information

Article

Toward a Simulation Model Complexity Measure

J. Scott Thompson [1,*], Douglas D. Hodson [2], Michael R. Grimaila [3], Nicholas Hanlon [1] and Richard Dill [2]

[1] Air Force Research Laboratory, Aerospace Systems Directorate, Dayton, OH 45433, USA
[2] Department of Electrical & Computer Engineering, Air Force Institute of Technology, Dayton, OH 45433, USA
[3] Department of Systems Engineering & Management, Air Force Institute of Technology, Dayton, OH 45433, USA
* Correspondence: j.scott.thompson.12@gmail.com

Abstract: Is it possible to develop a meaningful measure for the complexity of a simulation model? Algorithmic information theory provides concepts that have been applied in other areas of research for the practical measurement of object complexity. This article offers an overview of the complexity from a variety of perspectives and provides a body of knowledge with respect to the complexity of simulation models. The key terms model detail, resolution, and scope are defined. An important concept from algorithmic information theory, Kolmogorov complexity, and an application of this concept, normalized compression distance, are used to indicate the possibility of measuring changes in model detail. Additional research in this area can advance the modeling and simulation body of knowledge toward the practical application of measuring simulation model complexity. Examples show that KC and NCD measurements of simulation models can detect changes in scope and detail.

Keywords: simulation model; complexity; Kolmogorov complexity; normalized compression distance; resolution; scope; model families

check for updates

Citation: Thompson, J.S.; Hodson, D.D.; Grimaila, M.R.; Hanlon, N.; Dill, R. Toward a Simulation Model Complexity Measure. *Information* **2023**, *14*, 202. https://doi.org/10.3390/info14040202

Academic Editors: Bernard P. Zeigler, Tuncer Ören and Andreas Tolk

Received: 1 February 2023
Revised: 17 March 2023
Accepted: 22 March 2023
Published: 24 March 2023

1. Introduction

Modeling and simulation (M&S) has contributed greatly to the development of highly capable and reliable systems; it is a powerful tool for developing, understanding, and operating systems of both great simplicity and increasing complexity. As techniques progress and as system designers strive for more capability, M&S practitioners and stakeholders struggle to comprehend how systems operate and, often more importantly, how to protect against catastrophe when systems enter regimes of failure. Some stakeholders assume that M&S necessarily should be implemented with "high fidelity", expecting that systems be represented in exquisite detail. Practitioners know that M&S is largely a software-intensive activity, and requires considerable resources devoted to design, development, verification, and validation before M&S tools can be applied for experimentation, analysis, or training. Yet, limited resources require economical considerations to drive when and how much M&S to apply; stakeholders may be impatient to realize their expected return on investing in M&S. Through hard-won experience, practitioners build an intuitive sense of how much detail is required to address a given simulation problem. Is there a faster way to arrive at the insight provided by this intuition? The amount of complexity in a simulation model is driven by the amount of model detail, which is a function of the model's resolution, scope, size, and interactions [1]. Measuring simulation model complexity would be useful for a number of purposes, driven by several foundational principles that M&S practitioners must consider.

The parsimony principle, often referred to as Occam's razor after fourteenth century Franciscan philosopher, William of Occam, suggests an inherent bias toward more simple explanations of phenomena [2]. As George Box described of statistical models, "Since all models are wrong the scientist must be alert to what is importantly wrong. It is inappropriate to be concerned about mice when there are tigers abroad" [3]. The principle of

representational parallelism (not computational parallelism) asserts that key components of systems should be decomposed to roughly equivalent levels of detail in order to make valid comparisons and inferences. Further, the M&S practitioner not only has to contend with referential complexity (what in the source system to represent) but also methodological complexity (how that representation is implemented in software) [4]. Measuring complexity in some relative or absolute sense would assist with the simulation model selection problem—that is, which representation of a source system is the right one for answering the question at hand? Such a measure would help to locate a given representation within a multi-level approximation family or to compare across representations that purport to have similar referential complexity but different methodological complexity.

A number of complexity measures have been defined in other areas of research. The following section reviews key perspectives and terms and introduces the concept of Kolmogorov complexity as the most promising for simulation models. Section 4 introduces a set of simple models that incrementally increase the explicitly modeled scope and uses static source code analysis tools to investigate other source code complexity metrics. Then, Section 5 demonstrates how Kolmogorov complexity could be a superior measure. Section 6 discusses the limitations and challenges for calculating simulation model complexity and suggests future research before Section 7 emphasizes key points and conclusions.

2. Many Facets of Complexity

Complexity is a complicated word. Scientific rationalism and reductionism triumph when logical reasoning allows one to pick apart complicated systems and examine simple constituent parts as a means to understand a complicated whole. (However, one should be cautious in applying too much reductionism to complex systems [5,6], and we have the likes of Heisenberg and Gödel to thank for helping us understand some critical limits. Moon and Blackman describe how a healthy dose of pragmatism helps keep the rationalist house of cards from tumbling down [7].) In the fields of computer science and information systems theory, and even more broadly in management or natural systems, researchers have proposed a number of uses, meanings, and measures for complexity, both broad and narrow. Examining these different meanings provides insight into the different ways one can regard a system, whether that system be simple, complicated, complex, chaotic, or disordered.

2.1. Defining Complexity

In the broadest sense, Rickels, et al., give excellent definitions of the related terminology.

> "*System* is simply the name given to an object studied in some field and might be abstract or concrete; elementary or composite; linear or non-linear; simple or complicated; complex or chaotic. *Complex* systems are highly composite ones, built up from very large numbers of mutually interacting subunits (that are often composites themselves) whose repeated interactions result in rich, collective behavior that feeds back into the behavior of the individual parts. *Chaotic* systems can have very few interacting subunits, but they interact in such a way as to produce very intricate dynamics. *Simple* systems have very few parts that behave according to very simple laws. *Complicated* systems can have very many parts too, but they play specific functional roles and are guided by very simple rules. Complex systems can survive the removal of parts by adapting to the change; to be robust, other systems must build redundancy into the system (e.g., by containing multiple copies of a part)" [8].

Constituent parts of a complicated system have much less interaction and fewer feedback mechanisms than with parts of a complex system.

