ChatGPT

Platonic Spin Networks for Quantum Geometry and Attention Interfaces

Spin Networks on Platonic Solid Graphs (Quantum Geometry Basics)

Spin Networks in LQG: In loop quantum gravity (LQG), space is described by **spin networks** – graphs with edges labeled by $SU(2)$ representations (spins) and nodes where those edges meet [1]. Each edge's half-integer *spin label* corresponds to a quantum of area, and each node (with multiple edges) carries an *intertwiner* describing a quantum volume [2] [3]. In simple terms, a spin network is a web of "chunks" of geometry: edges are like elementary surface patches and nodes like grains of volume [4] [3]. This duality can be visualized by pairing a node and its emanating edges with a polyhedron: the node sits at the polyhedron's center, each adjacent edge punctures one face, and the spin on that edge is related to the area of that face [5] [3]. For example, a node with six edges can be dual to a cube (six faces), as illustrated below, where the central dot represents the cube's volume and each line pierces a face of the cube:

A node with 6 emanating edges (purple) can be visualized as the center of a cube (blue) – each edge is dual to one face of the cube [4]. In LQG, labeling each edge with a spin j endows the corresponding face with a quantized area proportional to $j(j+1)$ [3].

Polyhedra as Quantum Geometry: A remarkable result in LQG is that **any intertwiner (node) of valence N can be interpreted as a quantum polyhedron with N faces** [6] [7]. The classical counterpart is given by Minkowski's theorem: a set of N vectors (outward face normals) with magnitudes (areas) that close tip-to-tail defines a unique convex polyhedron [6]. At the quantum level, the $SU(2)$ spin on each edge fixes the face area (to an eigenvalue of the area operator), and the intertwiner imposes the **closure constraint** (the sum of face vectors is zero) so that the faces can form a closed polyhedron [6] [7]. In this way, a spin-network state with many nodes can be viewed as a patchwork of "quantum polyhedra," each face shared by two polyhedra (since each edge connects two nodes) [8] [9]. For instance, a single 4-valent node (four edges) corresponds to a **quantum tetrahedron**, while a node with 20 edges would correspond to a dodecahedron, etc. This provides a beautiful **geometric interpretation**: spin networks are essentially discrete 3D geometries, with each node's polyhedron contributing volume and each edge's spin contributing area to the adjacent faces [7]. Coherent states peaked on classical geometry even allow these quantum polyhedra to approximate smooth shapes in a semiclassical limit [7].

Edges, Holonomies, and Phases: In LQG each edge of the graph also carries a **holonomy** – essentially an $SU(2)$ group element obtained by exponentiating the gravitational connection along that edge [10]. The spin network state is a gauge-invariant function of these holonomies. Intuitively, you can think of each edge as not only having a discrete area (spin) but also an orientation/phase information (related to how the frames of the two polyhedra at its ends are rotated relative to each other) [11] [12]. These conjugate "twist" angles between faces are part of what are known as **twisted geometries** [13]. They encode intrinsic curvature: if you go around a closed loop of edges on the network, the product of holonomies (a Wilson

loop) may have a net rotation – indicating a deficit angle or curvature concentrated on that loop (analogous to Regge calculus) [14] [15]. In a Platonic graph, such as an **icosahedral spin network**, one could imagine associating deficit angles to closed cycles that correspond to curvature quanta at those "faces" of the discrete geometry. **Phase indicators** can thus be assigned to edges or loops: for example, using color hue or rotating arrows on an edge to show the $U(1)$ phase corresponding to the holonomy's trace (exponential of curvature) – a technique akin to color-coding phase in complex wavefunctions or spherical harmonic plots.

