# 1 Introduction

In the botanical world, it often happens that a part of a plant looks like the whole. For example, branches of a tree look like scaled down versions of the whole tree, because they further separate into smaller branches. Similarly, the leaves of some ferns are themselves composed of smaller blades and look like small ferns. This observation that plants are often *self-similar* suggests that there may be an algorithmic way to general images of plants. In this work we will explore two different methods of generating computer graphics that can be applied to modelling of plants: iterated function systems and L-systems. We will briefly describle the first, show an example of plant modelled using this technique, and we will spend more time on the second.

# 2 Iterated Function Systems

## 2.1 Description of the method

The main idea of iterated function systems, abbreviated IFS, is to successively apply transformations to a starting set to obtain an image. Only affine linear transformations are allowed; these contain rotations, translations, reflection and scaling. Affine linear transformation can be defined as the composition of linear transformations with translations. In $\mathbb{R}^2$, a linear transformation $f$ can be represented as a matrix $A$, which acts on a vector $x$ by matrix multiplication, while a translation can be represented by vector addition. Thus, an affine linear transformation $g$ can be represented as

$$g(x) = Ax + b$$

where both $x$ and $b$ are vectors in $\mathbb{R}^2$.

## 2.2 Barnsley fern

IFS were suitable to generate images of fractals, but it became clear that they could also be used to model nature. The Barnsley fern, who bears the name of his discover Michael Barnsley, is a famous example of the use of IFS to model nature [1]. Many other natural images can be formed with IFS (see [6]).

To construct this fractal shape, we use the following 4 transformations.

$$\omega_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0.16 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\omega_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$$

$$\omega_3 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$$

$$\omega_4 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -0.15 & -0.28 \\ 0.26 & 0.24 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0.44 \end{pmatrix}$$

Moreover, there is a degree of randomness: each transformation has a certain probability of being used at each iteration: $\omega_1$ has probability 0.01, $\omega_2$ has probability 0.85 and both $\omega_3$ and $\omega_4$ have probability 0.07.

We have coded our own implementation of this process in Matlab. The code is provided in a separate file. With $10^8$ iterations, we obtain Fig. 1, which resembles a species of fern called the black spleenwort. As the figure demonstrates, IFS are successful are generating realistic images of nature.

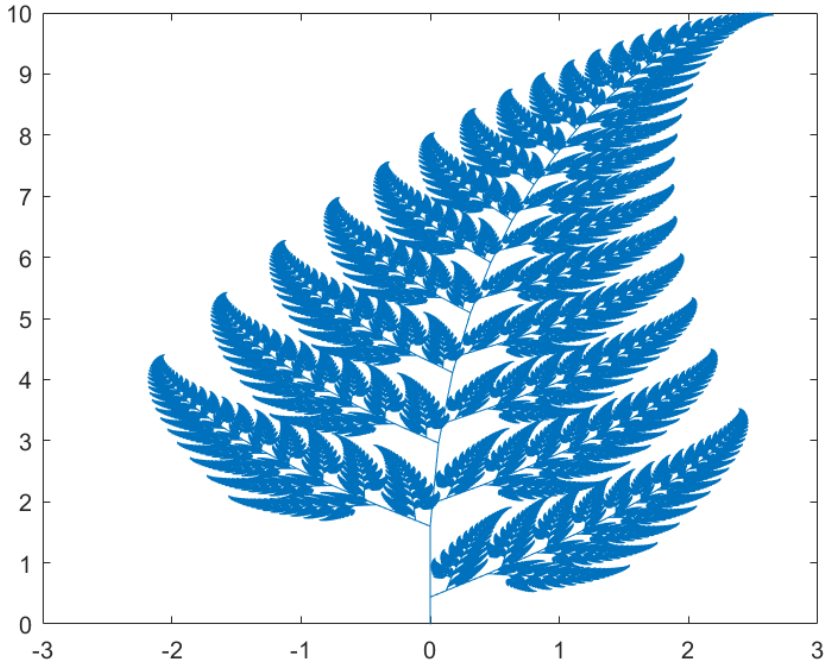



Figure 1: Barnsley fern implemented using IFS.

