile.close
    for key in theData:
        setattr(self, key, theData[key])

  def makePlatonicSeedDict(self, ymlList, Species1):
    theGarden=self
    i=0
    for s in ymlList:
      theSeed=Species1()
      fileLoc= "Species/"+ymlList[i]
      theSeed.importPrefs(fileLoc)
      theSeed.name="Platonic %s" % (ymlList[i])
      theGarden.platonicSeeds[ymlList[i]]=theSeed
      i=i+1


  def plantSeed(self, theSeed):
    #self=garden, obj=seed
    theGarden=self
    theNameList= theSeed.name.split()
    if len(theNameList)<2:
      idNumb=str(random.random())
    else:
```



```
      idNumb=theNameList[1]
    theSeed.timePlanted=time.time()
    theSeed.name="plantedSeed %s" % (idNumb)
    if theSeed.motherPlant==0:
      theSeed.motherPlant=theSeed
    theGarden.soil.append(theSeed)
    theGarden.numbSeeds=self.numbSeeds+1
    ####If there are subregions defined, see what subregion does it belong in this seed belongs
    #print "the Garden: %s" % (theGarden)
    #print "the region: %s" % (theGarden.theRegions)
    if len(theGarden.theRegions)>0:
      for aRegion in theGarden.theRegions:
        #print aRegion.name
        #print theSeed.name
        if aRegion.shape=='square':
          inSubregion=geometry_utils.pointInsideSquare(aRegion.x, aRegion.y, aRegion.size,
theSeed.x, theSeed.y)
        elif aRegion.shape=='circle':
          #size needs to be radius but region defines diameter
          inSubregion=geometry_utils.pointInsideCircle(aRegion.x, aRegion.y, aRegion.size/2.0,
theSeed.x, theSeed.y)
        if inSubregion:
          if not aRegion in theSeed.subregion:
            theSeed.subregion.append(aRegion)
    return theSeed

  def kill(self, theObject):
    theGarden=self
    if theObject in self.soil:
      #die!
      if len(theObject.seedList)>0:
        for theSeed in theObject.seedList:
          if theSeed.massSeed>0.0:
            ###the plant has seeds. Drop them prior to killing plant###
            ###just drop the seed straight down###
            theSeed.countToGerm=theObject.delayInGermination
            theGarden.plantSeed(theSeed)
          theObject.seedList.remove(theSeed)
      theObject.seedList=[]
      if theObject.isSeed==1:
        self.numbSeeds=self.numbSeeds-1
      else:
        self.numbPlants=self.numbPlants-1
      self.deathNote.append(theObject)
      self.soil.remove(theObject)

  def calcEulerGreenhill(self, plant):
    theGarden=self
    #calculate the critial force at which a upright cylinder will deflect
    E=plant.youngsModulusStem*10000000000
    Ds=plant.radiusStem*2.0
    ps=plant.densityStem
    ###let the subregion the plan is in override the gravity###
    ##use only the final one if there are multiple###
    if len(plant.subregion)>0:
      g=plant.subregion[-1].gravity
    else:
      g=theGarden.gravity
    #r=plant.radiusStem
    #z=plant.z
    #Pacr=(math.pow(math.pi, 3)/16.0)*(math.pow(r, 4)/math.pow(z, 2))*E
    heightCritical= 0.79*((E/(g*ps))**0.3333)*(Ds**0.6667)
    #massCritical=0.785*ps*heightCritical*Ds*Ds
    #forceCritical=0.483736625*E*(Ds*Ds*Ds*Ds)
    #denom=(2.0*heightCritical)*(2.0*heightCritical)
    #forceCritical=forceCritical/denom
    return heightCritical
```

```
  def checkEulerGreenhillViolation(self, plant):
    theGarden=self
    if plant.isSeed==1: return 0 #don't check it if it's a seed.
    #theGravity=theGarden.gravity
    #if len(plant.subregion)>0:
    # theGravity=plant.subregion[-1].gravity
    #determine if a plant violates the Euler-Greenhill rule
    #ps=plant.densityStem
    #Ds=plant.radiusStem*2.0
    heightCritical=self.calcEulerGreenhill(plant)
    ###
    #massCritical=0.785*ps*heightCritical*Ds*Ds
    #forceCritical=massCritical*theGravity
    #massTotal=plant.massTotal+plant.massSeedsTotal
    #theForce=massTotal*theGravity
    #if plant.heightStem>=heightCritical or massTotal>=massCritical or theForce>= forceCritical:
    if plant.heightStem>=heightCritical:
      #omg! violates euler-greenhill!
      return 1
    else:
      return 0

  def outOfBounds(self, theObject):
    ##returns 1 if out of bounds
    ##returns 0 if in bounds
    theGarden=self
    #if the object's radius goes off world, die
    if theObject.isSeed:
      r=theObject.radiusSeed
    else:
      r=theObject.radiusStem
    if theObject.x+r>theGarden.theWorldSize/2.0:
      return 1
    elif theObject.x-r<0.0-theGarden.theWorldSize/2.0:
      return 1
    elif theObject.y+r>theGarden.theWorldSize/2.0:
      return 1
    elif theObject.y-r<0.0-theGarden.theWorldSize/2.0:
      return 1
    else:
      return 0