Toy Universes on Regular Polyhedral Complexes

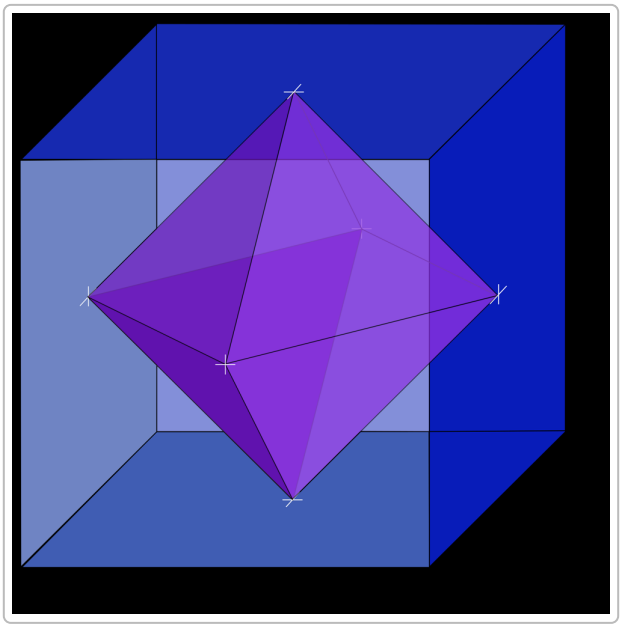
Real quantum gravity calculations often use simple graphs as **mini-universes**. Interestingly, **Platonic solids** and their networks serve as excellent toy models due to their high symmetry. Researchers have studied cosmological scenarios using minimal graphs that correspond to triangulations of space. A prime example is the **"dipole cosmology"**, where physical space is taken as a 3-sphere discretized into just two tetrahedral cells glued together [16]. The dual graph of this triangulation is two nodes connected by four edges – essentially the graph of an octahedron (since an octahedron has 6 vertices, but here we have 2 "super-nodes" each connected by 4 distinct links) representing the gluing of 4 faces. This simple spin network captures a homogeneous universe with only a few degrees of freedom. In fact, the dipole model can reproduce the dynamics of an anisotropic Bianchi IX cosmology (mixmaster universe) when interpreted properly [17]. **Loop quantum cosmology** thereby reduces to tweaking a handful of spin labels (edges) and intertwiners (nodes) – a drastic simplification that still yields insights into quantum bounces and anisotropy behavior [18].

Another elegant toy universe arises from the **complete graph on 5 nodes (K$_5$)**, which is the dual of a decomposition of the 3-sphere into 5 tetrahedra (forming a 4-simplex) [19]. This K$_5$ spin network has $\binom{5}{2}=10$ edges and 5 nodes, and can be viewed as the simplest closed "quantum space" with all nodes maximally interconnected. Such a highly symmetric graph could serve as a model for a small closed universe where each node/polyhedron represents a region of space and every pair of regions shares a boundary. *Dalton (2021) even speculates on Platonic solids forming the fabric of space:* he proposes a picture where space consists of Platonic cells whose corners and edges correspond to Rovelli's spin-network nodes and links [20]. In that spirit, one might imagine a **cubic** spin network (8 nodes, 12 edges) or an **icosahedral** spin net (12 nodes, 30 edges) as a "mini-universe" on which fields live. These highly symmetric graphs make it easier to impose homogeneous or isotropic conditions – every edge might carry the same spin j, for example, encoding a nearly uniform geometry (perhaps a discrete analog of a round sphere or torus when embedded appropriately).

Spatial Adjacency vs. Network Topology: It's important to note that while we often draw these graphs embedded in space for intuition (e.g. a cube graph drawn as a cube), in quantum gravity the edges **do not have predetermined lengths** – the geometry is entirely encoded in the spin labels [21]. One can therefore choose any convenient graph topology as a "scaffold" for a toy model, and assign spins to edges to define a geometry. A cube graph could represent 8 "chunks" of space with some adjacency relationships (like a 3D grid with periodic identification), whereas a dodecahedral graph (the graph of a dodecahedron's vertices and edges) might represent a spherical pack of 20 regions. By tuning the edge spins, one can introduce curvature: e.g. if all spins are equal, the network might be in a quasi-regular state, whereas if one region has very high spin on edges to its neighbors, that could indicate a larger area face and thus a "bump" in geometry (like a concentrated curvature around that region). This suggests an avenue to simulate phenomena like **inhomogeneities** or curvature concentrations on a toy universe by adjusting a few spins.