From a machine learning perspective, if a data set is merely complicated (that is, not complex), then one can often build a decision tree (a hierarchical data structure of multi-dimensional breakpoints: "Is an input parameter higher or lower than *X*?") to understand

the data space. Sports broadcasters often share complicated statistics with their viewing audiences, perhaps ironically attempting to pass it off as giving insight into the complexity of sport. A representative example might go something like, "This pitcher has the best earned run average over the past seven years when starting a night game on a Thursday after having an extra day of rest between starts." A savvy baseball fan will pick out the last of the four factors listed here (past seven years, night game, Thursday, and extra day of rest) as the dominant factor, and the others having dubious influence on the pitcher's success.

However, if a data set is complex, a decision tree alone is not sufficient; one must use a random forest (a probabilistic ensemble of decision trees) or a neural network, or some other algorithm with the capacity to account for the interaction and nonlinearity of the data. Complex systems often exhibit *emergence*, where individual agent-to-agent and agent-to-environment interactions develop into more complex behaviors in the aggregate—the whole is greater or unexplainable as the sum of the parts [9].

In thermodynamics and statistical mechanics, *entropy* quantifies the amount of order or disorder in a system and relates microstates to macrostates [10]. This approach was borrowed by Shannon, whose information theory definition of entropy is discussed below [11].

In management theory, Snowden and Boone defined the cynefin (pronounced kə-*nev*-in, Welsh for neighborhood or environment) framework as a taxonomy of leadership approaches that map neatly into the definitions used by Rickels, et al., and they discuss how understanding complexity can advance scientific management beyond the foundations laid by Taylor [12].

Another broad treatment of complexity comes from the field of cybernetics, the study of communications and automatic control systems in machines, living things, and organizations. Beer used the term *variety* to measure complexity in a system, defined as the number of possible states of the system [13]. Building on this, Ashby defined the law of requisite variety, which states that for any system with a regulatory process, in order to obtain a desired outcome, the system must have at least as many states in the regulatory process as the number of possible input states [14,15].

The field of complex adaptive systems explores natural and engineered systems that have high levels of composition and interaction, and also learn and adapt to internal and external changes [16,17]. In this field, researchers sometimes use the term "holon", coined by Koestler in 1968, to describe nodes in a hierarchy (or "holarchy") that are at once a whole and a part, illustrating the potential for interaction and inter-dependency among and across levels [18].

2.2. Complexity in Computer Science

In a more narrow sense related to computer science, programmers aspire to write code that accomplishes useful tasks efficiently (programmer communicating to the machine typically via a compiler) and also is maintainable (programmer communicating to other programmers). Code complexity correlates with cost and difficulty in development and maintainability, and thus there is great focus in software engineering on simplicity in design and construction [19,20] and several attempts to provide measures for code complexity. *Computational* complexity, sometimes called asymptotic computational complexity, aims at measuring the resource requirements for executing a given algorithm in terms of space and time—more precisely, memory use and the number of processor cycles [21]. *Cyclomatic* complexity (CyC), developed by McCabe in 1976, counts the number of linearly independent paths through a piece of code; however, its use in guiding programmers to more elegant and maintainable code has come into question [22]. Recently, Campbell developed an alternative metric called *cognitive* complexity (CoC) that assesses code complexity based on the following rules: "1. Ignore structures that allow multiple statements to be readably shorthanded into one; 2. Increment (add one) for each break in the linear flow of the code; 3. Increment when flow-breaking structures are nested." [23] CoC also accounts for nesting and control flow structures. The chief aim is to make concrete which portions of

code break the flow while reading it (however, this metric does not appear to have any empirical support from experiments by cognitive scientists, so the use of the term cognitive is potentially problematic).

Early in the development of computer science, researchers recognized the need for methods to measure the runtime performance of software, and analyzing code with a software profiler is now a standard technique for software engineers. One such profiling technique is to keep track of which part of a program is running, and for how long—the GNU profiler `gprof` calls this the flat profile [24].

Software profiling is naturally an important consideration for M&S practitioners, as much of M&S involves implementing models and simulations in software. Control and simulation language (CSL), first described in 1962, allowed programmers to define a number of *activities* which would be invoked through simple or complex logical tests [25]. This approach enables *activity scanning*, counting the number of times an activity is activated [26]. Muzy, et al. tweaked the definition of activity to be, more generically, the number of events occurring during a simulation [27], and Capocchi, et al., employed this definition to develop a methodology for selecting the level of detail in a hierarchical simulation model [28].

2.3. Information Theory Views of Complexity

In the field of information theory, three related concepts are *Kolmogorov complexity* (also referred to as *algorithmic complexity*), *Shannon's entropy* and *Bennett's logical depth*, and Lemberger and Morel give all three ideas excellent treatment [29]. These characterize complexity in terms of the description of a system, "assuming that we have a specific goal in mind" for the description. Shannon's entropy considers the amount of randomness present in a system, often in the context of ensuring the error-free transmission of information through a communication channel. It relies on the fact that characters in a given language occur in words at a certain frequency, and the goal is to "encode strings of characters, whose probabilities are known, in such a way that their average encoded length is as short as possible" [29], and entropy is the average length of the optimal encoding. Logical depth, meanwhile, has the goal of finding "the description that requires as little computation as possible to reproduce the given string" [29]. This allows for a longer description, provided that it reduces computation. Contrast this with Kolmogorov complexity, which is more often thought of as the length of the most possible compression of a string, or as the minimum description length of an object. In all three ideas, the metric provides a useful estimate of optimality, but there is never a guarantee of finding the global optimum. Another key assertion of Lemberger and Morel is that there is no single concept of complexity relevant for all purposes. One must specify in what context or for what purpose one wants to consider a system's complexity, reminiscent of how a wave function in quantum physics collapses when one makes an observation.

2.4. Kolmogorov Complexity

A review of the precise definition of Kolmogorov complexity (KC) is useful here, as later sections will build on these ideas. In the 1960s, three scientists—Ray Solomonoff, Andrey Kolmogorov, and Gregory Chaitin—separately discovered very similar theoretical results in the fields of mathematics and computer science that together formed the foundation of the concept of Kolmogorov complexity and algorithmic information theory (AIT) [30–34]. Formally, the Kolmogorov complexity C of an object X is defined as

$$\mathrm{C}(X) = \min_p \{\mathrm{length}(p) : \mathrm{U}(p) = X\} \tag{1}$$

where p is a computer program that, when executed on a universal Turing machine U, produces X as an output. KC is sometimes referred to as the minimum description length for an object, and is often expressed in bits, and scaled by $\log_2$.