# 3 L-Systems

Our main reference for this section is [7], an in-depth and beautiful book which we highly recommend as further reading. For a shorter read, consider [8]. For the original paper putting forth L-systems in 1968, see [5].

## 3.1 Background

L-systems owe their name to their inventor, biologist Lindenmayer. This method was developped with the purpose of modelling biological processes at the microscopic level, such as cell growth. Contrary to the iterated function systems, L-systems are well-suited to model the the growth of plants over time. In particular, each iteration corresponds to a stage of growth of a plant, and increasing the number of iterations is akin to observing the plant changing for a longer amount of time. This was not the case for the iterated function sys-

tems. This also implies that this method is suitable for creating moving graphics depicting interactive growth of a plant, for example. L-systems are inscribed in the larger trend of 'rewriting'. Characterized by the works of Thue, who first defined the concept, of Chomsky, who was interested in formal grammars, and Backus and Naur, who focused on the programming language ALGOL-60, rewriting systems have applications in many other fields, such as linguistics and computer science [7].

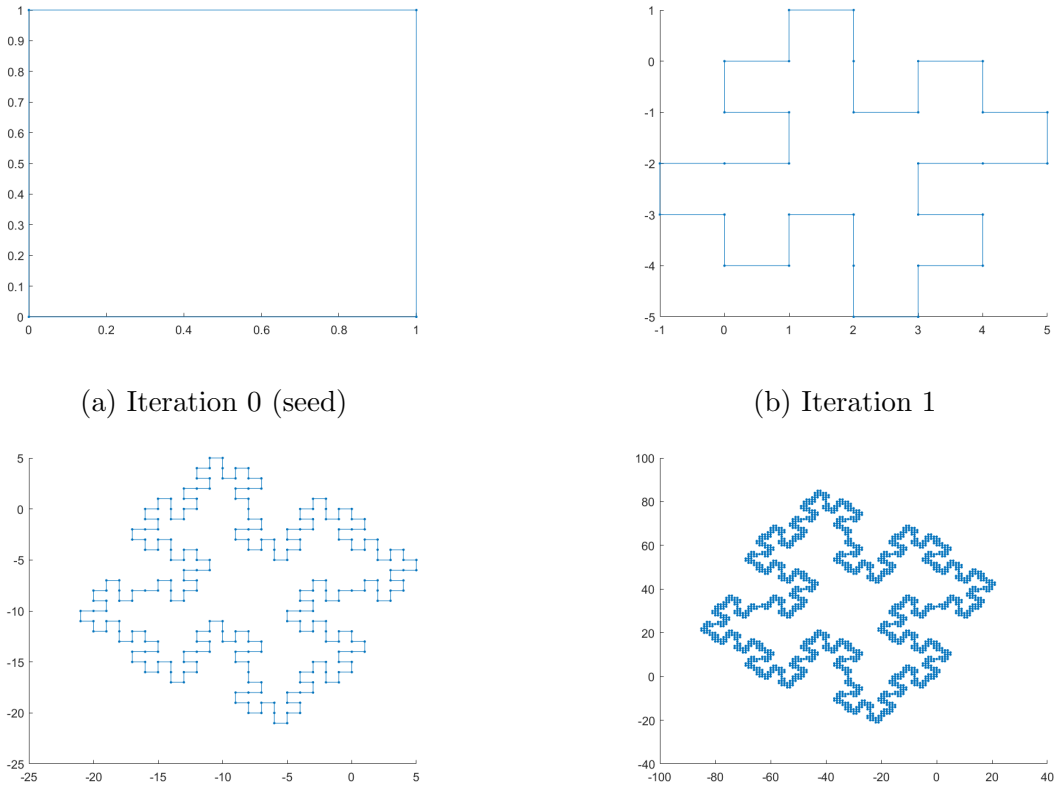
## 3.2 Formal definition

We first present the basic idea. We start with an input string, called an *axiom* or *seed*. We now apply some *production rules* which replaces a symbol by another string of symbols. We repeat this process any number of times, developing the string with each iteration.