Duals, Symmetries, and Cognitive Metaphors

One special feature of Platonic graphs is the presence of **dual polyhedra** and symmetry mappings. The five Platonic solids come in dual pairs (except the tetrahedron which is self-dual): **Cube–Octahedron** and **Dodecahedron–Icosahedron**. A spin network on one can be naturally mapped to the dual network on the other by exchanging nodes and faces. For example, the **cube graph** (8 nodes, 12 edges, each node valence 3) is dual to the **octahedral graph** (6 nodes, 12 edges, each node valence 4). In the cube network, nodes represent volumes at cube corners and edges represent shared faces between cubes; in the octahedral network, nodes represent volumes at octahedron centers and edges represent shared faces between octahedra. **Toggling between a Platonic solid and its dual** is thus a powerful visual idiom – it exchanges the roles of "cells" and "holes," reminiscent of Poincaré duality in meshes. In an interactive setting, one could allow the user to switch view from a spin network (edges connecting volume nodes) to a **dual surface network** (faces connecting dual nodes). The Donut of Attention's UI could exploit this by an overlay where, say, the cube's vertices (original spin network nodes) fade out and the cube's face centers (dual nodes of an octahedron) light up, with connecting lines drawn between adjacent face centers (forming an octahedral graph). This would illustrate abstract dualities or complementary perspectives on the data.



Dual polyhedra as an interactive metaphor: the cube (blue) and octahedron (purple) are dual – the octahedron's vertices align with the cube's face centers, and vice versa. In a spin network overlay, a cube-graph of 8 nodes (each at a cube vertex) could be morphed into an octa-graph of 6 nodes (each at a cube face center) to highlight dual network structures or "complementary" sets of relationships.

Such dualities resonate beyond geometry. In cognitive and attention networks, one often encounters complementary systems (for instance, the brain's **Default Mode Network vs. Dorsal Attention Network** are antagonistic yet jointly exhaustive). It is enticing to use the dual Platonic graphs as metaphors: e.g. an idea that one network of nodes (say, blue cube nodes) represents an inward-focused set of processes, while the dual network (purple octa nodes connecting through the cube's faces) represents an outward-focused set – together filling the cognitive "space." *This isn't just fantasy:* researchers have conjectured that brain functional activity may reside on toroidal or spherical manifolds. Notably, Tozzi & Peters (2016) propose that

thoughts trace out a donut-like trajectory in the brain, i.e. neural activity lives on a torus where opposite points correspond to complementary brain states [22]. In such a toroidal model, one could imagine circular overlays for two dual networks on the torus's surface. Likewise, Platonic spheres (icosahedral nets) have been used to map cortical data – for example, an **icosahedral geodesic grid** is often used in brain imaging to model the cortical surface for spherical convolutional networks [23]. These examples indicate that using **regular polyhedral graphs as cognitive network metaphors** is quite plausible. A highly symmetric network can represent "ideal" or baseline connections in a brain or AI attention architecture, and deviations or colorings on that network can show activation patterns, stresses, or misalignments from the ideal symmetry.