Note that X as an "object" does not mean an instance of an object-oriented class loaded in the memory of a running computer program but rather is used in the generic sense of the word. A Wikipedia article about the Kamchatka Peninsula (selected to be mentioned

here by choosing geographic coordinates using the website http://random.org, accessed on 4 February 2022), a password for user authentication on a computer network or website, the third coffee mug in your kitchen cabinet, the song *Heat Waves* by Glass Animals (number 1 streamed song on Spotify on 4 February 2022), and Andrew Wyeth's painting *Christina's World* are all objects, and each has a KC that can be calculated, at least in theory.

In practice, directly computing KC turns out to be impossible most of the time. Much as it is not possible to determine if an arbitrary computer program will run to completion (defined by Alan Turing as the "halting problem"), KC is non-computable for non-trivial examples. This is so because there is no guarantee that a given p is the shortest program that will produce X. Despite the direct non-computability of KC, there are methods to approximate its value, establish a bound, or otherwise employ the concept in useful ways. For example, suppose X is stored in a list in computer memory. To produce X, one needs only the index of X in that list, and the KC of X is its index. Cilibrasi and Vitányi used this approach to define the Google Similarity Distance [35], which uses the position of words and phrases in Google search results as a complexity measure and distance metric.

Another approach to approximate KC is through data compression. The information carried by an object is what remains when all redundancy is removed, or in other words, what is left over after maximal lossless compression. Cilibrasi and Vitányi describe how compression/decompression programs exhibit desirable properties that make them excellent choices as the program p in the definition of KC [36]. While their approach is robust to a number of compression algorithms, consider the Lempel–Ziv–Welch (LZW) compression algorithm [37], and the related Lempel–Ziv–Markov chain (LZMA) algorithm [38], which create a dictionary of the data to be compressed. The dictionary can be stored as a hash table and the compressed representation of the data stored as a sequence of keys. The concatenation of the hash values and the sequence of keys is in most cases smaller than the original data. The concatenation of the decompression algorithm and the compressed data takes the place of the program p in Equation (1), and the $\log_2$ of the size of the compressed data file is a suitable approximation of the KC of the original uncompressed data.

At the heart of the approach from Cilibrasi and Vitányi is their concept of *normalized compression distance* (NCD), which they demonstrated as a distance metric for clustering a wide variety of data types with no requirement for a priori knowledge of data features or subject matter.

$$\mathrm{NCD}(x,y) = \frac{\mathrm{C}(xy) - \min\{\mathrm{C}(x), \mathrm{C}(y)\}}{\max\{\mathrm{C}(x), \mathrm{C}(y)\}} \tag{2}$$

Here, $\mathrm{C}(x)$ is the compressed size of data object x, and $\mathrm{C}(xy)$ is the compressed size of concatenated data objects x and y. When concatenated data objects have a high compression ratio and low NCD, this indicates that the two objects share a significant amount of mutual information. Section 5 provides an example application of these concepts for measuring simulation model complexity, while the next section explores the established theory of simulation model complexity, proposes refined definitions of key terms, and illustrates these terms with a set of simulation model archetypes.

3. Simulation Model Complexity, Abstraction, and Model Families

One turns to modeling and simulation (M&S) in a number of situations with the expectation that time invested in developing and executing simulation experiments will result in a sufficient return of knowledge to justify the investment. The economics of M&S drive a desire to maximize the return on investment by developing a model that, in the ideal, contains just enough detail to inform the problem at hand [39,40]. For many applications of M&S, models are necessarily simplifications of the source system they are intended to represent, implying that the cost to develop and experiment with the model is less than the cost to develop and experiment with the source system. Zeigler, et al., define model abstractions as valid simplifications of the source system [1]. For very complicated and complex systems-of-systems (SoS), a single representation is insufficient to inform

all of the questions and decisions encountered through the SoS life cycle. A number of researchers have explored multi-perspective and multi-resolution modeling, and in the United States Department of Defense (DoD) and United States Air Force (USAF), it is common to categorize broad model families in terms of the size, scope, and resolution they provide to investigate the source system. These categories are often depicted in the shape of a triangle, with higher resolution models at the base and more abstracted but more broadly scoped models at the top [41–46].

3.1. Defining Simulation Model Complexity

Even more narrowly than the above discussion in Section 2, with respect to simulation models, Zeigler, et al., distinguished between three kinds of complexity. A simulation model's overall complexity is driven by the size/resolution/interaction product, which are illustrated in Figure 1 [1]. Quoting from Zeigler, "*Scope* refers to how much of the real world is represented, *resolution* refers to the number of variables in the model and their precision or granularity." Further, Zeigler, et al. define *size* as "the number of components in a coupled model", and is strongly determined by scope most of the time. Building on this foundation, the three kinds of complexity are analytic, simulation, and development complexity. *Analytic* complexity is the number of states that fully explore the global state space of a model. The *simulation* complexity is essentially the computational complexity of the simulation program, although they characterize it in more specific simulation terms, such as the number of interactions. Third, the *exploratory* complexity is the computational complexity of searching a modeling space.

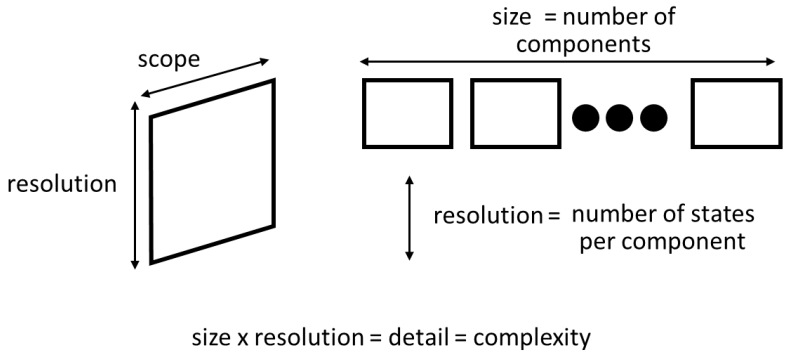


Figure 1. Zeigler, et al. define a simulation model's complexity as driven by its level of detail, which is the product of the model's size and scope, resolution, and interactions. Reprinted with permission from Ref. [1]. Copyright 2019 Elsevier Inc.