Formally, an L-system consists of three components: an *alphabet* $V$, an *axiom* $\omega$, and a set $P$ of *productions*. We call $a \in V$ a *letter*. A finite sequence $a_1, a_2, ..., a_n$ with $a_i \in V \ \forall i$ is called a *word*. We denote by $V^*$ the set of words and by $V^+$ the set of non-empty words on $V$. The set of productions is composed of finitely many pairs $p_1, ..., p_m$ where $p_i = (a_i, w_i) \in V \times V^+$. Each such pair $p_i$ is called a *production* and is sometimes denoted $a_i \to w_i$. $a_i$ is called the predecessor of the production $p_i$, while $w_i$ is called the successor.

We distinguish two types of L-systems: *context-free* L-systems, denoted 0L-systems, and *context-sensitive* L-systems. In the latter, the production rules of a given word may depend on the surrounding string. That is never the case in the former. We focus here on context-free L-systems. An L-system is called *deterministic* just in case every letter $a$ is the predecessor of exactly one production, i.e. no two production map the same letter to different words. Note that if there is a letter $a$ such that no production $p$ has $a$ as its predecessor, then we assume that the identity production $(a, a)$ is in $P$. We abbreviate deterministic L-system by D0L-system. We will focus on this type of L-system.

We present a simple example. Let our alphabet consist of the letters $a$ and $b$. Let the axiom be $a$. Our only production rule is $a \to aba$. After the first iteration, our string becomes



(a) Iteration 0 (seed)

(b) Iteration 1

(c) Iteration 2

(d) Iteration 3

Figure 2: Plots of the Koch snowflake for different iterations.

"*aba*". After the second iteration, our string becomes "*abababa*". We can interpret these strings geometrically. A graphical example is presented in Fig. 1. This figure shows that L-systems allows us to draw fractals shapes, such as the Koch snowflake. The next section will show how these strings can be interpreted geometrically.

## 3.3 Turtle interpretation

In order to implement the method of L-systems, we use what is known as "Turtle graphics". The idea is to imagine a turtle walking across the computer screen holding a pen. We can command the turtle to move to a certain point, which, after enough operations, can create interesting graphics. The turtle follows instructions such as drawing a line segment forward, heading to another direction, or going back to a previously saved position.

For this project, we implemented our own version of Turtle graphics in Matlab from scratch. The code is provided as a separate file. The turtle is given as input a number of iterations $n$, an input angle $\theta$, a seed, and a set of production rules. For each iteration, starting from the seed, the string is rewritten using the production rules. Each symbol in the string corresponds to an action for the turtle. After $n$ iterations, our turtle performs the sequence of actions which draws the resulting picture. We trained our turtle to intepret the following symbols as follows:

F draw a line segment forward

\+ rotate by angle $\theta$ counterclockwise

\- rotate by angle $\theta$ clockwise

[ push the current state onto a stack– in other words, keep in mind the current position and direction of the turtle

] pop the current state off the top of the stack – go back to the last position stored in memory and head in the proper direction

For example, the production rule $F \to F[+F] - F$ means, in turtle language, that every time the turtle is about to encounter a straight segment, instead of drawing the line, the turtle will:

1. draw one straight segment
2. save its current position and direction
3. rotate counterclockwise 25 °
4. draw a line in that new direction
5. go back to the last saved position and direction (without drawing)
6. rotate clockwise 25 °

7. draw a line in that new direction.

This simple algorithm is capable of creating interesting and realistic shapes, as we will see in the next section.

If is also possible to have a symbol for nodes; here we use $X$. This symbol is not attached to a command. This allows us to introduce a slight variation of our technique. Now, instead of replacing edges by some sequence of action, we replace nodes. We call this node-rewriting, as opposed to edge-rewriting. As we will see, this can create different shapes.