Design and Visualization Features for "Spin Network Mode"

To integrate the above ideas into the **Donut of Attention UI** (a Three.js-based toroidal interface), we can formulate a set of visualization and interaction features. The goal is to present a **Spin Network Mode** where a Platonic solid (or torus-wrapped polyhedral complex) becomes a canvas for real-time geometric and graph-based overlays. Below we outline key features and how they can be implemented, referencing both scientific correctness and UI/UX inspirations:

- **Choice of Graph (Platonic Topology):** The mode could offer a selection of Platonic graphs – tetrahedron (4 nodes, 6 edges), cube/octahedron (dual 8–6 nodes, 12 edges), dodecahedron/icosahedron (dual 20–12 nodes, 30 edges). Each provides a different "world" to embed information. A table of their properties is given below for reference:

Platonic Solid (Graph)	Nodes (Volumes)	Edges (Links)	Node Degree	Dual Solid (Graph)
Tetrahedron (K$_4$ graph)	4	6	3	Self-dual (Tetrahedron)
Cube (hexahedron)	8	12	3	Octahedron (dual)
Octahedron	6	12	4	Cube (dual)
Dodecahedron	20	30	3	Icosahedron (dual)
Icosahedron	12	30	5	Dodecahedron (dual)

Each graph can be pre-stored (Three.js has `PolyhedronGeometry` for vertices; we can also hardcode adjacency). For instance, the **tetrahedral graph** (K$_4$) is explicitly constructed in Noah Mackay's Wolfram model by listing 4 vertices and all 6 edges connecting them [24]. We can similarly define the edge lists for cube, etc., or use existing libraries (e.g. networkx or hardcoded arrays). The UI could allow switching between these graphs, perhaps even animating a morph (using the fact that all Platonic solids can be inscribed in the same sphere for a smooth transition).

- **3D Layout on a Torus:** Given the Donut UI is torus-based, one creative approach is to **embed the graph on a torus surface** or around a torus shape. For example, the nodes of a cube could be positioned on a ring (with appropriate 3D offsets to mimic the cube shape). Alternatively, the torus itself might act as an interactive "ring menu" to select which Platonic network to display, while the network floats in the middle. It might be more intuitive to simply place the Platonic solid at the

center of the torus (like a crystal ball in a ring), and the torus rim provides UI controls. Whichever the case, Three.js can handle both the torus (with `TorusGeometry`) and the graph model simultaneously.

- **Real-Time Edge/Node Phase Indicators:** To convey dynamic phase or flow on the network, we can use visual cues on edges and nodes:

- *Edges:* Use **animated colors or oscillating glow** along edges to indicate phase angles. For instance, map phase $\phi \in [0,2\pi)$ to a color wheel (hue shift) – an edge could smoothly cycle through colors if its phase rotates over time. Another option is a small **arrowhead or moving texture** on the edge that rotates around or slides, indicating direction of a phase current (like arrows on oriented edges [1] but animated). This is similar to visualizing AC electrical phase on transmission lines or phase in quantum mechanics demos (where phase=0 is bright vs phase=π is dark, etc.). Nodes could display phase by a **rotating sprite** or orientation: e.g. each node could be a small disk that spins around its axis at a rate proportional to some phase or whose tilt indicates phase angle.

- *Holonomy and Curvature:* If the user creates a **closed loop** on the graph (by selecting a cycle of edges), the UI can display the net holonomy/phase gained. For example, upon selecting a loop, a **floating membrane** can appear spanning that loop (a translucent polygon) and colored according to the curvature (red for positive curvature/deficit angle, blue for negative, etc.). This directly illustrates how curvature would be localized on a spin network: a nontrivial holonomy means that membrane is "strained." Such an overlay is analogous to a **Wilson loop visualization** – helpful pedagogically to show how spin networks encode curvature.

- **Curvature and Stress Heatmaps:** Beyond binary phase, continuous values like curvature, probability weight, or stress can be mapped to color intensity or deformations:

- Use **color heatmaps** on edges or faces (membranes) to indicate magnitude of some quantity (e.g. an edge's deviation from flatness, or uncertainty). The Einstein Online demo in LQG used color to represent area eigenvalues on faces (red = small area, violet = large) [25]. We can adopt a similar scheme: for instance, edges with higher spin could glow brighter or thicker (since spin ~ area ~ "amount of geometry") and faces (floating membranes) could interpolate a color based on the average spin of their boundary.
- Use **thickness or distortion:** an edge's thickness or a slight pulsation can reflect its weight (like a spring under tension). If an edge's spin is increased by the user, we could subtly extend that edge's length or thickness to show it "growing" area. This is tricky because in a true quantum geometry sense, edges don't have physical length, but a bit of artistic liberty here aids intuition ("bigger spin = fatter connection"). Alternatively, **spherical harmonic modes** could be projected onto the polyhedron's surface to show distributions – e.g. a low-order spherical harmonic Y_{lm} pattern can illustrate an anisotropic mode of geometry or attention across the network (with lobes of positive/negative shown in different colors).