Other foundational research into defining model resolution comes from Davis and several collaborators. Davis and Hillestad defined resolution as having many facets, including the following:

- Entity: Modeled agents are individuals or aggregated as groups.
- Attribute: Account for specific component performance or overall system performance/effectiveness.
- Logical dependency: Do (or do not) place constraints on attributes and their interdependence.
- Process: State changes computed at different entity/attribute resolutions.
- Spatial and temporal: Finer or coarser scales of space and time [42].

Davis and Hillestad's perspective seems to lump Zeigler's concept of scope into the idea of attribute resolution and logical-dependency resolution. Merging the two approaches, the definitions in Table 1 frame the concept of simulation model complexity.

Table 1. New definitions of simulation model complexity and related terms synthesized from Zeigler, Davis, and others.

Term	Definition
Scope	The breadth of the model; how much of reality is represented, particularly the number of components and parameters encoded as model variables to represent those components
Resolution	The depth of the model; the precision, granularity, quantization, and/or discretization of model parameters and structure, including internal interactions and logical dependencies
Detail	The product or integration of the model's scope and resolution
Complexity	The aggregation of a model's detail, external interactions and logical dependencies, and in many contexts its instantiation (input parameter settings) in a specific simulation scenario execution

3.2. Model Family Archetypes

As introduced previously, when developing a complex SoS, such as future force designs for defense acquisition, it is often useful to conceptualize a hierarchy of simulation model representations and analysis of the SoS. The levels in this modeling, simulation, and analysis (MS&A) hierarchy, depicted in Figure 2, align to levels of key performance parameters of the SoS that provide insight into its capabilities, namely, measures of performance (MoPs), measures of effectiveness (MoEs), and measures of objectives (MoOs) [46].

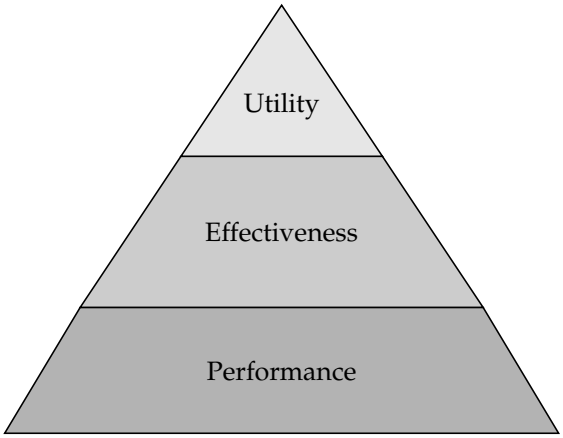


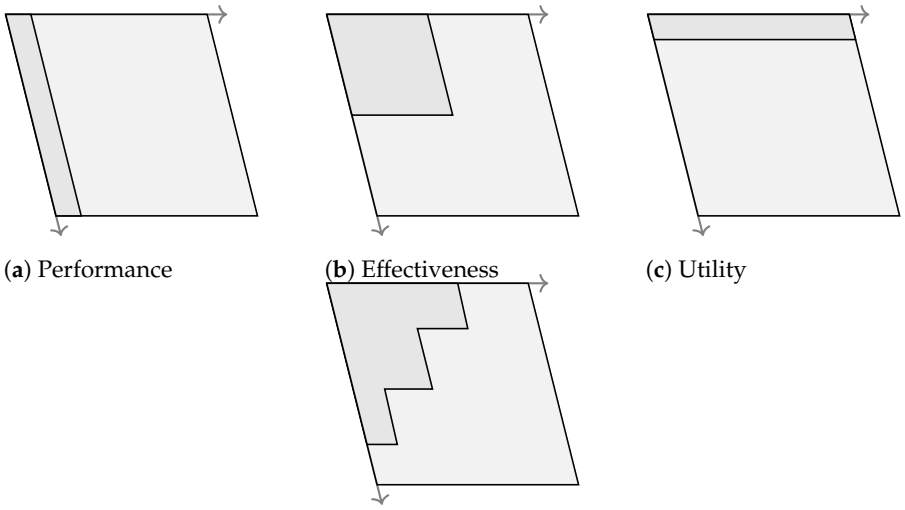
Figure 2. In LiCausi's depiction of the MS&A hierarchy, there are three levels of analysis: performance analysis, effectiveness analysis, and utility analysis. Simulation models typically support analysis at one of these levels, and a family of models are needed for multiple levels of analysis. Adapted from [46].

The following discussion and Figure 3 illustrate a number of simulation model archetypes for these levels of analysis:

- A performance model emphasizes resolution in a small area of scope, for example, a computational fluid dynamics model of airflow over a wing. While not true in all cases, simulations with performance models typically occur over a short duration (perhaps on the order of seconds or milliseconds), or may even be static in time.
- An effectiveness model trades away some of the resolution to consider broader scope. The flight dynamics models described below in Section 4.1 can be used for effectiveness analysis, in many cases, particularly when assessing the effectiveness of mission systems and payloads on an air vehicle. Effectiveness simulations typically span minutes to hours of simulated time.
- A utility model generally treats most components of a system with the lowest resolution in order to investigate the system in its most broad scope. For assessing an air

vehicle's utility to meet strategic and operational objectives, its movement may be characterized very simply, such as with an average cruising speed to compute the time it takes to travel to its destination. Utility simulations cover many hours, days, or an even longer duration.

The final archetype in Figure 3 is that of a typical model, intended to convey that simulation models tend to have a particular focus area with higher resolution in support of the model's intended use. A typical model does not match the resolution of the theoretical source system, nor does it cover the entire scope of the source system, as there will remain unknown and unobservable effects. In Figure 3, the darkly shaded areas of the figure depict the resolution and scope of each archetype, relative to the theoretical source system.



(**a**) Performance (**b**) Effectiveness (**c**) Utility

(**d**) Typical model

Figure 3. An archetypal performance model has high resolution and limited scope, whereas an archetypal utility model has wide scope and low resolution. A typical simulation model often emphasizes resolution in a particular area of concern, makes simplifying assumptions and implements sufficient abstractions for other areas still within scope, and assumes away other areas as out of scope.

4. Investigating Complexity Measures

This section explores two of the source code complexity measures discussed above in Section 2: cyclomatic complexity (CyC), and cognitive complexity (CoC).

