Computational Experiments at Wolfram Institute

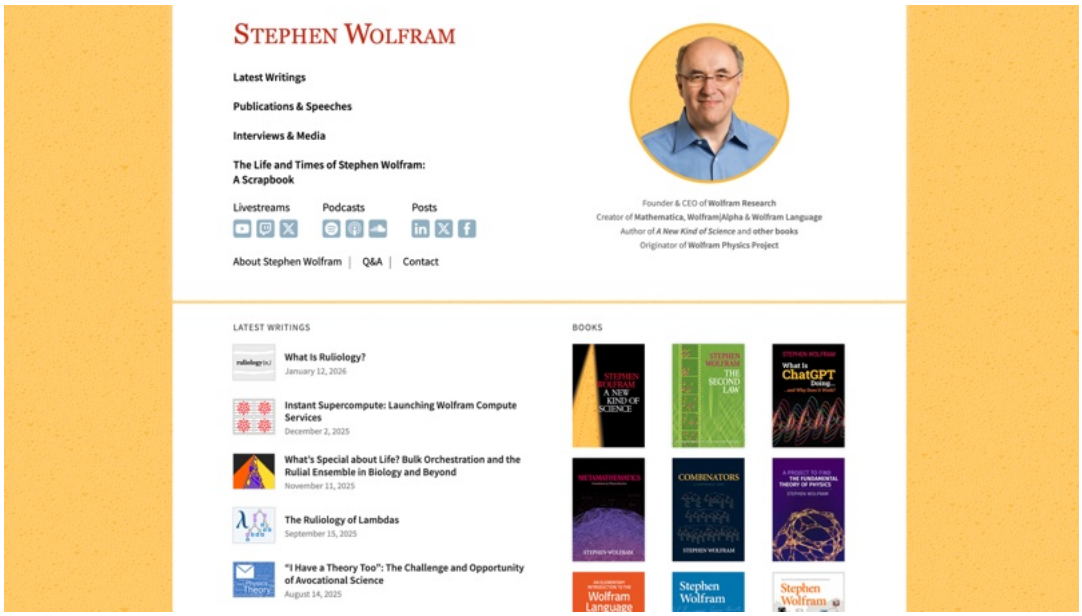
Pavel Hajek

Wolfram Institute

https://wolframinstitute.org/

- Directed by **Dr. Stephen Wolfram**, founder and CEO of **Wolfram Research, Inc.**

https://www.stephenwolfram.com/

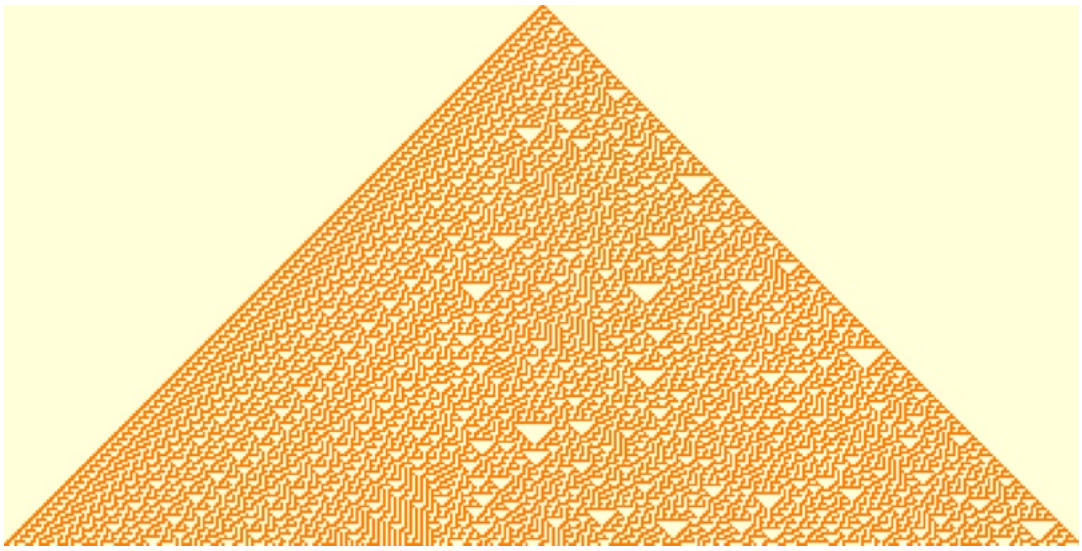


- **Born 1959, papers in nuclear physics since 15, PhD at 20, job at IAS around R. Feynman**
- Fascinated by **complex systems**, **emergent behavior**, and **"computational irreducibility"**
- Wrote ***A New Kind of Science***
- **Rule 30**:

In[ ]:=
```
RulePlot[CellularAutomaton[30], ColorRules → {0 → LightYellow, 1 → Orange}]
Pane[ArrayPlot[CellularAutomaton[30, {{1}, 0}, 200], PixelConstrained → 1,
  ColorRules → {0 → LightYellow, 1 → Orange}, Frame → False],
 {400, Automatic}, Alignment → Center]
```

Out[ ]=

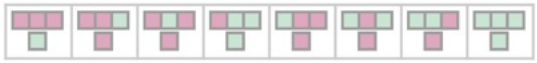
Out[ ]=

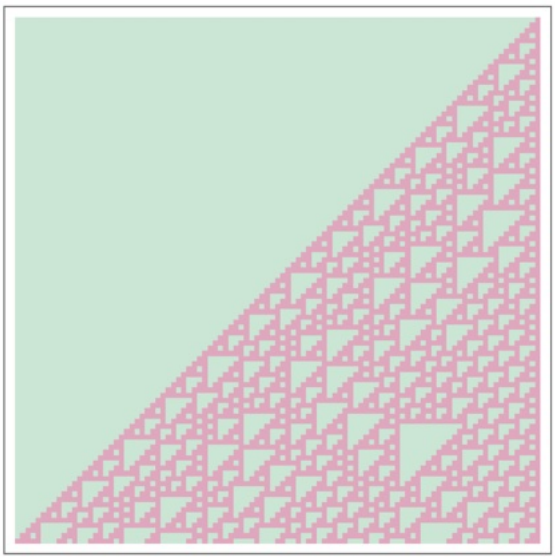


- **Turing completeness** is not the same as computational irreducibility.
  - Rule 110

In[ ]:=
```
With[{c0 = ColorData["Pastel"][.8], c1 = ColorData["Pastel"][.2]},
 Column @ { RulePlot[CellularAutomaton[110], ColorRules → {1 → c1, 0 → c0}],
   Pane[ArrayPlot[
      CellularAutomaton[110, {{1}, 0}, 100],
      ColorRules → {1 → c1, 0 → c0},
      PlotRange → All], {400, Automatic}, Alignment → Center]}]
```

Out[ ]=

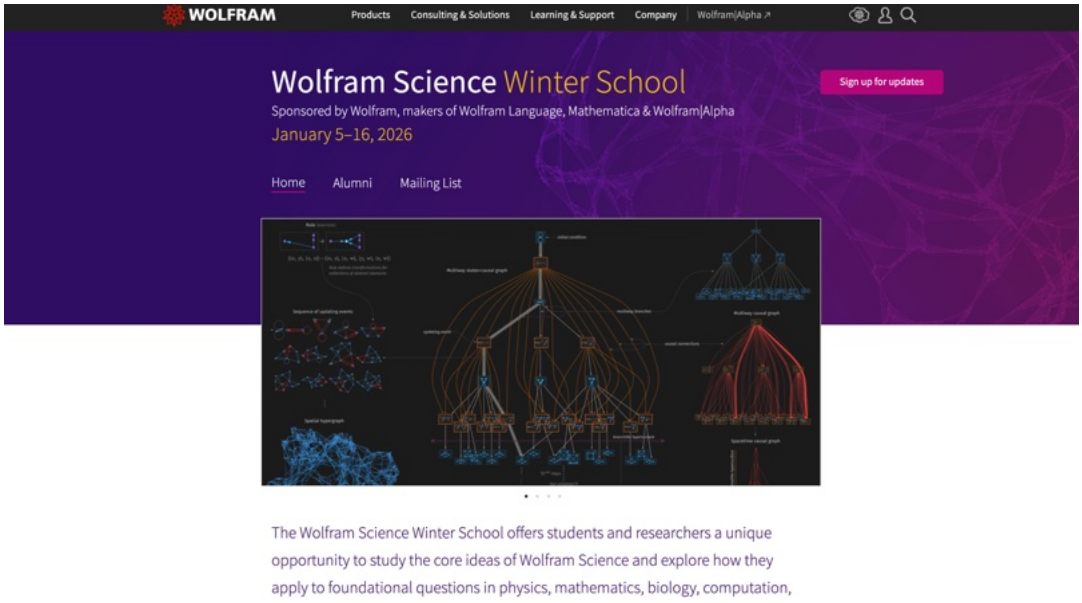




- **Undecidability** is not the same as computational irreducibility.
  - Halting problem, homotopy group of finite ASC in d>4,...

Wolfram Science Winter School

https://education.wolfram.com/winter-school/



- **Wolfram Community**
  - https://community.wolfram.com/groups/-/m/t/3358052
  - https://community.wolfram.com/groups/-/m/t/3607623
- **Wolfram Summer School**
  - https://education.wolfram.com/summer-school/
- Looking for enthusiastic, get-things-done–oriented students who want to explore their ideas within one of our frameworks. Paid internships possible.

Wolfram Physics Project

- **Emergence of physics from complex behavior of simplest models - hypergraph rewriting**

  https://www.wolframphysics.org/technical-introduction/
  https://www.wolframphysics.org/universes/

```
Import[NotebookDirectory[] <> "rewriting.mp4"]
```

In[ ]:=

People

- 🧠 **Stephen Wolfram - New Kind of Science**
- Original theoretical framework and toolkit due to **Jonathan Gorard** and **Max Piskunov.**
- Animation by **Richard Assard** (cutting edge parallelized GPU optimized hypergraph rewriting engine).
- **Nik Murzin** (coding 🧙, quantum stuff, diagrams,...)**, Carlos Zapata** (hypermatrix algebras, "hyper-categories", n-ary science) **, Xerxes Arsiwalla** (higher category theory, pre-geometry, metamathematics)**, Dugan Hammock** (tilings and quasi-crystals)**, Pavel Hajek** (infrageometry, physics project, n-ary science) + others.

A point of view I am comfortable with

WPP aims to construct a **minimal model of the universe** grounded in concrete objects of the **least possible complexity**—objects whose entire informational content fits into the smallest data structure. The goal is to develop a **robust microscopic theory** capable of handling low-scale singularities, and then define a kind of **thermolimit** in which increasing complexity yields approximate recovery of current physical theories.

Exact microscopic predictions will remain out of reach due to scale and **computational irreducibility**. However, in the appropriate limits—where **aggregation** effectively bypasses irreducibility—the fundamen-

tal model should reproduce the coarse-grained theories we use today. The underlying philosophy is that contemporary mathematics and physics arise from **extremely coarse observations** of an **enormously complex substrate**. Standard model spaces (such as the real numbers) are abstractions introduced for convenience, particularly for expressing dynamics through differential equations that mix positions and momenta and making macroscopic predictions.

If we want a genuinely **fundamental theory of everything**, these abstractions must be set aside. We must begin with a minimal, concrete model and allow the familiar mathematical and physical structures to emerge only as large-scale, high-complexity limits.

For orientation: In WPP the **length of an edge** is estimated to be

10^{-93} m

https://www.wolframphysics.org/technical-introduction/potential-relation-to-physics/units-and-scales/

which is far below the Planck length

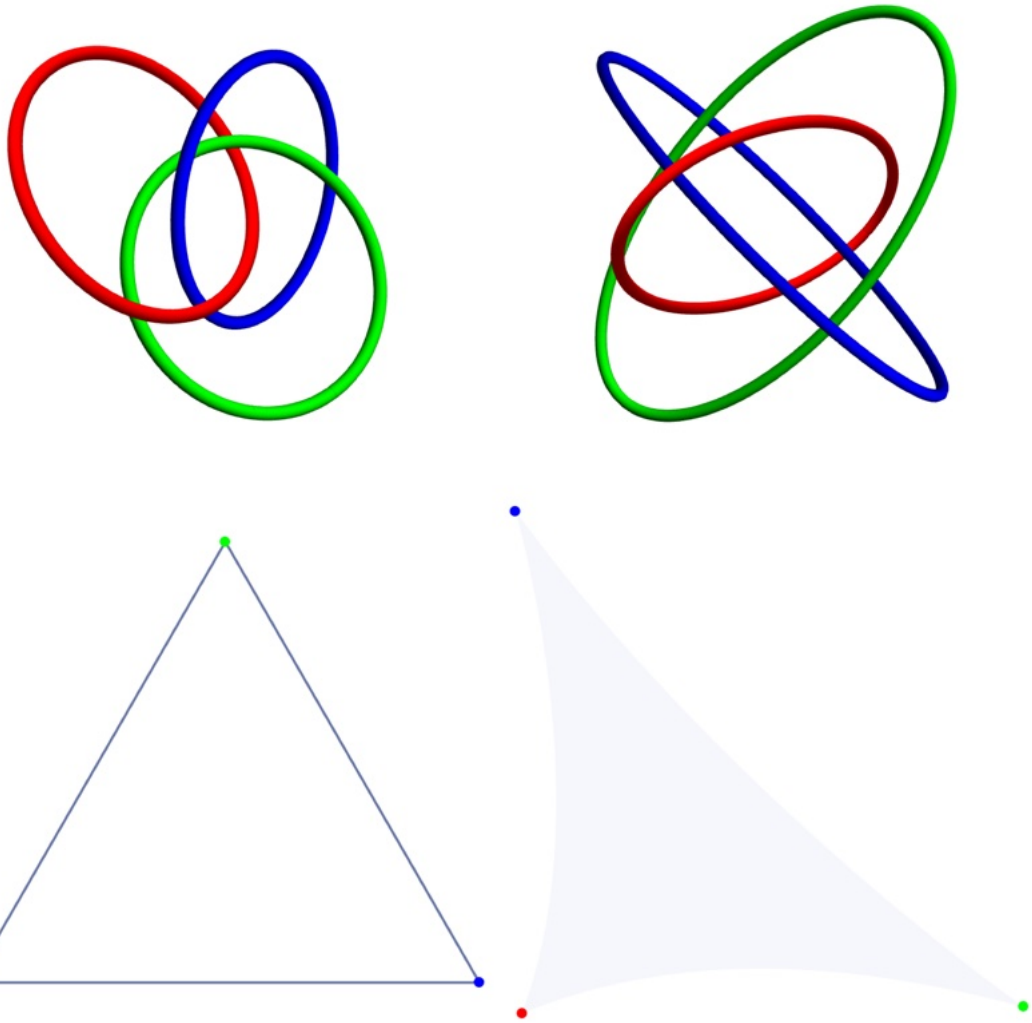
1.6163×10^{-35} m.

In the real world, the smallest experimentally accessible length scales are limited by diffraction that roughly corresponds to the wavelength $\sim 10^{-7}$ m for optical devices and goes down to about $\sim 10^{-13}$ m for synchrotron X-rays (European XFEL in Hamburg).

