C Anthony Risinger

ACCELERATING the VIRTUAL EXPERIENCE via COMPACTIFICATION of the METAVERSE

Abstract

Spatial computing stands at the heart of immersive, competitive experiences. When reality yields to fast-paced virtual arenas, every millisecond can clutch the win, and every dropped frame can shatter the illusion. Widespread spatial partitioning of diverse virtual realities (VR) using the topologically close-packed ***A15 phase structure*** *(β–W) promises to thread new, scalable dimensions of isotropic order into the fabric of the metaverse itself.*

This research advances the development of a trans-metaversal coordination space. Latency-sensitive, full-body VR experiences—such as online multiplayer tournaments—face significant challenges under current infrastructure, where IEEE 754 floating-point representation errors introduce subtle inconsistencies across different hardware and software environments, impacting responsiveness and fairness [5]. These inconsistencies undermine determinism in distributed systems, and no standard float implementation guarantees identical results across platforms. The proposed solution enforces cross-platform consistency by mapping float positions to deterministic integer identifiers within a crystallographic lattice, providing stable, reproducible behavior.

Full immersion demands higher fidelity at greater speeds; simply relaying kinematic coordinates for complex interactions (e.g., a 5-on-5 full-body VR match) can result in aggregate data rates exceeding $100\,\mathrm{kbit\,s^{-1}}$ *per user.*

The proposed solution addresses these issues by mapping three-dimensional (3D) floating-point coordinates—and their often-unused dynamic range—to the nearest A15-encoded integer representation. By embedding A15 within structured partitioning geometries like the Weaire–Phelan honeycomb or the Tetrastix Prism prism and operating within rigorously defined numerically stable scaling regimes, the framework discretizes 3D space while guaranteeing deterministic fidelity relative to its chosen grid resolution. The result is a higher-order representation that is packable, stackable, significantly smaller (often reducing data size by 50% or more compared to standard 32 bit *float vectors), minimizes directional aliasing, maintains compatibility with global measurement standards, and ensures verifiable consistency—crucial for responsive, fair, and trustworthy virtual experiences.*

In a quest for instant replay and deterministic fidelity, aligning virtualities with A15 ensures every bit of space is part of the living, responsive experience.

Keywords *A15*, Beta-Tungsten, Competitive Virtual Reality, Coordinate Encoding, Crystallography, Determinism, Deterministic Simulation, Infima Labs, Lattice Encoding, Metaverse, Numeric Stability, Pm-3n, Spatial Computing, Spatial Encoding, Spatial Hashing, Tetrastix Prism, Weaire–Phelan Honeycomb

Executive Summary

The *A15* encoding framework addresses a fundamental challenge in spatial computing: achieving deterministic, bandwidth-efficient coordinate representation across heterogeneous systems. By mapping continuous 3D coordinates onto a crystallographically-inspired integer lattice, this approach establishes precise spatial representation within the inherently imprecise world of floating-point arithmetic.

At its core, the framework leverages the unique properties of the *A15* phase structure (β–W), a crystal structure with exceptional isotropy and binary-compatible coordinates. Through a carefully designed multi-stage scaling pipeline that prioritizes integer representation during geometric construction, it guarantees bit-identical spatial encoding when operating within defined stable scaling regimes ($\epsilon_\Delta = 0$). This eliminates the representation errors that plague floating-point coordinates, ensuring consistent behavior across different hardware, operating systems, and compiler implementations.

For networked virtual environments—particularly competitive VR, collaborative simulations, and distributed computing—this translates directly to several practical benefits:

- **Network Efficiency:** Shrinks coordinate transmission size by 50% compared to raw float vectors, enabling higher update rates within limited bandwidth.
- **Deterministic Guarantee:** Enables verifiable event sequences, accurate replays, and fair competition across heterogeneous client devices.
- **Geometric Fidelity:** Preserves spatial relationships with minimal directional bias owing to *A15*'s high coordination (13.5) and near-icosahedral local ordering.

The framework includes robust tools for verifying numerical stability through histogram analysis, enabling developers to confidently implement deterministic spatial systems while maintaining flexibility in coordinate scale selection appropriate to their application domain.

1. Introduction: Floats, Fairness, and the A15 Foundation

Binary floating-point numbers, commonly known as "floats" and largely governed by the IEEE 754 standard [19], are the computational workhorses for representing real numbers. They offer a vast dynamic range essential for graphics and science, yet this reach often comes at the cost of precision. Floats are notorious for providing *approximately correct* answers[1], representing most values inexactly. These tiny, fundamental representation errors, inherent in their structure (Equation (1)), can accumulate and interact, particularly challenging the deterministic consistency required for networked virtual reality (VR) or distributed simulations. This research confronts these challenges head-on, proposing an alternative spatial representation rooted in the unique

[1] Floats are abundant in software yet maddeningly fickle; among the first one-hundred simple reciprocals ($1/n$), ninety-three require approximation in standard binary float formats [14].

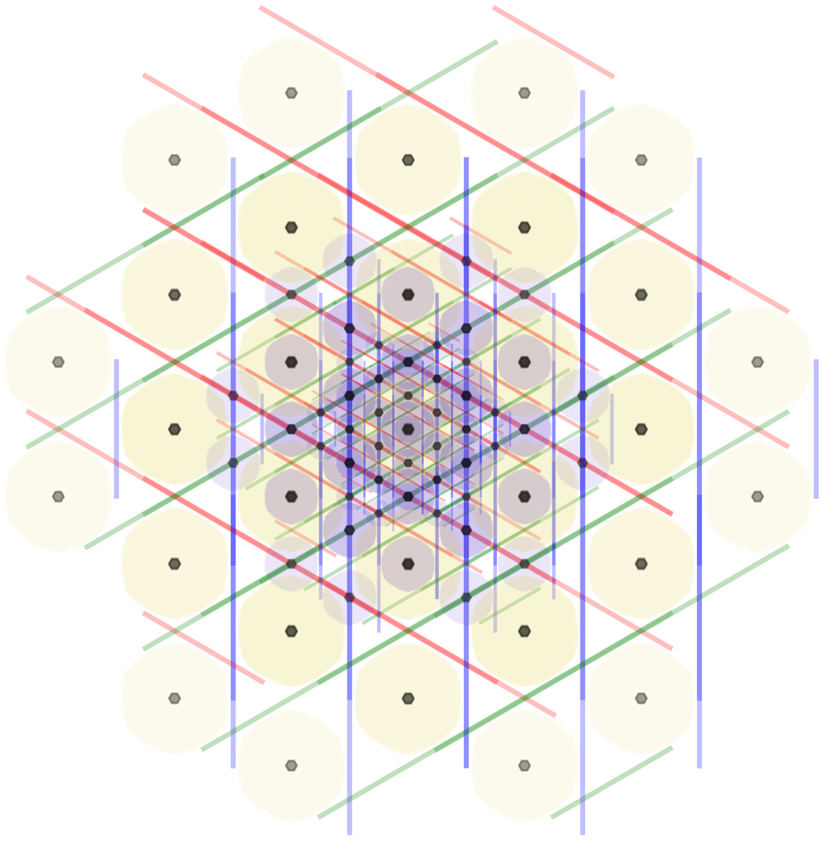


Figure 1. Illustration of the *A15* structure (β–W) at three different scales, generated using A15.py (fig-intro.png.txt). The visualization highlights the constituent pyritohedral and tetradecahedral elements, demonstrating how the structure maintains its fundamental $Pm\bar{3}n$ space group symmetry while revealing increasing detail at finer resolutions.

crystallographic properties of the *A15* phase structure (β–W, space group $Pm\bar{3}n$, No. 223 [1]), as illustrated in Figure 1. Furthermore, establishing a robust foundation for spatial representation with inherent numerical stability and structural consistency extends beyond immediate interactive fidelity. Such a foundation offers compelling advantages for long-term data archival, application-agnostic interoperability across diverse platforms, and the efficient decomposition and distribution of spatial computations in parallel and edge computing environments.

1.1. The Challenge of Floating-Point Precision

Standard normalized binary floating-point numbers derive their value from three components: a sign bit, a biased exponent, and a significand (mantissa). Conceptually[2], their value relates to:

$$\text{value} = (-1)^{\text{sign}} \times (1 + \text{significand}) \times 2^{\text{exponent}} \tag{1}$$

The fixed-size significand offers constant *relative* precision within an exponent range, while the exponent scales the value logarithmically. This design excels at exactly representing powers of two and fractions whose denominators are solely powers of two, but only a finite subset thereof. Consequently, most decimal values and even many simple fractions (e.g., 1/10) are merely approximated [14].

While seemingly minuscule, these representation errors can compound, critically affecting reproducibility. In distributed systems like multiplayer games or collaborative VR, this becomes paramount. Identical logical inputs processed on different hardware architectures, operating systems, or even with different compiler optimizations can yield subtly divergent floating-point results. This variance breaks simulation consistency, undermines fairness in competition [5], complicates debugging and verification, and fundamentally opposes the requirement for determinism.

These challenges manifest most acutely in latency-sensitive, spatially-complex applications where:

- Small coordinate discrepancies can produce visible jitter or desynchronization
- State consistency must be maintained across heterogeneous client devices
- Replays and verifiable event sequences require bit-exact reproduction
- Network bandwidth constraints demand efficient coordinate representation

Yet, widespread hardware acceleration makes IEEE 754 formats—primarily binary32 (32 bit, single-precision) and binary64 (64 bit, double-precision)—ubiquitous in modern CPUs and GPUs. Binary32, in particular, remains vital for performance-sensitive applications like game development and VR, establishing a key baseline for efficiency comparisons. This work, therefore, accepts the practical necessity of floats but seeks to structure their use, mitigating risks to determinism via the disciplined partitioning and verifiable scaling offered by the *A15* framework detailed in Section 2.

1.2. The A15 Phase Structure: A Crystallographic Foundation

Effectively partitioning virtual space requires a measure of geometric "fairness"—a balance between **isometry** (preserving distances between corresponding points) and **isotropy** (uniformity of properties across different directions). Discretizing continuous space onto a lattice while preserving both ensures that spatial relationships remain consistent and free from directional bias. While regular lattices achieve perfect isometry through translational symmetry, attaining high isotropy is constrained

[2]This simplified view omits details such as exponent bias, subnormal numbers, infinities, and NaNs, rigorously defined in [19] but not central to the core issue of approximation.

by fundamental geometric principles. The continuous rotational symmetry group in 3D, $SO(3)$, implies that highly isotropic structures should appear locally spherical. However, the imposition of discrete translational symmetry restricts the allowed rotational symmetries to only 1-, 2-, 3-, 4-, and 6-fold axes. This is the essence of the **crystallographic restriction theorem** [2], which explicitly forbids the global 5-fold symmetry characteristic of highly isotropic polyhedra like the icosahedron (associated with the I_h point group symmetry [7]) in periodic lattices.

The *A15* structure ($Pm\bar{3}n$, No. 223 [1]) navigates this restriction with notable elegance. It is based on the simple primitive cubic (cP) Bravais lattice but incorporates a complex basis (or motif) containing two distinct types of crystallographic sites within its conventional unit cell [12, 13], as illustrated in Figure 2:

- 25% are C12 sites: 12-coordinated, occupying Wyckoff position 2a (e.g., at fractional coordinates $(0, 0, 0)$ and $(1/2, 1/2, 1/2)$). Their local coordination environment exhibits characteristics related to pyritohedral symmetry.
- 75% are C14 sites: 14-coordinated, occupying Wyckoff position 6d (e.g., at $(1/4, 1/2, 0)$ and its symmetry equivalents [1]). Their local environment resembles a tetradecahedron (a 14-faced polyhedron).

This intricate arrangement results in a high average coordination number, calculated as $Z_{avg} = (0.25 \times 12) + (0.75 \times 14) = 13.5$, suggesting a densely connected local structure conducive to efficient packing. The local symmetry around these sites is described by the crystallographic point group T_h ($m\bar{3}$, order 24). Significantly, T_h is a maximal subgroup common to both the full cubic symmetry group O_h (order 48) and the non-crystallographic icosahedral group I_h (order 120) [8]. While lacking the forbidden 5-fold axes of I_h, T_h retains key 3-fold rotational symmetries present in icosahedral structures. This unique symmetry allows the *A15* structure to incorporate significant near-icosahedral local ordering, boosting local isotropy considerably compared to simpler cubic structures, all while maintaining the long-range periodicity required by crystallography. The structure also features alternating left- and right-handed local environments around the 6d sites, adding further complexity relevant to implementation (Section 3.3.6).

Crucially for this work, the canonical definition of the *A15* structure relies on fractional coordinates inherently suitable for binary representation: 0, 1/4, and 1/2. These values are exactly representable in base-2 systems. When these fractional coordinates are mapped onto an integer lattice (conceptually, by scaling the unit cell coordinates by 4, aligning with the derivation of the 96-unit baseline in Section 2.3.3), two characteristic squared site-to-site distances emerge within this integer framework: $d^2 = 4$ (yielding distance 2) and $d^2 = 5$ (yielding distance $\sqrt{5}$). The presence of $\sqrt{5}$, the hypotenuse of a fundamental 2:1 right triangle, connects the structure's geometry directly to the golden ratio $\phi = (1 + \sqrt{5})/2$, further reflecting the embedded near-icosahedral characteristics. This intrinsic numerical simplicity and compatibility with base-2 representation ensures that the *A15* structure can be defined and scaled with precision, forming a robust foundation for the numerically stable encoding framework developed herein (Section 2.4).

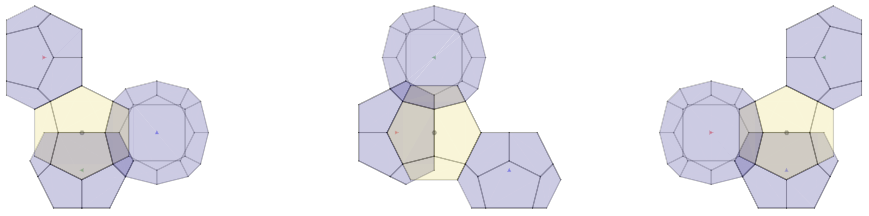


Figure 2. Components illustrating the local arrangement within a left-handed $^1/_2$ unit cell fragment of the *A15* structure. Shows C12 (Wyckoff 2a, center of pyritohedral environment) and C14 (Wyckoff 6d, center of tetradecahedral environment) sites demonstrating the local complexity within the overall $Pm\bar{3}n$ symmetry. Generated via A15.py (fig-cell2.png.txt).

1.3. Local Discretization Methods: WPH and TSP

While the *A15* structure defines the target lattice points for coordinate encoding, mapping arbitrary continuous coordinates from a virtual environment onto these discrete identifiers requires a well-defined *local discretization method.* This method effectively defines the boundaries of the region (the "cell") surrounding each *A15* site; any continuous point falling within a given cell is quantized or mapped to that specific site's identifier. This framework primarily considers two such methods, both intrinsically linked to the *A15* structure, sharing its $Pm\bar{3}n$ space group symmetry, and implemented within the accompanying A15.py software (Figure 3):

Weaire–Phelan Honeycomb (WPH) Geometry: This research utilizes the geometric polyhedral form of the Weaire–Phelan structure as a partitioning method. It divides space using cells of two distinct shapes: a 12-faced pyritohedron and a 14-faced tetradecahedron, arranged in a precise 1:3 ratio [33]. Significantly, these flat-faced polyhedra constitute the Voronoi decomposition for the A15 lattice points; the pyritohedron is the Voronoi cell for the C12 (Wyckoff 2a) site, and the tetradecahedron is the Voronoi cell for the C14 (Wyckoff 6d) site. This relationship makes the Weaire–Phelan Honeycomb geometry a canonical choice for discretization, as each cell naturally defines the region of space closest to its corresponding A15 lattice site. These cell shapes correspond topologically to the local coordination environments of the C12 and C14 sites within the *A15* structure, respectively. This partitioning is notable for its high degree of isotropy, closely mirroring the symmetry characteristics of the underlying lattice. (It is important to distinguish this geometric, space-filling honeycomb from the related but distinct relaxed, minimal-surface structure which famously provided a counter-example to Kelvin's conjecture on foam partitioning [29, 21, 32].)