4.1. Flight Dynamics Simulation Models

To illustrate the application of measuring the complexity of a family of related simulation models, consider a family of models that represent the motion of an air vehicle. The first model is the most detailed, inspired by a description of the equations of motion provided in a textbook for game developers by Grant Palmer, hereafter called the Palmer model. The Palmer model calculates the four primary forces of flight in three dimensions: lift, thrust, gravity, and drag [47]. The authors implemented this model in Rust (https://www.rust-lang.org, accessed on 4 February 2022) and use a fourth-order Runge–Kutta integration scheme to solve the differential equations of motion for a Cessna 172 Skyhawk. Note that the Palmer model does make several simplifying assumptions, such as modeling the aircraft as a point mass, not an extended rigid or flexing body. This level of abstraction is suitable for many applications, such as a virtual flight simulation, where the goal of the flight dynamics model is to provide a user with a sufficiently realistic experience of piloting a specific aircraft under typical flight conditions.

At an even higher level of abstraction, consider three models implemented as individual Rust functions for simple kinematic equations of motion with two or three degrees of freedom: 2DOF, 2DOF-TRC, and 3DOF. The equations are based on a white paper by Weintraub [48]. The model 2DOF-TRC applies a turn rate constraint that limits the air vehicle's

ability to change its heading in a single time step. Each function updates an input state vector of 12 components consisting of the aircraft's position, velocity, orientation, and orientation rates. The functions perform direct propagation (Euler's method) and no more sophisticated numerical integration, requiring that the time step remains small. Together, they represent an incremental relaxing of assumptions about the aircraft's motion and the incremental addition of details that progressively better describe the motion of a true air vehicle. Even as simplistic as they may seem, such representations are used frequently in MS&A of future air vehicle concept exploration and requirements development.

4.1.1. Static Analysis Tools and Methodology

Members of the Rust community developed two static analysis tools that calculate relevant metrics. Clippy is an open-source linter that at one time calculated CyC then switched to CoC (although not without dissenting voices about CoC's usefulness; see https://github.com/rust-lang/rust-clippy/issues/3793#issuecomment-542457771, accessed on 2 February 2022) [49]. Clippy does not report CoC by default but can be forced to output CoC for any function that has non-zero CoC by setting the parameter `cognitive-complexity-threshold` to 1. The rust-code-analysis (RCA) library calculates CyC, CoC, and a number of other metrics [50]. The most appropriate comparisons with the simple models are with those in the Palmer file `fdm.rs`, as these are the equations directly involved in updating aircraft state. This analysis ignores other functions.

4.1.2. Results and Interpretation

Results are shown in Table 2. Clippy only provided CoC metrics for the three functions in `fdm.rs` of the Palmer model, not the simple models, indicating that the simple model functions all have zero CoC. Clippy and RCA disagree on the reported value of CoC for the Palmer functions and for `update_2dof_trc`, although the trends are mostly consistent. The RCA CoC and RCA CyC values almost never match—one would not expect them to—yet there is general agreement in the trends between them.

Table 2. The static analysis tools Clippy and rust-code-analysis generated metrics for cognitive complexity (CoC) and cyclomatic complexity (CyC) for each of the functions in the Palmer and the simple models.

Model	Function	Clippy CoC	RCA CoC	RCA CyC
Palmer	`calculate_forces`	6	7	7
Palmer	`plane_rhs`	5	7	6
Palmer	`eom_rk4`	3	2	3
2DOF	`update_2dof`	0	0	1
2DOF-TRC	`update_2dof_trc`	0	2	2
3DOF	`update_3dof`	0	0	1

Inspecting the code for the Palmer model, these functions account for various situations with logic checks in IF statements. In `calculate_forces`, there are logic checks for computing lift (below or above maximum angle of attack, flap position, and ground effects) and for modeling the ground by setting the gravitational force to zero if the altitude is zero. In `plane_rhs`, there is a loop to compute intermediate values of the state vector, as well as logic checks to avoid division by zero, and another check for zeroing the gravitational force if the altitude is zero. In `eom_rk4`, there are two FOR loops for updating the state vector. Conversely, the only simple model function to implement a logic check is 2DOF-TRC, which does so to enforce the maximum turn rate constraint. The RCA CoC and RCA CyC values are higher for the functions in `fdm.rs` of the Palmer model than for the simple model functions, indicating that these metrics picked up on the distinct increase in representation detail. There is no difference between the 2DOF and 3DOF models, though. The metrics are not able to detect the increase in information content because it does not require more

complex code structure to implement. This analysis demonstrates that CoC and CyC are not sensitive to changes in size or scope for the Palmer or simple models.

5. Measuring Kolmogorov Complexity of Simulation Models

Previously in this article, Section 2.4 introduced the concept of Kolmogorov complexity and how researchers have used compression as a way to approximate KC. After exploring some alternative measures of source code complexity in Section 4, this section describes how to measure KC of a simulation model, considering two case studies.

5.1. Case Study 1: Kolmogorov Complexity of the Simple Air Vehicle Models

Each model introduced in Section 4.1 was separated into individual files and compressed using the 7zip program. Table 3 lists the KC of each model ($\log_2$ of the size of the compressed archive). The impact of incrementally adding scope to the equations of motion is seen in the incremental increase in KC.

Table 3. Compressed archive file size and KC values of the simple air vehicle models.

Model	Compressed Size (Bytes)	Kolmogorov Complexity
2DOF	375	8.551
2DOF-TRC	448	8.807
3DOF	477	8.898

5.2. Case Study 2: Normalized Compression Distance of a Family of AFSIM Models

The Air Force Research Laboratory maintains and distributes the Advanced Framework for Simulation, Integration, and Modeling (AFSIM) that supports a wide variety of MS&A [51,52]. AFSIM uses an object-oriented entity-component software design together with a run-time plugin capability to interface simulation models of components not supported by the core framework. This design allows AFSIM users to build simulations choosing from a variety of representations, including movement models of different scopes and resolutions. One of the built-in movement models is called AIRMOVER and is similar to the 3DOF simple air vehicle model described above. AIRMOVER does not account for explicit aerodynamics, and vertical transitions are discontinuous, although it does produce continuous, smooth motion in the horizontal plane by enforcing maximums for linear acceleration, velocity, and radial acceleration. Users can define an entity and add an AIRMOVER component using a domain-specific language of AFSIM commands and scripts.