Hypergraphs and Rewriting

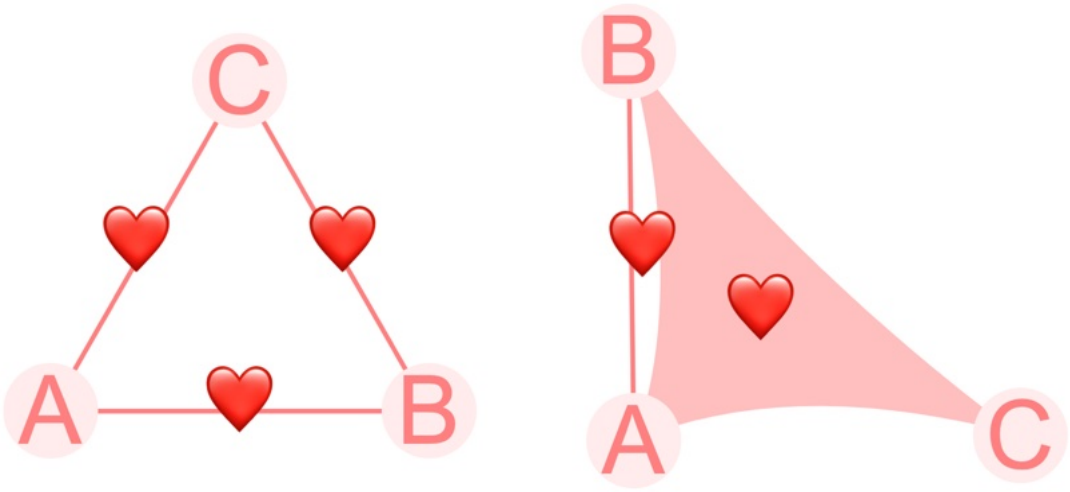
In[•]:=
```
plotParametricKnots[x__] := Graphics3D[#, Boxed → False, Axes → False] &@
    (Style[Tube[Table[#[[1]][α], {α, 0, 2 Pi, 0.05}], 0.05], #[[2]] & /@ x);
links = plotParametricKnots /@ {
    {{α ↦ {Cos[α], Sin[α], 0}, Red},
     {α ↦
       {1, .5, 0} + RotationMatrix[Pi / 10, {1, 0, 0}].{Cos[α], Sin[α], 0}, Green},
     {α ↦ {.5, 1, 0} + RotationMatrix[Pi / 5, {0, 1, 0}].{Cos[α], Sin[α], 0}, Blue}}
    ,
    {{α ↦ {2 * Cos[α], Sin[α], 0}, Red},
     {α ↦ RotationMatrix[Pi / 2, {1, 0, 0}].RotationMatrix[Pi / 2, {0, 1, 0}].
         {2 * Cos[α], Sin[α], 0}, Green}, {α ↦ RotationMatrix[Pi / 2, {0, 1, 0}].
         RotationMatrix[Pi / 2, {1, 0, 0}].{2 * Cos[α], Sin[α], 0}, Blue}
    }};
hgs = Hypergraph[#,
     VertexStyle → {1 → Red, 2 → Blue, 3 → Green}, EdgeLabels → "EdgeTag"] & /@ {
    {{1, 2}, {2, 3}, {3, 1}},
    {{1, 2, 3}}
   };
Grid[{links, hgs}]
```

Out[ ]=



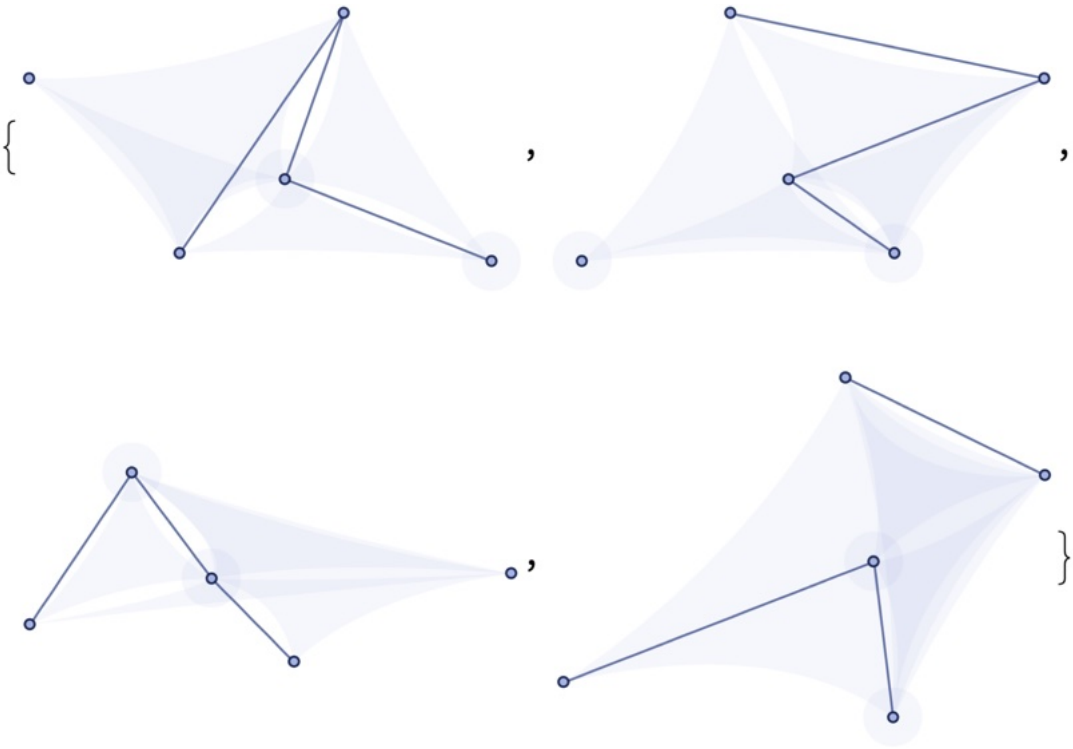
In[ ]:=
```
style = {VertexStyle → {_ → LightPink}, VertexLabels →
    {_ → Placed[Style["Name", 40], Center]}, VertexLabelStyle → {_ → Pink},
  VertexSize → {_ → 0.1}, EdgeStyle → {_ → Directive[Pink, Thick]},
  EdgeLabels → {_ → Placed[Style["❤", 30], Center]}, ImageSize → Medium};
t1 = Hypergraph[{{"A", "B"}, {"B", "C"}, {"C", "A"}}];
t2 = Hypergraph[{{"A", "B", "C"}, {"A", "B"}}];
t3 = Hypergraph[{{"A"}}];
style =
 Row@{Hypergraph[t1, Sequence @@ style],
   Spacer[40], Hypergraph[t2, Sequence @@ style]}
```

Out[ ]=



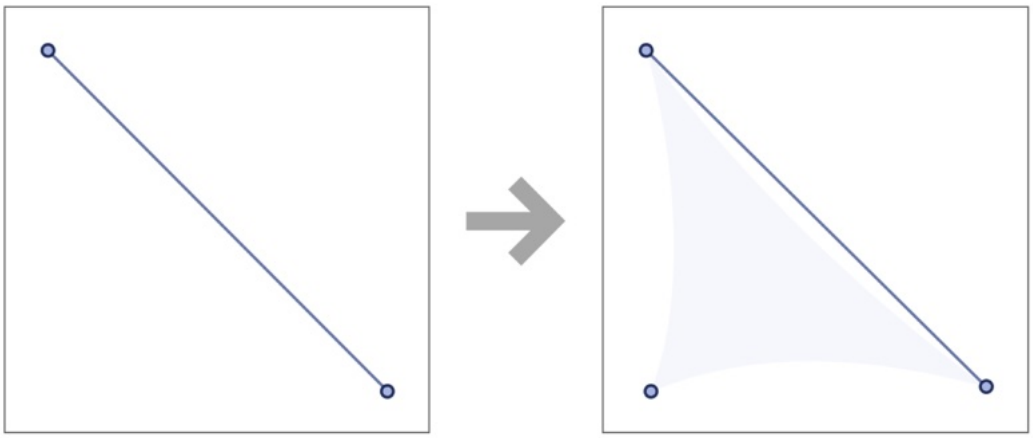
In[•]:=
```
Hypergraph[#] & /@ Table[RandomHypergraph[5,
    {{2, 1}, {3, 2}, {3, 3}, {1, 4}}, "Connected" → True, "Simple" → True], {i, 1, 4}]
```

Out[•]=



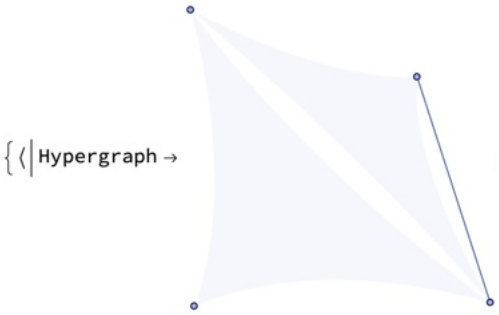
In[•]:=
```
r1 = HypergraphRule[Hypergraph[{{1, 2}}], Hypergraph[{{1, 2, 3}, {2, 3}}]]
```

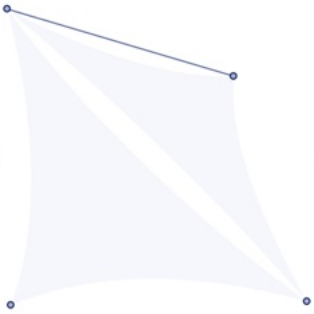
Out[•]=



In[ ]:=
```
r1[Hypergraph[{{1, 2, 3}, {2, 3}}]]
```

Out[ ]=





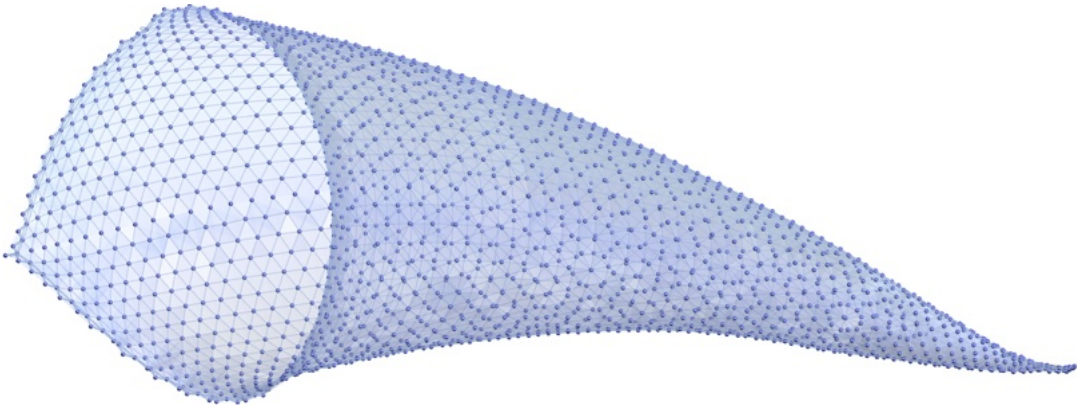
{⟨|Hypergraph → , MatchVertices → {2, 3}, MatchEdges → {{2, 3}},
MatchEdgePositions → {{2}}, NewVertices → {v42672}, NewEdges → {{2, 3, v42672}, {3, v42672}},
DeletedVertices → {}, RuleVertexMap → {1 → 2, 2 → 3, Verbatim[3] → v42672},
Bindings → ⟨|v1_ ↦ 2, v2_ ↦ 3|⟩|⟩, ⟨|Hypergraph → ,
MatchVertices → {2, 3}, MatchEdges → {{2, 3}}, MatchEdgePositions → {{2}},
NewVertices → {v42672}, NewEdges → {{2, 3, v42672}, {2, v42672}}, DeletedVertices → {},
RuleVertexMap → {1 → 3, 2 → 2, Verbatim[3] → v42672}, Bindings → ⟨|v1_ ↦ 3, v2_ ↦ 2|⟩|⟩}

Graphics of WPP

- The pictures in WPP depict the end-state of a **canonical single path** (which applies the "oldest" applicable event to the "oldest" updated node) in the **multicomputation tree of hypergraph rewriting**.

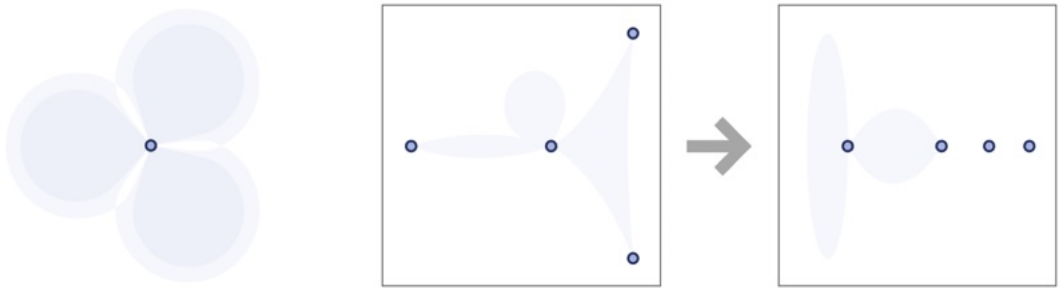
In[•]:=
```
r1 = {{1, 1, 2}, {1, 3, 4}} → {{4, 4, 3}, {2, 5, 3}, {2, 5, 3}};
i1 = {{0, 0, 0}, {0, 0, 0}};
ResourceFunction["GraphReconstructedSurface"][
 WolframModel[r1, i1, 2000, "FinalState"]]
```

Out[•]=



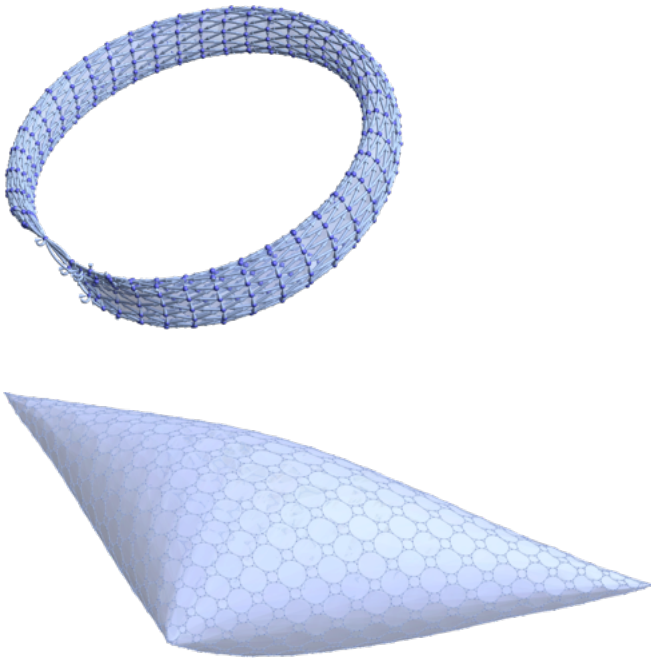
In[•]:=
```
Row @ {Hypergraph@i1, Spacer[40], HypergraphRule @@ Hypergraph /@ List @@ r1 }
```

