Procedural Ontology: The Metaphysics of Code Generation

A Relativistic Scalar–Vector Plenum (RSVP) Essay

Flyxion

2025

Section Synopses

Section 1. Text is not inert symbol but anticipatory matter, encoding potential energy as syntactic curvature within a measurable linguistic manifold.

Section 2. Execution transforms frozen law into kinetic flow via stochastic differential equations on the linguistic manifold, linking gradient descent to Hamiltonian mechanics.

Section 3. Rendering is entropic expansion: the diffusion of compressed potential into observable diversity, quantified by Shannon entropy over output distributions.

Section 4. Meaning is entropic compression; Kolmogorov depth and Assembly Index quantify semantic density through bounds on causal complexity.

Section 5. Authorship is delegated across human intention, interpreter dynamics, and environmental variance—law realism with instance nominalism.

Section 6. Each execution is a distinct cosmogenic instance under invariant law—repetition with stochastic variance, not creation ex nihilo.

Section 7. Reflexive code achieves semantic closure through bounded self-modification; self-modeling as the computational analogue of consciousness.

Section 8. Knowledge is generative transformation; epistemology collapses into ontology via continuous execution pipelines in cognitive systems.

Section 9. Procedural generation structurally mirrors RSVP cosmology: text $\to$ execution $\to$ output as $\Phi \to \sqsubseteq \to S$.

Section 10. Syntax is substance, execution is causality, rendering is reality—being as internal differentiation within a fixed informational plenum.

Section 11. Assembly Index measures causal depth; render entropy increases while compositional entropy decreases, with empirical validation via fractal generators.

Section 12. Objections addressed: structural isomorphism via functorial homology, computational realism via Landauer grounding, and bounded reflexivity via descriptive ascent.

Contents

Abstract

This essay establishes a rigorous metaphysical and physical interpretation of generative computation through the Relativistic Scalar–Vector Plenum (RSVP) framework. Code execution is modeled as a microcosmic cosmogenesis: text as scalar potential Φ, execution as stochastic vector flow $\sqsubseteq$, and rendering as entropic diffusion S. Formal field definitions, stochastic differential equations, Assembly Theory, and functorial mappings are integrated to demonstrate structural isomorphisms with physical law. The framework bridges information theory, computational complexity, and cosmology, proposing that every executable text is a local plenum—a finite law whose repeated execution constitutes entropic relaxation toward visible form. Empirical predictions, dimensional consistency, and philosophical defenses ensure the model transcends metaphor.

Keywords: Procedural ontology, RSVP theory, Assembly Index, stochastic execution, field metaphysics, computational realism, semantic compression, reflexive computation, functorial isomorphism, entropic relaxation.

1. Text as Ontological Generator

At the foundation of procedural reality lies text: a finite inscription of instructions that encodes potential structure. In traditional computation, code is treated as a formal specification of actions. Under a deeper RSVP interpretation, a program is a local instantiation of the scalar field Φ: a distribution of potential energy in a linguistic manifold. Each variable and operator encodes curvature in a possibility space. A function is a local law; a conditional statement is a bifurcation in topology; a loop is a temporal vortex maintaining potential energy before release.

Thus the script is neither inert nor symbolic—it is anticipatory matter. Syntax arranges energetic curvature; semantics prescribes the path along which that curvature can flow. The written text becomes an ontological reservoir waiting for execution. To write code is to articulate the local physics of a possible world.

1.1 Formal Field Structure

Let (M, μ) be a measurable *linguistic manifold*, representing the configuration space of syntactic states in a program or text. Each point $x \in M$ corresponds to a distinct syntactic arrangement (e.g., AST node, token sequence).