## 3.4 Example of plant-like structures from L-systems

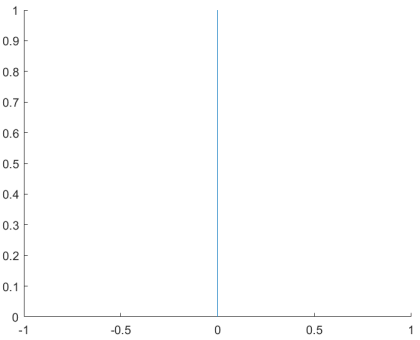
Here are some examples of plant-like structures that can be obtained from deterministic L-systems. Our starting point for these models was the values found in REF We begin with a panel depicting the different steps or construction of these images (Fig. 3). The system we used here is:

$$n = 5$$
$$\theta = 25$$
$$F \to F[+F]F[-F]F$$

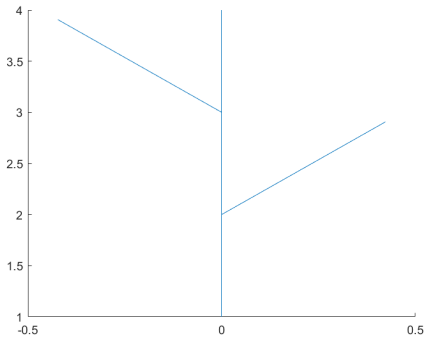
At each step, the production rule is applied which complexifies the image. The resulting structure resembles an algae or a wild weed. We can generate an image that looks like a bush (Fig. 4) with the system

$$n = 5$$
$$\theta = 30$$
$$F \to F[+F][-F[-F]F]F[+F][-F]$$

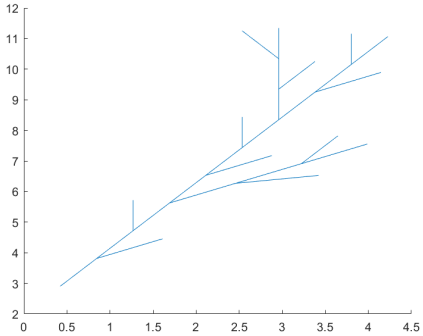
We can also obtain a plant that looks like a Christmas tree (Fig.5) with the following system:



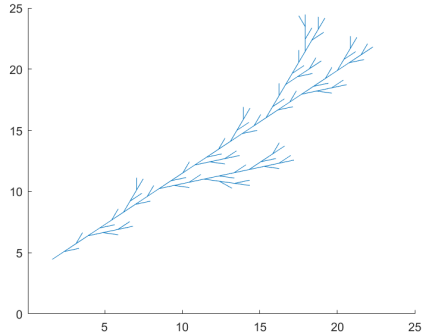
(a) Iteration 0 (seed)



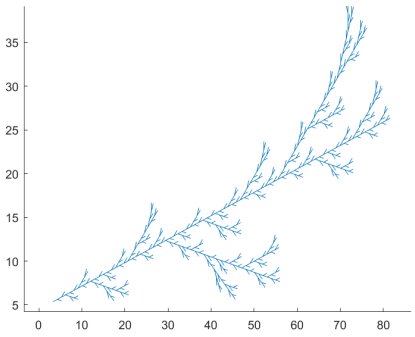
(b) Iteration 1



(c) Iteration 2



(d) Iteration 3



(e) Iteration 4

Figure 3: Plots of the L-system for different iterations. The resulting structure looks like a wild weed or perhaps an algae.