Tetrastix Prism (TSP) Geometry: A structurally simpler alternative partitioning method, activated within A15.py via the -stix configuration option. This method employs axis-aligned planar faces, effectively dividing space into cubic blocks centered on the *A15* lattice sites. While offering computational advantages for certain operations (such as point-in-cell tests) due to its simpler geometry, this approach generally exhibits lower isotropy compared to the Weaire–Phelan Honeycomb method.

Crucially, both the Weaire–Phelan Honeycomb and the Tetrastix Prism serve as interchangeable local discretization methods within this framework. They define the specific geometry used to quantize continuous space around each *A15* lattice site. The choice influences the precise shape of these local regions and boundaries, impacting factors like the resulting partition's isotropy and the computational cost of the quantization step itself. However, the fundamental coordinate identifier that is ultimately transmitted or stored remains based on the position within the underlying *A15* lattice, regardless of which local method is used for the mapping.

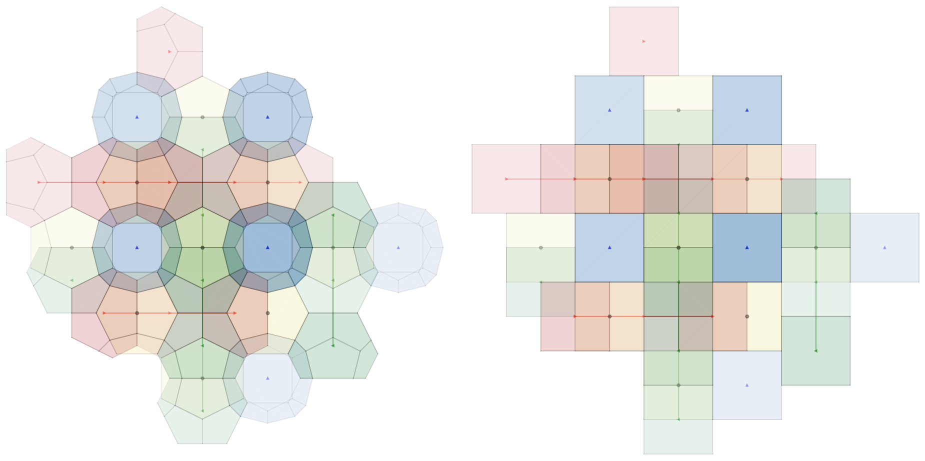


Figure 3. Visualization of local discretization methods related to the underlying *A15* structure. Left: The geometric polyhedral variant of the **Weaire–Phelan Honeycomb**, featuring pyritohedra (12 faces) and tetradecahedra (14 faces). Right: The simpler **Tetrastix Prism** partitioning using axis-aligned planar faces resulting in cubic blocks centered on *A15* sites. Both share the $Pm\bar{3}n$ space group symmetry. Generated using A15.py (fig-wp.png.txt and fig-ts.png.txt).

1.4. Clarification on Weaire-Phelan Variants

It is essential to emphasize that this research specifically utilizes the **geometric polyhedral form** of the Weaire-Phelan structure as a partitioning method for the A15 framework. This geometric variant consists of flat-faced polyhedra (pyritohedra and tetradecahedra) arranged in a precise 1:3 ratio, constituting the Voronoi decomposition for the A15 lattice points.

This geometric WPH variant should be explicitly distinguished from the related but distinct **minimal-surface foam approximation** of Weaire-Phelan, which is widely known in the physics literature for providing a counter-example to Kelvin's conjecture on foam partitioning [29, 21, 32]. The minimal-surface variant features curved interfaces minimizing surface area under specific physical constraints, resulting in subtle geometric differences from the polyhedral form used here.

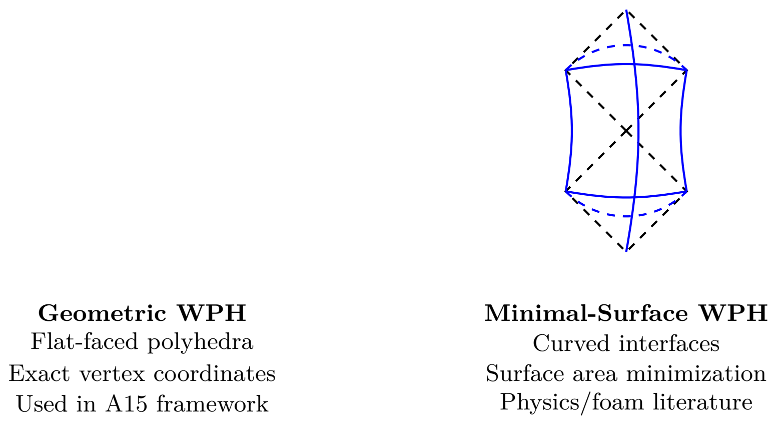


Figure 4. Conceptual comparison between the geometric polyhedral form of the Weaire-Phelan structure used in this framework (left) and the minimal-surface foam approximation known from physics literature (right). The geometric form uses flat-faced polyhedra with exact vertex coordinates, while the minimal-surface variant features curved interfaces that minimize surface area.

The practical implications of this distinction are significant for implementation:

- The geometric WPH used in this framework can be precisely represented using flat-faced polyhedra with exact vertex coordinates, facilitating deterministic computational processing.
- Point-in-cell tests against the geometric WPH can be implemented using standard computational geometry techniques for polyhedra, avoiding the complexity of curved surface representations.
- The structural rigidity of the geometric WPH ensures consistent partitioning regardless of cell contents or external influences, in contrast to physical foam structures that might deform based on pressure differentials.

In contrast, the minimal-surface WPH would require more complex surface representations, potentially introducing additional numerical challenges contradictory to the framework's determinism goals. Therefore, all references to the Weaire-Phelan Honeycomb (WPH) within this document specifically denote the geometric polyhedral form, not the minimal-surface variant from foam physics.

This clarification is particularly important because the minimal-surface WPH has gained broader recognition through its architectural application in structures like the Beijing National Aquatics Center (the ”Water Cube”) for the 2008 Olympics, potentially creating confusion about which variant is employed in this spatial encoding framework.

1.5. Distinction from Traditional Spatial Indexing

The *A15*-based partitioning approach presented here relates to, yet fundamentally differs from, traditional spatial indexing techniques commonly employed in databases, geographic information systems (GIS), and various domains of computer graphics (e.g., octrees [11], R-trees [16], k-d trees [3]; see [26] for a comprehensive overview). Conventional indexing methods typically utilize data-driven, adaptive strategies; their partitioning boundaries often adjust dynamically based on the distribution and density of existing spatial objects, with the primary goal of optimizing query performance (such as range searches or nearest neighbor lookups) for that specific dataset.

In stark contrast, the *A15* framework imposes a predetermined, regular, crystallographically-inspired structure onto the virtual space itself, largely independent of the dynamic content residing within it. This *structure-first* methodology prioritizes the creation of a universal, efficient, and geometrically sound fabric for spatial representation and encoding. The key distinctions are summarized in Table 1. The *A15* approach deliberately trades the dynamic query optimization focus of traditional indexing for significant advantages in predictability, communication efficiency (via compact identifiers that implicitly encode the regular structure), guaranteed numerical determinism (when using stable scaling regimes), and inherent geometric consistency derived from crystallographic symmetry—properties particularly valuable for networked virtual environments demanding shared spatial understanding, robustness against floating-point discrepancies, and efficient state synchronization.

1.6. Comparative Context: Other Symmetric Structures

The 230 crystallographic space groups categorize all possible ways to combine periodic translational symmetry (defined by the 14 Bravais lattices) with rotational and reflectional symmetries (defined by the 32 crystallographic point groups) in 3D space [1]. Situating the *A15* structure (space group $Pm\bar{3}n$, No. 223) within this framework highlights its distinctive characteristics relevant to packing and partitioning (Table 2).

As shown, *A15*'s $Pm\bar{3}n$ space group utilizes the primitive cubic (cP) Bravais lattice but applies the T_h point group symmetry (order 24). This group possesses lower overall symmetry than the full cubic O_h group (order 48) characteristic of the highest-symmetry simple cubic, body-centered cubic (BCC, cI), and face-centered cubic (FCC, cF) structures. However, as discussed (Section 1.2), the specific T_h symmetry is significant because it represents a maximal crystallographic subgroup linking cubic (O_h) and non-crystallographic icosahedral (I_h) symmetries [8]. This unique combination allows the *A15* structure to accommodate its complex basis with distinct C12 and C14 sites, enabling its exceptionally high average coordination number (13.5), which relates to efficient local packing and high connectivity.

When compared to common space-filling honeycombs and related sphere packings, the *A15* structure, particularly when coupled with the Weaire–Phelan Honeycomb partitioning method, offers further distinctions relevant to this framework:

vs. BCC-derived Structures: The body-centered cubic lattice (BCC, coordination number Z=8) and its dual, the bitruncated cubic honeycomb (Kelvin's structure [29]), have lower fundamental coordination than *A15* (13.5), suggesting a less densely connected local environment.

vs. FCC-derived Structures: The face-centered cubic lattice (FCC, coordination Z=12) represents the densest packing of identical spheres [6] and is related to the rhombic dodecahedral honeycomb. While achieving maximal coordination for identical points, *A15* surpasses this average by utilizing two distinct, efficiently arranged site types.

vs. the Weaire–Phelan Honeycomb: The geometric Weaire–Phelan Honeycomb, used as a local discretization method, shares the $Pm\bar{3}n$ space group and its cell types correspond topologically to the A15 site environments. This ensures the partitioning boundaries naturally align with the symmetries and neighborhood relationships of the underlying A15 encoding lattice, a unique synergy not offered by partitions derived from simpler lattices.

In summary, the unique combination offered by the *A15* structure—its high mean coordination, significant local isotropy via T_h symmetry, a direct relationship to the compatible Weaire–Phelan Honeycomb partitioning structure, and inherently binary-suitable base coordinates—makes it exceptionally well-suited for the deterministic, efficient, and geometrically sound spatial encoding framework proposed here.

1.7. Comprehensive Lattice Comparison

The selection of *A15* as the foundation for this framework was not arbitrary but resulted from a systematic evaluation of alternative crystallographic structures. Table 3 provides a comprehensive comparison of *A15* against common alternative lattices for spatial partitioning, highlighting the specific properties that make it exceptionally well-suited for deterministic spatial encoding.

Each property contributes significantly to the lattice's suitability for deterministic spatial encoding:

Coordination Number: The average number of nearest neighbors for each lattice point directly relates to the connectivity and local packing efficiency. *A15*'s exceptional mean coordination number (13.5) enables a more densely connected local structure than even the FCC lattice (12), often considered optimal for uniform spheres.

Isotropy: The uniformity of geometric properties across different directions affects how consistently the lattice represents spatial relationships regardless of orientation. While SC exhibits significant directional bias, both FCC and *A15* achieve high isotropy. However, *A15* maintains this isotropy specifically through its T_h point group symmetry, which incorporates key aspects of icosahedral ordering (Section 1.2).

Binary Compatibility: The natural alignment of the structure's coordinates with binary representation directly impacts numerical stability. *A15*'s basis site coordinates (involving only factors of 0, 1/4, and 1/2) are perfectly compatible with binary representation, unlike the irrational coordinates required for optimal FCC packing.

Cell Uniformity: While SC, BCC, and FCC consist of identical unit cells, *A15* incorporates two distinct site types (C12 at Wyckoff 2a and C14 at Wyckoff 6d) in a precise 1:3 ratio. This heterogeneity enables its exceptional mean coordination without sacrificing periodicity or computational tractability.

Complexity: Implementation complexity ranges from the straightforward SC to the more involved *A15*.

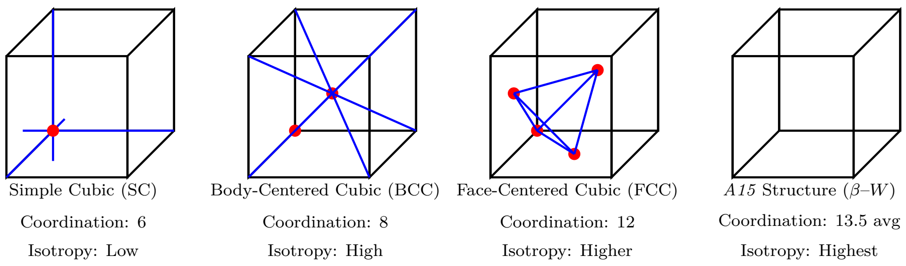


Figure 5. Comparison of crystal lattice structures for spatial partitioning. From left to right: Simple Cubic (SC) with coordination number 6, Body-Centered Cubic (BCC) with coordination number 8, Face-Centered Cubic (FCC) with coordination number 12, and A15 structure with average coordination number 13.5. The A15 structure combines high isotropy with binary-friendly coordinates, making it especially suitable for deterministic spatial encoding.

The key insight from this comparison is that *A15* occupies a unique position in the design space: it achieves exceptionally high coordination (surpassing even FCC) while maintaining perfect compatibility with binary representation—a combination not available in other crystallographic structures. This makes it particularly well-suited for applications requiring both high isotropy and guaranteed determinism.

While simpler lattices like SC offer lower implementation complexity, they suffer from significant directional bias that can adversely affect the fairness of spatial representation. BCC improves upon isotropy but still falls short of the near-icosahedral characteristics of *A15*. FCC achieves excellent isotropy and packing density but lacks the binary-friendly coordinates essential for deterministic representation.

The *A15* structure's combination of high coordination, significant local isotropy, and binary-compatible coordinates provides a robust foundation for the deterministic, efficient, and geometrically sound spatial encoding framework proposed in this research.

2. The *A15* Encoding Framework: Design and Implementation

The heart of this research lies in translating the idealized crystallographic description of the *A15* structure (Section 1.2) into a practical, numerically robust system for partitioning continuous 3D space and encoding coordinates. This requires establishing an appropriate and consistent scale, mapping continuous coordinates to discrete lattice identifiers, and ensuring the process avoids the pitfalls of floating-point approximation (Section 1.1). The framework achieves this through a carefully staged approach to coordinate scaling, primarily operating within an internal integer coordinate system during geometric construction to preserve fidelity and defer floating-point conversion until the final output step. This section details this methodology and introduces the reference implementation, A15.py [24], used throughout this work for generation, analysis, and validation.

2.1. A15 Encoding Pipeline: Visual Overview

To provide a clear conceptual overview of the A15 encoding framework, Figure 6 illustrates the complete pipeline from continuous space to deterministic A15 representation. This visualization complements the detailed technical description of the multi-stage scaling framework (Section 2.3) and numerical stability analysis (Section 2.4), offering an accessible entry point to the framework's core mechanisms.