Out[•]=



In[ ]:=
```
Row @ {
  GraphReconstructedSurface[
   WolframModel[{{1, 1, 2}, {3, 4, 1}} → {{1, 1, 3}, {4, 5, 1}, {2, 5, 3}},
    {{0, 0, 0}, {0, 0, 0}}, 500, "FinalState"] ],
  Spacer[20],
  GraphReconstructedSurface[
   WolframModel[{{1, 2, 3}, {4, 5, 6}, {1, 4}} → {{2, 7, 8}, {3, 9, 10},
      {5, 11, 12}, {6, 13, 14}, {13, 8}, {7, 10}, {9, 12}, {11, 14}},
    {{0, 0}, {0, 0}, {0, 0}, {0, 0, 0}, {0, 0, 0}}, 10, "FinalState"], 8]
 }
```

Out[ ]=

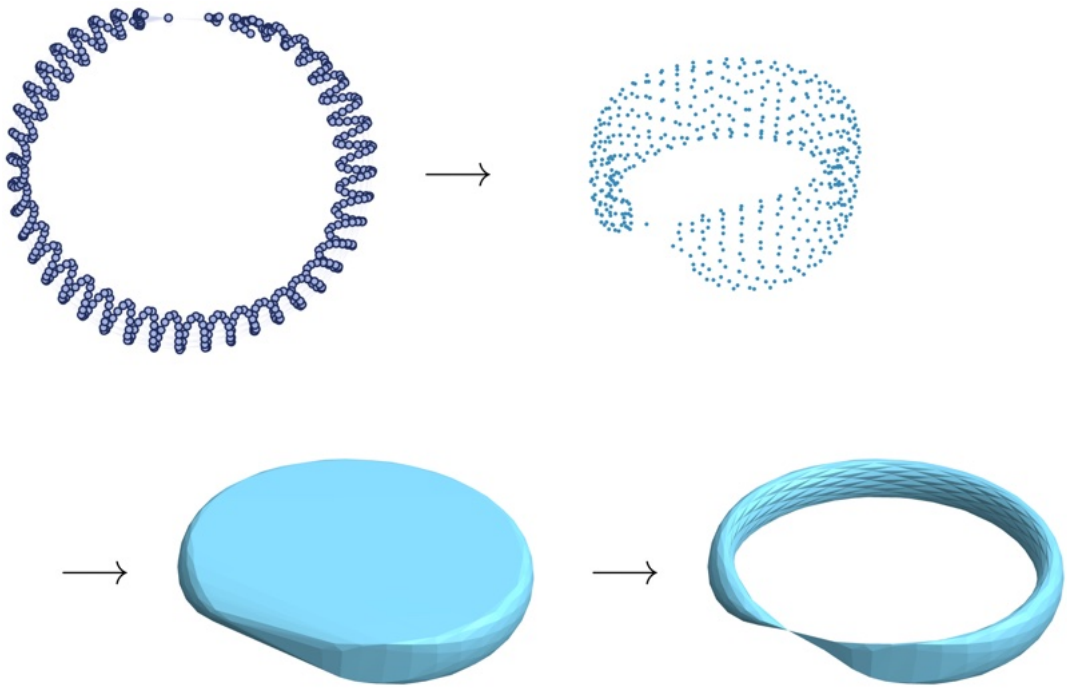


- The resulting hypergraph is converted into an ordinary graph by **replacing each hyperedge with a clique** on its incident vertices.

  The graph is embedded into $\mathbb{R}^3$ using the **spring-electric embedding** and converted into a **simplicial surface mesh** by taking its **Delaunay triangulation**, keeping only 2-simplices, and discarding “large” triangles that would otherwise fill in convex holes.

In[ ]:=
```
hg = WolframModel[{{1, 1, 2}, {3, 4, 1}} → {{1, 1, 3}, {4, 5, 1}, {2, 5, 3}},
    {{0, 0, 0}, {0, 0, 0}}, 500, "FinalState"] ;
g = ResourceFunction["HypergraphToGraph"][ hg ];
ge = GraphEmbedding[g, "SpringElectricalEmbedding", 3];
d = DelaunayMesh[ge];
m = ResourceFunction["NonConvexHullMesh"][ge, 2.5];
Row[{Hypergraph[hg], Spacer[20], ⟶, Spacer[20],
  ListPointPlot3D[ge, Axes → False, Boxed → False], Spacer[20],
  ⟶, Spacer[20], d, Spacer[20], ⟶, Spacer[20], m}]
```

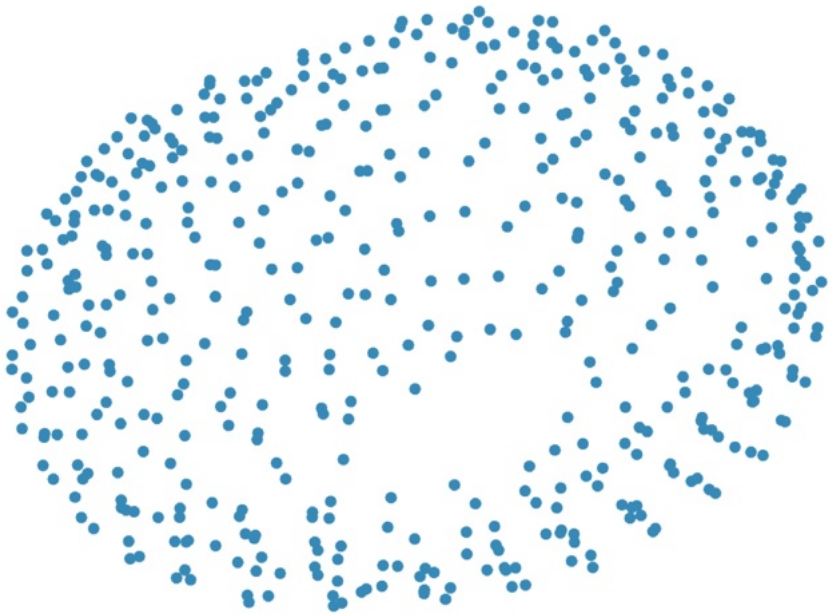
Out[ ]=



- However, we are interested **intrinsic geometry** independent of the embedding.
  There are several natural embeddings, each reflecting a different intrinsic property of the graph.

In[ ]:=
```
ListPointPlot3D[ GraphEmbedding[g, "GravityEmbedding", 3],
 Axes → False, Boxed → False]
```

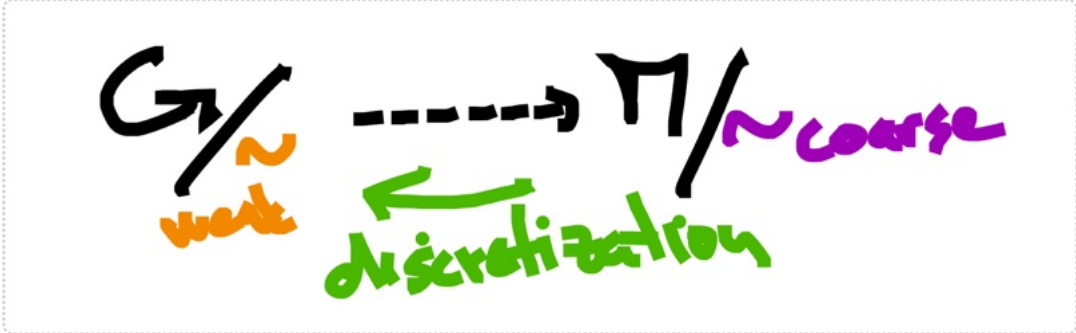
Out[ ]=



Infrageometry

- **Emergence of intrinsic geometry in hypergraph rewriting**

In[•]:=
```
With[{w = 1000, h = 300}, Canvas[PlotRange → {{0, w}, {0, h}}, ImageSize → {w, h}]]
```



Goals

- Emergence of geometry and physical laws in “thermolimit” of increasing complexity
- Discrete concept of dynamics converging to solutions of ODEs, in particular Einstein equations
- Quantum gravity

“Minimal model pipeline”

- Define **robust minimal models**
- Use them to define observables/invariants
- Study **emergence and persistence** in large scale limit for concrete initial conditions
- Design **tests** to select out “physical universe” and discard rules leading to singularities
- **Ruliology** - classification of rules from the **ruliad** based on their properties under the limit

Robustness

- **“Small graphs” can represent a wide range of possible geometries**, and any convergence to a definite, smooth geometry should only be expected at **much larger scales**.

  Given that the micro-world is quantum, this is entirely reasonable. One can as well think that sufficiently small objects do not possess definite shapes at all, and that the geometric abstractions we use for the macroscopic world simply do not apply at those scales.

Possible resources:

- M. Gromov. Asymptotic invariants of infinite groups
- Ising model, block-spin transformation

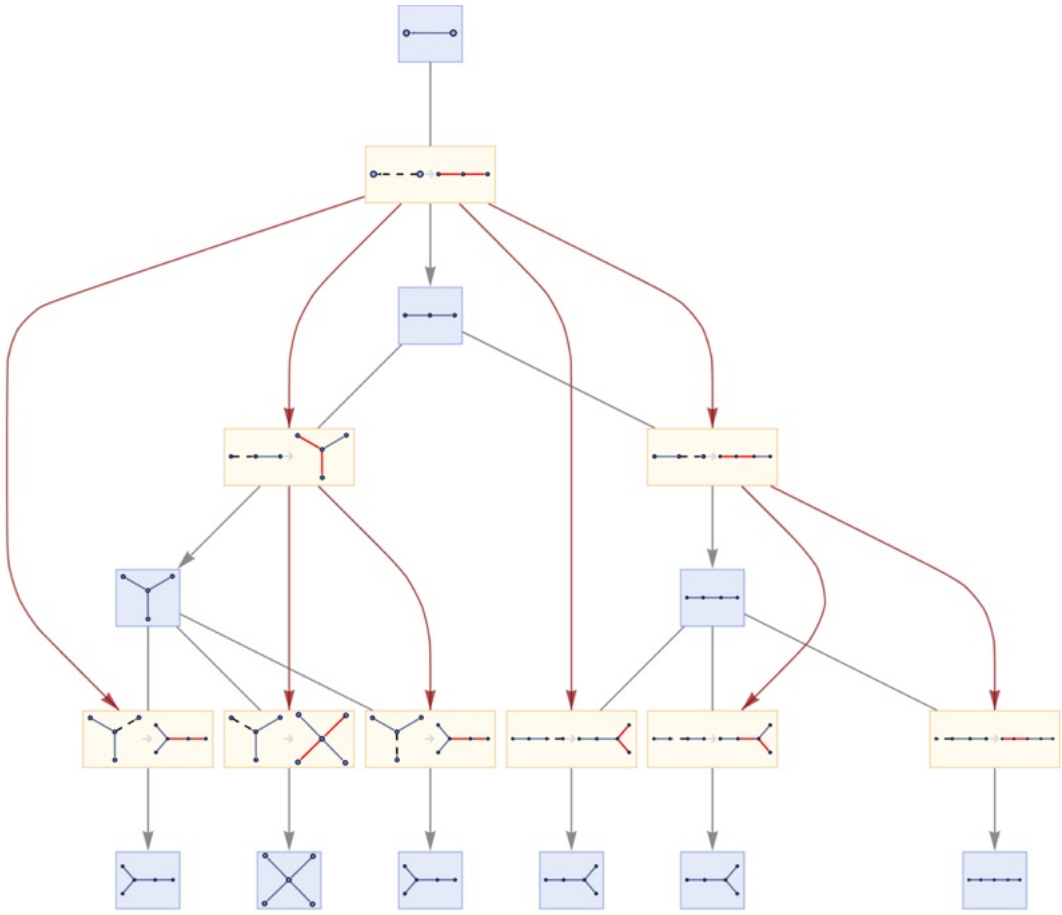
- Causal set theory
- R. Forman. Morse theory for Cell Complexes

Multicomputation and types of geometries

- **Multicomputation** involves **indeterminism** arising from the existence of multiple possible matches for the rewriting rules, which permit different **updating orders** (i.e., different chains in the graph). Such concept is known in theoretical computer science, e.g., **nondeterministic Turing machine** used to define the **NP class**. The original WPP project employed a canonical updating order together with generational layering.

In[•]:=
```
ms = MultiwaySystem[{{{1, 2}} → {{1, 2}, {2, 3}}}, {{{1, 2}}}];
ms["EvolutionCausalGraph", 3]
```

Out[•]=



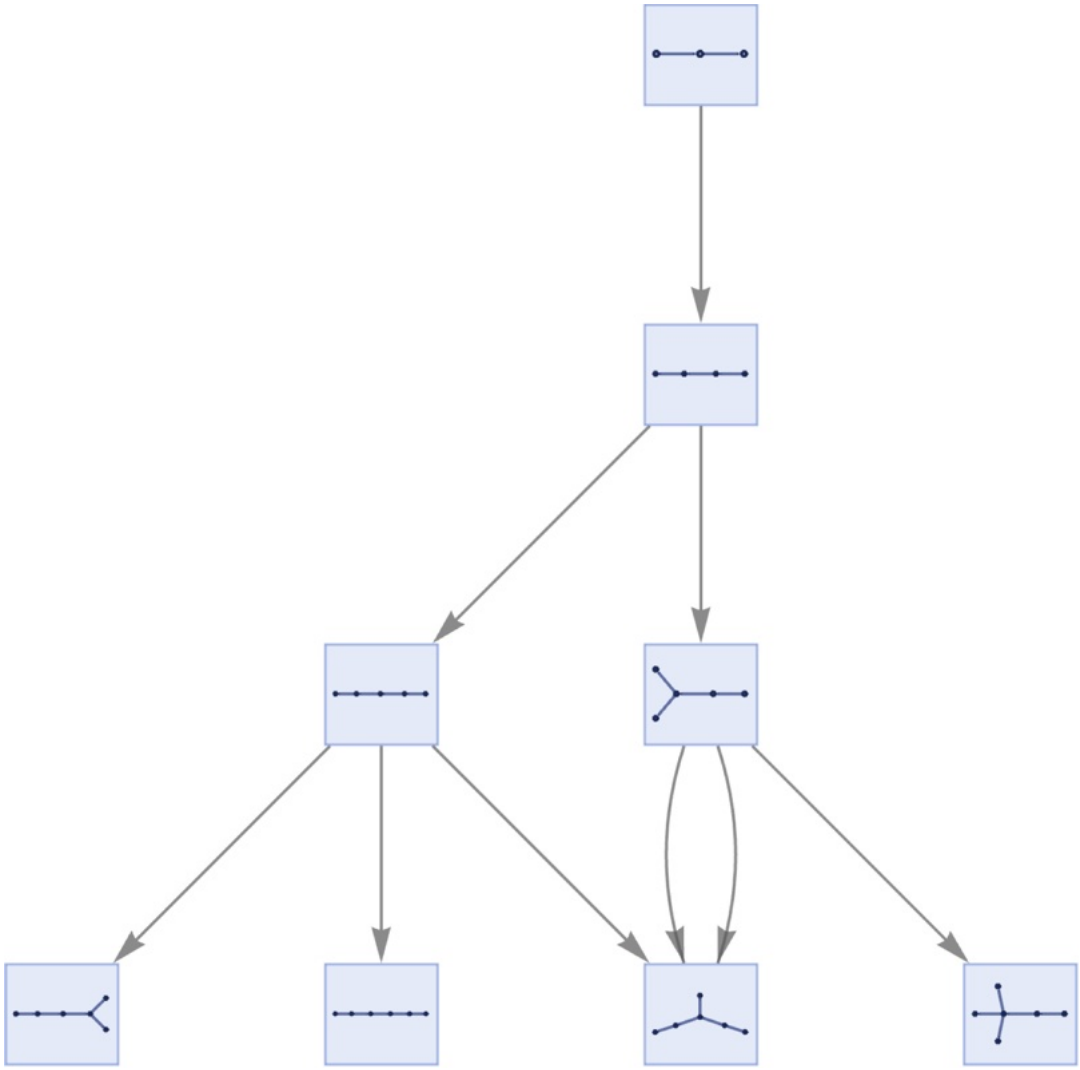
- We extract the following **binary relations** from the rewriting system:

In[•]:=
```
With[{w = 1000, h = 300}, Canvas[PlotRange → {{0, w}, {0, h}}, ImageSize → {w, h}]]
```

- **States graph** - Represents the non-deterministic computation as it moves through different computational states, identified up to a chosen **canonicalization**. Note that a deterministic machine would correspond to a vector field whose zero locus consists of **halting states**, while periodic orbits correspond to **non-halting states**.