Another air vehicle movement model option for AFSIM users is called P6DOF, short for the pseudo six-degree-of-freedom model. This represents an air vehicle as a rigid body, and calculates aerodynamic forces and computes moments. The "pseudo" qualifier indicates abstraction in various modeling design choices—for example, assuming off-diagonal elements in the moment of inertia tensor are zero. P6DOF is implemented as an AFSIM plugin and has little relationship to the core framework. Similar to AIRMOVER, users define an entity and add a P6DOF component using extensions of the AFSIM command and scripting language.

The AFSIM release comes with a number of examples and demos, several of which were selected for a demonstration of calculating NCD, listed in Table 4. Source code (C++) and model definitions (AFSIM command and script language) were extracted from the main folder structure and copied into separate directories. Using the 7zip program, these directories were compressed alone and in pairs and the size of the compressed archives inserted into Equation (2). Compressed archive file size and NCD values are reported in Table 5.

Table 4. Select examples from the AFSIM release.

Name	Description	Modeled Components
AirMover	Single-engine light fighter aircraft	• 3DOF movement
FA-LGT	Single-engine light fighter aircraft	• Aero for fixed surfaces and each flight control surface • Flight controls • Mass • Propulsion – Centerline engine – Fuel tank capacity – Throttle and fuel flow rates
FA-LGT + Ext Tanks	Extends FA-LGT, adds external fuel tanks	• All from FA-LGT • 2 wing-mounted fuel tanks – Aero – Mass – Fuel capacity
C-HVY	Four-engine heavy airlift cargo aircraft	• Aero for fixed surfaces and each flight control surface • Flight controls • Mass • Propulsion – 4 under-wing engines – Fuel tank – Throttle and fuel flow rates

Table 5. Compressed archive file size and normalized compression distance values.

Model	Compressed Size (Bytes)	KC	NCD
AirMover	63,421	15.9527	N/A
FA-LGT	21,258,004	24.3415	N/A
FA-LGT + ext tanks	22,078,546	24.3961	N/A
C-HVY	20,867,455	24.3078	N/A
FA-LGT and FA-LGT + ext tanks	22,114,939	24.3985	0.038813
FA-LGT and C-HVY	23,697,162	24.4982	0.128165
FA-LGT and AirMover	21,322,768	24.3459	1.0

The relative simplicity of the AirMover model is seen in the order of magnitude difference in compressed archive size compared to the P6DOF models. The small NCD of FA-LGT and FA-LGT + external tanks demonstrates the large amount of mutual information in the two models, while FA-LGT and C-HVY have a slightly larger NCD. The NCD of FA-LGT and the AirMover model maximizes the theoretical value, indicating that the two models share no detectable mutual information.

6. Challenges and Future Research

The approach introduced above suggests that the size of model source code and data after compression can be interpreted as a measure of the complexity of a simulation model. Before this approach can be widely adopted by M&S practitioners, additional research must be done to overcome limitations and to more precisely define the approach. Several of these challenges and opportunities for further work are described here.

1. How far afield can this approach be used for valid comparison? It is well documented that there is a wide range in the complexity of source code written by different programmers, or written in different languages. The invariance theorem of Kolmogorov complexity (see Chapter 2 of Li and Vitányi [53]) implies that these differences can be accounted for with an additive constant, but how often does the constant dominate, washing out any observability of complexity differences due to changes in model detail?
2. What gets measured? What gets included on the scale? To interpret an increase in KC as an increase in model detail and complexity, one would need confidence that all model details are included in the measurement, and irrelevant data are not included. However, how does one draw a line to encapsulate a model? Formal modeling specifications, such as discrete event system specification (DEVS), can provide a clear boundary. Research into M&S ontology can help to parse semantics. More work should be conducted to investigate tying these other areas of M&S research into model complexity.
3. How does using KC compare to the approach defined by Cappochi, et al., [28] in using activity as the base metric for simulation complexity? Can KC of a DEVS model's event transition functions be substituted for activity in Cappochi's methodology?
4. Is the textual representation of the model source code and input data the right representation for this type of analysis? Other representations may be better, such as intermediate representations, such as LLVM used in the code compilation process [54]. This can take advantage of the automatic code optimization that compilers perform.
5. How to measure? How should the compression program be configured? The 7zip program, used in the above demonstrations, allows a user to select from a number of parameters, including compression algorithm, level of compression, dictionary size, word size, and solid block size.
6. How should the measurement be calculated? These demonstrations use the size of the compressed data, sometimes scaling by $\log_2$.
7. How is complexity related to model accuracy and precision? Can the complexity analysis be paired with verification, validation, and uncertainty quantification (VVUQ) approaches to inform model selection?
8. How can resolution and scope be measured independently? Is it possible to treat them as separate dimensions of complexity?

7. Conclusions

Understanding a variety of perspectives on complexity can help M&S practitioners in developing and employing M&S tools. Using Kolmogorov complexity to interpret the size of a compressed archive containing a simulation model, it may be possible to measure the complexity of the simulation model and relate that directly to the model's resolution and scope. This article investigates existing source code metrics that do not indicate changes in model detail, and also how KC and NCD may provide such an indication. Measurement of model complexity can assist M&S practitioners in selecting a representation that is best suited for meeting the requirements. Even if the selection is not a difficult one (choosing a utility model over a performance model when addressing questions of broad scope), a deeper understanding of complexity as being driven by resolution and scope can lead to better conceptual models in designing and implementing simulation models in software. Overall, expanding this area of research will advance the body of knowledge of M&S toward a practical means to measure simulation model complexity.

Author Contributions: Conceptualization, writing—review and editing, J.S.T., D.D.H., M.R.G., N.H. and R.D.; software designs and implementations, J.S.T. and D.D.H.; methodology, formal analysis, and writing—original draft preparation, J.S.T.; supervision, D.D.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The flight dynamics models described in Section 4.1 are available at https://github.com/jscott-thompson/se-models, accessed on 20 December 2022 and https://github.com/jscott-thompson/gphys-palmer, accessed on 20 December 2022. The AFSIM models described in Section 5.2 are not available for public release.