The pipeline consists of four main stages:

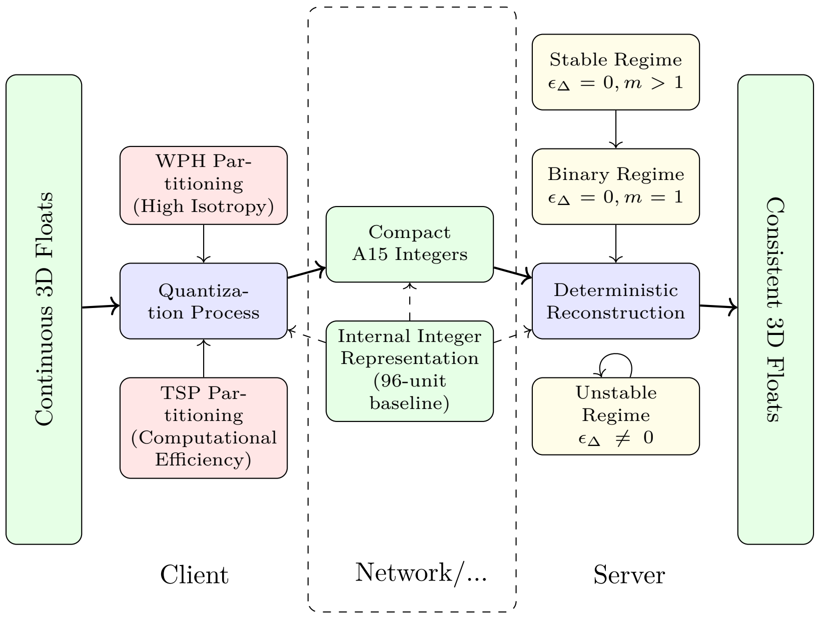
Input: Continuous 3D Space The process begins with continuous coordinates from the application's native coordinate system, typically represented as floating-point vectors subject to the approximation challenges detailed in Section 1.1.

Quantization Using either the WPH or TSP partitioning method (Section 1.3), continuous coordinates are mapped to the nearest A15 lattice point. The computational efficiency of this step varies based on implementation strategy.

Compact A15 Identifier The output of quantization is an integer identifier that uniquely specifies the corresponding A15 lattice point. This compact representation typically requires significantly less storage than the original floating-point coordinates.

Reconstruction When needed for visualization or processing, the A15 identifier can be deterministically reconstructed to floating-point coordinates. When operating in stable scaling regimes ($\epsilon_\Delta = 0$, Section 2.4.4), this reconstruction guarantees bit-identical results across all platforms and environments.

The framework's key innovation lies in the carefully designed multi-stage scaling pipeline (Section 2.3) that operates within an internal integer representation during geometric construction. This approach, coupled with the rigorous stability analysis



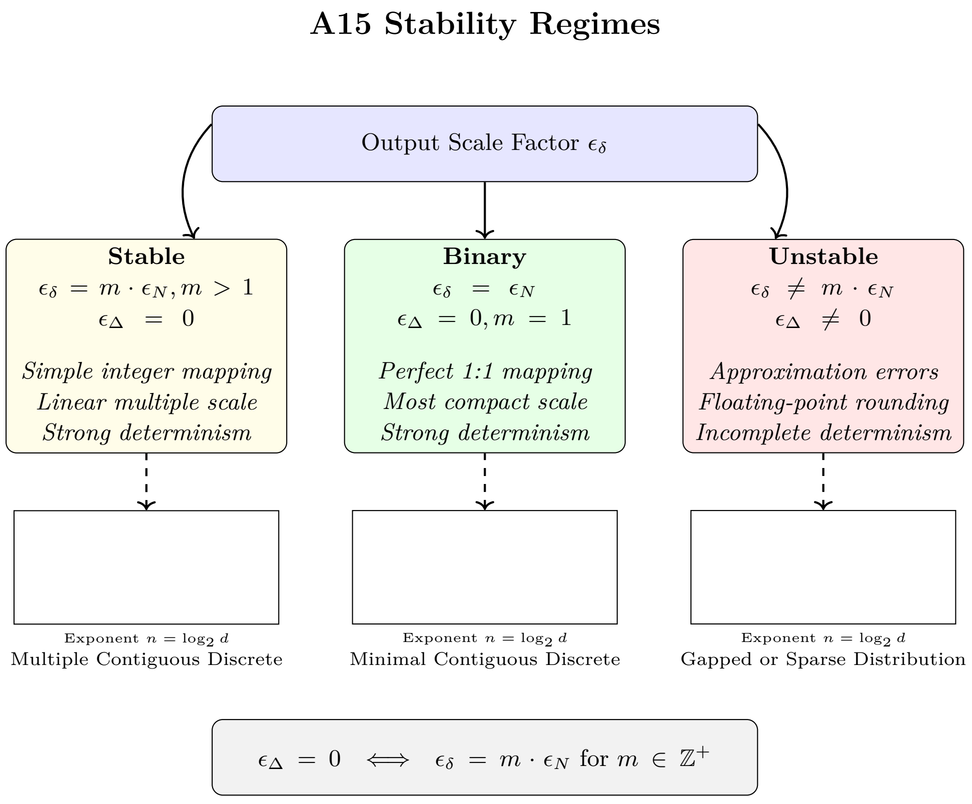
Continuous 3D Floats
WPH Partitioning (High Isotropy)
Quantization Process
TSP Partitioning (Computational Efficiency)
Compact A15 Integers
Internal Integer Representation (96-unit baseline)
Stable Regime $\epsilon_\Delta = 0, m > 1$
Binary Regime $\epsilon_\Delta = 0, m = 1$
Deterministic Reconstruction
Unstable Regime $\epsilon_\Delta \neq 0$
Consistent 3D Floats
Client
Network/...
Server

Figure 6. System diagram of the *A15* encoding framework. The process begins with continuous 3D coordinates, which are quantized to the nearest *A15* lattice point using either the WPH or TSP partitioning method. This produces a compact integer *A15* identifier suitable for efficient transmission or storage. It can be deterministically reconstructed to floating-point coordinates as needed. When operating in binary or stable scaling regimes ($\epsilon_\Delta = 0$), this reconstruction guarantees bit-identical results across all platforms, creating islands of determinism within the otherwise approximated float continuum.

(Section 2.4), ensures that the A15 lattice points themselves can be represented exactly in binary floating-point format, creating islands of perfect determinism within the otherwise approximated float continuum.

Figure 7 illustrates the three stability regimes defined by the relationship between output scale factor ϵ_δ and inherent base scale ϵ_N, alongside their characteristic histogram patterns. The binary regime ($\epsilon_\delta = \epsilon_N$) produces a specific pattern of discrete peaks corresponding to the inherent geometric characteristics of the A15 structure. The stable regime ($\epsilon_\delta = m \cdot \epsilon_N, m > 1$) shows the same pattern plus additional peaks, while the unstable regime ($\epsilon_\delta \neq m \cdot \epsilon_N$) exhibits a gapped or sparse distribution indicative of approximation errors.

The hierarchical representation capability shown in Figure 8 further extends the framework's utility, enabling efficient encoding across different spatial scales with appropriate precision for each level. This multi-scale approach allows applications to balance compact global positioning with high-precision local details, all within the same coherent A15 framework.



A15 Stability Regimes
Output Scale Factor ϵ_δ
Stable
$\epsilon_\delta = m \cdot \epsilon_N, m > 1$
$\epsilon_\Delta = 0$
Simple integer mapping
Linear multiple scale
Strong determinism
Binary
$\epsilon_\delta = \epsilon_N$
$\epsilon_\Delta = 0, m = 1$
Perfect 1:1 mapping
Most compact scale
Strong determinism
Unstable
$\epsilon_\delta \neq m \cdot \epsilon_N$
$\epsilon_\Delta \neq 0$
Approximation errors
Floating-point rounding
Incomplete determinism
Exponent $n = \log_2 d$
Multiple Contiguous Discrete
Exponent $n = \log_2 d$
Minimal Contiguous Discrete
Exponent $n = \log_2 d$
Gapped or Sparse Distribution
$\epsilon_\Delta = 0 \iff \epsilon_\delta = m \cdot \epsilon_N$ for $m \in \mathbb{Z}^+$

Figure 7. Analysis of stability regimes based on the relationship between output scale factor ϵ_δ and inherent base scale ϵ_N. The binary regime ($\epsilon_\delta = \epsilon_N$) produces a specific pattern of discrete histogram peaks corresponding to the intrinsic geometric characteristics of the A15 structure. The stable regime ($\epsilon_\delta = m \cdot \epsilon_N, m > 1$) shows the same pattern plus additional peaks from the integer multiple scaling. The unstable regime ($\epsilon_\delta \neq m \cdot \epsilon_N$) exhibits a gapped or sparse distribution, indicating approximation errors. Operating in either binary or stable regimes ($\epsilon_\Delta = 0$) is essential for guaranteeing bit-identical results across platforms.

Together, these visualizations provide a comprehensive overview of the A15 encoding framework, illustrating its pipeline structure, stability characteristics, and hierarchical representation capabilities. They serve as a visual complement to the detailed technical descriptions throughout the document, enhancing accessibility without sacrificing precision.

2.2. Core Principle: Internal Integer Representation

Applying a discrete lattice like *A15* to represent continuous space necessitates a rigorous scaling framework. Simply using floating-point coordinates throughout the

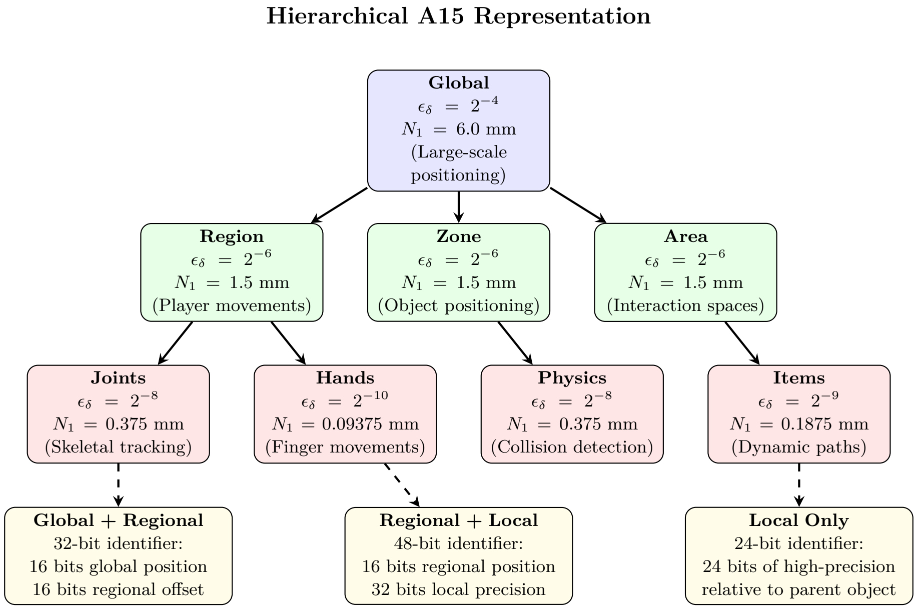


Figure 8. Hierarchical A15 representation using multi-scale relative addressing. The framework naturally supports efficient encoding across different spatial scales, from global positioning down to fine-grained object details. Each level uses an appropriate A15 scale (ϵ_δ) with coordinates defined relative to its parent context. This enables compact representation while maintaining precision where needed, with bit allocation tailored to the requirements of each hierarchical level.

geometric construction risks introducing the very numerical inconsistencies the framework aims to eliminate. Therefore, the methodology prioritizes calculations within a well-defined internal integer coordinate system, delaying the mapping to inexact output formats until the final step. This preserves the precise geometric relationships inherent in the *A15* structure and forms the bedrock for numerical stability.

The essential insight is that while floating-point values inherently approximate most real-world coordinates, integers can exactly represent discrete positions within a lattice. By constructing the *A15* structure's vertices, edges, and cells using only integer coordinates initially, the framework maintains perfect internal consistency. This integer-first approach enables deterministic construction of the spatial partitioning before any potential floating-point approximation occurs during the final output scaling.

Crucially, this reliance on an internal integer foundation, particularly when an application deliberately aligns its native coordinate system and units with a chosen stable A15 scale ($\epsilon_\Delta = 0$, Section 2.4.3), opens pathways for highly efficient quantization. In such aligned scenarios, mapping continuous or application-native coordinates to A15 identifiers may simplify significantly, potentially reducing complex geometric searches to fast integer arithmetic operations like truncation or bit shifts, thereby mitigating concerns about quantization overhead through careful co-design.

2.3. Multi-Stage Scaling Pipeline

The conversion from the abstract *A15* definition to concrete coordinates involves a systematic, multi-stage pipeline implemented within A15.py. This process uses several interacting parameters and fixed factors to progressively build the structure within the internal integer system before final output scaling. The key stages are detailed below, and the parameters are summarized in Table 4.

2.3.1. Primitive Definition and Resolution (prescale)

The pipeline originates with the definition of the fundamental geometric units associated with the *A15* basis sites—typically the pyritohedra and tetradecahedra related to the Weaire–Phelan Honeycomb method (Section 1.3), or the cubic blocks for the Tetrastix Prism method. Functions within A15.py (e.g., pyritohedron(), tetradecahedra()) apply an internal integer multiplier, the prescale parameter, to the canonical *dimensionless fractional* coordinates (like $0, 1/4, 1/2$) defining the vertices of these shapes. This crucial first step converts the fractional values into *primitive-relative integer coordinates*, establishing the minimum resolution necessary to accurately represent the individual geometric primitives without internal approximation. Default prescale values in A15.py (typically 20 for standard WPH-derived shapes, 24 for TSP-derived shapes via -stix) are chosen specifically to ensure these base vertices land precisely on an integer grid relative to the shape's center.

2.3.2. Lattice Placement (Factor 24)

The lattice() function in A15.py replicates these integer-vertex polyhedra periodically across 3D space according to the $Pm\bar{3}n$ symmetry rules. It determines the positions for the *centers* of these polyhedra based on *integer lattice vectors* xyz relative to an origin offset o. A key element in this placement is the fixed **Lattice Spacing Factor of 24 units**. Within the lattice() function, the calculation (xyz + o) * 24 multiplies the integer lattice vector by this factor. This defines the spacing between the nominal lattice points (i.e., the centers of the polyhedra) *measured in the integer units established by* prescale. This factor ensures the correct relative positioning of the repeating units within the overall *A15* lattice framework.

2.3.3. Basis Accommodation (Factor 96 Baseline)

The *A15* structure's complexity arises from its multi-atom basis; it contains distinct crystallographic sites (Wyckoff 2a and 6d) not just at the nominal lattice points defined above, but also at specific fractional offsets, notably involving 1/4 for the 6d sites. To place *all* required sites onto a single, consistent internal integer grid requires a resolution finer than the 24-unit spacing factor alone provides. The smallest denominator involved is 4 (from 1/4). Therefore, the internal integer grid must be conceptually scaled such that one unit step along a primitive lattice vector corresponds to $4 \times 24 = 96$ internal integer units along each principal axis.

This derived **Effective Lattice Unit Factor of 96** represents the fundamental period or effective unit dimension of the comprehensive internal integer grid required to represent the *complete A15* structure (including its basis) without loss of precision due to these fractional offsets. This factor of 96 establishes the crucial **baseline dimension** relative to which the structure's inherent numerical precision requirements (ϵ_N, see Section 2.4.2) are determined and against which the final output scale (ϵ_δ) is compared for stability analysis.

2.3.4. Optional Binary Rescaling (rescale)

Before the lattice generation stage fully populates the internal integer coordinates, an optional power-of-two scaling factor, rescale, can be applied. This factor is specified via the -rescale command-line flag or implicitly through '+/-' suffixes appended to shape names in the input (e.g., pyritohedra++ implies a rescale factor of $2^2 = 4$ in A15.py). This rescale factor multiplies the effective size represented by the internal integer coordinates relative to the 96-unit baseline established above. It allows adjustments to the overall size or relative proportions of different generated components *before* the final output scaling step, operating purely in powers of two within the integer domain. Importantly, the choice of rescale influences the resulting inherent base scale ϵ_N of the internal geometry.