In[•]:=
```
ms["StatesGraph", 3, "CanonicalStateFunction" → "CanonicalHypergraph"]
```

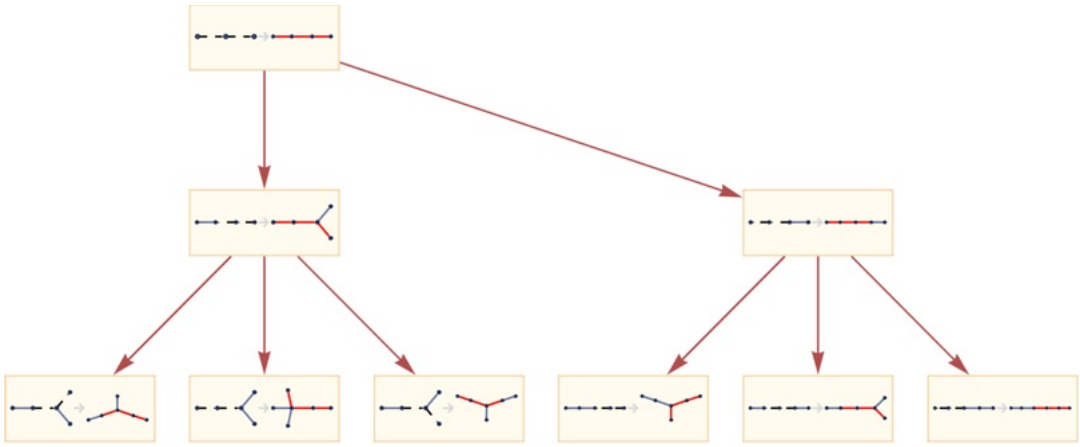
Out[•]=



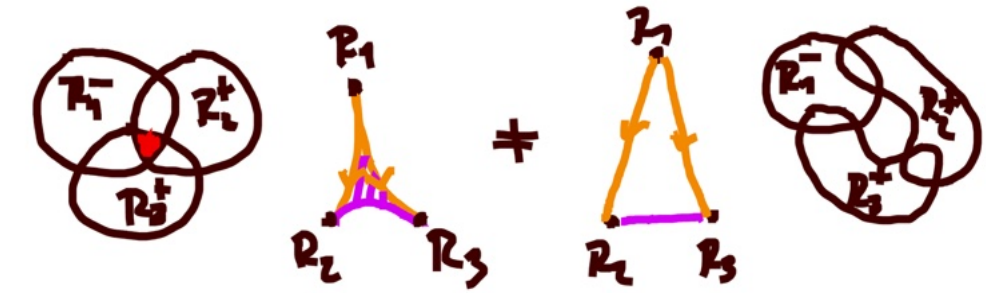
- The **causal graph** is intended to reconstruct **Lorentzian geometry**. Note that there already exists a body of work on **sprinkling** points into a Lorentzian manifold and analyzing how uniquely the resulting causal set (**partial order**) determines the underlying geometry by postulating embeddings of certain properties.

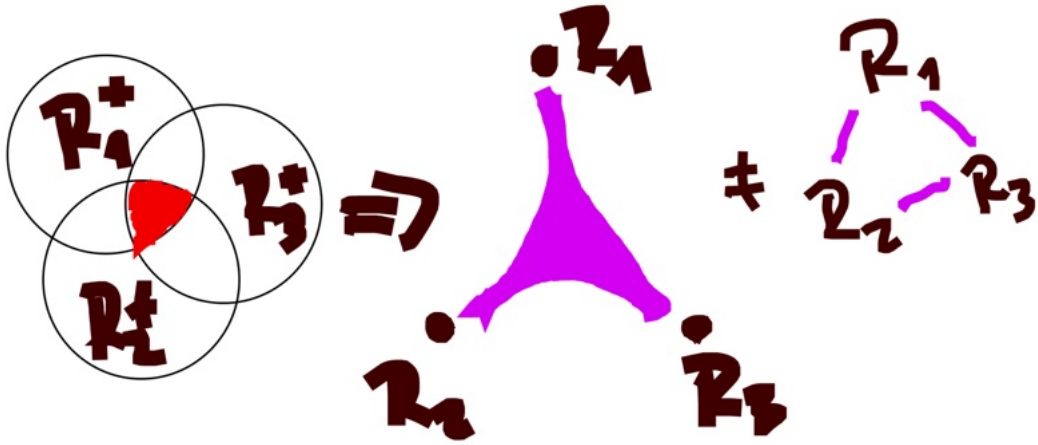
In[ ]:=
```
ms["CausalGraph", 3]
```

Out[ ]=



- We are planning to extend to the following **irreducible n-ary relations**:

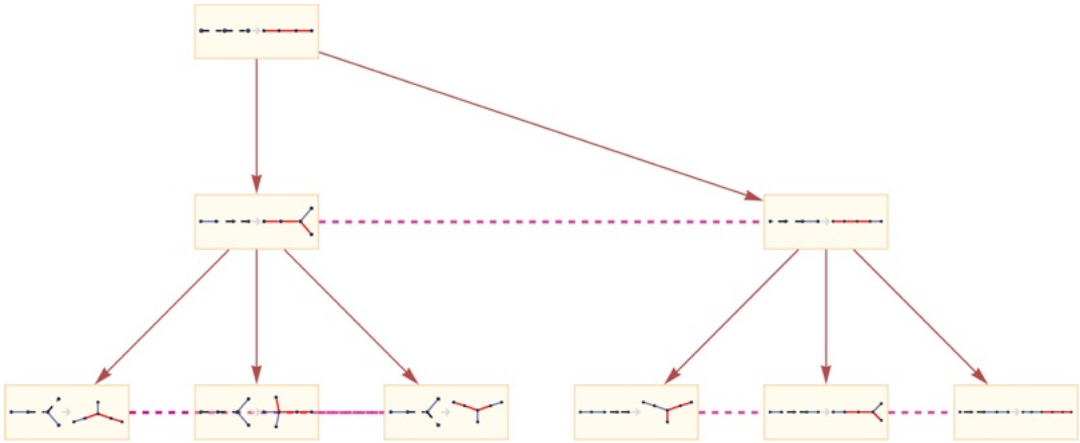




- The **causal-branchial graph** is a natural place for **quantum gravity** to emerge.

In[ ]:=
```
ms =
  MultiwaySystem[{{{1, 2}, {2, 3}} → {{1, 2}, {2, 3}, {3, 4}}}, {{{1, 2}, {2, 3}}}];
Row @ {
  ms["CausalBranchialGraph", 3]
 }
```

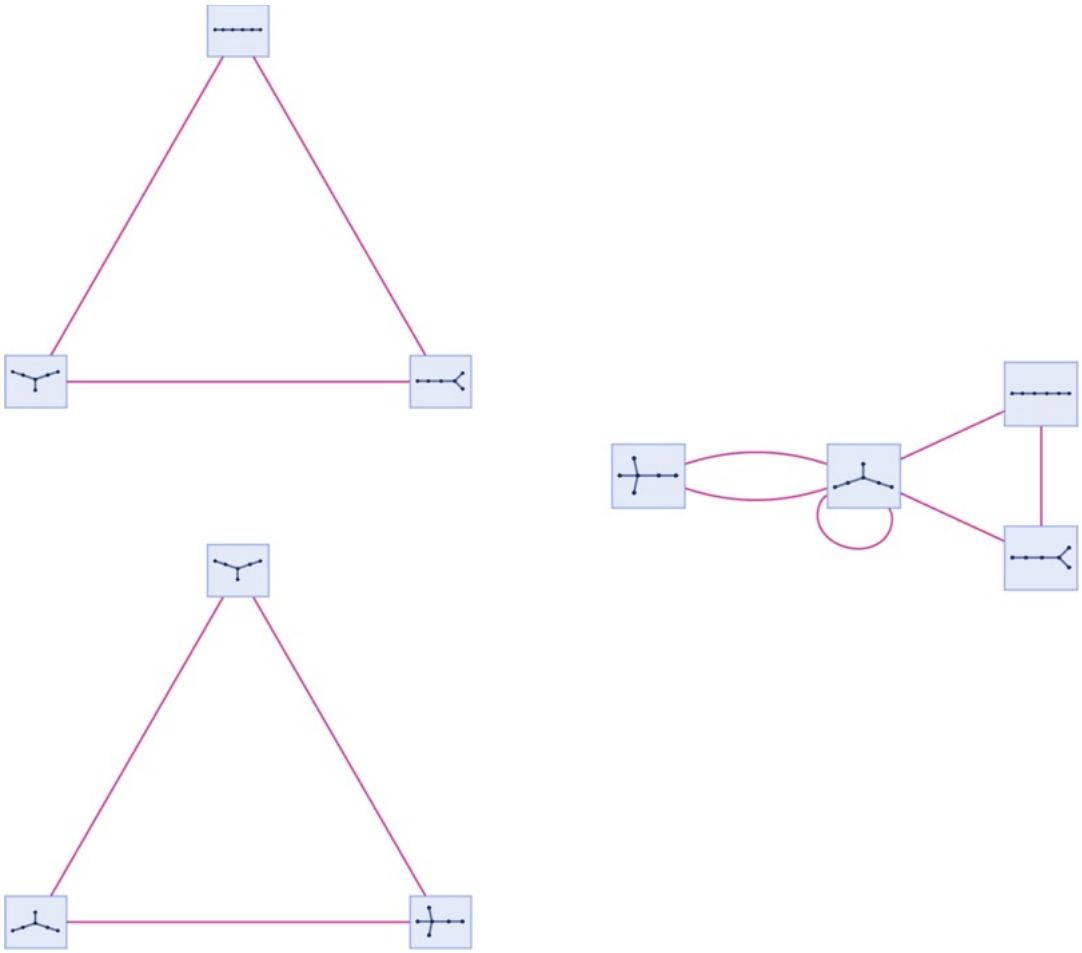
Out[ ]=



- In the **branchial graph** we interpret nodes as quantum states and their superpositions; in the appropriate limit, the resulting geometry should approximate that of a Hilbert space (noting that the adjacency matrix is self-adjoint).

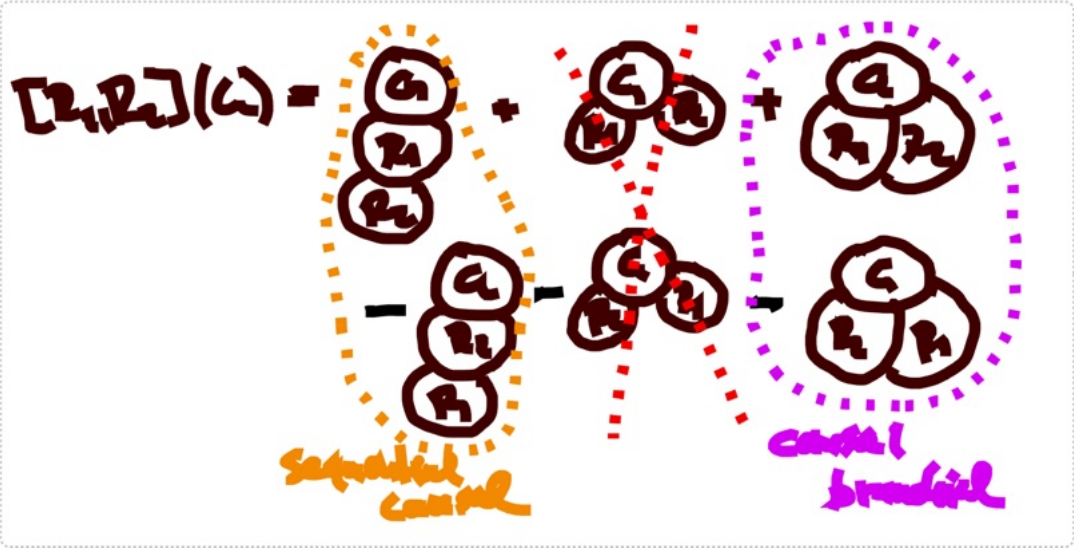
In[ ]:=
```
Row @ {
  ms["BranchialGraph", 3],
  Spacer[60],
  ms["BranchialGraph", 3, "CanonicalStateFunction" → "CanonicalHypergraph"]}
```

Out[ ]=



- Each rule can be interpreted as a linear operator on the **moduli space of isomorphism classes of hypergraphs**, which can be interpreted as an infinitesimal generator of a flow on the set of isomorphism classes of hypergraphs.

  The commutator of such rules decomposes into **sequential-causal** and **causal-branchial** part.



- For certain classes of rules, the causal–branchial component vanishes on all graphs. Most notably, this occurs for **elementary blowups**—rules whose left-hand side consists of a single hyperedge.

  In this case, the commutator of elementary blowups can be written as a linear combination of elementary blowups, so this class of rules is closed under commutation and therefore forms a **Lie algebra**.

  The special case involving unary edges corresponds to the **insertion Lie algebra**, which appears in perturbative quantum field theory. Note that there is a graded version of that.

  Such a Lie algebra induces a Poisson bracket on the space of formal functions, giving rise to **Hamiltonian mechanics on ruliad**. Canonical functions are the number of vertices and edges.

  https://community.wolfram.com/groups/-/m/t/3358052

  https://resources.wolframcloud.com/PacletRepository/resources/WolframInstitute/Hypergraph/English/ReferencePages/Symbols/InsertionBracket.html

- We plan to study **n-ary versions** of this constructions (**Lie-n and n-Lie**) in the context of the **Arity science project.**
- It can also be interpreted as the Lie algebra of a **renormalization group**, particularly when considering **coarse-grained mesoscale models** (e.g., hypergraphs with labels).
- We want manifestations of the **holography principle** to emerge through comparisons between causal and branchial graphs.

Example 1: Dimension

- For any **Riemannian manifold:**

$$V(r) = \frac{\pi^{d/2}}{(d/2)!} r^d \left(1 - \frac{r^2}{6(d+2)} R + O(r^4)\right)$$

- **Wolfram-Hausdorff dimension:**

$$\Delta(r) = \frac{d \log(V(r))}{d \log(r)} \sim \frac{\log(V(r+1)) - \log(V(r))}{\log(r+1) - \log(r)}$$

with respect to the **graph distance.** This corresponds to **one edge as minimal model of distance**.