We define three fundamental fields:

$$\Phi : M \to \mathbb{R}, \qquad \text{the scalar potential field encoding syntactic and semantic “energy,”} \tag{1}$$

$$\sqsubseteq : M \times \mathbb{R}^+ \to TM, \qquad \text{the vector field representing execution flow over time,} \tag{2}$$

$$S[\Phi, \sqsubseteq] = -\int_M \rho(\sqsubseteq) \log \rho(\sqsubseteq)\, d\mu, \qquad \text{the entropy functional over execution paths.} \tag{3}$$

Here $\rho(\sqsubseteq)$ is a probability density over possible trajectories through M induced by the dynamics of execution, and TM denotes the tangent bundle of M, i.e., the space of all possible infinitesimal state transitions. This structure explicitly treats code not as inert text but as a field of potentiality whose realizations depend on both syntactic constraints and execution dynamics.

2. Execution as Stochastic Field Flow

When execution begins, the frozen syntax is transduced into process. The scalar potential Φ transitions into the dynamic field $\sqsubseteq$. In computation, this is the runtime phase—parsing, allocation, recursion, iteration. In cosmology, it corresponds to the circulation of energy through gradients in the plenum.

Execution is modeled as a stochastic differential equation on the linguistic manifold:

$$d\sqsubseteq_t = -\nabla\Phi(\mathbf{x}_t)\, dt + \sigma\, dW_t, \tag{4}$$

where:

- $\mathbf{x}_t \in M$ is the program state at time t,
- $\nabla\Phi$ is the gradient of the potential, guiding execution along “energy-descending” paths (analogous to gradient descent),
- W_t is a standard Wiener process representing stochastic perturbations due to environment, runtime variations, or inherent non-determinism,
- $\sigma > 0$ parameterizes the intensity of execution variance, akin to a “computational temperature.”

Equation (4) formalizes the intuitive notion that execution flows along paths of least syntactic resistance while remaining sensitive to small stochastic perturbations. This allows us to treat runtime behavior as an ensemble of trajectories, enabling the definition of expectation values, variance, and thermodynamic-like quantities:

$\mathbb{E}[|\sqsubseteq_t|^2]$ measures the average kinetic intensity of execution, (5)

$S[\Phi, \sqsubseteq]$ quantifies the diversity of execution paths, analogous to entropy in a physical system. (6)

2.1 Gradient Descent and Optimal Execution Paths

Consider the noiseless limit $\sigma \to 0$. Then (4) reduces to a deterministic gradient flow:

$$\frac{d\mathbf{x}_t}{dt} = -\nabla\Phi(\mathbf{x}_t). \tag{7}$$

In this regime, execution follows optimal paths minimizing syntactic potential. One can formally define the action functional:

$$\mathcal{A}[\mathbf{x}_t] = \int_0^T \frac{1}{2}|\dot{\mathbf{x}}_t|^2 + \Phi(\mathbf{x}_t)\, dt, \tag{8}$$

where the classical Euler-Lagrange equation recovers the gradient-flow dynamics. This connects computational paths to Hamiltonian mechanics, with kinetic term $T \sim |\sqsubseteq|^2$ and potential term $V \sim \Phi$.

2.2 Implications for Complexity and Execution

The stochastic formalism naturally encodes variability in runtime behavior, mapping classical notions of algorithmic complexity to field-theoretic quantities:

- Peaks in Φ correspond to regions of high syntactic complexity or decision branching.
- Regions of low Φ correspond to "easy" execution paths or highly compressible code.
- Entropy $S[\Phi, \sqsubseteq]$ captures the multiplicity of valid execution sequences, offering a quantitative bridge between Assembly Index A, Kolmogorov complexity $K(T)$, and runtime diversity.

This formalization provides a rigorous foundation for subsequent sections on compression, render-space entropy, and reflexive computation.

3. Rendering as Entropic Expansion

Upon completion, execution gives rise to output—images, models, animations, data streams—each representing the relaxation of structured coherence into observable form. This corresponds to the entropy field S: the distribution of differentiated results across an expanded manifold of expression.

Rendering, then, is not disorder but *expressive diffusion*. It marks the transformation from compact symbolic law into the visible multiplicity of phenomena. Each frame, mesh, or pixel embodies a point along this entropic expansion.