- **Stress or uncertainty:** Uncertainty in a measurement (say a superposition of spin values) might be shown by a **fuzzy halo or jitter** on the corresponding edge or node. For example, a transparent glow that expands and contracts (as if the edge is "blurry") could indicate the edge is not in a definite state. This metaphor aligns with uncertainty clouds in atomic orbital visuals or error bars in charts, translated into the network domain.

- **Interactive Editing of Spins/Weights:** Users should be able to click on an edge or node and adjust its parameters in real-time. For edges, the primary parameter is the spin (an integer or half-integer). A UI slider or even a direct manipulation (e.g. "pinch" an edge to increase its weight) could be provided. On change, the visualization should update: the edge label (if numeric labels are shown) updates to the new $2j$ or j value, and any dependent visuals (thickness, color, adjacent face area color) update accordingly. Since these networks are small, the updates are essentially instantaneous. For educational feedback, one could compute derived quantities like the node's **volume** from its surrounding spins and display that. (Bianchi et al. gave formulas for a polyhedron's volume from face areas [26]; implementing those might be complex, but one could approximate volume qualitatively by e.g. a bubble size at the node).

- Additionally, allow toggling the **orientation (phase)** of an edge's holonomy: perhaps the user can twist an edge like a dial. This could be represented by a small dial widget when an edge is selected, or keyboard input to shift phase. As the user changes it, the edge's color will rotate through the hue cycle, and if a closed loop's total phase is being tracked (as described above), the user can attempt to "tune" it to zero to flatten the curvature – a fun interactive challenge.

- **Labels and Information:** Each edge and node can have optional labels. Edges might be labeled with spin values j or even the full representation dimensionality $(2j+1)$. Nodes might be labeled with an index or with the type of intertwiner (for example, a node could be labeled by volume or by a set ${j_f}$ of its incident spins for clarity). In an attention-architecture context, labels could also be something semantic (like each node is a concept or each edge an interaction strength). The **Graph-Visualization** library by David Piegza (Three.js-based) uses spheres for nodes and lines for edges and notes that adding edge labels or making edges selectable is straightforward [27] – this existing code (610 stars on GitHub) could be repurposed for the core graph rendering, then extended with our custom overlays.

- **Floating Membranes (Surfaces):** As mentioned, the UI can draw polygonal translucent surfaces to represent faces of the polyhedron or cycles in the graph. These "membranes" help in visualizing dual structures or areas. For a given Platonic solid, one can predefine the faces (each face is a cycle of edges). By toggling "membrane view," the user would see a soap-film like sheet over each face of the polyhedron, turning the wireframe graph into a solid-looking polyhedron. This is not just cosmetic: the membranes can be colored or animated to convey data. For example, if the user wants to see the **alignment or duality** between two different networks, one network could be drawn as the wireframe and another as membranes. Consider overlaying a cube and an octahedron: we draw the cube's edges and the octahedron's faces. The result (as shown in the image above) is an octahedron suspended in a cube, clearly illustrating dual correspondence. In terms of data, imagine the cube's edges carry one type of information (say "internal network connections") while the octahedron's faces (which correspond to cube faces) carry another (say "external pressures on each facet"). The UI could let the user flip between these or view both in combination, using color and transparency to distinguish. This multi-layer network visualization is analogous to **Arena3D** in systems biology, which visualizes multilayer networks in 3D space [28] – we can achieve a similar effect by using the dual polyhedral layers.