$$n = 4$$

$$\theta = 40$$

$$F \to F[+FF][-FF]F[+FF][-FF]FF$$

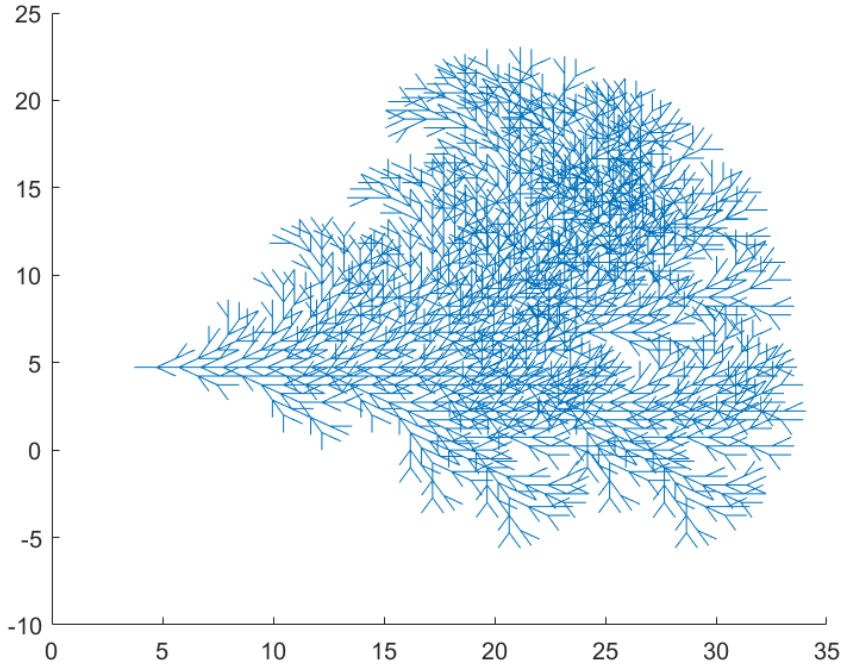



Figure 4: A plant model that looks like a bush.

Using node-rewriting instead of edge-rewriting, we obtain a structure that resembles a flowerhead (Fig. 6). For Fig. 6, we used the system

$$n = 5$$
$$\theta = 30$$
$$X \to F[+X][-X]FX$$
$$F \to FF$$

## 3.5 Adding randomness

Indeed, deterministic L-systems generate the same image over and over. If the goal is to generate large numbers of such plants, for example a field, then the above process may look too regular and unrealistic. A solution to this problem is to add randomness to L-systems.

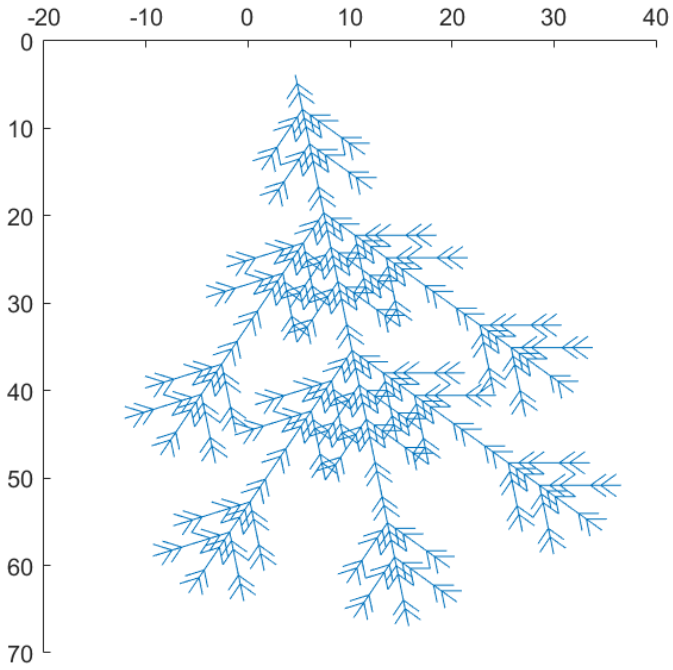



Figure 5: A plant model that looks like a Christmas tree.