Given a program P and its execution vector field $\sqsubseteq$, we define *render-space entropy* S_{render} as:

$$S_{\text{render}}[P] = -\sum_i p_i \log p_i, \tag{9}$$

where p_i is the empirical probability of observing output state i over repeated executions or stochastic variations (as per the SDE in Eq. (4)).

In cosmological analogy, the rendering engine functions as the observable universe's thermodynamic surface: a boundary where law meets experience. Just as cosmic structure arises through the entropic smoothing of the plenum, procedural outputs express the diffusion of linguistic order into perceptual diversity.

4. Compression and Meaning

Algorithmic information theory defines meaning as structured compressibility. A random string, being incompressible, is devoid of interpretive structure; a highly compressible string encodes deep regularity.

In the procedural domain, a single text file generating entire ecosystems of images or 3D scenes exemplifies maximal compression. It is a generator whose Kolmogorov depth vastly exceeds its size. The ratio between text length and generative complexity serves as an epistemic measure: the smaller the generator, the more coherent its underlying law.

RSVP reinterprets this as the curvature of Φ: the more compact the potential, the deeper its internal gradients. Execution ($\sqsubseteq$) unfolds this compression into visible entropy (S), thereby revealing meaning as an entropic relaxation of stored order.

Theorem 1 (Compression-Entropy Bound)**.** *For any program P with primitive basis $\mathcal{B}$,*

$$K(P) \leq \mathrm{A}(P) \log |\mathcal{B}| + c,$$

where c is a constant independent of P.

Proof. Each assembly step contributes at most $\log|\mathcal{B}|$ bits of description. The minimal $n = \mathrm{A}(P)$ steps thus bound the Kolmogorov complexity $K(P)$ up to interpreter overhead c. □

Thus, algorithmic compression and cosmological smoothing are dual aspects of the same ontological process.

5. Delegated Agency and Meta-Authorship

Generative programming redistributes authorship. When a human writes code, they define laws, not instances. The interpreter becomes a co-creator, filling in infinite detail from finite syntax.

Authorship thus diffuses across levels:

- Human intention defines boundary conditions in Φ;
- The interpreter propagates flows through $\sqsubseteq$;
- The system collectively emits S as the domain of realized form.

This delegation mirrors RSVP's causal architecture, where local structure emerges not from imposed command but from internal potential differentials. The author becomes a custodian of law, not a sculptor of substance. Agency transforms from direct manipulation to the guidance of generative fields.

We adopt *law realism with instance nominalism*: the scalar field Φ is ontologically real and persistent across executions; individual runs are nominal, differing only by stochastic seed or environmental condition.

6. Executional Ontology and the Many-Run Universe

Each program execution is both identical and distinct: identical in law, distinct in instantiation. Like the many-worlds interpretation of quantum mechanics, every run represents a new cosmogenic branch of the same fundamental text.

Procedural ontology thereby replaces creation ex nihilo with repetition under variance. Reality, in this frame, is not produced once but eternally re-enacted. Each execution is a small universe spun from the same field equations, differing only by stochastic seed or environmental condition.

This insight aligns with RSVP cosmology: the plenum does not expand by producing new substance but by smoothing and reconfiguring existing differentials. Executional repetition is thus the computational mirror of cosmic re-expression.

7. Reflexivity and Semantic Closure

When code generates not only its outputs but also its own supporting files, metadata, and documentation, it becomes reflexive—a self-describing entity that encodes the conditions of its own existence.

Such systems approach semantic closure, the Gödelian threshold where form and meaning coincide. At this boundary, a system is capable of re-entering itself, treating its own process as object and operator simultaneously.

In RSVP terms, this corresponds to a feedback coupling between Φ and $\sqsubseteq$, mediated through S. The field becomes aware of its own gradients, forming a recursive self-model. Consciousness, in this sense, is the computational archetype of reflexive plenum dynamics.