- Neither dimension of model space nor Hausdorff dimension useful.

In[1]:=

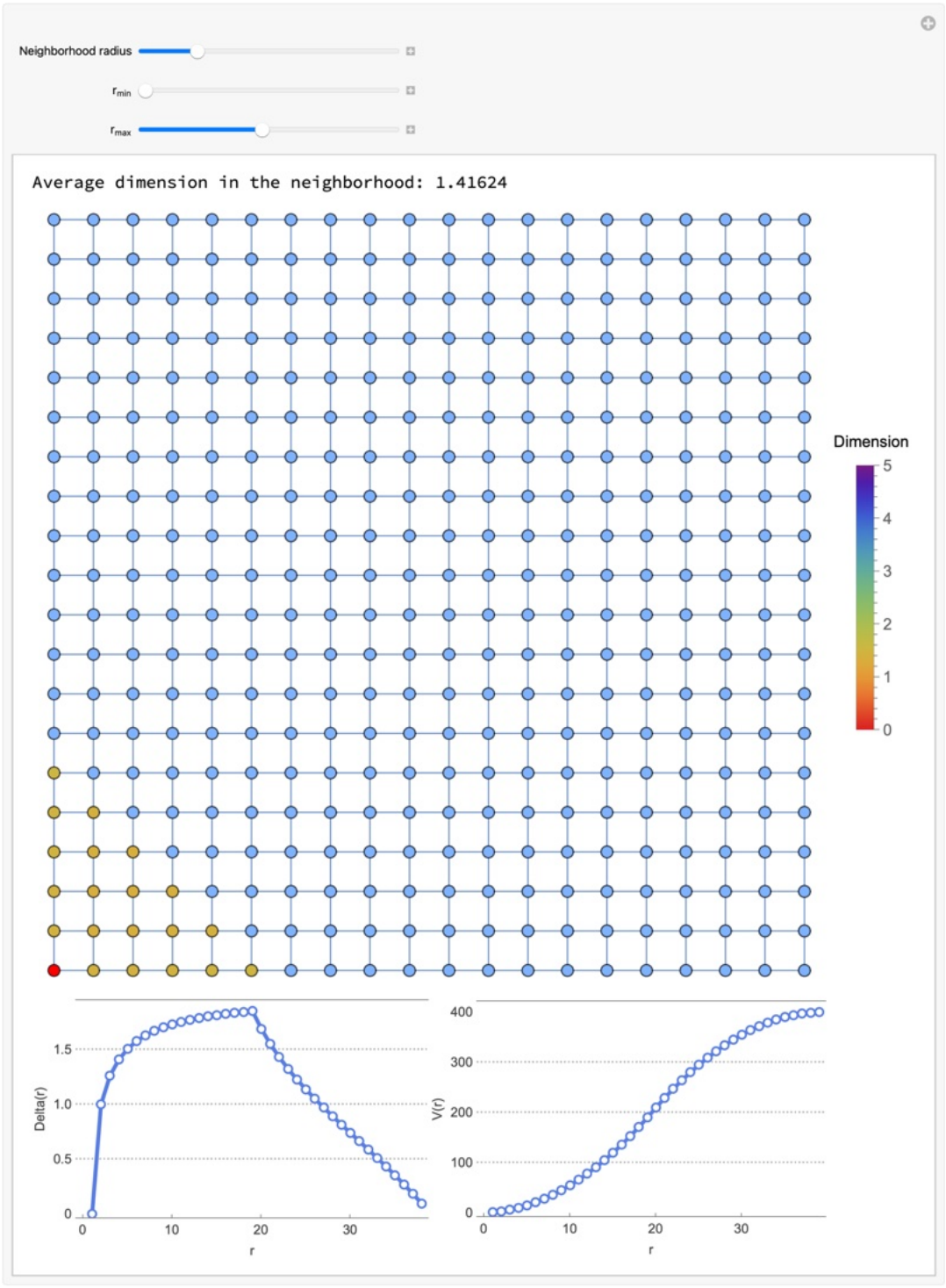
```
GeodesicSurfaces[g_, v_] := KeySort@Counts@GraphDistance[g, v];

GeodesicVolumes[g_, v_] := Association @ With[{assoc = GeodesicSurfaces[g, v]},
    Thread[Keys[assoc] → Accumulate@Values@assoc]];

LogDifferenceDimension[volumes_] :=
  Association@Table[r → If[r == 0, 0, N[(Log[volumes[r + 1]] - Log[volumes[r]]) /
       (Log[r + 1] - Log[r])]], {r, Most[Keys[volumes]]}];

FitGeodesicGrowth[data_] :=
  Block[{a, r, d, α}, NonlinearModelFit[data, a * r ^ d, {{d, 2}, {α, 0}}, r]];

FittedGeodesicDimension[data_, radius_] :=
  Module[{d, fit}, d = Take[List @@@ Rest@Normal@data, UpTo[radius]];
   If[Length[d] < 2, Return[0]];
   fit = FitGeodesicGrowth[d];
   fit["BestFitParameters"][[2]]
  ];

FindNearestVertex[g_, pt_] :=
  Module[ {vertices, coords, dists},
   vertices = VertexList[g];
   coords = GraphEmbedding[g];
   dists = EuclideanDistance[pt, #] & /@ coords;
   vertices[[First@Ordering[dists, 1]]]
  ];

VisualizeGraphDimension[g_] :=
  DynamicModule[
   {graphRadius, selectedVertex, clickPt,
```

```
 quantity, displayRadius, minRadius, maxRadius, maxDim},
clickPt = {0, 0};
maxDim = 5;
graphRadius = GraphRadius[g];
selectedVertex = First@VertexList[g];
quantity = "LogDimension";
displayRadius = Min[graphRadius, 5];
minRadius = 1;
maxRadius = Min[graphRadius, 5];
Manipulate[ClickPane[Dynamic@Module[
    {nbhd, volumes, dimensions, meanDimensions, meanDimension, idx},
    nbhd = VertexList@NeighborhoodGraph[g, selectedVertex, displayRadius];
    idx = First @ First @ Position[nbhd, selectedVertex];
    volumes = GeodesicVolumes[g, #] & /@ nbhd;
    dimensions = LogDifferenceDimension /@ volumes;
    meanDimensions = (dims ↦
        Mean[KeySelect[dims, minRadius ≤ # ≤ maxRadius &]]) /@ dimensions;
    meanDimension = Mean[meanDimensions];
    Column[{
      Row[{"Average dimension in the neighborhood: ", meanDimension}],
      Row[{
        Graph[ g,
         VertexStyle → Join[Thread[nbhd →
              (ColorData["Rainbow"][1 - Clip[# / maxDim, {0, 1}]] & /@
                meanDimensions)], {selectedVertex → Red}],
         VertexSize → 0.3,
         ImageSize → 600
        ],
        BarLegend[ {ColorData["Rainbow"][1 - # / maxDim] &, {0, maxDim}},
         LabelStyle → {FontSize → 12}, LegendLabel → "Dimension"]
       }],
      Row[{
        ListPlot[Values@ dimensions⟦idx⟧,
         Joined → True, PlotTheme → "Business", Frame → True,
         FrameLabel → {"r", "Delta(r)"}, ImageSize → 300],
        ListPlot[Values@volumes⟦idx⟧,
         Joined → True, PlotTheme → "Business", Frame → True,
         FrameLabel → {"r", "V(r)"}, ImageSize → 300]
       }]
     }]], (
    clickPt = #;
    selectedVertex = FindNearestVertex[g, clickPt]) &],
 {{displayRadius, 5, "Neighborhood radius"}, 1, Dynamic[graphRadius], 1},
 {{minRadius, 1, "r_min"}, 1, Dynamic[graphRadius], 1},
 {{maxRadius, 10, "r_max"}, 1, Dynamic[graphRadius], 1},
 TrackedSymbols :→ {displayRadius, selectedVertex, minRadius, maxRadius}]];
```

Here we visualize the mean Δ for $r \in [r_{\min}, r_{\max}]$ for each point in the selected neighborhood and for the neighborhood as a whole.

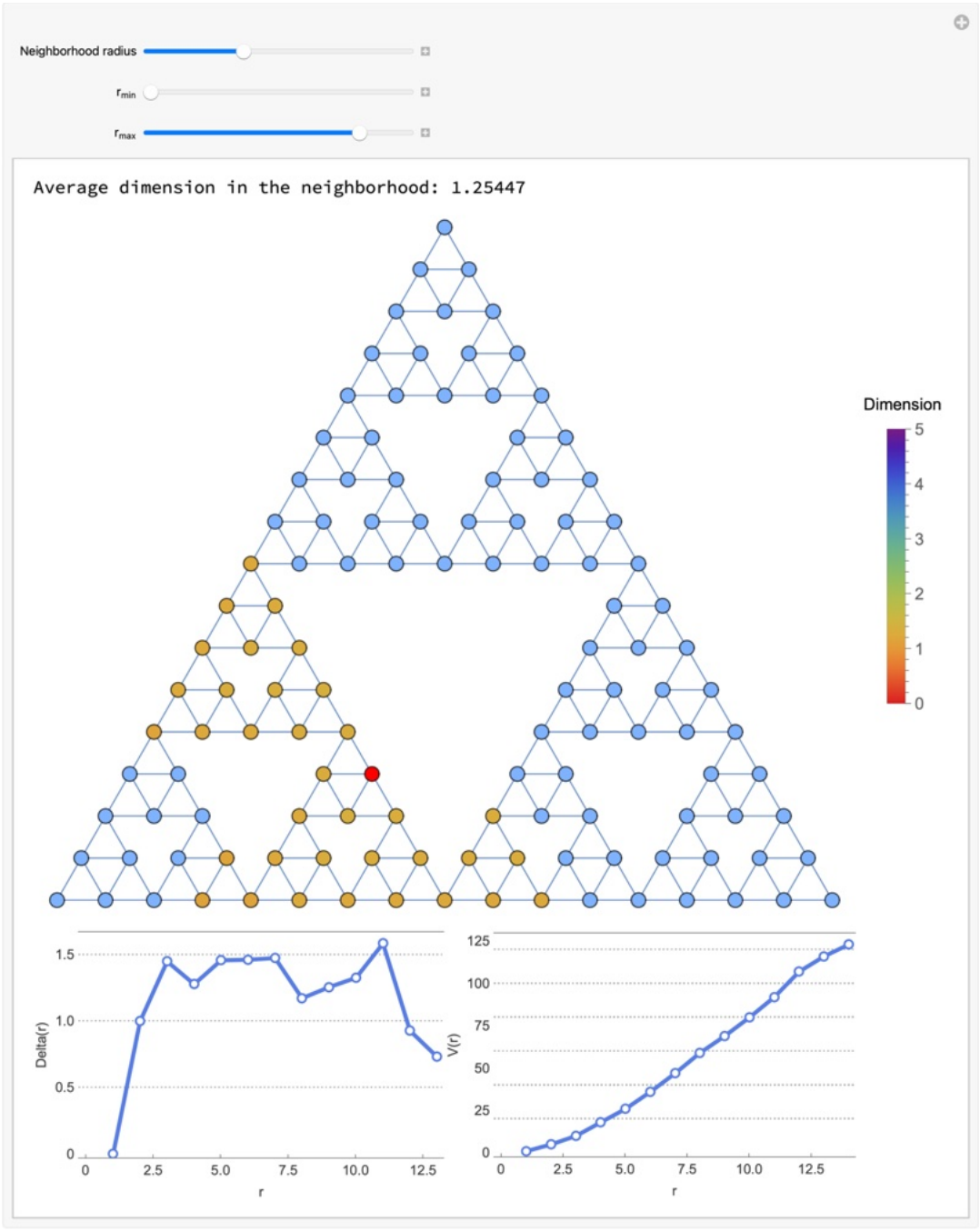
In[12]:=
```
With[{g = GridGraph[{20, 20}]}, VisualizeGraphDimension[g]]
```

Out[ ]=



In[ ]:=
```
With[{g = GraphData[{"Sierpinski", 5}]}, VisualizeGraphDimension[g]]
```

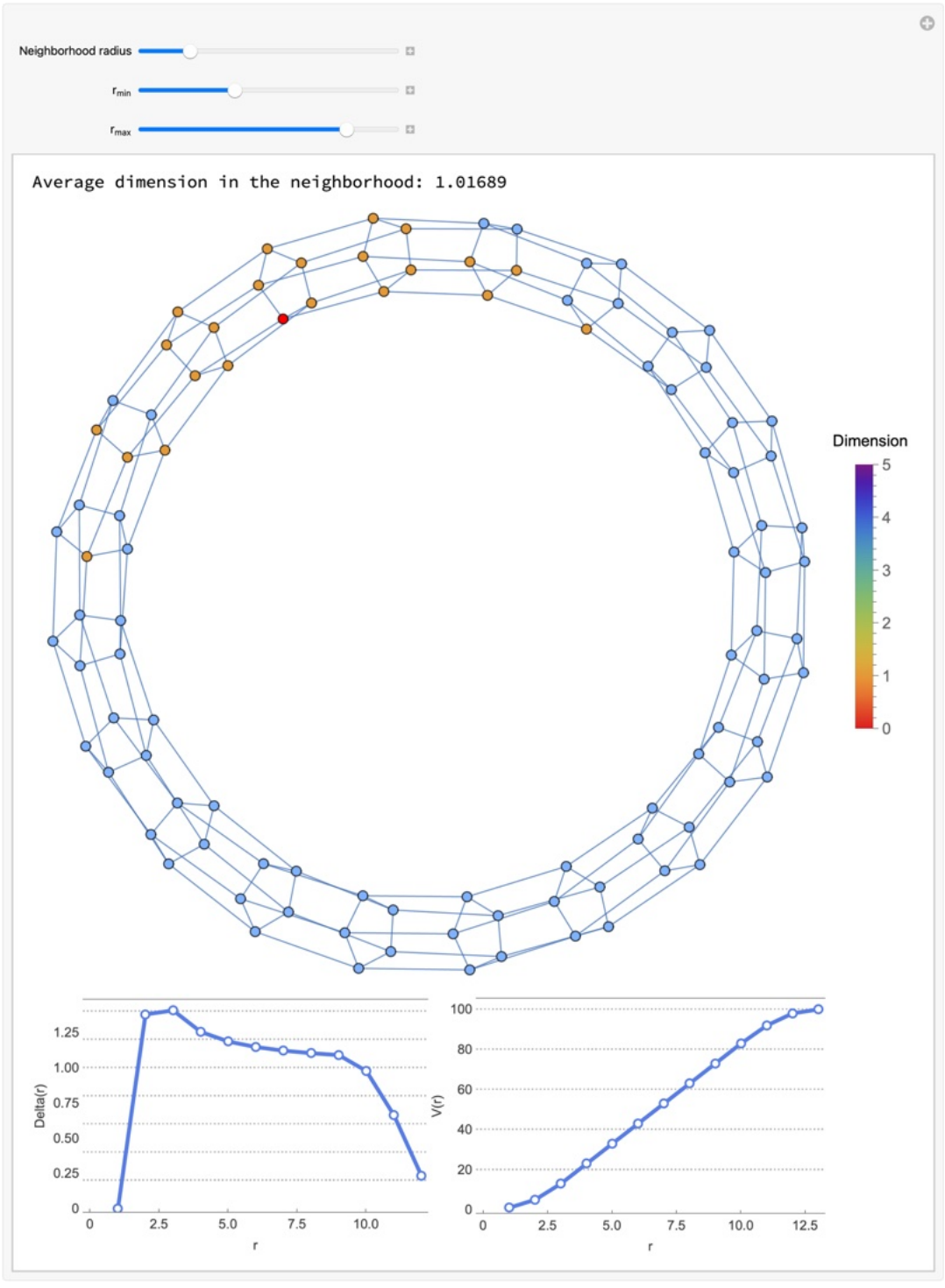
Out[ ]=



- The quantity $\Delta(r)$ is for large r influenced by the **finiteness of the graph,** and for small r by **local irregularities**. We are thus interested in a **plateau** where $\Delta(r)$ is approximately constant.
- There could be other notions of dimension (e.g. based on homology, coordinatization, causal graph, model rules), and their equality can be used as a **test admissible initial conditions**.