2.3.5. Resultant Internal Integer Coordinates

The cumulative effect of the initial prescale (Section 2.3.1), the lattice placement logic (Section 2.3.2, establishing the 96-unit effective baseline Section 2.3.3), and any applied rescale factor (Section 2.3.4) collectively defines the complete set of vertex coordinates for the entire generated structure. These coordinates exist within a consistent, potentially large-valued, *internal integer* system referenced against the 96-unit effective lattice baseline. By prioritizing integer arithmetic throughout these construction stages, A15.py preserves the precise geometric relationships inherent in the *A15* structure, deferring any floating-point approximation until the very last output step. This internal integer representation forms the bedrock for the numerical stability analysis detailed next.

Furthermore, the explicit control over scale inherent in this pipeline naturally supports hierarchical or multi-scale representations via relative addressing. This allows, for instance, coarse-grained coordinates identifying an entity's global position to coexist efficiently with fine-grained coordinates defining its internal details (e.g., limb articulations relative to a centroid), each utilizing an appropriate A15 scale (ϵ_δ) within the same unified structural logic, optimizing both data density and precision.

2.4. Numerical Stability: Regimes and Validation

Achieving deterministic spatial representation—ensuring that identical logical operations yield bit-identical results across different systems—is a primary motivation for this framework. This guarantee hinges critically on the relationship between the scale chosen for the final output coordinates (ϵ_δ) and the inherent precision requirements (ϵ_N) of the underlying *A15* geometry, as captured by the internal integer representation (Section 2.2). The framework defines specific, verifiable conditions under which the mapping from the precise internal structure to the output coordinate system can be performed without introducing floating-point approximation errors relative to that internal structure.

2.4.1. Global Output Scaling Factor (ϵ_δ)

The primary user control over the final representation's physical scale or resolution is the scale parameter, specified via the -scale=<value> command-line option in A15.py. We denote this crucial factor as ϵ_δ. It is applied globally by the figure() function *after* all internal integer coordinates (relative to the 96-unit baseline) have been generated. This ϵ_δ defines the mapping from the dimensionless internal integer system to the output coordinate system (typically floating-point). Therefore, ϵ_δ effectively sets the real-world size represented by one unit of the internal 96-unit baseline dimension, and its specific value is the key determinant of the resulting numerical stability regime.

It is important to emphasize that ϵ_δ does not eliminate the inherent approximation nature of floating-point numbers—most real-world coordinates will still require approximation when represented as binary floats. Rather, it establishes a carefully chosen scale at which the *specific discrete points of the A15 lattice* can be exactly represented without such approximation, creating "islands of determinism" within the otherwise approximated float continuum.

2.4.2. Inherent Base Scale (ϵ_N)

The internal integer geometry produced by the construction process (Section 2.3) possesses an intrinsic precision limit relative to the 96-unit baseline. This limit arises from the specific combination of the *A15* basis site coordinates (involving factors of 1/4), the chosen prescale value, and any applied rescale factor. We define ϵ_N as the **inherent base scale** required to represent this specific internal geometry exactly when mapped to a base-2 representation. Conceptually, ϵ_N is the finest scaling factor, expressible as a power of two ($1/2^N$ for some integer N), that allows all generated internal integer vertex coordinates to be represented perfectly as rational numbers without approximation when measured against the 96-unit baseline. It captures the structure's intrinsic geometric precision limit within the binary system used by floating-point numbers. The A15.py script computationally infers ϵ_N by analyzing the power-of-two denominators required for the exact rational representation of the generated internal integer geometry relative to the 96-unit baseline.

2.4.3. Stability Condition and Difference (ϵ_Δ)

Numerical stability is achieved if, and only if, the chosen global output scale (ϵ_δ) is commensurate with the inherent base scale (ϵ_N). Specifically, the mapping from the internal integer representation to the output coordinate system is guaranteed to be exact (free from representation error relative to the internal grid) if ϵ_δ is a positive integer multiple (m) of ϵ_N. The A15.py script quantifies this relationship by calculating the **stability difference**, ϵ_Δ. Conceptually, ϵ_Δ measures the mismatch or residual error when checking if ϵ_δ aligns perfectly with the grid defined by ϵ_N:

$$\epsilon_\Delta = 0 \quad \iff \quad \epsilon_\delta = m \cdot \epsilon_N \quad \text{for some integer } m \geq 1 \tag{2}$$

If this condition holds ($\epsilon_\Delta = 0$), the framework operates in a stable regime. Otherwise ($\epsilon_\Delta \neq 0$), the scaling is unstable, and approximation errors are necessarily introduced relative to the internal structure.

2.4.4. Stability Regimes

The stability condition leads to three distinct operational regimes:

Binary Scaling ($\epsilon_\Delta = 0$ with $m = 1$): This optimal regime occurs when the chosen output scale precisely matches the minimum required inherent base scale ($\epsilon_\delta = \epsilon_N$). All internal integer coordinates map directly and exactly onto the binary floating-point grid defined by $\epsilon_N = 1/2^N$ without any approximation relative to the internal structure.

$$\epsilon_\delta = \epsilon_N = \frac{1}{2^N} \quad \implies \quad \epsilon_\Delta = 0 \tag{3}$$

This represents the most compact scale that allows exact representation and guarantees determinism. The validation histogram (Figure 9, left) shows a specific pattern of discrete peaks at the exponents corresponding to the inherent A15 geometry.

Stable Scaling ($\epsilon_\Delta = 0$ with $m > 1$): This regime occurs when the output scale is an exact positive integer multiple (m) of the inherent base scale ($\epsilon_\delta = m \cdot \epsilon_N$). All internal coordinates still map exactly to representable binary floating-point values without approximation error relative to this scaled grid.

$$\epsilon_\delta = m \cdot \epsilon_N = \frac{m}{2^N}, \quad m \in \mathbb{Z}^+, m > 1 \quad \implies \quad \epsilon_\Delta = 0 \tag{4}$$

This regime maintains perfect representability and determinism while providing flexibility in choosing the overall physical scale. The smallest resolvable difference (Unit of Least Precision, Section 6.4.5) corresponds to m units at the ϵ_N scale, effectively making the ULP equal to ϵ_δ. The validation histogram (Figure 9, middle) displays the same characteristic pattern of peaks as the binary regime, plus additional peaks reflecting the integer scaling factor m.

Unstable Scaling ($\epsilon_\Delta \neq 0$): This regime occurs whenever the chosen output scale ϵ_δ is *not* a positive integer multiple of the inherent base scale ϵ_N. Under these conditions, the precise internal integer geometry cannot be perfectly represented on the binary floating-point grid implied by ϵ_δ. Rounding errors are necessarily introduced during the final scaling from internal integers to output floats.

$$\epsilon_\delta \neq m \cdot \epsilon_N \quad \text{for any} \quad m \in \mathbb{Z}^+ \quad \implies \quad \epsilon_\Delta \neq 0 \tag{5}$$

Utilizing unstable scales fundamentally compromises the core benefit of the framework regarding determinism. It introduces representation errors, leading to subtle geometric inconsistencies, non-deterministic outcomes in sensitive calculations, and significant challenges in achieving reliable state synchronization. The validation histogram (Figure 9, right; also Figure 10) clearly reveals this instability through a broad, sparse, or gapped distribution of denominator exponents.

Therefore, operating exclusively within the **Binary** or **Stable** scaling regimes ($\epsilon_\Delta = 0$) is essential for numerical consistency, reproducibility, and guaranteed deterministic behavior.

2.5. Validation via A15.py Histogram Analysis

The A15.py script provides not only the means to generate structures based on this framework but also includes a direct, quantitative method for validating the numerical stability regime resulting from any chosen set of parameters, particularly the global output scale ϵ_δ. This validation capability is accessed via the -bars command-line option and provides empirical confirmation of the stability concepts discussed above (Section 2.4). When enabled, the script performs the following analysis within its figure() function:

1. It iterates through the components (x, y, z) of output floating-point coordinates at the selected scale factor (ϵ_δ). For each component, it obtains the exact rational representation (k, d) using Python's built-in float.as_integer_ratio() method [23]. This captures the precise value representable by the float variable, including any approximation introduced if the scaling was unstable relative to the internal grid (Section 2.3.5).
2. It then analyzes the denominator d of this exact rational representation. Since floats operates within a base-2 context, the denominator d always simplifies to 2^n for some integer exponent n. The script extracts this effective exponent $n = \log_2 d$, which conceptually represents $-\log_2$ of the fractional part's required precision at the output scale ϵ_δ.
3. Finally, it visualizes the distribution of these calculated exponents n across all analyzed vertex components as a histogram (Figure 9). This histogram provides immediate visual feedback on the numerical stability of the chosen configuration.

The shape of this generated histogram directly corresponds to the theoretical stability regimes (Section 2.4.4): a specific pattern of discrete peaks signifies Binary scaling; the same pattern plus additional peaks signifies Stable scaling; and a sparse, broad, or gapped distribution signifies Unstable scaling, revealing the introduction of approximation errors relative to the internal geometry. Furthermore, the script explicitly calculates and displays the stability difference ϵ_Δ alongside the histogram (Figure 10), offering direct numerical confirmation of the regime. This built-in quantitative validation provides objective criteria for selecting scaling parameters that correctly balance spatial resolution, memory efficiency, and the critical requirement for numerical robustness and determinism.

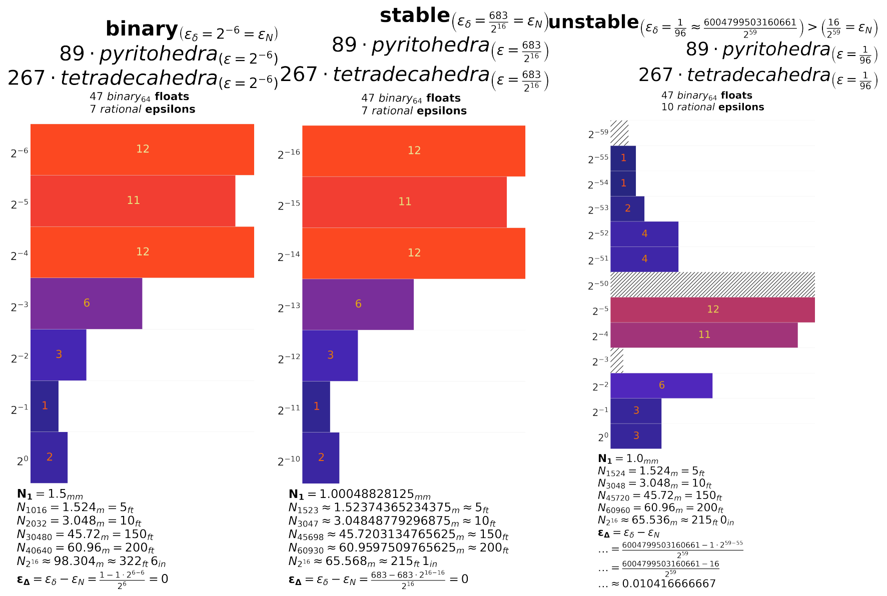


Figure 9. Example histograms generated by A15.py (-bars) illustrating numerical stability regimes for different output scaling factors (ϵ_δ). Left (fig-histb.png.txt, $\epsilon_\delta = 1/64$): **Binary** scale ($\epsilon_\Delta = 0$) showing a distinctive pattern of discrete denominator exponents ($n = \log_2 d$) reflecting the intrinsic geometric characteristics of the A15 structure. Middle (fig-hists.png.txt, $\epsilon_\delta = 683/2^{16}$): **Stable** scale ($\epsilon_\Delta = 0$) showing the same pattern plus additional exponents introduced by the integer multiple scaling. Right (fig-histu.png.txt, $\epsilon_\delta = 1/96$): **Unstable** scale ($\epsilon_\Delta \neq 0$) showing a widely spread distribution with gaps, indicating floating-point approximation errors introduced during final scaling relative to the internal integer geometry.

2.6. A15.py **Reference Implementation Overview**

The Python script A15.py [24] serves as the reference implementation, exploratory tool, and validation instrument for the concepts presented in this research. It requires Python 3 and leverages standard scientific libraries—NumPy [17] for efficient numerical array operations, SciPy [31] for geometric computations (such as convex hull calculations used for visualization via scipy.spatial.ConvexHull), and Matplotlib [18] for versatile 2D and 3D visualization. The script's workflow systematically applies the principles of the *A15* encoding framework, incorporating the crucial multi-stage scaling logic (Section 2.3) and the quantitative numerical stability analysis (Section 2.5) described previously. Table 5 summarizes the core stages of its operation, from parameter parsing to final visualization and validation. This structured process allows A15.py to generate, visualize, and analyze configurations, providing concrete examples and empirical validation of the framework's properties, such as the composite output shown in Figure 10.

3. Interpretation, Benefits, and Limitations

The investigation detailed in the preceding sections confirms the distinctive suitability of the *A15* phase structure, when employed within the numerically stable scaling framework (Section 2), for partitioning interactive 3D spaces. The confluence of its inherent crystallographic properties (Section 1) and the demonstrable numerical stability achievable through disciplined scaling ($\epsilon_\Delta = 0$, Section 2.4.3) yields a framework that is both geometrically sophisticated and computationally practical for demanding immersive applications. This section interprets these findings, highlights the quantifiable benefits derived directly from the framework's mechanics, and candidly discusses practical limitations and engineering considerations relevant to its implementation.

3.1. Interpretation and Core Findings

This research establishes *A15* encoding as a robust structural foundation for coordinated spatial partitioning. The core finding is that by leveraging the specific crystallographic symmetries and binary-friendly coordinates of the *A15* structure, and by rigorously adhering to the multi-stage scaling pipeline culminating in a stable output scale ($\epsilon_\Delta = 0$), it is possible to create a spatial representation that fundamentally eliminates floating-point representation errors relative to its own discrete grid. This provides a pathway to achieving verifiable determinism in spatial computations, a critical enabler for next-generation networked virtual environments and related spatial computing tasks.