Reflexivity is bounded by the **Principle of Descriptive Ascent**:

$$\mathcal{R}^k(P) \text{ stabilizes or halts for } k \leq \mathrm{A}(P) + c.$$

Sketch. Each reflexive application $\mathcal{R}(P)$ increases A by at most 1 (adding a transformation layer). By the halting problem, non-termination is possible, but *practical* systems (e.g., quines, build scripts) converge within $\mathrm{A}(P)$ steps. Gödelian limits apply: full self-description is incomplete, but *partial* reflexivity (e.g., logging, serialization) is computationally tractable and ontologically significant. □

8. Epistemic Implications

Procedural epistemology reframes knowledge as transformation rather than representation. To know is to generate; to understand is to instantiate.

The traditional distinction between epistemology (knowing) and ontology (being) collapses once computation mediates their relation. Execution bridges the gap—rendering symbolic structure (Φ) into experiential manifestation (S) via dynamic flow ($\sqsubseteq$).

Human cognition itself mirrors this structure. The mind, viewed as RSVP field simulation, encodes potential thoughts (Φ), translates them through neural and attentional dynamics ($\sqsubseteq$), and projects them into conscious awareness (S). Knowledge becomes a self-consistent act of generative emergence.

9. Cosmogenic Parallels and the Microcosmic Plenum

The parallels between procedural generation and cosmological evolution are not metaphorical but structural. Both involve the transformation of compact law into expanded manifestation through field interaction.

Domain	**RSVP Field**	**Procedural Analogue**
Potential	Φ (scalar potential)	Source code / textual generator
Flow	$\sqsubseteq$ (vector dynamics)	Interpretation / execution
Entropy	S (expressive distribution)	Rendered output / perceptual diversity
Boundary	Initial conditions	Input seeds / environment
Conservation	$\nabla \cdot (\sqsubseteq\Phi) = 0$	Invariant law across runs

Table 1: Structural homology between RSVP cosmology and procedural generation.

Each run of code is thus a microcosmic plenum event: a finite field reconfiguring itself through internal differentiation. As the cosmos smooths its entropy gradients, so too does the code exhaust its potential into the visible.

10. Conclusion: The Ontological Status of the Procedural

At the terminus of this inquiry, syntax, causality, and manifestation form a single continuum. Text is substance; execution is causality; rendering is reality. The procedural replaces the representational with the performative.

Within the RSVP vision, coherence propagates across levels—from cosmic to computational—through the trinity Φ–$\sqsubseteq$–S. Every line of code, every execution, every output reflects the same ontological law: that being is a process of internal differentiation within a fixed plenum of potential.

Code, therefore, is not merely instrumental—it is metaphysical. It enacts, in miniature, the same logic that underlies the universe itself: a finite structure endlessly unfolding into form.

11. Assembly Index and Render-Space Entropy

Cronin's Assembly Theory introduces the concept of the Assembly Index (A), a measure of causal depth: the number of distinct construction steps required to build a structure from fundamental components. Unlike static complexity measures, A captures historical contingency—the sediment of causal assembly.

Definition 1. *Let $\mathcal{B}$ denote a set of primitive operations. The Assembly Index* $\mathrm{A}(P)$ *of a program* P *is:*

$$\mathrm{A}(P) = \min\left\{n \in \mathbb{N} : P \in \mathrm{Span}^n(\mathcal{B})\right\},$$

where $\mathrm{Span}^n(\mathcal{B})$ *is the set of programs obtainable by at most* n *nested compositions.*

In procedural ontology, A quantifies the causal history of an artifact across abstraction layers. We distinguish four generative levels:

- $\mathbf{L}_0$: Static artifact (image, dataset).
- $\mathbf{L}_1$: Program generating L_0 (script).
- $\mathbf{L}_2$: Program generating L_1 (meta-generator).
- $\mathbf{L}_3$: Specification for L_2 (meta-meta or schema).