- **Uncertainty & Quantum Effects Visual Idioms:** Since spin networks are quantum states, one might want to illustrate quantum superposition or uncertainty. One way is as mentioned: blurring or blending states. Another way is to enable a **"quantum jitter" mode** where the nodes of the graph

randomly but subtly shift around their mean position, to symbolize that geometry is not fixed. The amplitude of jitter could be tied to some parameter (like the Planck scale vs. current scale). Additionally, **particle-like effects** (sparks or pulsations) on edges could indicate transitions or interactions (like a grasped field operator exciting a spin). These are more speculative features, but could add an experiential layer for users to sense that this network is *alive* and not just static.

- **Academic Rigor and References Mode:** To satisfy educational use, the Spin Network Mode might include a "references overlay" that labels each element with a reference to theory. For example, clicking a node might pop up "Intertwiner = Quantum Polyhedron (Bianchi et al. 2010) [6] ", or clicking an edge might show "Spin-j representation of $SU(2)$ (area eigenvalue $=8\pi\ell_P^2\sqrt{j(j+1)}$) [29] ". While not always displayed, these could be available on demand to connect the visualization to literature.

Implementation Notes (Three.js, WebGL, Unity)

Building this in Three.js is very feasible. Three.js provides `LineSegments` or `TubeGeometry` for edges, `SphereGeometry` for node points, and `TextSprite` or CSS2D/CSS3D for labels (if we want HTML-based labels that hover over elements). The interactive controls can be done via **raycasting** (to pick edges/nodes with mouse or VR pointer) and dat.GUI or custom GUI for sliders to adjust values. Real-time updates are straightforward given the small graph sizes – we can recalc colors, geometries each frame or on demand. ShaderMaterial can produce glowing edges or phased coloring easily (with a uniform phase that is updated, or even using sine functions of time for autonomous oscillation).

For example, **MathBox** (built on Three.js) has capabilities for dynamic graphs and could be repurposed [30] . Alternatively, the project could fork an existing graph viz library: Piegza's Graph-Visualization provides basic 3D force-directed layout and interaction for generic graphs (we can lock our nodes to specific coordinates for Platonic shapes, disabling the force simulation) and it is MIT licensed open-source [27] . Using that as a starting point can save time in handling camera controls, selection, etc. We'd then layer our custom visual effects on top.

If we explore beyond Three.js: - **WebGPU** could offer performance boosts for very high-frequency updates or large graphs, but our use-case (dozens of edges at most) is well within Three.js/WebGL capabilities. - **Unity3D** is another route, especially if aiming for VR or AR delivery of the Donut of Attention. Unity has excellent 3D UI tools and one could script similar behaviors with its entity-component system. In fact, Unity's asset store might already have a "Polyhedron Viewer" or network graph library to jump-start. For instance, a Unity project could quickly render a spinning Platonic solid and allow triggers on edges for user input. However, the overhead of using Unity (native app or WebGL export) vs a pure web solution should be weighed. Since DonutOS is mentioned in context of Three.js, sticking to web tech seems preferable for integration. - **Processing or p5.js** could be used for rapid prototyping of visuals (many Processing demos exist for drawing polyhedra and even simple networks), but they would likely be replaced by Three.js for final due to the need for 3D interactivity in browser.

Inspiration from Existing Demos: The *Polyhedra Viewer* by @tesseralis [31] [32] demonstrates smooth transformations and dual relations between dozens of polyhedra in the browser, using a combination of three.js and custom geometry morphing. It shows that even complex operations like truncation and dualizing can be animated – confirming that drawing and morphing Platonic structures is quite doable. For dynamic physics or educational content, the Wolfram Language prototype by Mackay & Jackson (2023) is

instructive [24] [33]. They created a *tetrahedral spin-network model* and even ran a kind of diffusion dynamics on it. While their focus (quantum kernel propagator on a tetrahedral lattice) is different, the notion of treating a **4-vertex tetrahedral graph as a fundamental unit** and computing things like graph Laplacians and modes on it is directly relevant [34] [35]. We can borrow ideas like computing the graph Laplacian for potential "vibrational modes" of the network (imagine toggling a mode to see the network oscillate in a normal mode, as an attention resonance metaphor). Their code also explicitly sets coordinates for a tetrahedron and defines adjacency [24], which is a nice validation of our approach to hardcode Platonic graph geometry.