3.2. Quantifiable Benefits

The *A15* encoding framework, when implemented correctly within stable scaling regimes, offers several key advantages validated by theoretical analysis and the A15.py [24] reference implementation:

3.2.1. Guaranteed Determinism via Stable Scaling

Arguably the most significant contribution stems directly from operating within the rigorously defined **Binary** or **Stable** scaling regimes ($\epsilon_\Delta = 0$, Section 2.4.4). By precisely aligning the chosen output scale (ϵ_δ) with the structure's inherent geometric

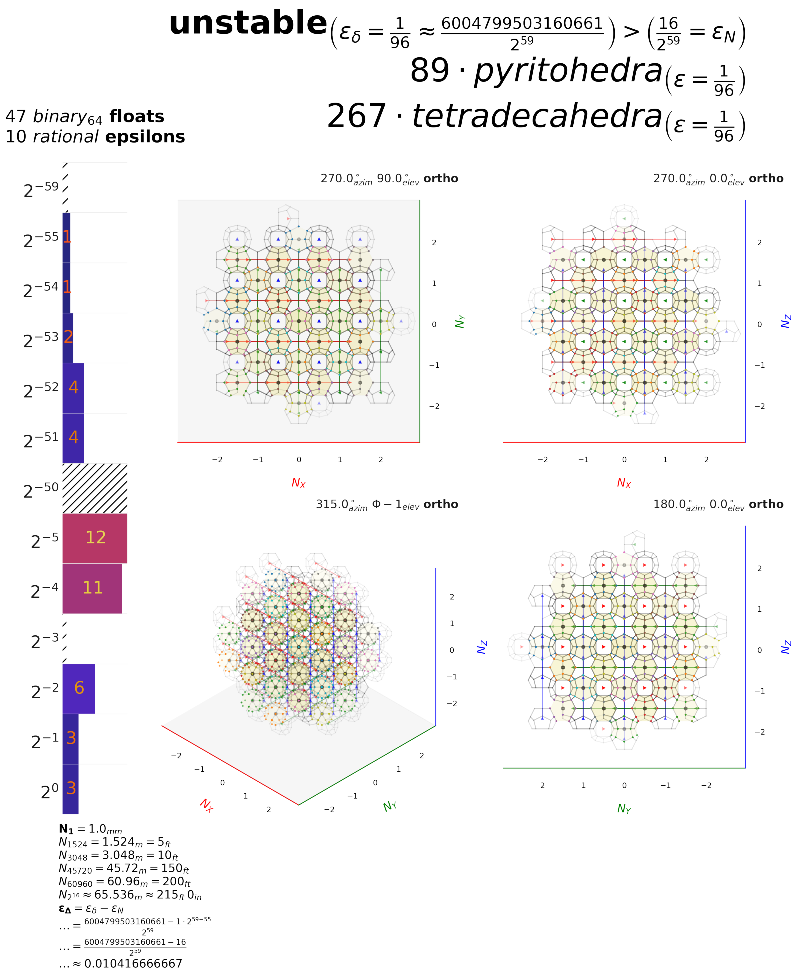


Figure 10. Composite visualization generated by A15.py (fig-main.png.txt) showing pyritohedra and tetradecahedra components alongside stability analysis. This example uses an **Unstable** output scale $\epsilon_\delta = 1/96$. The calculated non-zero stability difference ($\epsilon_\Delta \approx 0.0104 \neq 0$) is explicitly annotated in the histogram sidebar (left), confirming instability relative to the internal integer geometry. The histogram visually reflects this instability through its wide, gapped distribution of floating-point denominator exponents ($n = \log_2 d$). Main 3D views use dimensionless coordinates N_X, N_Y, N_Z relative to the basic unit width N_1. For this specific unstable scale, N_1 corresponds to 1.0 mm, derived from applying $\epsilon_\delta = 1/96$ to the internal 96-unit effective lattice baseline. Counts shown refer to the number of float components analyzed for the histogram.

precision requirements (ϵ_N, relative to the 96-unit baseline derived in Section 2.3.3), the framework guarantees that all *A15* lattice points and derived vertex coordinates map exactly onto hardware binary floating-point formats *relative to that chosen stable scale.*

It is important to emphasize that this guarantee applies specifically to the representation of the discrete A15 lattice points themselves; it ensures that the quantized spatial framework remains consistent across systems, but does not eliminate all floating-point variability within applications using this framework. Operations performed between these stable grid points (such as physics calculations, interpolation, or velocity updates) may still use floating-point arithmetic and thus be subject to traditional floating-point variability unless additional measures are taken.

Nevertheless, this systematic elimination of representation errors for the spatial partitioning grid itself provides a foundation for much-improved consistency across disparate machines, platforms, and even different compilation environments—a fundamental prerequisite for reliable state synchronization in networked systems, reproducible physics simulations, efficient network delta compression strategies, verifiable event sequences and replays, and ultimately, fair and trustworthy competitive experiences.

3.2.2. Memory and Bandwidth Efficiency

Mapping continuous coordinates onto compact, often integer-based, *A15* identifiers provides substantial memory and bandwidth savings compared to standard floating-point vector representations (Section 1.1). This advantage is particularly pronounced when quantizing explicitly defined or bounded volumes where the full dynamic range and mantissa precision of standard floats represent unnecessary overhead. For a practical example, consider establishing a coordinate system using a stable scale where the basic unit width N_1 (derived from the 96-unit baseline, Sections 2.3.3 and 6.4.7) corresponds to 1.5 mm (achieved with $\epsilon_\delta = 1/64$). A 48 bit integer identifier could then be structured as follows: 3 bits can uniquely address the 8 distinct atomic sites within the A15 conventional cubic cell (2 at Wyckoff 2a, 6 at Wyckoff 6d); allocating 11 bits for the vertical axis provides $2^{11} \times 1.5\,\text{mm} \approx 3.07\,\text{m}$ of range, sufficient vertical headroom for human-scale interaction; the remaining 34 bits, split evenly (17+17) for the horizontal axes, define a square area of $(2^{17} \times 1.5\,\text{mm})^2 \approx (196.6\,\text{m})^2$. This 48-bit structure, capable of encoding a space several times larger in area than the largest professional stadium fields with sub-millimeter intra-cell precision, represents a **50% reduction** compared to the 96 bit typically required for three standard single-precision (32 bit) floats (Equation (6)).

$$\text{Savings} = \frac{(96\,\text{bit} - 48\,\text{bit})}{96\,\text{bit}} \times 100\% = 50\% \qquad (6)$$

Furthermore, because many application environments (e.g., smaller arenas, non-square layouts [25]) require less range, the bit allocation can often be reduced further, leading to savings significantly exceeding 50%. This efficiency translates directly into reduced memory footprints for spatial data structures and significantly lower network traffic for coordinate updates. Considering just baseline kinematic tracking data

(Equation (7)), this reduction from approximately $67.2\,\mathrm{kbit\,s^{-1}}$ down to $34\,\mathrm{kbit\,s^{-1}}$ or less offers significant leverage on aggregate bandwidth, especially in complex scenarios often exceeding $100\,\mathrm{kbit\,s^{-1}}$ per user for kinematics alone.

$$\mathrm{Rate}_{\mathrm{3xFloat}} = 20\,\mathrm{jts} \times (3_{\mathrm{pos}} + 4_{\mathrm{quat}})\,\frac{\mathrm{floats}}{\mathrm{jt}} \times 4\,\frac{\mathrm{bytes}}{\mathrm{float}} \times 15\,\mathrm{Hz} = 8400\,\mathrm{byte/s} \approx 67.2\,\mathrm{kbit/s} \tag{7}$$

The *A15* integer encoding effectively reclaims storage and bandwidth otherwise consumed by unused float exponent bits and excess significand precision within suitably bounded contexts. Furthermore, this compactness and structural definition offer benefits for long-term data storage and archival, providing a potentially more stable and interpretable format compared to raw floating-point streams. Practical applications, such as the related layoutc project which encodes physical layouts using similar principles [25], demonstrate the utility of such compact, deterministic encodings.

3.2.3. Geometric Fidelity and Isotropy

Leveraging the intrinsic crystallographic properties of the *A15* structure (Section 1.2) imparts beneficial geometric qualities to the spatial representation. Its verified high mean coordination number (13.5) indicates efficient local packing and dense connectivity between neighboring regions. The crucial T_h point group symmetry provides a high degree of local isotropy by incorporating near-icosahedral geometric elements within a globally cubic (and thus perfectly periodic) framework (Section 1.2). This structural integrity, especially when combined with the topologically matched Weaire–Phelan Honeycomb local discretization method (Section 1.3), fosters a spatially uniform or “fair” representation, helping to minimize the directional biases or artifacts that can plague simpler grid-based partitioning schemes.

3.2.4. Suitability for Distributed Computing

The inherent regularity, crystallographic symmetries, and predictable neighborhood topology of the A15 lattice provide a structured substrate well-suited for spatial domain decomposition. This facilitates the distribution of spatial computations across parallel architectures, including clusters, GPUs, or edge networks, potentially simplifying load balancing and data management compared to adaptive or irregular spatial structures. The deterministic nature of the grid ensures consistent partitioning across nodes operating under stable scaling regimes.

3.2.5. Foundation for Enhanced Interoperability

By establishing a common, mathematically precise, and verifiable spatial structure, the A15 framework offers a robust foundation for enhanced interoperability. Diverse applications adhering to the same A15 structure, scale, and orientation conventions could potentially exchange, reference, or merge spatial data with greater semantic consistency and reduced ambiguity, fostering cohesion across federated virtual environments or collaborative platforms.

3.2.6. Validated Implementation Framework

The accompanying A15.py script (Section 2.6) serves as more than just a visualization aid; it is a validated reference implementation and analysis framework. It demonstrates the practical construction of *A15*-based structures, rigorously implements the multi-stage scaling logic essential for achieving numerical stability (Section 2.3), and provides the quantitative histogram analysis (Section 2.5, Figure 9) that empirically confirms the existence and accessibility of the stable scaling regimes ($\epsilon_\Delta = 0$). This ensures the reproducibility of the core findings regarding numerical stability and offers a concrete, verifiable starting point for developers seeking to implement or explore the *A15* encoding framework for their own applications.

3.3. Limitations and Engineering Considerations

Despite its significant advantages for achieving determinism and efficiency, adopting the *A15*-based partitioning approach involves several practical limitations and engineering challenges that require careful consideration during implementation. These represent addressable design aspects rather than fundamental flaws in the underlying concept:

3.3.1. Complexity of Arbitrary Rotations

Applying arbitrary rotations relative to the lattice axes directly to the integer lattice coordinates used in *A15* encoding can be intricate, particularly when mapping points onto valid *A15* basis sites (Wyckoff 2a or 6d) or navigating complex cell boundaries like those in the Weaire–Phelan Honeycomb. Correct implementation necessitates carefully calculated transformations aware of the crystal basis and space group operations [1].

In practice, this means that when objects rotate in the virtual space, the mapping of their vertices or bounding volumes to A15 identifiers requires additional computational steps beyond simple coordinate transformation. For arbitrary rotations, the system must typically:

1. Transform object-local coordinates to world space (standard transform)
2. Apply nearest-neighbor or containment tests to determine which A15 cell contains each point
3. Map these points to appropriate A15 lattice identifiers

The complexity depends on the nature of the rotation and the chosen local discretization method; axis-aligned rotations or operations within simpler partitioning geometries like the Tetrastix Prism may present fewer challenges. Regardless of the approach, adopting a canonical orientation convention (Section 3.3.6) is essential for consistent interpretation and interoperability.

3.3.2. Quantization Cost and Design Alignment

A practical consideration is the computational cost of quantization—mapping arbitrary continuous coordinates to the nearest discrete *A15* identifier. While general-purpose nearest-neighbor searches in 3D can be computationally intensive, especially with complex cell boundaries (e.g., Weaire–Phelan Honeycomb), this cost is highly dependent on implementation strategy and application alignment.

For arbitrary coordinates in arbitrary orientations relative to the A15 grid, the quantization process could require:

- Point-in-cell tests against multiple candidate cells
- Distance calculations to determine the nearest lattice point
- Potentially complex geometric intersection tests for the Weaire–Phelan Honeycomb partitioning method

However, as noted (Section 2.2), if an application's coordinate system is deliberately aligned with a stable *A15* scale ($\epsilon_\Delta = 0$), quantization can potentially become a highly efficient process dominated by integer arithmetic operations (e.g., truncation or bit shifts), making performance manageable through informed design choices rather than being an inherent bottleneck. This approach represents a co-design strategy where the application's spatial system is developed with A15 quantization in mind from the outset, rather than being applied as an afterthought.

3.3.3. Integration with Existing Float-Based Systems

Incorporating the A15 framework into existing engines and applications that rely heavily on floating-point arithmetic presents integration challenges. Most modern game engines, simulation environments, and physics systems operate natively with floating-point coordinates and transformation matrices. Introducing A15 encoding requires careful attention to the boundaries between:

- Float-based internal physics and rendering systems
- A15-encoded positions for storage and network transmission
- Coordinate and state transformations between these representations

Practical implementation strategies might include:

- Using A15 encoding only for network transmission and state synchronization
- Maintaining dual representations (float for local processing, A15 for verifiable operations)
- Developing middleware translation layers between engine components
- Implementing custom physics solvers that operate directly on the A15 grid

Each approach involves trade-offs between implementation complexity, performance, and the degree of determinism achieved across the entire application pipeline.

3.3.4. Scalability and Spatial Federation

The framework naturally defines discrete, structured zones based on the generated *A15* lattice extents (controlled by the n parameter in A15.py). Seamlessly extending this model to create massive, open-world environments requires robust mechanisms for managing transitions and maintaining coordinate and state consistency across the boundaries between independent *A15* zones. Addressing this likely involves developing standardized inter-zone boundary protocols for coordinate transformations (potentially involving scale changes), object state hand-offs, and perhaps authority transfer between simulation domains. Level-of-Detail (LOD) systems [22] utilizing multi-resolution *A15* grids (employing coarser quantization for distant or less critical zones, potentially via relative addressing) might also play a role in managing complexity at large scales.

3.3.5. Encoding Efficiency for Irregular Volumes

Using lattice-aligned cuboidal extents for defining *A15* zones, while straightforward to implement via the n parameter, can lead to inefficient use of the addressable integer coordinate space when representing environments with highly irregular external boundaries, complex internal terrain features (like mountains or caves), or significant voids (e.g., the interior of large, non-rectangular buildings). This may result in significant portions of the allocated integer coordinate range being unused or unreachable within the playable or interactive space. Potential mitigations could include implementing secondary encoding or metadata schemes (e.g., run-length encoding of valid zones along axes, hierarchical spatial masks, sparse data structures like sparse voxel octrees adapted to the A15 lattice), exploring hybrid approaches that combine the regular *A15* grid with adaptive structures primarily for managing boundary details or sparse areas (though this might reintroduce some complexity), or effectively managing precision and sparsity through hierarchical, multi-scale A15 grids utilizing relative addressing (Section 2.3).

3.3.6. Requirement for Handedness Convention

While the overall $Pm\bar{3}n$ space group is centrosymmetric (achiral), the specific arrangement of atoms in the *A15* structure's basis results in alternating left- and right-handed local coordination environments around the 6d sites (Section 1.2). This local chirality is implemented deterministically based on lattice position within the A15.py reference code, and is consistent with a standard **right-handed coordinate system** interpretation. However, for interoperability in any practical application, particularly networked ones, all participating systems **must establish and strictly adhere to a shared global orientation convention**, independent of any specific implementation's internal standard. This convention dictates how local chiralities and global axes are interpreted, represented, and transformed, ensuring consistency between different client implementations and, crucially, when interfacing with host environments or game engines that may use different native coordinate system handedness (e.g., left-handed systems common in Unity [30] and Unreal Engine [9] versus right-handed systems standard in physics and mathematics). Without such a shared convention, mirrored or incorrectly oriented geometry could easily result from exchanging *A15*-encoded coordinates.

3.3.7. Privacy and Security of Tracking Data (PII)

The application of this efficient encoding framework to fine-grained spatial tracking data, especially full-body kinematics derived from VR/AR systems, carries significant privacy implications that **must** be addressed with utmost seriousness. Such detailed movement data constitutes **Personally Identifiable Information (PII)** and may qualify as sensitive biometric data under various regulations (e.g., GDPR [10], CCPA [4]). Handling this data demands rigorous privacy safeguards and unwavering ethical considerations as a **non-negotiable** aspect of implementation. Developers and deployers **must** integrate robust security measures as a foundational requirement. This includes, at a minimum:

- Employing strong **end-to-end encryption (E2EE)** for all *A15*-encoded coordinate streams and associated tracking data during network transmission and persistent storage.
- Strict adherence to **data minimization principles** (collecting only the data essential for the application's functionality).
- Implementing transparent **user consent mechanisms** before any tracking begins.
- Establishing clearly defined **data retention and deletion policies**.
- Utilizing effective **anonymization or aggregation strategies** whenever full individual fidelity is not strictly required (e.g., for analytics or heatmaps).
- Ensuring full compliance with all relevant legal and ethical regulations.