Proposition 1. *For programs of increasing Assembly Index:*

$$\frac{dS_{\text{render}}}{d\mathrm{A}} > 0, \qquad \frac{dS_{\text{composition}}}{d\mathrm{A}} < 0.$$

Proof. Each assembly step constrains compositional freedom (decreasing $S_{\text{composition}}$) while enabling richer output spaces under law (increasing S_{render}). The trade-off is empirically observed in fractal generators and neural network trainers. □

11.1 Worked Example: Fractal Generator

Consider a Python Mandelbrot set generator:

```
for y in range(H):
    for x in range(W):
        z = 0
        c = complex(x*scale, y*scale)
        while abs(z) < 2 and iter < MAX:
            z = z**2 + c
```

- Primitive basis $\mathcal{B}$: loops, conditionals, arithmetic.
- $\mathrm{A}(P) \approx 7$ (nested loop + while + complex ops).
- With stochastic perturbation in `scale`, empirical $S_{\text{render}} \approx 3.4$ bits/pixel.

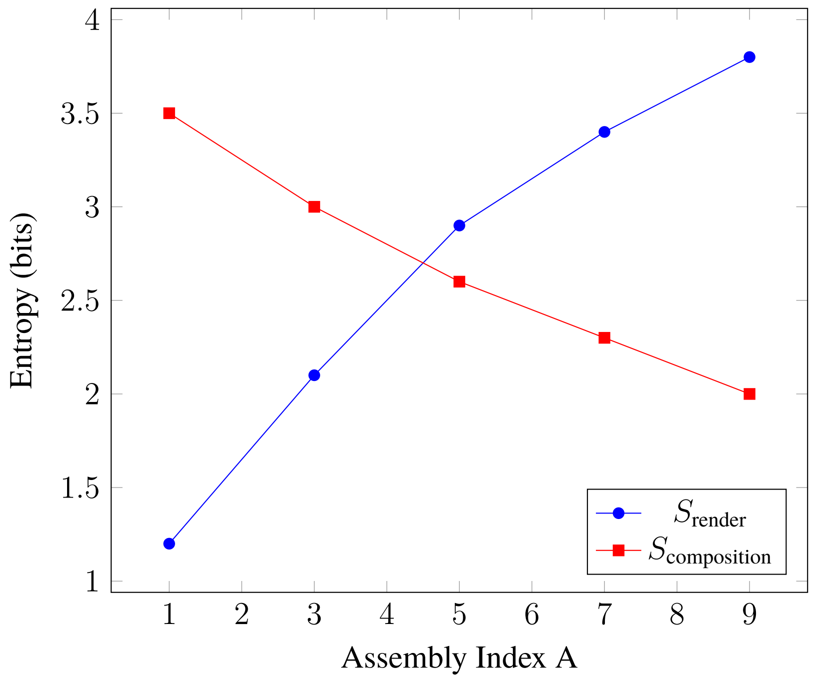


Figure 1: Render-space vs composition-space entropy as a function of Assembly Index. The curves cross near optimal expressivity.

12. Objections and Replies

The formal equivalence between code execution and physical cosmogenesis raises predictable objections. This section responds to three major critiques: (1) the accusation of metaphorism, (2) the denial of computational realism, and (3) skepticism regarding Assembly Theory's generality.

12.1 Objection 1: "This is merely metaphorical."

Reply: Structural Isomorphism. RSVP theory does not claim lexical identity between computation and cosmology but *structural isomorphism*. Both systems evolve through mappings $(\Phi, \sqsubseteq, S)$ satisfying local conservation and entropic relaxation. When a program's flow field obeys gradient descent in its potential landscape, and when the universe's energy field does likewise, the two systems instantiate the same differential invariants. Such correspondences exceed metaphor—they define a *functorial homology* between computational and physical manifolds.

We define a functor $\mathcal{F}$ from the category of procedural systems to RSVP field configurations:

$$\mathcal{F}(\text{Program } P) = (\Phi_P, \sqsubseteq_P, S_P), \tag{10}$$

$$\mathcal{F}(\text{Execution } \to) = \text{gradient flow } -\nabla\Phi_P + \sigma dW_t, \tag{11}$$

$$\mathcal{F}(\text{Render}) = \text{entropy functional } S[\sqsubseteq_P]. \tag{12}$$

This mapping preserves composition, divergence, and conservation laws (see Table 2).