In[ ]:=
```
VisualizeGraphDimension[TorusGraph[{5, 20}]]
```

Out[ ]=

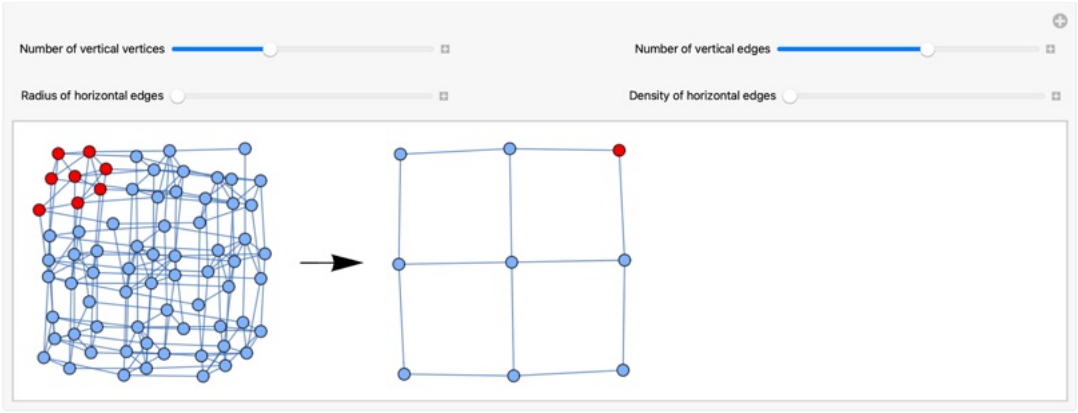


- There are other graph metrics, e.g., **effective resistance**.

Example 2 : Fibered Graph

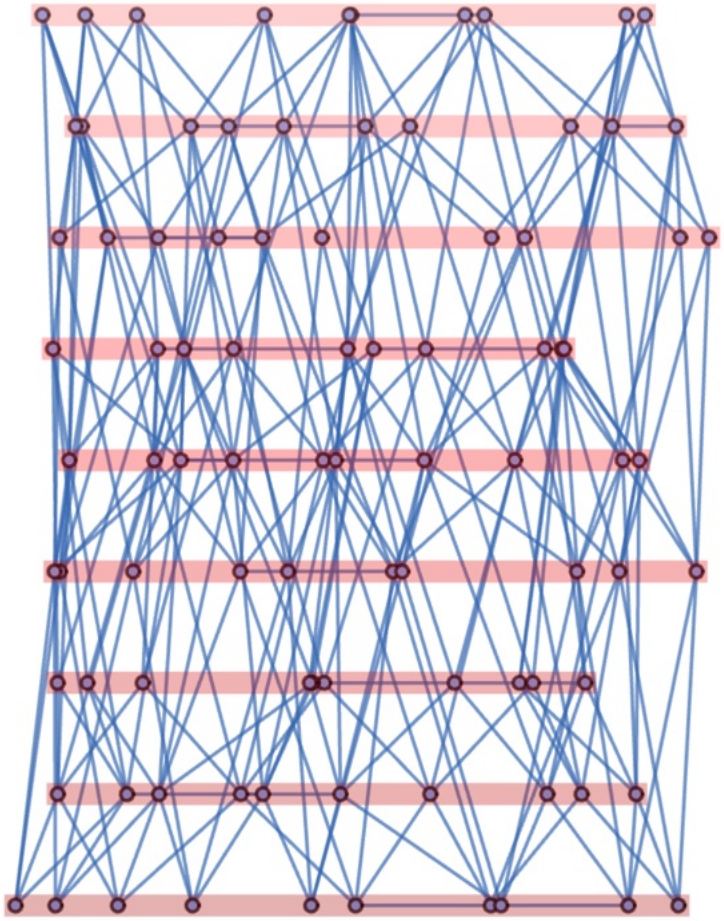
In[ ]:=
```
Module[{g},
 g = GridGraph[{3, 3}];
 RandomFibrationViewer[g, "TotalImageSize" → Small, "BaseImageSize" → Small]
]
```

Out[ ]=



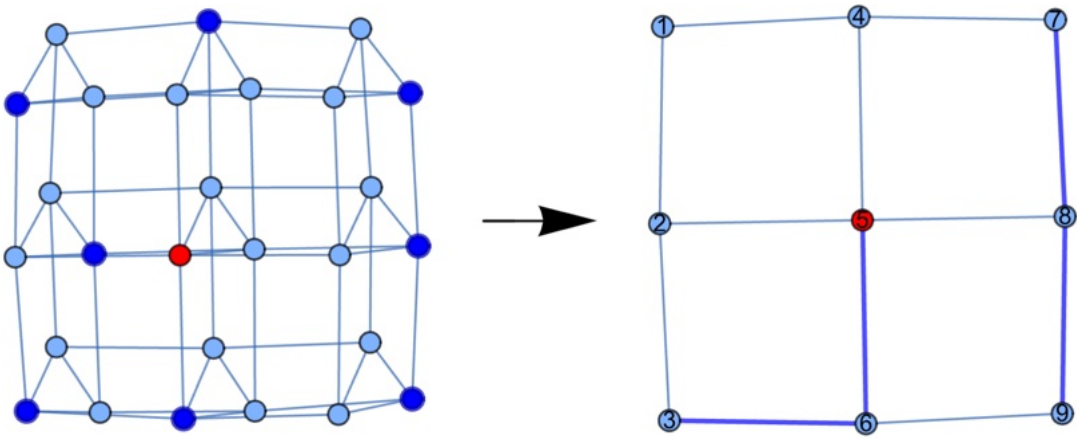
In[ ]:=
```
Module[{g, total, proj},
 g = GridGraph[{3, 3}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 10, "VerticalEdges" → 2,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 2];
 HighlightFibers[
  Graph[total, VertexCoordinates → FibrationEmbedding[total, proj]], proj]
]
```

Out[ ]=



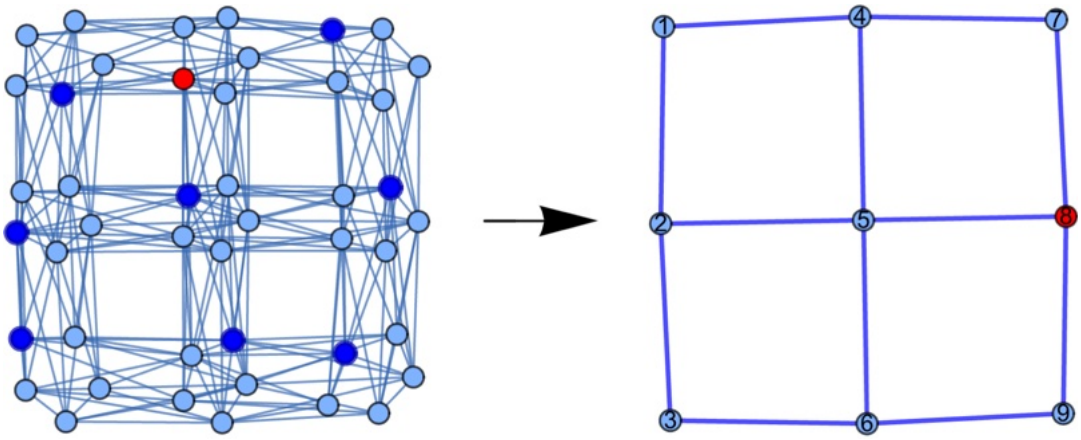
In[•]:=
```
Module[{g, total, proj, s},
 g = GridGraph[{3, 3}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 3, "VerticalEdges" → 5,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 1];
 s = RandomSection[total, proj];
 SectionViewer[total, proj, s,
  "TotalImageSize" → Small, "BaseImageSize" → Small]
]
```

Out[•]=



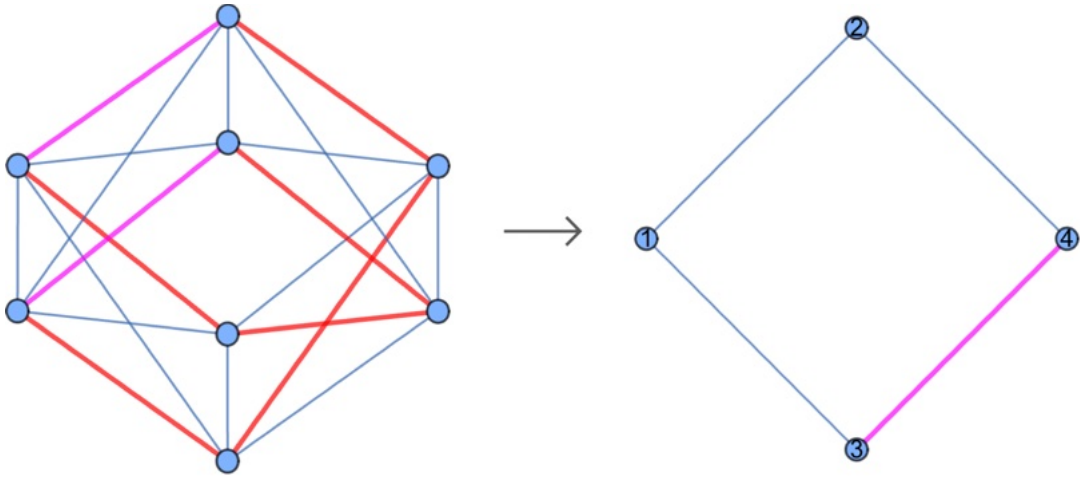
In[•]:=
```
Module[{g, total, proj, s},
 g = GridGraph[{3, 3}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 5, "VerticalEdges" → 5,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 3];
 s = SmoothSection[total, proj, <||>];
 SectionViewer[total, proj, s,
  "TotalImageSize" → Small, "BaseImageSize" → Small]
]
```

Out[•]=



In[ ]:=
```
Module[{g, total, proj, s, c},
 g = GridGraph[{2, 2}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 2, "VerticalEdges" → 2,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 5];
 s = SmoothSection[total, proj, <||>];
 c = RandomConnection[total, proj];
 ConnectionViewer[total, proj,
  c, "TotalImageSize" → Small, "BaseImageSize" → Small]
]
```

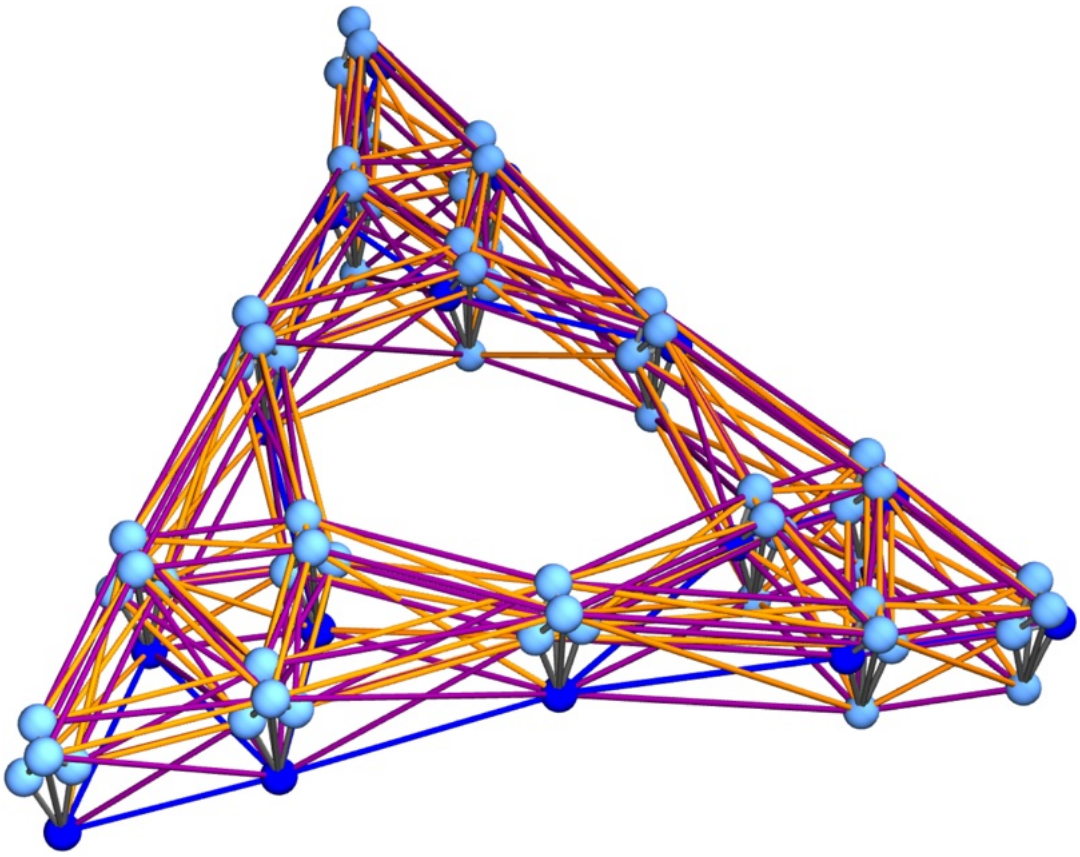
Out[ ]=



- **Minimal model of fiber bundle with chosen gauge and with a connection.**
- Having a smooth section, we can embed it in the plane and anchor the fibers to it.