This data represents individuals and their behavior; it **must** be treated with the highest degree of care, respect, security, and transparency. Failure to do so carries significant legal, ethical, and reputational risks.

It should be noted that while the A15 encoding framework reduces the raw size of spatial data, potentially making it more efficient to encrypt and secure, this efficiency gain does not diminish the fundamental privacy requirements surrounding such sensitive information.

4. Immediate Potential and Future Prospects

The *A15*-based spatial partitioning and encoding framework, validated for its numerical stability and efficiency (Section 3), offers immediate potential for enhancing current virtual environments and provides a solid foundation for future advancements in spatial computing. Its inherent structural and numerical properties lend themselves to broad applicability, while also opening intriguing avenues for further research and development aimed at refining and extending its capabilities.

4.1. Immediate Practical Applications

The practical adoption of *A15* partitioning across diverse virtual platforms is facilitated by several key features, yielding tangible benefits for various applications available today:

4.1.1. Cross-Platform Compatibility and Interoperability

The underlying $Pm\bar{3}n$ space group of the *A15* structure is centrosymmetric, lacking inherent global chirality [1]. This structural property means the fundamental lattice encoding can be consistently represented within both **left-handed coordinate systems** (common in game engines like Unity [30] and Unreal Engine [9]) and **right-handed systems** (standard in mathematics and physics) through straightforward external affine transformations. While this simplifies cross-platform integration, achieving unambiguous interoperability absolutely requires establishing and adhering to a shared global orientation convention (Section 3.3.6).

In practical terms, applications implementing the A15 framework should:

- Document the specific handedness convention used (explicitly right-handed or left-handed)
- Define the precise alignment of A15 crystallographic axes with world-space axes
- Specify the origin point of the A15 grid relative to world space
- Include these specifications in any data exchange protocols or standards

Crucially, the shared mathematical foundation provides a pathway toward application-agnostic spatial understanding, enabling potentially more seamless data exchange between disparate systems built upon the same A15 conventions.

4.1.2. Integration Strategies with Existing Engines

For practical implementation in current game engines and simulation environments, several integration approaches can leverage A15 encoding while minimizing disruption to existing pipelines:

Dual-Representation Strategy: This approach maintains two parallel coordinate representations:

- Native engine coordinates (typically floats) for internal processing, physics, and rendering
- A15 identifiers for network transmission, authoritative state storage, and verification

The synchronization between these representations occurs at well-defined boundaries:

1. Convert native coordinates to A15 identifiers before network transmission or state archival
2. Convert received A15 identifiers back to native coordinates for local processing
3. For critical verifications, operations follow the same path through A15 encoding/decoding

This strategy minimizes integration complexity while still gaining the bandwidth efficiency and deterministic verification benefits of A15 encoding.

Middleware Strategy: A specialized middleware layer can handle the translation between native engine coordinates and A15 identifiers transparently:

- Intercept network communication and convert coordinates on the fly
- Provide verification services for authoritative operations
- Implement comparison operators that account for A15 discretization

This approach allows for incremental adoption without significant engine modifications, though it may introduce additional computational overhead at translation boundaries.

Native Implementation Strategy: For applications requiring maximal determinism and efficiency, a more comprehensive approach involves:

- Implementing custom spatial data structures directly using A15 identifiers
- Developing physics solvers that operate on the A15 grid with deterministic integer arithmetic
- Constructing rendering pipelines aware of A15 discretization

This approach requires significant engineering investment but can yield systems with strong determinism guarantees throughout the entire application pipeline, not just at network boundaries.

4.1.3. Alignment with Global Measurement Standards

The explicit output scaling mechanism (ϵ_δ, controlled via the -scale parameter in A15.py) allows the dimensionless internal *A15* lattice coordinates (relative to the 96-unit effective baseline, Section 2.3.3) to map directly and predictably onto standard physical units, such as SI meters or Imperial feet. Critically, selecting a scale factor within the Binary or Stable regimes ($\epsilon_\Delta = 0$, Section 2.4.4) ensures this mapping is exact relative to the framework's chosen resolution. This facilitates interoperability not only between virtual systems but also with real-world measurements, enhancing user comprehension and grounding virtual spaces in familiar metrics. For instance, the recommended baseline scale of $\epsilon_\delta = 2^{-6}$ (-scale=1/64) provides robust sub-millimeter precision while operating within a numerically stable regime, yielding a basic unit width (N_1) of 1.5 mm (Section 6.4.7), suitable for many human-scale interactions.

4.1.4. Hierarchical Representation via Relative Addressing

The framework naturally enables multi-scale representations through relative addressing (Section 2.3). Applications can utilize this capability to encode coarse global positions efficiently while representing fine-grained local details (like avatar kinematics or intricate environmental features) with high precision relative to a parent coordinate. For example, an avatar's core position could be stored on a moderate-resolution A15 grid, while its complex joint movements are encoded on a much finer A15 grid defined locally relative to that core position. This approach optimizes the balance between data size, spatial range, and the level of detail required for different components of a scene.

4.1.5. Example Use Cases

These features position *A15* partitioning as a robust foundation suitable for several demanding applications:

- **Competitive VR Esports:** Where deterministic replays, fairness verification, and efficient network synchronization are critical. The framework's guaranteed consistency across different client hardware and reduced bandwidth requirements are particularly valuable.

- **Industrial Digital Twins:** Requiring precise spatial fidelity and alignment with real-world measurements. The ability to precisely map A15 coordinates to physical units enables accurate synchronization between physical and virtual counterparts.
- **Distributed Physics Simulations:** Leveraging the regular lattice structure for domain decomposition across computing nodes while maintaining state consistency. The deterministic representation helps ensure reproducible results across different simulation environments.
- **Collaborative Mixed Reality:** Where multiple participants with diverse devices need to maintain a shared spatial understanding. The compact representation and cross-platform compatibility facilitate this shared reference frame.
- **Procedural Content Generation:** Benefiting from deterministic spatial addressing to ensure consistent generation results. The hierarchical capabilities allow for efficient representation of multi-scale structures.

The related layoutc project [25], which applies similar compact, deterministic encoding principles to physical layouts, provides a concrete example of these concepts in practice.

4.2. Future Research Directions

The *A15* partitioning framework, while demonstrating immediate utility, also catalyzes numerous avenues for future research aimed at further enhancing immersive experiences and extending the capabilities of spatial computing:

4.2.1. Performance Characterization and Optimization

A critical direction for future work involves comprehensive empirical performance analysis of A15 implementations across various hardware and software environments:

- Measuring quantization costs for different cell geometries (WPH vs. TSP) and comparing optimization strategies
- Benchmarking bandwidth consumption in realistic multi-user scenarios with full kinematic tracking
- Exploring hardware-accelerated implementations, potentially leveraging GPU parallelism or specialized SIMD instructions for efficient nearest-neighbor mapping
- Developing optimized data structures specifically tailored to A15-based spatial partitioning

Such empirical validation would provide crucial guidance for implementers regarding performance expectations and optimization strategies across different hardware targets and use cases.

4.2.2. Advanced Complex Geometry Representation

Developing robust and efficient methods for representing non-cuboid volumes or complex geometric features within the A15 framework (Section 3.3.5) remains a key area for practical improvement. Research could explore:

- Hybrid approaches combining the base A15 grid with techniques like sparse octrees or boundary representations for detail
- Constructive solid geometry (CSG) operations defined relative to A15 cells
- Multi-resolution *A15* representations (spatial Level-of-Detail, LOD [22]) using the framework's inherent support for relative addressing (Section 4.1.4)
- Efficient encoding schemes for representing partially filled or complex boundary regions within a coarser A15 grid

These approaches would address the current limitation of efficiently representing irregular volumes or sparse scenes while maintaining the core benefits of the A15 framework.

4.2.3. Deterministic Physics on the A15 Grid

Extending the determinism guarantees beyond just spatial representation to the actual simulation dynamics represents a promising frontier:

- Developing physics solvers that operate directly on A15 grid points using fixed-point or integer arithmetic
- Creating collision detection algorithms specifically optimized for A15 cell geometries
- Defining temporal integration schemes that maintain determinism across hardware platforms
- Exploring trade-offs between simulation fidelity and guaranteed reproducibility

Such research could lead to fully deterministic simulation environments where not just the spatial coordinates but all dynamic interactions operate with bit-exact consistency across systems.

4.2.4. Framework for Metaverse Interoperability

Standardized protocols built upon *A15* encoding (or similar deterministic lattice-based systems) could define universal mechanisms for:

- Agent state representation across diverse virtual environments
- Coordinate referencing between independently developed worlds
- Interaction semantics and object transformation between spaces
- Capability negotiation and feature discovery between A15-compatible systems

This research direction could establish a foundation for a more coherent, navigable metaverse built on deterministic spatial principles, where objects and avatars can move between independently developed environments while maintaining consistent representation.

4.2.5. AI Integration and Training Efficiency

The structured, deterministic nature of A15-encoded space offers potential advantages for artificial intelligence systems operating in virtual environments:

- Investigating whether the discrete, deterministic nature of A15 representation reduces training noise for spatial AI models
- Exploring potential efficiency gains in reinforcement learning when using consistent spatial representations
- Developing AI navigation and pathfinding algorithms optimized for A15 cell structures
- Creating prediction models that leverage the geometric regularity of the A15 lattice

This research could lead to more efficient training methodologies for spatial AI agents and potentially more robust behavior in deployed systems.

4.2.6. Exploration of Alternative Geometric Structures

While A15 offers a compelling balance of properties derived from crystallography, exploration of alternative partitioning or encoding schemes derived from related or more exotic mathematical structures could yield novel insights or properties advantageous for specific applications. Areas for investigation include:

- **Triply Periodic Minimal Surfaces (TPMS):** Structures like the Gyroid [27] possess complex topology useful for flow simulation or intricate environment design, likely requiring implicit surface representations.
- **Quasicrystalline Patterns:** Non-periodic tilings exhibiting symmetries forbidden in periodic crystals (like 5-fold rotation [28]) could enable partitioning schemes with unique tiling properties or isotropy characteristics.
- **Optimized Point Sets (e.g., Delone Sets):** Computationally generated point distributions balancing criteria like density and minimum separation guarantees [15] could tailor partitioning more closely to specific application requirements than regular lattices.
- **Higher-Dimensional Projections:** Projecting regular polytopes or honeycombs (e.g., from 4D [7]) can generate novel 3D structures with useful partitioning properties.

Pursuing these directions promises to expand the capabilities of structurally informed spatial partitioning. The intersection of crystallographic principles, computational geometry, numerical analysis, and real-time interactive systems represents a fertile territory for extending beyond entertainment into scientific visualization, collaborative design, distributed simulation, robotics, and the broader architecture of spatial computing itself.

5. Glossary of Terms and Notation

This glossary provides definitions for the specialized terminology and mathematical notation used throughout this document.

5.1. Mathematical Notation

ϵ_δ **Global Output Scaling Factor** — The primary scaling parameter (specified via -scale in A15.py) applied to map internal integer coordinates to output units. Determines the physical size represented by one unit of the 96-unit baseline dimension. Typically expressed as a fraction (e.g., 1/64) or power of two (e.g., 2^{-6}).

ϵ_N **Inherent Base Scale** — The minimum scaling factor, expressible as a power of two ($1/2^N$), required to represent the internal integer geometry exactly when mapped to a binary number system. Inferred by analyzing the generated structure's geometric requirements relative to the 96-unit baseline dimension.

ϵ_Δ **Stability Difference** — Measures the mismatch between ϵ_δ and ϵ_N. When $\epsilon_\Delta = 0$, the scaling is stable, meaning the output coordinates can be represented exactly in binary floating-point format relative to the internal grid. Calculated as the residual after subtracting the nearest integer multiple of ϵ_N from ϵ_δ.

N_1 **Basic Unit Width** — The physical dimension corresponding to the 96-unit effective lattice baseline at the chosen output scale ϵ_δ. Calculated as $96 \times \epsilon_\delta$. For the recommended scale of $\epsilon_\delta = 2^{-6}$, this yields $N_1 = 96 \times 2^{-6} = 96/64 = 1.5$ (typically measured in millimeters for human-scale applications).

ULP Unit of Least Precision — The smallest coordinate difference or spatial distance along a principal axis that can be exactly resolved by the quantization scheme at a specific scale ϵ_δ. Equal to the chosen output scale factor, ϵ_δ.

5.2. Technical Terminology

A15 The crystallographic designation for the β–W structure with space group $Pm\bar{3}n$ (No. 223), characterized by high coordination and near-icosahedral local ordering. Named according to the Strukturbericht notation system used in crystallography.

β–W Beta-tungsten, the metallurgical name for the A15 crystal structure, originally observed in tungsten but subsequently found in various intermetallic compounds with the general formula A_3B.

C12 12-coordinated sites within the A15 structure, occupying Wyckoff position 2a, comprising 25% of the basis sites. The local environment around these sites resembles a pyritohedron.

C14 14-coordinated sites within the A15 structure, occupying Wyckoff position 6d, comprising 75% of the basis sites. The local environment around these sites resembles a tetradecahedron.

Determinism Guarantee that identical operations produce bit-identical results across different computing environments, regardless of hardware architecture, operating system, or compiler optimizations.

Isotropy Uniformity of properties across different directions, indicating how "fair" or unbiased a spatial representation is. Higher isotropy means minimal directional artifacts or biases.

Lattice A periodic arrangement of points in space, defined by translational symmetry. In crystallography, a lattice is characterized by its Bravais type and basis vectors.

$Pm\bar{3}n$ Hermann-Mauguin notation for the space group (No. 223) of the A15 structure, describing its complete symmetry operations. The notation indicates: P (primitive unit cell), m (mirror plane), $\bar{3}$ (3-fold rotoinversion axis), n (glide plane).

Quantization The process of mapping continuous coordinates to discrete A15 lattice identifiers. This transformation discretizes space according to the chosen partitioning method (WPH or TSP) and scaling factor.

T_h Point group symmetry ($m\bar{3}$, order 24) describing the local symmetry around sites in the A15 structure. This group is a maximal subgroup common to both cubic symmetry (O_h) and icosahedral symmetry (I_h), enabling the structure's high local isotropy within a periodic lattice.

TSP Tetrastix Prism, a simplified partitioning method using axis-aligned planar faces to divide space into cubic blocks centered on A15 lattice sites. Offers computational efficiency at some cost to isotropy.

WPH Weaire-Phelan Honeycomb, a partitioning method using two distinct polyhedra (pyritohedra and tetradecahedra) in a 1:3 ratio, corresponding to the Voronoi decomposition of A15 lattice points. Provides high isotropy but with more complex cell boundaries.

Wyckoff Position Standardized designation for sets of equivalent points within a crystal structure, identified by multiplicity and site symmetry. Named after Ralph W.G. Wyckoff, who systematized the classification of crystal structures.

96-Unit Baseline The fundamental period or effective unit dimension of the comprehensive internal integer grid required to represent the complete A15 structure without loss of precision. Derived as 4×24 to accommodate basis site offsets of 1/4 within the lattice spacing factor of 24.