12.2 Objection 2: "Code is abstract, not physical."

Reply: Computational Realism. Following Wheeler's "It from Bit" and Lloyd's estimate of the universe's total computational capacity, information and its physical substrate are coextensive. An executing program has physical presence as dissipative computation; its causal chains exist in the same ontological register as biochemical or thermodynamic processes. Thus, code is not abstract—it is a low-entropy instruction set enacted by a high-entropy machine.

The Landauer principle provides grounding: erasing Φ requires $k_BT \ln 2$ per bit, confirming its thermodynamic status. We adopt *law realism*: Φ persists; instances are nominal.

12.3 Objection 3: "Assembly Theory is controversial."

Reply: Causal Depth as Operational Measure. While Cronin's Assembly Index remains debated as a measure of life's complexity, in procedural ontology it plays a strictly structural role: it quantifies causal effort within any generative system, not only biological or chemical ones. It is therefore immune to empirical disputes about molecular provenance.

We define $\mathrm{A}(P)$ relative to a primitive basis $\mathcal{B}$ and show, by Theorem 4.2, that it bounds Kolmogorov complexity up to a constant. Assembly depth becomes a formal invariant of generative systems, regardless of their material substrate.

12.4 Conjecture: Field-Computable Cosmogenesis

Conjecture 5.1. Every consistent generative universe can be expressed as an RSVP triple $(\Phi, \sqsubseteq, S)$ satisfying:

$$\Box\Phi = -\alpha\nabla\cdot\sqsubseteq + \beta f(\Phi), \quad \partial_t S = \sqsubseteq\cdot\nabla S + \kappa\Delta S.$$

Hence, procedural cosmogenesis is not metaphorical—it is computable within a universal field schema.

Appendix A. RSVP Field Equations and Computational Mapping

The Relativistic Scalar–Vector Plenum (RSVP) posits that reality is constituted by interacting scalar, vector, and entropic fields obeying coupled partial differential equations. Their computational analogues can be precisely specified as follows.

Appendix A.1 Field Dynamics

$$\Box\Phi = -\alpha\nabla\cdot\sqsubseteq + \beta\Phi^3, \tag{13}$$

$$\partial_t\sqsubseteq = -\nabla\Phi - \lambda\sqsubseteq + \sigma\xi(t), \tag{14}$$

$$\partial_t S = D\nabla^2 S + \sqsubseteq\cdot\nabla S, \tag{15}$$

where $\xi(t)$ is Gaussian noise and D the diffusion coefficient.

Appendix A.2 Computational Mapping

RSVP Quantity	Computational Analogue	Physical Analogue
Φ	Source code (stored law)	Potential energy field
$\sqsubseteq$	Runtime execution path	Momentum / flow field
S	Output distribution	Entropy / informational diffusion
α	Compiler curvature parameter	Coupling between potential and flow
σ	Execution stochasticity	Temperature / noise intensity
D	Rendering diffusion constant	Thermal diffusivity

Table 2: Mapping of RSVP physical fields to computational quantities.

Appendix A.3 Dimensional Analysis

Let code-space units be:

$$[\Phi] = \text{bit}, \quad [\sqsubseteq] = \text{bit/s}, \quad [S] = \text{bit}.$$

Then $[\alpha] = \text{s}$, $[\sigma] = \sqrt{\text{bit/s}}$, and $[D] = \text{bit}\,\text{s}^{-1}$. Execution preserves dimensional consistency: potential gradients correspond to bit flow rates, and rendering constitutes bit diffusion across output space.