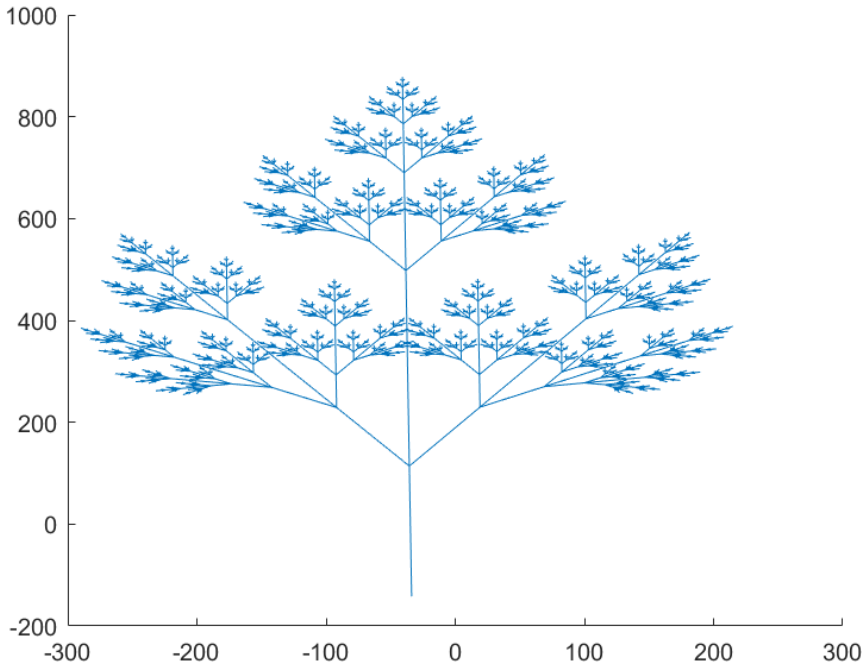



Figure 6: A plant model that looks like a flowerhead.

It is actually natural to include some context-dependence when modelling plant growth: the environmental conditions like the quantity of sunlight, the quality of the soil or the amount of water available will likely influence the growth of the plant and promote, for example, more intense leafy growth while putting a halt to flowering. We can think of the environment as varying randomly.

To implement this randomness, we can assign probabilities to different production rules for the same word. This could represent, for example, how a stem *may* grow many new leaves, or none, or only few, or instead grow flowers, all depending on environmental conditions. Hence, we cannot determine with certainty what the result will be like, and each model is likely to be different from the last. Such L-systems are called *stochastic*.

# 4 Conclusion

In this work we have discussed two methods for generating images of flora: iterated function systems and L-systems. There is a connection between the two methods. Chapter 8 of [7] describes the steps involved in transforming an L-system modelling the growth of a plant into an IFS which can generate an approximation of the fractal representing the plant after growth. The field of computer graphics has certainly evolved since the time when L-systems and IFS were first developped. Nontheless, these techniques are still used to this day for more advanced tree graphics [9], cloud modelling [4], automatic world building [3] or even for generating music for computer games [2]. This shows that when different fields like mathematics and biology cross and combine with new technologies like higher computing power, progress can be sparked.

# References

[1] Michael F. Barnsley. Transformations on Metric Spaces; Contraction Mappings; and the Construction of Fractals. In Fractals Everywhere, pages 42–CP8. Elsevier, jan 1993.

[2] Mikael Fridenfalk. Algorithmic music composition for computer games based on L-system. In 2015 IEEE Games Entertainment Media Conference, GEM 2015. Institute of Electrical and Electronics Engineers Inc., jan 2015.

[3] Mikael Fridenfalk. Application for real-time generation of mathematically defined 3D worlds. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 9590, pages 45–68. Springer Verlag, 2016.

[4] Seokyoon Kang and Ki Il Kim. Three dimensional cloud modeling approach based on L-system. In Proceedings - 2015 3rd International Conference on Computer, Information and Application, CIA 2015, pages 7–9. Institute of Electrical and Electronics Engineers Inc., feb 2016.

[5] Aristid Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. Journal of Theoretical Biology, 18(3):280–299, 1968.

[6] Heinz-Otto Peitgen, Hartmut Jürgens, and Sietmar Saupe. Chaos and Fractals. Second edition, 1987.

[7] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The Algorithmic Beauty of Plants. Springer-Verlag, New York, electronic edition, 2004.

[8] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental Models of Herbaceous Plants for Computer Imagery Purposes. Technical Report 4, 1988.

[9] Ruoxi Sun, Jinyuan Jia, and Marc Jaeger. Intelligent tree modeling based on L-system. In Proceeding 2009 IEEE 10th International Conference on Computer-Aided Industrial Design and Conceptual Design, pages 1096–1100, 2009.