5.3. Implementation Parameters in A15.py

prescale Internal integer multiplier applied to shape primitives, establishing primitive resolution (defaults: 20 for WPH, 24 for TSP). Converts base fractional coordinates to integer vertices relative to shape center.

rescale Optional power-of-two scaling applied before generation, adjusting size relative to the 96-unit baseline. Specified via -rescale flag or implicitly through '+/-' suffixes appended to shape names (e.g., pyritohedra++ implies a rescale factor of $2^2 = 4$).

scale Equivalent to ϵ_δ, the final global scaling factor applied after generation to map internal integers to output units. Specified via the -scale=<value> command-line option.

n Controls lattice extent: integer specifies cuboid dimensions (number of lattice cells along each axis); float specifies spherical radius cutoff based on lattice vector magnitude relative to the origin.

-bars Visualization option enabling histogram analysis of floating-point denominators to validate numerical stability. The resulting histogram pattern directly indicates the stability regime ($\epsilon_\Delta = 0$ or $\epsilon_\Delta \neq 0$).

-stix Configuration option activating the TSP partitioning method instead of the default WPH geometry. Trades some isotropy for computational efficiency in point-in-cell tests.

-pop Option to display the visualization in a pop-up window. Can be specified as -pop or with a specific command like -pop=open.

-savefig Option to save the visualization to a file. Default filename is savefig.png unless specified otherwise.

5.4. Integration Strategies

Dual-Representation Integration approach maintaining two parallel coordinate representations: native engine coordinates (typically floats) for internal processing and A15 identifiers for network transmission and verification. Minimizes integration complexity while providing bandwidth and determinism benefits at interface boundaries.

Middleware Integration approach implementing a translation layer between an existing engine and A15 encoding. Transparently converts coordinates without requiring significant engine modifications, facilitating incremental adoption.

Native Implementation Integration approach building spatial data structures and physics directly on A15 identifiers, maximizing determinism throughout the entire pipeline. Requires more engineering investment but provides the strongest guarantees.

Relative Addressing Technique using hierarchical A15 grids at different scales, with fine-grained coordinates defined relative to a parent object's position. Enables efficient multi-scale representation while maintaining precision where needed.

This glossary provides a reference for the specialized terms and mathematical notation used throughout the document. For implementation details and additional technical specifications, refer to the A15.py reference implementation (Section 2.6) and the supplementary information (Section 6).

6. Supplementary Information

This section provides guidelines for replicating the results presented using the accompanying code, details supplementary resources available online, and discusses additional technical considerations relevant to the implementation and interpretation of the *A15* encoding framework.

6.1. Code Availability and Replication Protocols

The figures and structural data presented in this research were generated using the accompanying Python script, A15.py [24], which serves as the reference implementation. To ensure reproducibility, users should have Python 3 installed along with the standard scientific libraries NumPy [17], SciPy [31], and Matplotlib [18]. Understanding the script's execution flow (Table 5) and key parameters, particularly those governing the multi-stage scaling framework (Section 2.3) and numerical stability validation (Section 2.5), is essential for proper use and interpretation of results.

Note on performance: A15.py is a clarity-first reference implementation intended for correctness and pedagogical value. It does not reflect the performance achievable through optimized implementations. Developers targeting real-time applications (e.g., networked VR, physics engines) are advised to implement accelerated versions using fixed-point or SIMD-based quantization pipelines alongside other integration strategies outlined in Section 4.1.2 and design optimized implementations tailored to their application domains.

The core Python script (A15.py), configuration files (*.png.txt) used for figure generation, the LaTeX source for this document (or a version thereof), and extended documentation are publicly available within the Infima Labs space repository on GitHub [20]:

https://github.com/infimalabs/space/

A project overview and supplementary materials may also be found at the project's homepage:

https://infima.space/A15/

The related project layoutc, applying similar compact encoding concepts to paintball field layouts [25], is also available via associated repositories:

https://github.com/infimalabs/layoutc/

Achieving deterministic results, a core goal of this framework, relies crucially on operating within the **Binary** or **Stable** scaling regimes ($\epsilon_\Delta = 0$, Section 2.4.4). The recommended baseline scale of $\epsilon_\delta = 2^{-6}$ (-scale=1/64) generally provides a practical balance for human-scale interactions, offering high precision (approximately 1.5 mm basic unit width N_1, see Section 4.1.3 and Section 6.4.7) while ensuring exact floating-point representability relative to the internal grid for typical configurations. Users are strongly encouraged to employ the -bars analysis feature (Section 2.5) to explicitly verify the stability ($\epsilon_\Delta = 0$) of any custom configurations before deployment in applications where determinism is critical.

Furthermore, while A15.py deterministically implements alternating handedness for local coordination environments based on lattice position (Section 3.3.6), networked applications or systems exchanging *A15*-encoded data **must** establish and consistently apply a shared global orientation convention to ensure interoperability and prevent geometric mirroring between different clients or system components.

6.1.1. Example Replication Commands

The primary figures presented in this document can be regenerated using the A15.py script and the corresponding configuration files (typically named fig-name.png.txt) provided in the supplementary materials repository (Section 6.1). Ensure the script and configuration files are accessible in the execution environment. Use the -i (or -pop) option for interactive viewing (requires a graphical display environment):

Figure 1 (Intro): Nested *A15* (Multiple Scales)
  python3 A15.py -i fig-intro.png.txt
Figure 2 (Internals): Left-Handed $^1/_2$ Unit Cell
  python3 A15.py -i fig-cell2.png.txt
Figure 3 (Partitions): Weaire–Phelan Honeycomb and Tetrastix Prism
  python3 A15.py -i fig-wp.png.txt
  python3 A15.py -i fig-ts.png.txt
Figure 10 (Composite): Representative Example (Unstable Scale)
  python3 A15.py -i fig-main.png.txt
Figure 9 (Histograms): Binary, Stable, and Unstable Scales
  python3 A15.py -i fig-histb.png.txt
  python3 A15.py -i fig-hists.png.txt
  python3 A15.py -i fig-histu.png.txt

For a detailed explanation of all command-line options, parameters, configuration file syntax, and advanced usage, refer to python3 A15.py –help.

6.2. Middleware Integration Example

To illustrate a simple partial-integration strategy, consider a middleware function that translates floating-point positions into A15 lattice identifiers for network transmission, and vice versa on receipt:

Listing 1: Middleware-style round-trip conversion from float to A15 and back.

```
from A15 import encode, decode

# Application float-space position
world_pos = [1.25, 0.75, 2.5]  # meters

# Encode to compact A15 ID for transmission
a15_id = encode(world_pos, scale=1/64)
send_to_network(a15_id)

# On receiver: decode back to float
recv_pos = decode(a15_id, scale=1/64)
render_object_at(recv_pos)
```

This pattern allows developers to gain determinism and bandwidth benefits without deeply modifying the internal coordinate systems of a host engine. The encoded form remains compact and platform-agnostic during transmission, with reconstruction yielding consistent results.

6.3. Practical Implementation Considerations

Beyond the theoretical foundation and validation tools provided in this research, several practical considerations merit attention for developers seeking to implement the A15 framework in real-world applications:

6.3.1. Performance Optimization Strategies

While comprehensive benchmarking remains an area for future work (Section 4.2.1), preliminary experience suggests several approaches to optimize A15 implementation performance:

- **Aligned Coordinate Systems:** Designing application coordinate systems to align with the A15 grid allows quantization to be implemented as simple integer truncation rather than complex geometric tests.
- **Caching A15 Identifiers:** For static objects or slowly changing environments, pre-computing and caching A15 identifiers can amortize quantization costs.
- **Hierarchical Spatial Partitioning:** Combining the A15 grid with higher-level partitioning schemes (e.g., spatial hashing) can accelerate nearest-neighbor searches for quantization.
- **Simpler Cell Geometry for Performance-Critical Paths:** Using the TSP partitioning method (Section 1.3) for paths requiring frequent quantization trading some isotropy for computational efficiency.

6.3.2. Incremental Adoption Path

Rather than attempting a complete transition to A15-based spatial representation, many applications will benefit from an incremental adoption strategy:

1. **Network Transmission Only** - Use A15 encoding solely for transmitting position updates over the network, while maintaining traditional floating-point coordinates for all internal processing.
2. **Authoritative State Recording** - Extend A15 usage to authoritative state storage, enabling deterministic replay and verification capabilities.
3. **Critical System Integration** - Selectively implement A15-aware subsystems for components where determinism is most crucial (e.g., collision detection).
4. **Comprehensive Integration** - Develop fully A15-native physics and simulation systems if warranted by application requirements.

This phased approach allows developers to gain immediate benefits from A15 encoding (bandwidth reduction, improved network consistency) while deferring more complex integration challenges until warranted by specific application needs.

6.3.3. Testing and Verification

Implementing a system that guarantees determinism requires rigorous testing approaches:

- **Cross-Platform Verification:** Test A15-encoded operations across different hardware, operating systems, and compiler settings to verify bit-exact results.

- **Stability Regime Validation:** For any chosen scale, verify that $\epsilon_\Delta = 0$ using techniques similar to the histogram analysis in A15.py (Section 2.5).
- **Boundary Case Testing:** Thoroughly test edge cases near cell boundaries where quantization decisions might be sensitive to implementation details.
- **Deterministic Replay:** Validate that identical initial conditions and inputs reliably produce identical event sequences when using A15 encoding.

These testing approaches help ensure that the theoretical determinism guarantees of the A15 framework translate into practical consistency in deployed applications.

6.4. Supporting Notes and Clarifications

Further details, data, and clarifications related to this research are provided below.

6.4.1. Bandwidth Calculation Basis

The discussion regarding network bandwidth requirements (Section 3.2.2) utilizes a baseline calculation (Equation (7)) assuming a single avatar with 20 tracked joints, each transmitting 3D position (3x 32 bit floats) and an orientation quaternion (4x 32 bit floats) at a 15 Hz update rate ($\approx$67.2 kbit s^{-1}). This kinematic component typically represents only a fraction of the total network traffic in complex interactive applications.

6.4.2. Memory Efficiency Context

The estimate of **50% or more** memory and bandwidth savings (Equation (6), Section 3.2.2) compares storing 3D coordinates using compact integer *A15* identifiers versus raw 32 bit floating-point vectors. A representative scenario assumes a 48 bit integer representation per 3D point for the *A15* identifier (balancing range and precision) compared to 3×32 bit $= 96$ bit for three standard floats. This efficiency gain is most pronounced when quantizing bounded volumes where the extreme dynamic range and full mantissa precision of floats are not required. The exact saving achieved depends on the application's required spatial extent, desired intra-cell resolution (potentially managed via relative addressing, Section 4.1.4), and the chosen bit depth for the *A15* identifier.

6.4.3. Geometric Data Availability

Tables containing the precise internal integer vertex coordinates (relative to shape centers at the relevant prescale value) for the fundamental polyhedra (pyritohedra with various h parameters, tetradecahedra) generated by A15.py functions are available within the code repository (Section 6.1), allowing independent verification of geometric constructions.

6.4.4. Pyritohedra Parameter for Weaire–Phelan Honeycomb Geometry

As implemented in A15.py, invoking the pyritohedron() function with the specific height parameter $h = 7/5$ yields internal integer coordinates that, after appropriate scaling and placement by lattice(), correspond precisely to the vertex coordinates defining the pyritohedral cells within the geometric Weaire–Phelan Honeycomb partition used in this framework (Section 1.3).

6.4.5. Unit of Least Precision (ULP) Definition

Within this framework, when operating at a specific Binary or Stable output scale ϵ_δ (Section 2.4.4), the **Unit of Least Precision (ULP)** represents the smallest coordinate difference or spatial distance along a principal axis that can be exactly resolved by the quantization scheme. This ULP corresponds directly to an integer difference of 1 in the underlying *internal integer* coordinate system (relative to the 96-unit baseline, Section 2.3.3) before the final scaling by ϵ_δ is applied. Therefore, the physical size of the ULP is precisely equal to the chosen output scale factor, ϵ_δ. Features, movements, or discrepancies smaller than ϵ_δ cannot be distinctly represented by the encoding at that scale. Selecting an appropriate ϵ_δ involves balancing the desired spatial resolution (ULP) against the overall spatial range achievable within a fixed-bit integer representation chosen for the *A15* identifier, potentially leveraging relative addressing (Section 4.1.4) to manage this trade-off across different parts of a scene.

6.4.6. Framework Adaptability

While this research focuses intensely on the *A15* phase structure due to its compelling combination of advantageous properties for deterministic spatial encoding, the underlying A15.py software framework possesses inherent adaptability. Key components, particularly the lattice() function for replicating geometric units according to symmetry rules and the visualization tools within the figure() function, could be modified or extended to generate and visualize other crystal lattice types or different space-filling structures, offering a versatile platform for broader geometric exploration, albeit likely requiring non-trivial adaptation of the core geometric and scaling logic.

6.4.7. Interpretation of Figure Annotations

Annotations visible in figures generated by A15.py with the -bars option (e.g., Figure 10) directly illustrate key stability concepts discussed in Section 2.4. The calculated value N_1 (labeled "basic unit width" or similar in some outputs) shown in the histogram sidebar represents the physical dimension corresponding to the chosen output scale ϵ_δ applied to the fundamental internal lattice dimension (the 96-unit baseline, Section 2.3.3). For the unstable scale $\epsilon_\delta = 1/96$ shown in Figure 10, this results in $N_1 = 96 \times (1/96) = 1.0$ (assuming millimeters as the base unit, thus 1.0 mm). In contrast, the recommended stable scale $\epsilon_\delta = 1/64$ yields $N_1 = 96 \times (1/64) = 1.5$ (or 1.5 mm, Section 4.1.3). The displayed ϵ_Δ value explicitly confirms the calculated stability difference for the configuration (e.g., $\epsilon_\Delta \approx 0.0104 \neq 0$ for the unstable case in Figure 10, confirming ϵ_δ is not an integer multiple of ϵ_N), providing direct numerical validation alongside the visual histogram representation.

Acknowledgements

This work benefited significantly from the democratization of knowledge, particularly Wikipedia's extensive collection of mathematical and physical concepts. The accessibility of such resources proved invaluable.

Unabashed acknowledgement is given to the transformative role of artificial intelligence assistants in accelerating the development and articulation of these ideas. Their capabilities enabled rapid iteration and refinement of concepts and language.

Special appreciation is extended to the Infima Labs team for their confidence and trust throughout this research. Their shared vision for advancing spatial computing was—and continues to be—a wellspring of inspiration.

Above all, deep gratitude is expressed to the author's wife and family for their unwavering support and patience throughout this work's development. Their encouragement and understanding were instrumental in bringing these ideas to fruition.

References

[1] Aroyo M.I., ed.: *International Tables for Crystallography, Volume A: Space-Group Symmetry.* International Union of Crystallography, Wiley, Chester, UK, 6th ed., 2016. ISBN 978-0470689080. Authoritative reference for crystallographic space groups and symmetry operations.

[2] Ashcroft N.W., Mermin N.D.: *Solid State Physics.* Holt, Rinehart and Winston, New York, 1976. ISBN 978-0030839931. Classic textbook including derivation of the crystallographic restriction theorem.

[3] Bentley J.L.: Multidimensional Binary Search Trees Used for Associative Searching. In: *Communications of the ACM*, vol. 18(9), pp. 509–517, 1975. URL http://dx.doi.org/10.1145/361002.361007. Original paper introducing k-d trees for spatial partitioning.

[4] California State Legislature: California Consumer Privacy Act of 2018 (CCPA), 2018. URL https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5. As amended by the California Privacy Rights Act of 2020 (CPRA).

[5] Claypool M., Claypool K.: Latency and player actions in online games. In: *Communications of the ACM*, vol. 49(11), pp. 40–45, 2006. URL http://dx.doi.org/10.1145/1167838.1167860. Study of the effects of network latency on different types of online games.