Acknowledgments: The authors wish to acknowledge the Air Force Research Laboratory, Aerospace Systems Directorate, Strategic Analysis and Planning Division, Modeling, Simulation, and Analysis Branch, for supporting this research effort through shared duty assignment with the Air Force Institute of Technology, Department of Electrical and Computer Engineering. We also thank the academic reviewers for their constructive feedback that improved the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Disclaimer: The views expressed in this document are those of the authors and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. Approved for public release, unlimited distribution. Case #AFRL-2022-5979, 19 December 2022.

Abbreviations

The following abbreviations are used in this manuscript:

2DOF	Two degree-of-freedom movement model
2DOF-TRC	Two degree-of-freedom movement model with a turn rate constraint
3DOF	Three degree-of-freedom movement model
AFSIM	Advanced Framework for Simulation, Integration, and Modeling
DoD	Department of Defense
CKC	Conditional Kolmogorov complexity
CoC	Cognitive complexity (of source code)
CyC	Cyclomatic complexity (of source code)
KC	Kolmogorov complexity
LZMA	Lempel–Ziv–Markov chain algorithm
LZW	Lempel–Ziv–Welch algorithm
MoE	Measure of effectiveness
MoO	Measure of objective
MoP	Measure of performance
MBSE	Model-based systems engineering
M&S	Modeling and simulation
MS&A	Modeling, simulation, and analysis
NCD	Normalized compression distance
P6DOF	Pseudo six degree-of-freedom movement model
RCA	Rust-code-analysis library
SoS	System-of-systems
TRC	Turn rate constraint
USAF	United States Air Force
VVUQ	Verification, validation, and uncertainty quantification

References

1. Zeigler, B.P.; Muzy, A.; Kofman, E. Chapter 16—Abstraction: Constructing Model Families. In *Theory of Modeling and Simulation*, 3rd ed.; Zeigler, B.P., Muzy, A., Kofman, E., Eds.; Academic Press: Cambridge, MA, USA, 2019; pp. 405–443.
2. Thorburn, W.M. Occam's Razor. *Mind* **1915**, *24*, 287–288. [CrossRef]
3. Box, G.E.P. Science and Statistics. *J. Am. Stat. Assoc.* **1976**, *71*, 791–799. [CrossRef]
4. Hofmann, M.; Palii, J.; Mihelcic, G. Epistemic and normative aspects of ontologies in modelling and simulation. *J. Simul.* **2011**, *5*, 135–146. [CrossRef]
5. Feher, J. The Core Principles of Physiology. In *Quantitative Human Physiology*; Feher, J., Ed.; Academic Press: Boston, MA, USA, 2012; pp. 3–11.
6. Hankey, A. The ontological status of western science and medicine. *J. Ayurveda Integr. Med.* **2012**, *3*, 119–123. [CrossRef] [PubMed]
7. Moon, K.; Blackman, D. A guide to understanding social science research for natural scientists. *Conserv. Biol.* **2014**, *28*, 1167–1177. [CrossRef]