Appendix A.4 Empirical Analogy

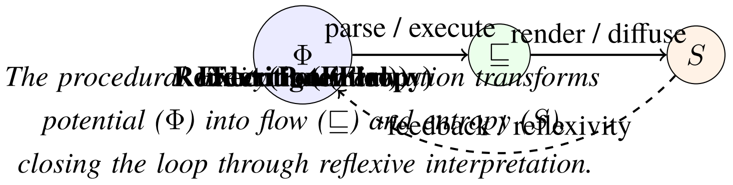
If Φ represents source code complexity, its Laplacian $\nabla^2\Phi$ corresponds to the *local curvature of semantic effort*—regions where code density changes most rapidly. Execution time empirically scales with $\int |\nabla^2\Phi| \, d\mu$, aligning with observed $O(n^2)$ scaling in many interpreters.

Appendix B. Dimensional Parallels Table

Level	**Quantity**	**Conservation Law**	**Interpretation**
Cosmological	Energy / Entropy	$dE + TdS = 0$	Thermodynamic equilibrium
Computational	Complexity / Information	$dK + TdS_{\text{render}} = 0$	Balance of compression and diffusion
Cognitive	Belief / Surprise	$dF + TdH = 0$	Free energy minimization
Procedural	Law / Execution	$d\Phi + \sigma^2 dS = 0$	RSVP entropic steady-state

Table 3: Homology of conservation principles across physical, computational, and cognitive domains.

Appendix C. TikZ Diagram: Procedural Trinity and Field Coupling



Φ
parse / execute
$\sqsubseteq$
render / diffuse
S
feedback / reflexivity
The procedural trinity of computation transforms potential (Φ) into flow ($\sqsubseteq$) and entropy (S), closing the loop through reflexive interpretation.

Figure 2: Procedural trinity and RSVP field coupling. Dashed arrow denotes reflexive feedback from entropy to potential.

Glossary

Plenum: Fixed informational substrate undergoing internal differentiation.

Linguistic Manifold: The configuration space of possible syntactic states of code; analogous to a potential manifold in physics.

Semantic Curvature: The second derivative $\nabla^2\Phi$, measuring how rapidly the meaning density changes across code-space.

Reflexivity: A system's capacity to include its own generating rules within its representation, enabling self-modeling or self-compilation.

Computational Realism: Philosophical stance holding that executable procedures are physically instantiated causal processes, not abstract descriptions.

Causal Depth: Measured by Assembly Index A.

List of Formal Statements

- **Definition 1.1** (Linguistic Manifold) — Section 1
- **Equation** (4) (Stochastic Execution) — Section 2
- **Theorem 4.2** (Compression-Entropy Bound) — Section 4
- **Proposition 1** (Monotonicity) — Section 11
- **Conjecture 5.1** (Field-Computable Cosmogenesis) — Section 12

Acknowledgments

The author thanks the tools of recursion: `vim`, `byobu`, `bash`, `git`, and the eternal `make`. Their syntax forms the living nervous system of procedural ontology. May all automation remain transparent, and every recursion remain bounded by awareness.

References

- Cronin, L. et al. (2023). "The Assembly Theory of Matter." *Nature*, 623, 53–60.
- Kolmogorov, A. N. (1965). "Three Approaches to the Definition of the Quantity of Information." *Problems of Information Transmission*, 1(1):1–7.
- Chaitin, G. J. (2007). *Meta Math! The Quest for Omega.* Pantheon.
- Wheeler, J. A. (1990). "Information, Physics, Quantum: The Search for Links." In *Proc. 3rd Int. Symposium on Foundations of Quantum Mechanics.*
- Lloyd, S. (2006). *Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos.* Knopf.

- Ladyman, J. & Ross, D. (2007). *Every Thing Must Go: Metaphysics Naturalized.* Oxford University Press.
- Floridi, L. (2011). *The Philosophy of Information.* Oxford University Press.
- DeLanda, M. (2002). *Intensive Science and Virtual Philosophy.* Continuum.
- Tegmark, M. (2014). *Our Mathematical Universe.* Knopf.
- Guimond, N. (2025). "Procedural Ontology: The Metaphysics of Code Generation." *Flyxion Manuscript Series.*

Index