Finally, **UX and Educational Impact:** By integrating these elements, the Spin Network Mode can cater to multiple audiences – a physics student can use it to intuitively see how discrete quantum geometry works (with correct citations and ability to tweak spins to see volume changes), an enthusiast can explore a toy universe (adjusting parameters to induce a "big bounce" or curvature and seeing it color-code on the network), and a cognitive scientist or AI researcher can even use the mode metaphorically to think about networks and attention (e.g. using the Platonic graph to visualize an attention head's connectivity, with phase as alignment and spin as weight). The donut-shaped UI underscores the idea of cycles and recurrence, which resonates with the toroidal models of brain activity [22].

In summary, using Platonic solid graphs as toy models provides a **unifying platform** where quantum geometry, cosmology, and attention networks can all be represented in a visually compelling, interactive way. By leveraging open-source 3D libraries and the rich analogies from literature, the Donut of Attention's "Spin Network Mode" can both educate and inspire – showing that whether it's **spin foam or neural foam**, a lot can be learned from playing with a beautifully symmetric network in the palm of your hand (or floating in front of your eyes).

Sources: The design above draws from loop quantum gravity research (spin networks and quantum polyhedra) [6] [7] [29], discrete cosmology models [18] [19], and ideas in cognitive science and network visualization [22] [23]. In developing the interface, we can build on existing frameworks in Three.js for graph visualization [27] [30] and take inspiration from interactive polyhedral tools and simulations [24] [33]. By grounding each feature in both literature and proven software patterns, we ensure the Spin Network Mode is not only flashy but scientifically meaningful and technically robust.

---

[1] [2] [3] [4] [5] [21] [25] [29] The fabric of space: spin networks « Einstein-Online
https://www.einstein-online.info/en/spotlight/spin_networks/

[6] [7] [26] [1009.3402] Polyhedra in loop quantum gravity
https://arxiv.org/abs/1009.3402

[8] [9] [10] [11] [12] [13] [14] [15] [19] arxiv.org
https://arxiv.org/pdf/1805.05856

[16] [17] [18] Triangulated loop quantum cosmology: Bianchi IX universe and inhomogeneous perturbations | Phys. Rev. D
https://journals.aps.org/prd/abstract/10.1103/PhysRevD.81.064019

[20] How could mathematics be the foundation for reality? (# 7) | by Michael Dalton | Medium
https://medium.com/@md_44220/how-could-mathematics-be-the-foundation-for-reality-7-63537d289cb0

22 The Borsuk-Ulam theorem in the context of a gauge theory. As...
https://www.researchgate.net/figure/The-Borsuk-Ulam-theorem-in-the-context-of-a-gauge-theory-As-Borsuk-Ulam-theorem-and-its_fig1_317972909

23 Deep learning in cortical surface-based neuroimage analysis
https://www.sciencedirect.com/science/article/pii/S2667102622000493

24 33 34 35 A lattice physics approach to spin-networks in loop quantum gravity - Online Technical Discussion Groups—Wolfram Community
https://community.wolfram.com/groups/-/m/t/3526043

27 30 Best way to draw graphs with Three.js - Stack Overflow
https://stackoverflow.com/questions/14858518/best-way-to-draw-graphs-with-three-js

28 Arena3Dweb: interactive 3D visualization of multilayered networks
https://pmc.ncbi.nlm.nih.gov/articles/PMC8128064/

31 32 Polyhedra Viewer
https://polyhedra.tessera.li/