8. Rickles, D.; Hawe, P.; Shiell, A. A simple guide to chaos and complexity. *J. Epidemiol. Community Health* **2007**, *61*, 933–937. [CrossRef]
9. King, D.W. (Air Force Institute of Technology, Department of Electrical & Computer Engineering, Wright-Patterson AFB, OH, USA); Peterson, G.L. (Air Force Institute of Technology, Department of Electrical & Computer Engineering, Wright-Patterson AFB, OH, USA) Classifying Emergence. 2020, Unpublished manuscript.
10. Wehrl, A. General properties of entropy. *Rev. Mod. Phys.* **1978**, *50*, 221–260. [CrossRef]
11. Ebeling, W.; Molgedey, L.; Kurths, J.; Schwarz, U. Entropy, Complexity, Predictability, and Data Analysis of Time Series and Letter Sequences. In *The Science of Disasters: Climate Disruptions, Heart Attacks, and Market Crashes*; Bunde, A., Kropp, J., Schellnhuber, H.J., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 2–25.
12. Snowden, D.J.; Boone, M.E. A Leader's Framework for Decision Making. *Harv. Bus. Rev.* **2007**, *85*, 68.
13. Beer, S. *The Heart of Enterprise*; John Wiley & Sons Ltd.: Bath, UK, 1979.
14. Raadt, J.D.R.d. Ashby's Law of Requisite Variety: An Empirical Study. *Cybern. Syst.* **1987**, *18*, 517–536. [CrossRef]
15. Ashby, W.R.; Goldstein, J. Variety, Constraint, and the Law of Requisite Variety. *Emerg. Complex. Organ.* **2011**, *13*, 190–207.
16. Valckenaers, P.; Van Brussel, H. Chapter Two—On the Design of Complex Systems. In *Design for the Unexpected*; Valckenaers, P., Van Brussel, H., Eds.; Butterworth-Heinemann: Oxford, UK, 2016; pp. 9–18.
17. Johnson, B.; Hernandez, A. Exploring Engineered Complex Adaptive Systems of Systems. *Procedia Comput. Sci.* **2016**, *95*, 58–65. [CrossRef]
18. Koestler, A. *The Ghost in the Machine*; Macmillan: Oxford, England, 1968; Volume 384.
19. Boehm, B.W. Software Engineering Economics. *IEEE Trans. Software Eng.* **1984**, *SE-10*, 4–21. [CrossRef]
20. McConnell, S. *Code Complete: A Practical Handbook of Software Construction*, 2nd ed.; Microsoft Press: Redmond, WA, USA, 2004.
21. Arora, S.; Barak, B. *Computational Complexity: A Modern Approach*; Cambridge University Press: Cambridge, UK, 2007.
22. Ebert, C.; Cain, J.; Antoniol, G.; Counsell, S.; Laplante, P. Cyclomatic Complexity. *IEEE Softw.* **2016**, *33*, 27–29. [CrossRef]
23. Campbell, G.A. *A New Way of Measuring Understandability*; Technical Report; SonarSource: Geneva, Switzerland, 2021.
24. Graham, S.L.; Kessler, P.B.; McKusick, M.K. gprof: A call graph execution profiler. *SIGPLAN Not.* **2004**, *39*, 49–57. [CrossRef]
25. Buxton, J.N.; Laski, J.G. Control and Simulation Language. *Comput. J.* **1962**, *5*, 194–199. [CrossRef]
26. Balci, O. The implementation of four conceptual frameworks for simulation modeling in high-level languages. In Proceedings of the 1988 Winter Simulation Conference Proceedings, San Diego, CA, USA, 12–14 December 1988; pp. 287–295.
27. Muzy, A.; Touraille, L.; Vangheluwe, H.; Michel, O.; Traoré, M.K.; Hill, D.R.C. Activity regions for the specification of discrete event systems. In Proceedings of the 2010 Spring Simulation Multiconference, Orlando, FL, USA, 11–15 April 2010; Society for Computer Simulation International: San Diego, CA, USA, 2010; Number Article 136 in SpringSim '10, pp. 1–7.
28. Capocchi, L.; Santucci, J.F.; Pawletta, T.; Folkerts, H.; Zeigler, B.P. Discrete-Event Simulation Model Generation based on Activity Metrics. *Simul. Model. Pract. Theory* **2020**, *103*, 102122. [CrossRef]
29. Lemberger, P.; Morel, M. Complexity, Simplicity, and Abstraction. In *Managing Complexity of Information Systems: The Value of Simplicity*; ISTE Ltd: London, UK; John Wiley & Sons: London, UK, 2011; pp. 18–75.
30. Solomonoff, R.J. A formal theory of inductive inference. Part I. *Inf. Control* **1964**, *7*, 1–22. [CrossRef]
31. Solomonoff, R.J. A formal theory of inductive inference. Part II. *Inf. Control* **1964**, *7*, 224–254. [CrossRef]
32. Kolmogorov, A.N. Three approaches to the definition of the concept 'quantity of information'. *Probl. Peredachi Inf.* **1965**, *1*, 3–11. (In Russian)
33. Kolmogorov, A.N. Three approaches to the quantitative definition of information. *Int. J. Comput. Math.* **1968**, *2*, 157–168. [CrossRef]
34. Chaitin, G.J. On the Length of Programs for Computing Finite Binary Sequences. *J. ACM* **1966**, *13*, 547–569. [CrossRef]
35. Cilibrasi, R.L.; Vitányi, P.M.B. The Google Similarity Distance. *IEEE Trans. Knowl. Data Eng.* **2007**, *19*, 370–383. [CrossRef]
36. Cilibrasi, R.L.; Vitányi, P.M.B. Clustering by compression. *IEEE Trans. Inf. Theory* **2005**, *51*, 1523–1545. [CrossRef]
37. Welch, T.A. A Technique for High-Performance Data Compression. *IEEE Comp* **1984**, *17*, 8–19. [CrossRef]
38. Pavlov, I. LZMA Software Development Kit. 2022. Available online: https://www.7-zip.org/sdk.html (accessed on 9 November 2022).
39. Nutaro, J.; Zeigler, B.P. Towards a theory of economic value for modeling and simulation: incremental cost of parallel simulation (wip). In Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences, Baltimore, MD, USA, 15–18 April 2018; Association for Computing Machinery: New York, NY, USA, 2018; Number Article 9 in ICCES'18, pp. 1–11.
40. Kobren, E.; Oswalt, I.; Cooley, T.; Waite, W.; Waite, E.; Gordon, S.; Severinghaus, R.; Feinberg, J.; Lightner, G. Calculating Return on Investment for U.S. Department of Defense Modeling and Simulation. 2011. Available online: https://apps.dtic.mil/sti/pdfs/ADA539717.pdf (accessed on 27 October 2022).
41. Friel, J. Air Battle Models. In *Military Modeling*; Hughes, W.P., Jr., Ed.; The Military Operations Research Society: Alexandria, VA, USA, 1984; pp. 127–145.
42. Davis, P.K.; Hillestad, R. Aggregation, disaggregation, and the challenge of crossing levels of resolution when designing and connecting models. In Proceedings of the 1993 4th Annual Conference on AI, Simulation and Planning in High Autonomy Systems, Tucson, AZ, USA, 20–22 September 1993; pp. 180–188.
43. Davis, P.K.; Hillestad, R. Families of Models that Cross Levels of Resolution: Issues for Design, Calibration and Management. In Proceedings of the 1993 Winter Simulation Conference-(WSC '93), Los Angeles, CA, USA, 12–15 December 1993; pp. 1003–1012.

44. Trevisani, D.A.; Sisti, A.F. Air Force hierarchy of models: A look inside the great pyramid. In Proceedings of the Enabling Technology for Simulation Science IV, Orlando, FL, USA, 25–27 April 2000; pp. 150–159.
45. Gallagher, M.A.; Caswell, D.J.; Hanlon, B.; Hill, J.M. Rethinking the Hierarchy of Analytic Models and Simulations for Conflicts. *Mil. Oper. Res.* **2014**, *19*, 15–24. [CrossRef]
46. LiCausi, A. MS&A of MOPs, MOEs, and MOOs: The Analysis Pyramid Reimagined. *Phalanx* **2019**, *52*, 46–51.
47. Palmer, G. *Physics for Game Programmers*; Apress: New York, NY, USA, 2005.
48. Weintraub, I.E. *Various Air Platform Models*; Technical Report; Air Force Research Laboratory, Aerospace Systems Directorate, Wright-Patterson AFB: Dayton, OH, USA, 2019.
49. rust-lang. Rust-Clippy: A Bunch of Lints to Catch Common Mistakes and Improve Your Rust Code. 2022. Available online: https://github.com/rust-lang/rust-clippy (accessed on 26 April 2022).
50. Ardito, L.; Barbato, L.; Castelluccio, M.; Coppola, R.; Denizet, C.; Ledru, S.; Valsesia, M. rust-code-analysis: A Rust library to analyze and extract maintainability information from source codes. *SoftwareX* **2020**, *12*, 100635. [CrossRef]
51. Clive, P.D.; Johnson, J.A.; Moss, M.J.; Zeh, J.M.; Birkmire, B.M.; Hodson, D.D. Advanced Framework for simulation, integration and modeling (AFSIM). In Proceedings of the 2015 International Conference on Scientific Computing, Las Vegas, NV, USA, 27–30 July 2015; pp. 73–77.
52. AFSIM Product Management Team. *Advanced Framework for Simulation, Integration, and Modeling (AFSIM) v2.7.1 Basic User Training Presentation*; Air Force Research Laboratory, Wright-Patterson AFB: Dayton, OH, USA, 2020.
53. Li, M.; Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*; Texts in Computer Science; Springer: Cham, Switzerland, 2019.
54. Lattner, C.A. LLVM: An Infrastructure for Multi-Stage Optimization. Master's Thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2002.