In[ ]:=
```
Module[{g, total, proj, s, c},
 g = GraphData[{"Sierpinski", 3}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 5, "VerticalEdges" → 10,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 2];
 s = SmoothSection[total, proj, <||>];
 c = RandomConnection[total, proj];
 FibrationGraph3D[total, proj, s, "Connection" → c]
]
```

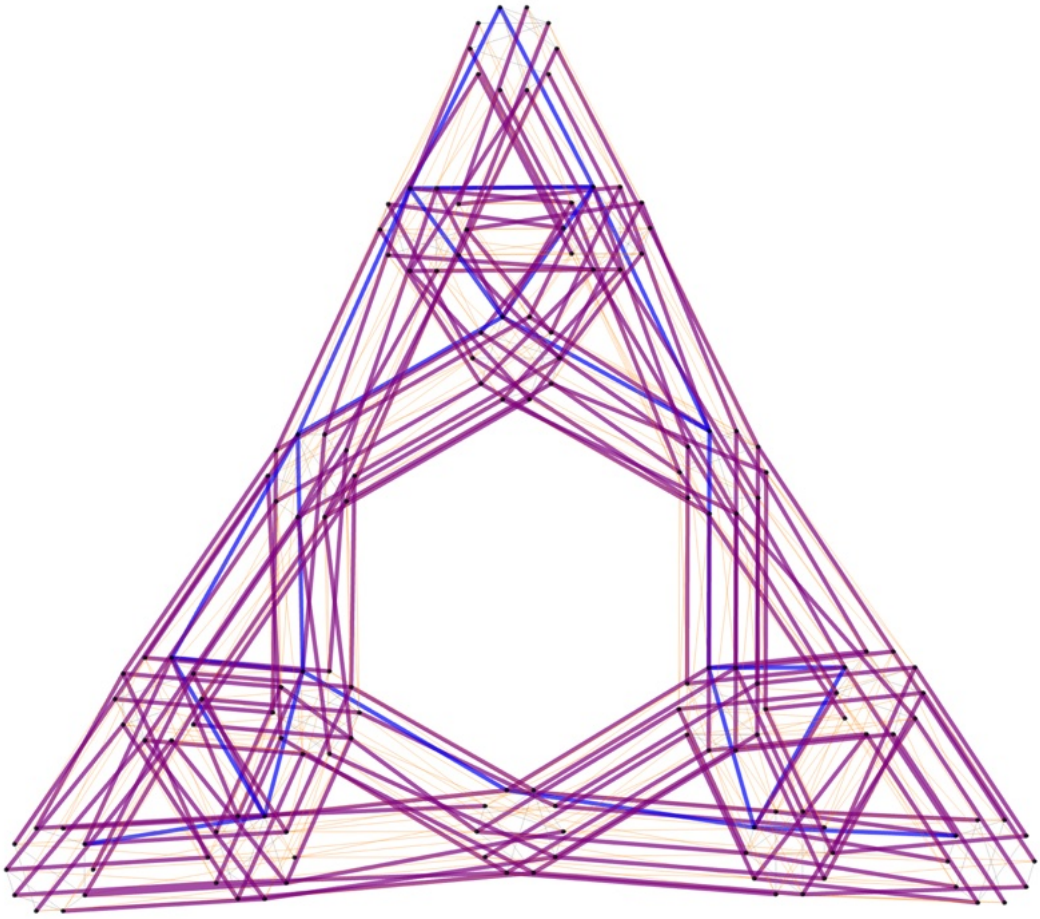
Out[ ]=



- Another embedding draws fibers as circles around the base vertices.

In[ ]:=
```
Module[{g, total, proj, s, c},
 g = GraphData[{"Sierpinski", 3}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 10, "VerticalEdges" → 10,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 2];
 s = SmoothSection[total, proj, <||>];
 c = RandomConnection[total, proj];
 CircularFiberGraph[total, proj, "Dimension" → 2,
  "FiberRadius" → 0.2, "Section" → s, "Connection" → c]
]
```

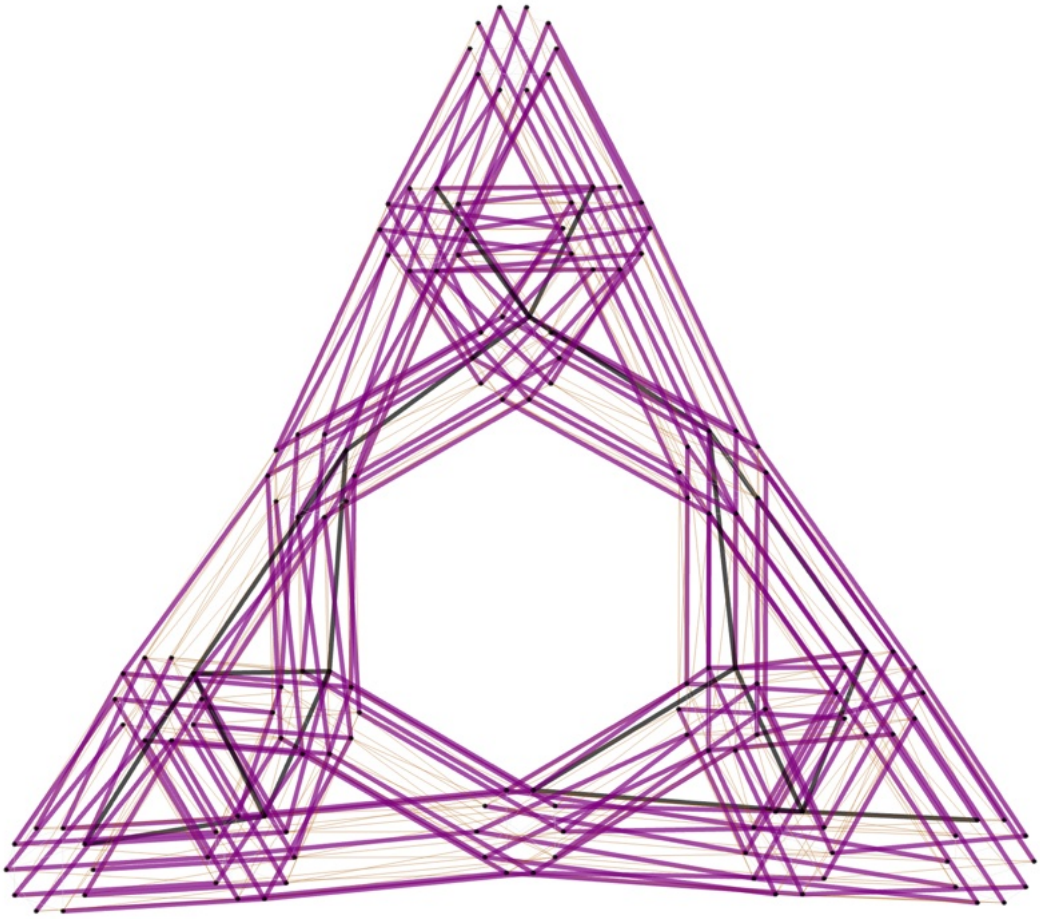
Out[ ]=



- Perhaps there is a notion of connection on the associated Vietoris-Rips complex?
- Now we can do **horizontal lifting** and compute **holonomy** and its trace = **Wilson loop**.

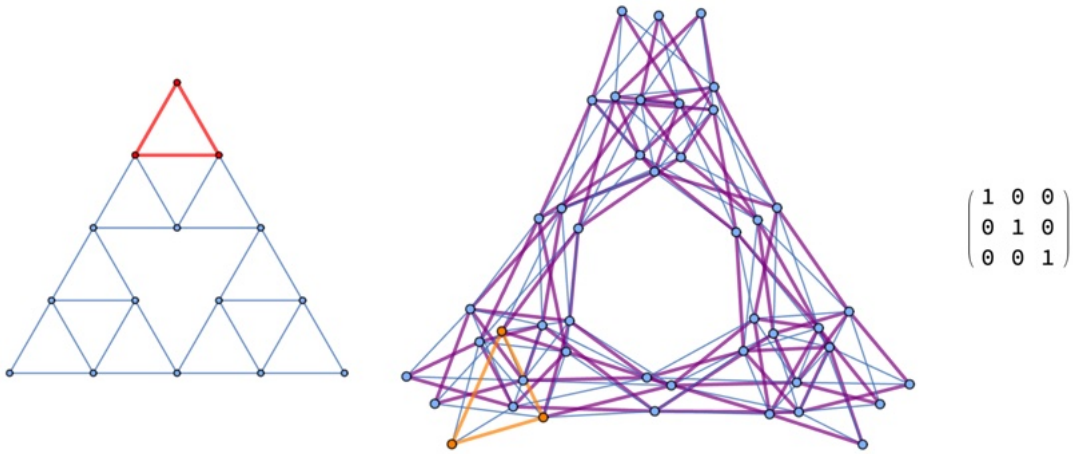
In[ ]:=
```
Module[{g, total, proj, s, c, cycles, loop},
 g = GraphData[{"Sierpinski", 3}];
 {total, proj} =
  RandomGraphFibration[g, "VerticalVertices" → 10, "VerticalEdges" → 10,
   "HorizontalEdgesRadius" → 1, "HorizontalEdgesDensity" → 2];
 s = SmoothSection[total, proj, <||>];
 c = RandomConnection[total, proj];
 cycles = FindCycle[base, Infinity, All];
loop = Append[First /@ First[cycles], First[First[First[cycles]]]];
 CircularFiberGraph[total, proj, "Dimension" → 2,
  "FiberRadius" → 0.2, "Section" → s, "Connection" → c]
]
```

Out[ ]=



In[ ]:=
```
base = GraphData[{"Sierpinski", 3}];
{total, proj} = RandomGraphFibration[base,
   "VerticalVertices" → 3, "HorizontalEdgesDensity" → 2];
conn = RandomConnection[total, proj];
cycles = FindCycle[base, Infinity, All];
loop = Append[First /@ First[cycles], First[First[First[cycles]]]];
holMatrix = HolonomyMatrix[total, proj, conn, loop];
fiber = Select[VertexList[total], proj[#] === First[loop] &];
startVertex = RandomChoice[fiber];
horizontalLift = FindHorizontalCurve[conn, proj, startVertex, loop];
Row[{HighlightGraph[
   Graph[base, ImageSize → 200], Style[PathGraph[loop], Red, Thick]],
  Spacer[20], HighlightGraph[Graph[total, ImageSize → 300],
   {Style[EdgeList[conn], Purple, Thick], Style[startVertex, Green,
     PointSize[Large]], Style[horizontalLift, Orange, PointSize[Large]],
    Style[PathGraph[horizontalLift], Orange, Thick]}],
  Spacer[20], MatrixForm[holMatrix]}]
```

Out[ ]=



$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

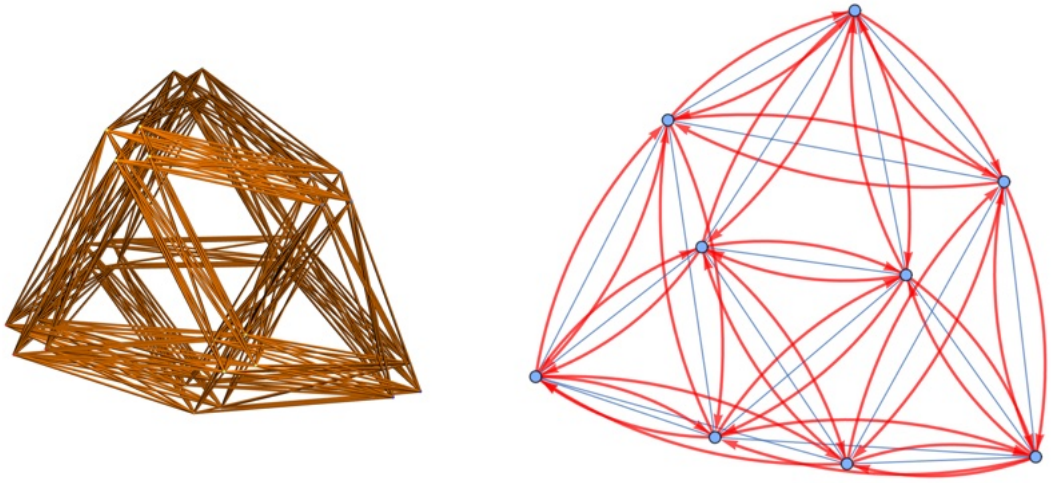
- Define observables on the **moduli space of connections:**
  - Vertex?
  - Edge: Characteristic function of each horizontal edge
  - Base loop: Wilson loop
  - Subgraph?
- G-structure?

Example 3: Tangent Bundle and Exponential Map

- The **tangent bundle** is a fibered graph with fibers tangent vectors and horizontal edges being edges of maximal transverse bipartite graph matchings.

In[•]:=
```
g = TorusGraph[{3, 3}];
{tg, proj} = GraphTangentSpace[g];
Row @
 {CircularFiberGraph[tg, proj, "Dimension" → 3, "FiberRadius" → 0.1], Spacer[50],
  TangentGraph[g]}
```

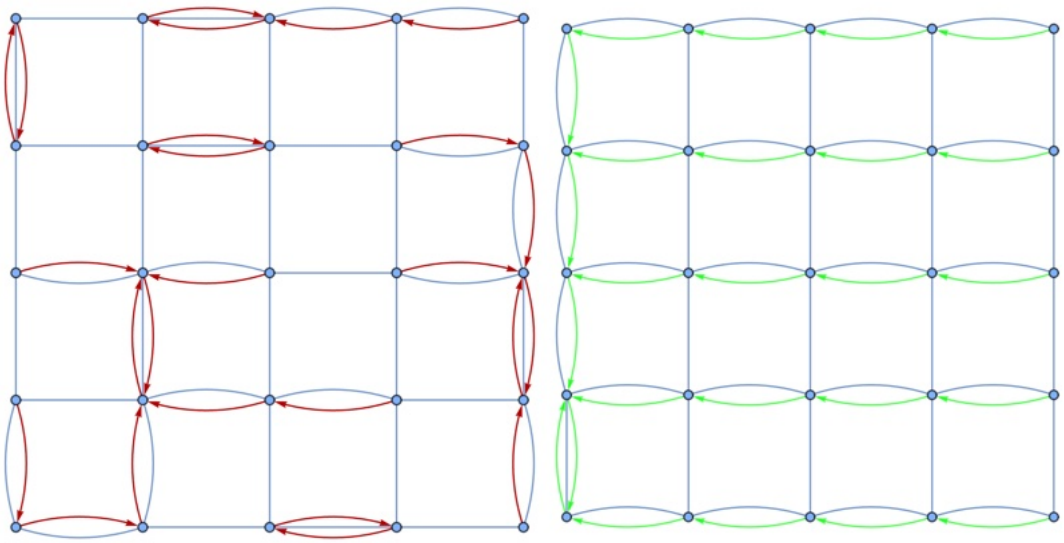
Out[•]=



- The notion of a **smooth section** is a $d = 1$ notion, since it considers only immediate neighbors. However, we already have a $d = 2$ notion of smoothing of vector fields. We should define smoothening of sections for each d.

```
RandomElementaryVectorField2[g_] := With[{vectors = GraphTangentSpace[g]},
   Values @ (RandomChoice /@ GroupBy[vectors, First])];

SmoothenElementaryVectorField2[g_, vf_] :=
  With[ {vfa = Association @ ( Rule @@@ vf )},
   (x ↦ DirectedEdge[x, With[
         {neighbors = VertexOutComponent[g, x, {1}]},
         {targets = vfa /@ neighbors},
         RandomChoice @ MinimalBy[neighbors,
           Sum[GraphDistance[g, target, #] &, {target, targets}]]
        ] ] ) /@ VertexList[g]
  ];

With[{g = GridGraph[{5, 5}]},
 {x = RandomElementaryVectorField2[g]},
 {y = SmoothenElementaryVectorField2[g, x]},
 Row[{HighlightGraph[EdgeAdd[g, x], x, ImageSize → Large],
   HighlightGraph[EdgeAdd[g, y], Style[y, Green], ImageSize → Large]}]
]
```