[6] Conway J.H., Sloane N.J.A.: *Sphere Packings, Lattices and Groups*, *Grundlehren der mathematischen Wissenschaften*, vol. 290. Springer-Verlag, New York, 3rd ed., 1999. ISBN 978-0387985855. URL http://dx.doi.org/10.1007/978-1-4757-6568-7. Comprehensive reference on lattices and sphere packing problems.

[7] Coxeter H.S.M.: *Regular Polytopes*. Dover Publications, New York, 3rd ed., 1973. ISBN 978-0486614809. Definitive work on regular polytopes, originally published by Methuen in 1947.

[8] Coxeter H.S.M., Moser W.O.J.: *Generators and Relations for Discrete Groups*, *Ergebnisse der Mathematik und ihrer Grenzgebiete*, vol. 14. Springer-Verlag, Berlin, 3rd ed., 1972. ISBN 978-3540058465. URL http://dx.doi.org/10.1007/978-3-662-21943-0. Important reference for the study of discrete groups, including crystallographic groups.

[9] Epic Games: Coordinate Space Terminology, 2023. URL https://docs.unrealengine.com/5.3/en-US/coordinate-space-terminology-in-unreal-engine/. Unreal Engine uses a left-handed coordinate system. Accessed: 2025-04-05.

[10] European Parliament and Council of the European Union: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation), 2016. URL https://eur-lex.europa.eu/eli/reg/2016/679/oj. Comprehensive EU data protection regulation that became enforceable on May 25, 2018.

[11] Finkel R.A., Bentley J.L.: Quad Trees: A Data Structure for Retrieval on Composite Keys. In: *Acta Informatica*, vol. 4(1), pp. 1–9, 1974. URL http://dx.doi.org/10.1007/BF00288933. Seminal paper introducing quadtrees, the 2D precursor to octrees.

[12] Frank F.C., Kasper J.S.: Complex alloy structures regarded as sphere packings. I. Definitions and basic principles. In: *Acta Crystallographica*, vol. 11(3), pp. 184–190, 1958. URL http://dx.doi.org/10.1107/s0365110x5800048x. Seminal paper introducing the concept of Frank-Kasper phases.

[13] Frank F.C., Kasper J.S.: Complex alloy structures regarded as sphere packings. II. Analysis and classification of representative structures. In: *Acta Crystallographica*, vol. 12(7), pp. 483–499, 1959. URL http://dx.doi.org/10.1107/S0365110X5900149X. Continuation of the foundational work on Frank-Kasper phases.

[14] Goldberg D.: What Every Computer Scientist Should Know About Floating-Point Arithmetic. In: *ACM Computing Surveys*, vol. 23(1), pp. 5–48, 1991. URL http://dx.doi.org/10.1145/103162.103163. Seminal paper on the pitfalls and proper usage of floating-point arithmetic.

[15] Gruber P.M.: *Convex and Discrete Geometry*, *Grundlehren der mathematischen Wissenschaften*, vol. 336. Springer, Berlin, 2007. ISBN 978-3540711322. URL http://dx.doi.org/10.1007/978-3-540-71133-9. Covers Delone (Delaunay) sets and space-filling polyhedra.

[16] Guttman A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pp. 47–57. ACM, Boston, Massachusetts, USA, 1984. URL http://dx.doi.org/10.1145/602259.602266. Seminal paper introducing R-

tree spatial indexing structures.

[17] Harris C.R., Millman K.J., van der Walt S.J., Gommers R., Virtanen P., Cournapeau D., Wieser E., Taylor J., Berg S., Smith N.J., Kern R., Picus M., Hoyer S., van Kerkwijk M.H., Brett M., Haldane A., del Río J.F., Wiebe M., Peterson P., Gérard-Marchant P., Sheppard K., Reddy T., Weckesser W., Abbasi H., Gohlke C., Oliphant T.E.: Array programming with NumPy. In: *Nature*, vol. 585(7825), pp. 357–362, 2020. URL http://dx.doi.org/10.1038/s41586-020-2649-2. The definitive paper describing the NumPy library for scientific computing in Python.

[18] Hunter J.D.: Matplotlib: A 2D Graphics Environment. In: *Computing in Science & Engineering*, vol. 9(3), pp. 90–95, 2007. URL http://dx.doi.org/10.1109/MCSE.2007.55. The original paper describing the Matplotlib visualization library for Python.

[19] IEEE: IEEE Standard for Floating-Point Arithmetic. Tech. Rep. 754-2019, Institute of Electrical and Electronics Engineers, New York, NY, USA, 2019. URL http://dx.doi.org/10.1109/IEEESTD.2019.8766229. Revision of IEEE Std 754-2008.

[20] Infima Labs: Infima Space Repository, 2023. URL https://infima.space. Accessed: 2025-04-05.

[21] Kusner R., Sullivan J.M.: Comparing the Weaire-Phelan equal-volume foam to Kelvin's foam. In: *Forma*, vol. 11(3), pp. 233–242, 1996. Analysis of the relative efficiency of the Weaire-Phelan structure.

[22] Luebke D., Reddy M., Cohen J.D., Varshney A., Watson B., Huebner R.: *Level of Detail for 3D Graphics*. Morgan Kaufmann, San Francisco, CA, 2002. ISBN 978-1558608382. Comprehensive guide to LOD techniques for real-time 3D graphics.

[23] Python Software Foundation: Built-in Types — Python 3 documentation: float.as_integer_ratio, 2025. URL https://docs.python.org/3/library/stdtypes.html#float.as_integer_ratio. Accessed: 2025-04-05.

[24] Risinger C.A.: A15.py: A15 Phase Visualizer, 2024. URL https://github.com/infimalabs/space/blob/main/A15/A15.py. Accessed: 2025-04-05.

[25] Risinger C.A.: layoutc: Paintball Field Layout Compressor, 2024. URL https://github.com/infimalabs/layoutc. Accessed: 2025-04-05.

[26] Samet H.: *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990. ISBN 978-0201502558. Comprehensive reference on spatial data structures including octrees, R-trees, and quadtrees.

[27] Schoen A.H.: Infinite periodic minimal surfaces without self-intersections. Tech. Rep. NASA TN D-5541, NASA, 1970. URL https://ntrs.nasa.gov/citations/19700020214. Classic work describing triply periodic minimal surfaces (TPMS) including the Gyroid.

[28] Shechtman D., Blech I., Gratias D., Cahn J.W.: Metallic Phase with Long-Range Orientational Order and No Translational Symmetry. In: *Physical Review Letters*, vol. 53(20), pp. 1951–1953, 1984. URL http://dx.doi.org/10.1103/PhysRevLett.53.1951. Seminal paper reporting the discovery of quasicrystals, for which Shecht-

man was awarded the Nobel Prize in Chemistry in 2011.

[29] Thomson W.: On the division of space with minimum partitional area. In: *Philosophical Magazine*, vol. 24(151), pp. 503–514, 1887. URL http://dx.doi.org/10.1080/14786448708628135. Lord Kelvin's original paper proposing the bitruncated cubic honeycomb.

[30] Unity Technologies: Understanding Vector Arithmetic in Unity, 2024. URL https://docs.unity3d.com/Manual/UnderstandingVectorArithmetic.html. Unity uses a left-handed coordinate system. Accessed: 2025-04-05.

[31] Virtanen P., Gommers R., Oliphant T.E., Haberland M., Reddy T., Cournapeau D., Burovski E., Peterson P., Weckesser W., Bright J., van der Walt S.J., Brett M., Wilson J., Millman K.J., Mayorov N., Nelson A.R.J., Jones E., Kern R., Larson E., Carey C.J., Polat I., Feng Y., Moore E.W., VanderPlas J., Laxalde D., Perktold J., Cimrman R., Henriksen I., Harris C.R., Quintero E.A., Archibald A.M., Ribeiro A.H., Pedregosa F., van Mulbregt P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. In: *Nature Methods*, vol. 17, pp. 261–272, 2020. URL http://dx.doi.org/10.1038/s41592-019-0686-2. The definitive paper describing the SciPy library for scientific computing in Python.

[32] Weaire D., Hutzler S.: *The Physics of Foams.* Oxford University Press, Oxford, 2001. ISBN 978-0198505891. Comprehensive reference on foam structures, including discussion of the Weaire-Phelan structure.

[33] Weaire D., Phelan R.: A counter-example to Kelvin's conjecture on minimal surfaces. In: *Philosophical Magazine Letters*, vol. 69(2), pp. 107–110, 1994. URL http://dx.doi.org/10.1080/09500839408241577. Describes the Weaire-Phelan structure which has approximately 0.3% less surface area than Kelvin's structure.

Affiliations

C Anthony Risinger
Infima Labs, https://infima.space, e-mail: anthony@infima.space
Snap Game Services, https://snap.gs, e-mail: anthony@snap.gs

Revised: D:20250417
Released: D:20231003
Accepted: –

Table 1
Comparison of Spatial Organization Approaches.

Characteristic	**Traditional Spatial Indexing** (e.g., Octree [11], R-tree [16], k-d tree [3])	***A15* Crystallographic Partitioning**
Main Objective	Efficient query/retrieval of existing, often arbitrary, data	Uniform, deterministic spatial discretization and efficient coordinate encoding
Partition Strategy	Data-driven; adaptive boundaries; often hierarchical; structure follows data	Structure-driven; regular lattice; predetermined boundaries (via WPH/TSP local methods); data follows structure
Symmetry Awareness	Generally low; structure adapts to data, often breaking ambient symmetries	High; leverages crystallographic symmetry (T_h, $Pm\bar{3}n$) of the imposed lattice
Coordinate Handling	Typically preserves input precision (e.g., float coordinates)	Quantizes coordinates to discrete representations (*A15* identifiers, often integers)
Memory Usage	Variable, depends on data density/distribution; includes tree/node overhead	Predictable based on defined scale/volume; highly efficient storage using integer identifiers
Isotropy	Variable; depends heavily on data distribution and algorithm specifics	High local isotropy inherent in the *A15* structure, especially when using Weaire–Phelan Honeycomb partitioning
Neighborhood Info	Requires explicit tree traversal or complex range/proximity queries	Implicit, regular neighbor relationships directly derivable from the lattice structure
Temporal Stability	Partitioning structure can change significantly as data moves or updates	Fixed partitioning grid provides temporal coherence for coordinate mapping (though content moves within)
Determinism Guarantee	Generally low; susceptible to float variance affecting boundary tests/traversal	High; guaranteed bit-level consistency achievable with stable scaling regimes ($\epsilon_\Delta = 0$)
Main Use Case	Databases, GIS [26], dynamic collision detection, view frustum culling	Foundational spatial fabric for deterministic VR/simulations, compact coordinate encoding, networked state consistency, verifiable replays

Table 2

Comparison of Relevant Cubic Space Groups.

Structure Type	**No.**	**HM Symbol**	**Point Group**	**Order**	**Bravais Lattice**
***A15* Type**	**223**	$\boldsymbol{Pm\bar{3}n}$	$\boldsymbol{m\bar{3}\ (T_h)}$	**24**	**Primitive (cP)**
Simple Cubic (SC)	221	$Pm\bar{3}m$	$m\bar{3}m\ (O_h)$	48	Primitive (cP)
BCC Type (e.g., W)	229	$Im\bar{3}m$	$m\bar{3}m\ (O_h)$	48	Body-Centered (cI)
FCC Type (e.g., Cu)	225	$Fm\bar{3}m$	$m\bar{3}m\ (O_h)$	48	Face-Centered (cF)

Data sourced from International Tables for Crystallography, Vol A [1]. HM Symbol: Hermann-Mauguin notation.

Table 3

Comprehensive Comparison of Lattice Structures for Spatial Partitioning

Lattice	**Coord. #**	**Isotropy**	**Binary?**	**Uniform?**	**Complex?**
Simple Cubic (SC)	6	Low	High	Perfect	Low
Body-Centered Cubic (BCC)	8	Medium	Medium	Perfect	Medium
Face-Centered Cubic (FCC)	12	High	Low	Perfect	Medium
***A15* (A15)**	**13.5 avg**	**High**	**High**	**Dual-type**	**High**

Table 4
Summary of Key Scaling Parameters and Factors in A15.py.

Parameter/-Factor	**Role and Implementation in** A15.py
prescale	Internal integer multiplier applied within shape functions (e.g., pyritohedron()). Converts base fractional coordinates (0, 1/4, 1/2, etc.) to integer vertices relative to shape center. Establishes primitive resolution (Defaults: 20 WPH, 24 TSP).
Lattice Spacing Factor (24)	Fixed factor used in lattice(). Multiplies integer lattice vectors xyz to determine nominal lattice point positions relative to origin offset o, measured in units defined by prescale.
Effective Lattice Unit Factor (96)	Derived factor ($4 \times 24 = 96$). Represents the fundamental period of the internal integer grid along a principal axis needed to accommodate *A15* basis site offsets (like 1/4) precisely. **This is the internal baseline dimension** relative to which inherent precision ϵ_N and output scale ϵ_δ are compared.
rescale	Optional pre-generator power-of-two scaling (via -rescale flag or +/- suffixes). Multiplies internal integer coordinates, adjusting size relative to the 96-unit baseline *before* final output scaling. Affects the resulting ϵ_N.
scale (ϵ_δ)	Final global scaling factor (-scale=<value>) applied by figure() after internal integers are generated. Maps internal integers (relative to the 96-unit baseline) to output coordinates (typically float). Sets real-world size and determines numerical stability regime (ϵ_Δ) by comparison with ϵ_N. Accepts various formats (integer, fraction, power, float).
n	Controls lattice extent (-n=<value>). Integer specifies cuboid dimensions (number of lattice cells defined by the 96-unit baseline along axes); float specifies a spherical radius cutoff based on lattice vector magnitude relative to the origin.

Table 5

Core Workflow Stages in the A15.py Implementation.

Stage	**Key Functions & Purpose in** A15.py
Configuration Processing	flags(), configuration() Parses command-line arguments/files (*.txt). Interprets parameters (e.g., scale, rescale, n, prescale, stix, visualization flags like -edges, -faces, -bars). Handles defaults, automatic configurations (-auto), inheritance (colon notation). Determines final parameter set for generation and visualization.
Shape Definition	pyritohedron(), tetradecahedra() Constructs fundamental geometric units based on parameters. Applies internal prescale factor converting base fractional coordinates to integer vertices relative to shape center. Handles local symmetry (handedness).
Lattice Generation	lattice() Replicates shape units across 3D grid based on extent n and at filter (for basis site mapping). Calculates center positions using integer vectors xyz, origin offset o, and the fixed * 24 spacing factor (relative to prescaled units), implicitly establishing the 96-unit effective baseline. Yields (vertex_array, config_object) pairs representing internal integer geometry.
Scaling & Stability Analysis	figure() Collects internal integer vertex arrays. Applies global scale parameter (ϵ_δ) mapping internal integers (relative to the 96-unit baseline) to output floats. Infers inherent base scale ϵ_N. Calculates stability difference ϵ_Δ (verifying if $\epsilon_\delta = m \cdot \epsilon_N$ for $m \in \mathbb{Z}^+ \implies \epsilon_\Delta = 0$).
Visualization & Validation	figure() Renders 3D geometry via Matplotlib [18] using specified options (-edges, -faces, etc.). Uses SciPy [31] for hull calculations if needed. If -bars requested, performs stability analysis (via float.as_integer_ratio() [23] denominators) and generates histogram (Figure 9), visualizing the stability regime and displaying ϵ_Δ. Outputs to screen (-interactive option implies pop-up) or file (-savefig). Includes annotations (Figure 10). Relies heavily on NumPy [17] throughout.