  def checkForOverlap(self, theObject):
    #specialized routine for detecting overlaps
    #accepts an object (plant or seed)
    #looks at all objects to see if seeds or stems touch
    #returns a list of objects the query object is touching
    theGarden=self
    if theObject.isSeed:
      r=theObject.radiusSeed
    else:
      r=theObject.radiusStem
    theIndex=0
    theOverlapList=[]
    if len(theObject.overlapList)==0:
      listToCheck=theGarden.soil
    else:
      listToCheck=theObject.overlapList
    for anObject in listToCheck:
      if not anObject==theObject: #don't look at yourself thx
        if anObject.isSeed:
          rr=anObject.radiusSeed
        else:
          rr=anObject.radiusStem
        theOverlap=geometry_utils.checkOverlap(theObject.x, theObject.y, r, anObject.x,
anObject.y, rr)
        if theOverlap==1 and anObject not in theOverlapList:
```



```
          ##occupy same space or completely covered
          theOverlapList.append(anObject)
        elif theOverlap==2 and anObject not in theOverlapList:
          ###there is partial overlap
          theOverlapList.append(anObject)
    return theOverlapList

  def removeOverlaps(self):
    if not self.allowOverlaps:
      theGarden=self
      if theGarden.showProgressBar:
        print "***Removing overlapping objects***"
        theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
        i=0
      for obj in theGarden.soil[:]:
        ###seeds and or stems that overlap are violating physics.
        ##Rules about overlapping objects:
        #####1.) If 2 objects overlap, the more massive object remains.
        #####2.) If 2 seeds overlap and they have the same mass, the oldest seed remains.
        objectList=theGarden.checkForOverlap(obj)
        for overlappingObject in objectList:
          if obj.massTotal>overlappingObject.massTotal:
            overlappingObject.causeOfDeath="crushed"
            theGarden.kill(overlappingObject)
            break
          elif obj.massTotal<overlappingObject.massTotal:
            obj.causeOfDeath="crushed"
            theGarden.kill(obj)
            break
          elif obj.massTotal==overlappingObject.massTotal:
            if obj.timePlanted>overlappingObject.timePlanted:
              obj.causeOfDeath="overlap violation"
              theGarden.kill(obj)
              break
            else:
              obj.causeOfDeath="overlap violation"
              theGarden.kill(overlappingObject)
              break
        if theGarden.showProgressBar:
          i=i+1
          theProgressBar.update(i)

  def removeOffWorldViolaters(self):
    if not self.allowOffWorld:
      theGarden=self
      if theGarden.showProgressBar:
        print "***Removing plants growing off world...***"
        theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
        i=0
      ###see if plants or seeds extend over the edge of the world
      ### if so, kill them off
      for obj in theGarden.soil[:]:
        if theGarden.outOfBounds(obj)==1:
          obj.causeOfDeath="stem off world"
          theGarden.kill(obj)
        if theGarden.showProgressBar:
          i=i+1
          theProgressBar.update(i)

  def removeEulerGreenhillViolaters(self):
    if not self.allowEulerGreenhillViolations:
      theGarden=self
      if theGarden.showProgressBar:
        print "***Checking for Greenhill-Euler violations...***"
        theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
        i=0
      for obj in theGarden.soil[:]:
        if theGarden.checkEulerGreenhillViolation(obj)==1:
```



```
          obj.causeOfDeath="violated Euler-Greenhill"
          theGarden.kill(obj)
        if theGarden.showProgressBar:
          i=i+1
          theProgressBar.update(i)

  def causeRandomDeath(self):
    if self.allowRandomDeath:
      theGarden=self
      if theGarden.showProgressBar:
        print "***A time to die...***"
        theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
        i=0
      for obj in theGarden.soil[:]:
        tooBad=random.random()
        theRegion=theGarden
        if len(obj.subregion)>0:
          #if there are multiple regions, only the most recently made one is used.
          theRegion=obj.subregion[-1]
        if tooBad<theRegion.randomDeath:
          obj.causeOfDeath="random death"
          theGarden.kill(obj)
        if theGarden.showProgressBar:
          i=i+1
          theProgressBar.update(i)

  def checkSenescence(self):
    if self.allowSlowGrowthDeath:
      theGarden=self
      if theGarden.showProgressBar:
        print "***Checking for too slow growth...***"
        theProgressBar= progressBarClass.progressbarClass(len(theGarden.soil),"*")
        i=0
      for obj in theGarden.soil[:]:
        if not obj.isSeed and obj.maxAvgHeightGrowthRate>0.0: #only look at growth if the
object is a plant, not a seed.
          fractOfMaxHeightGrowth=obj.avgHeightGrowthRate/obj.maxAvgHeightGrowthRate
          if obj.randomSlowGrowth>fractOfMaxHeightGrowth or
theGarden.randomSlowGrowth>fractOfMaxHeightGrowth:
            #the garden defines a value at which plants will start to stochastically die
            #each species also defines a value at which that species will start to
stochastically die
            #this way the garden can have a 'bad' environment that triggers problems earlier
than a species might define
            #to simulate a virus or something.
            tooBad=random.random()
            theRegion=theGarden
            if len(obj.subregion)>0:
              #if there are multiple regions, only the most recently made one is used.
              theRegion=obj.subregion[-1]
            chanceOfDeath=theRegion.randomDeath
            if tooBad<=chanceOfDeath:
              obj.causeOfDeath="growth too slow"
              theGarden.kill(obj)
        if theGarden.showProgressBar:
          i=i+1
          theProgressBar.update(i)

  def placeSeed(self, seedPlacement, sList, startPopulationSize, useDefaultYml, ymlList):
    theGarden=self
    #print sList
    #This block of code came from the main Vida.py. Moved and working on 2008.11.06 to allow
for calling during simulation runs at
    #not just at the start.
    if theGarden.cycleNumber<1: print "***Generating and placing seeds.***"
    ###initialize progress bar###
    if theGarden.showProgressBar or theGarden.cycleNumber<1:
      theProgressBar= progressBarClass.progressbarClass(startPopulationSize-1,"*") #why -1?
```



```
because index 0. So if total=100, 0-99.

    if seedPlacement=="square" or seedPlacement=="hex":
      seedDistance=geometry_utils.placePointsInGrid(startPopulationSize, theGarden.theWorldSize)
      prevX=-theGarden.theWorldSize/2.0
      prevY= theGarden.theWorldSize/2.0
      if seedPlacement=="hex":
        currentRow=0
        prevX= prevX-(seedDistance/2.0)
        prevY= prevY+(seedDistance/2.0)

    if seedPlacement=="defined" or len(sList)>0:
      theIndex=0
      #print sList
    for i in range(startPopulationSize):
      countToGerm=0
      theSpeciesFile=""
      if debug1 and startPopulationSize==1:
        x=0
        y=0
      elif len(sList)>0:
        if seedPlacement=='fromFile':
          x=sList[i][1]
          y=sList[i][2]
          countToGerm=sList[i][3]
          #radius of canopy would be sList[i][4]. Unused, but in place to be built on
          if sList[i][0] in ymlList:
            theSpeciesFile=sList[i][0]
          else:
            print "\nWARNING: The desired species %s was not found. \nA random species will be
used.\n" % (sList[i][0])
        else:
          if sList[i] in ymlList:
            theSpeciesFile=sList[i]
      if seedPlacement=="fromFile":
        x=sList[i][1]
        y=sList[i][2]
        countToGerm=sList[i][3]
        #radius of canopy would be sList[i][4]. Unused, but in place to be built on
      elif seedPlacement=='random':

x=random.randrange(-(theGarden.theWorldSize/2),(theGarden.theWorldSize/2))+random.random()

y=random.randrange(-(theGarden.theWorldSize/2),(theGarden.theWorldSize/2))+random.random()
      elif seedPlacement=="square" or seedPlacement=="hex":
        x=prevX+seedDistance
        y=prevY-seedDistance
        if x+seedDistance>=(theGarden.theWorldSize/2.0):
          prevX=-theGarden.theWorldSize/2.0
          if seedPlacement=="hex":
            if currentRow==0:
              currentRow=1
            else:
              currentRow=0
              prevX= prevX-(seedDistance/2.0)
          prevY= prevY-seedDistance
        else:
          prevX=x
      #print theSpeciesFile
      if useDefaultYml==False or theSpeciesFile=="":
        #print 'not default species'
        if theSpeciesFile=="" or theSpeciesFile=="_random_":
          whichSpecies=random.randint(0,len(ymlList)-1)##pick from the species randomly
          fileLoc= 'Species/'+ymlList[whichSpecies]
          theSpeciesFile=ymlList[whichSpecies]
        #print theSpeciesFile
        if theGarden.platonicSeeds.has_key(theSpeciesFile):
          platonicSeed=theGarden.platonicSeeds[theSpeciesFile]
```

```
        theSeed=copy.deepcopy(platonicSeed)
      #else:
      # print "something is very wrong."
    else:
      #if there are no species listed, use the default species.
      theSeed=Species1()
    theSeed=theGarden.plantSeed(theSeed)
    #print theSeed.subregion
    #now set some attributes
    theSeed.timeCreation=time.time()
    theSeed.countToGerm=countToGerm
    theSeed.x=x
    theSeed.y=y
    #this is just to get the properties calculated.Give the seed the max seed mass
    #bootstrap the seed
    theSeed.massSeed=theSeed.massSeedMax
    #convert mass to volume and the get the radius
    j= theSeed.massSeed/theSeed.densitySeed #this is volume in m^3
    j=j/(4.0/3.0)
    j=j/(math.pi)
    j=math.pow(j, 1.0/3.0)
    theSeed.radiusSeed=j
    theSeed.z= theSeed.radiusSeed
    theSeed.r= theSeed.radiusSeed
    #theSeed.growSeedOnPlant(theSeed.massSeedMax)
    #print "######"
    #print theSeed.__class__().name
    #print "######"
    #update the progress meter
    if theGarden.showProgressBar or theGarden.cycleNumber<1:
      theProgressBar.update(i)
```