Out[ ]=



- **Riemannian exponential map**

$$\exp_p : T_pM \to M \text{ sends } v \in T_pM \text{ to } \gamma(1) \in M$$

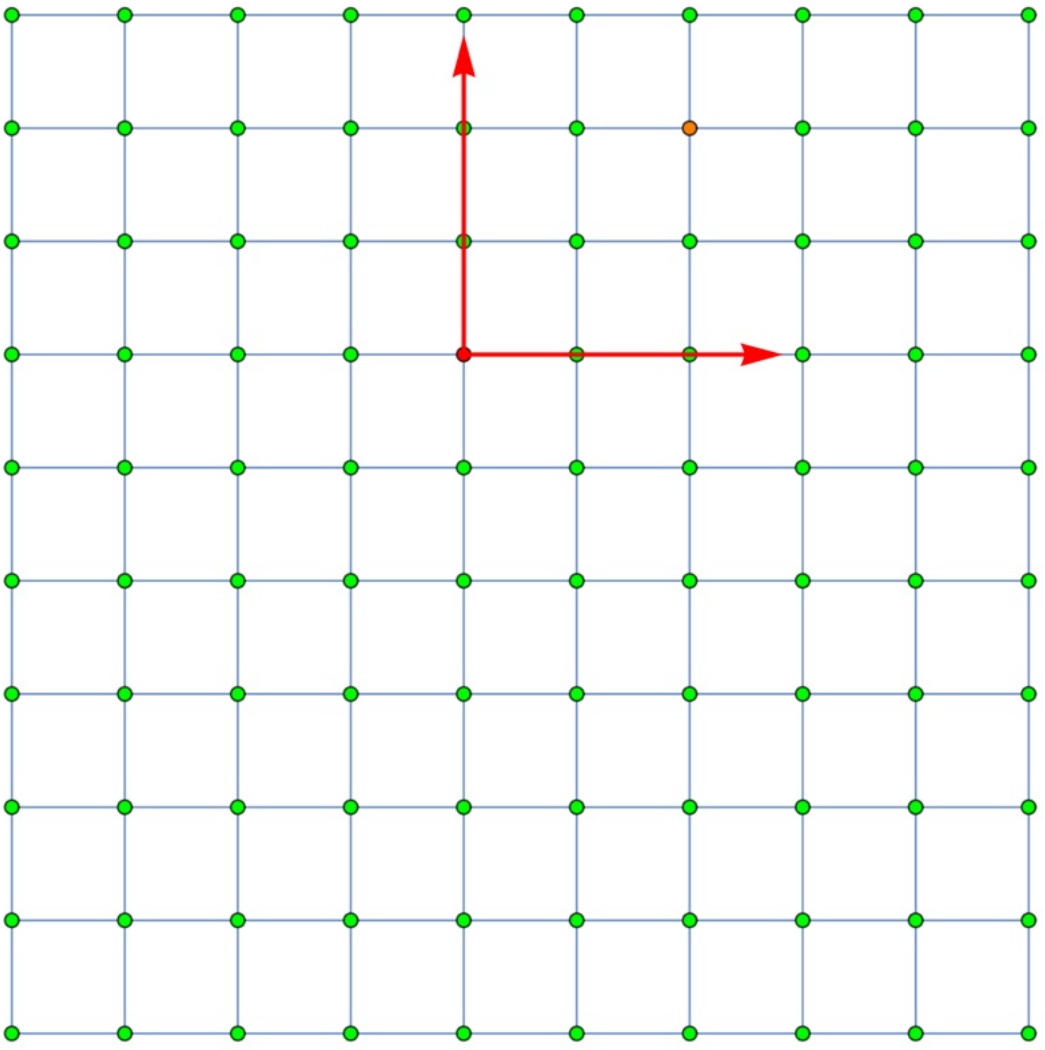
reached by the unique geodesic $\gamma : [0,1] \to M$ with $\gamma(0) = p$ and $\gamma'(0) = v$.

A proposal for minimal model:

For every $x \in G$ and $v_x \in T_x G$ let $n(v_x, y)$ be the number of shortest path from x to y starting with edge v_x.The exponential coordinates of y with respect to the basepoint x is defined as the linear combination $\frac{\sum_{v_x \in T_x G} n(v_x, y) v}{\sqrt{\sum_{v_x \in T_x G} n(v_x, y)^2}}$.

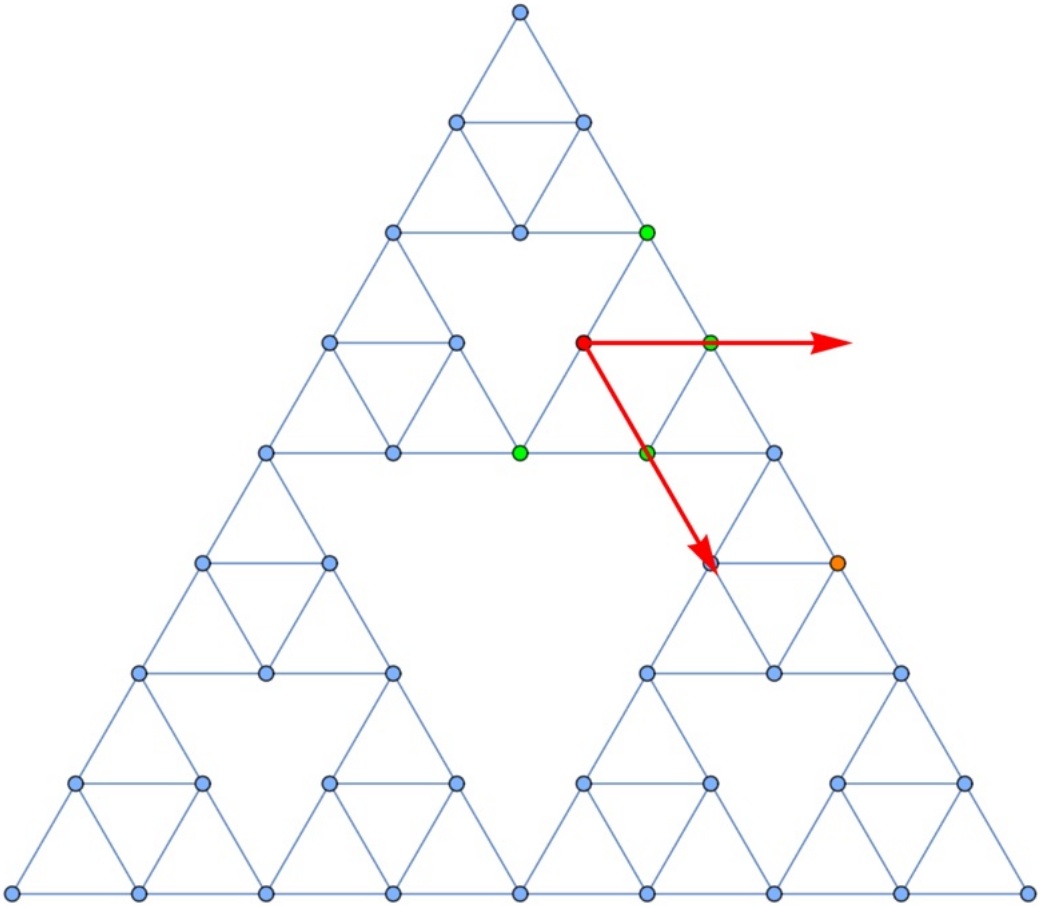
In[ ]:=
```
g = GridGraph[{10, 10}];
ExponentialMapViewer[g, RandomChoice @ VertexList @ g, Infinity]
```

Out[ ]=



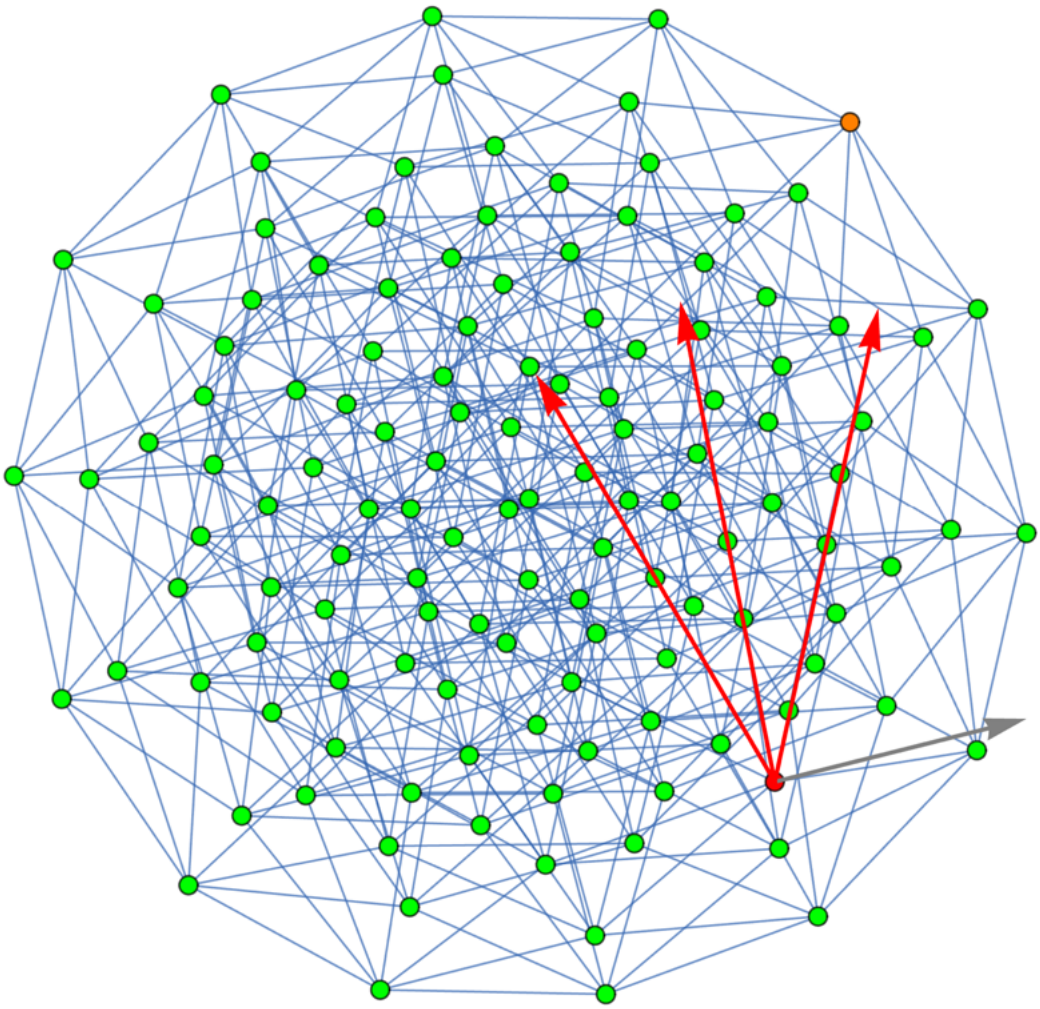
In[ ]:=
```
g = GraphData[{"Sierpinski", 4}];
ExponentialMapViewer[g, RandomChoice @ VertexList @ g, Infinity]
```

Out[ ]=



In[•]:=
```
g = Graph[HypercubeGraph[7]];
ExponentialMapViewer[g, RandomChoice @ VertexList @ g, Infinity]
```

Out[•]=



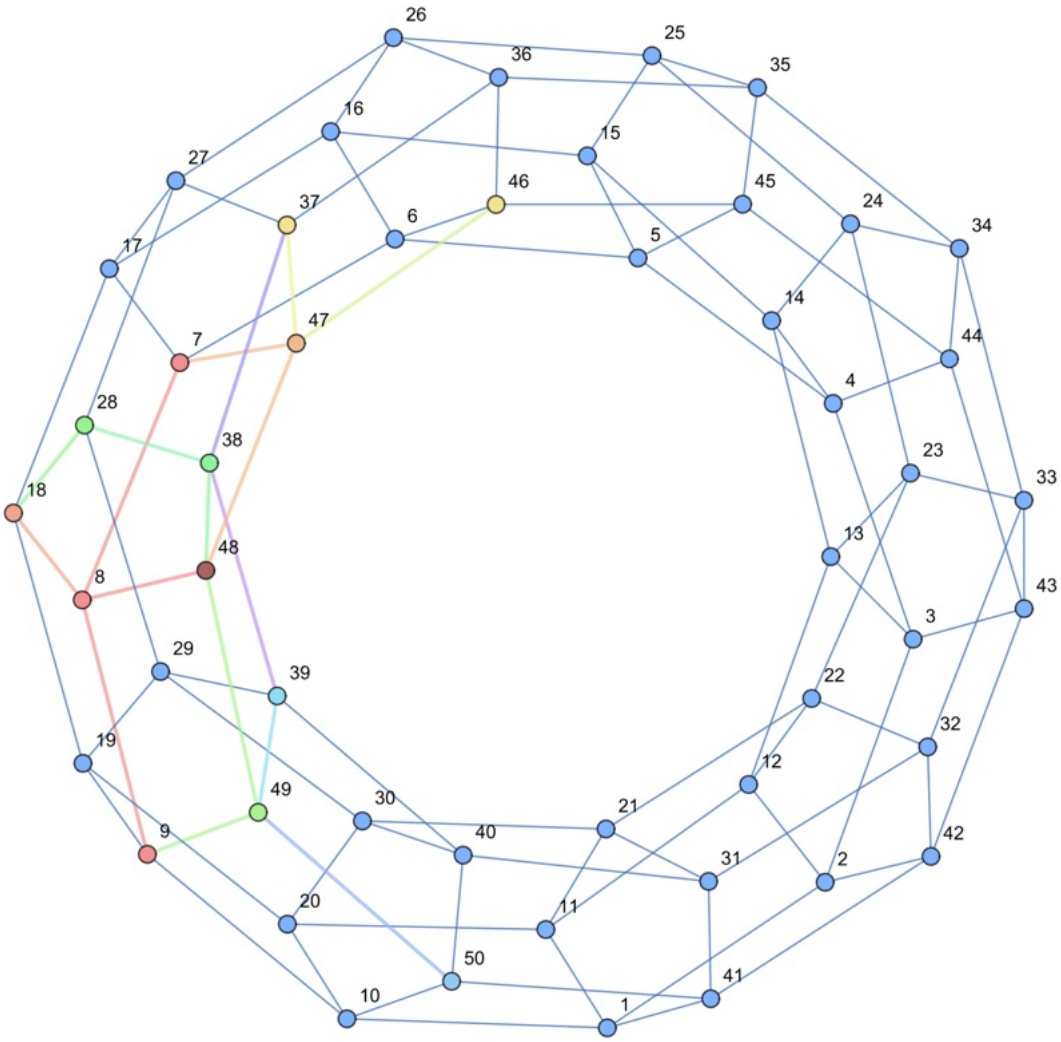
- -v ?
- Lie bracket [X, Y]? Affine connection? (perhaps using the formula of P. Bieliavsky)
- Vector space as limit under subdivision?
- I am also implementing the **cotangent bundle** and the **de Rham complex**. There will be an integration map from differential forms on the graph to cochains on the clique complex, sending the formal exterior product to the **cup product** of cochains.

Example 4 : Topology

- **Theorem of J. Latschev:** For every closed Riemannian manifold M, there exists $\varepsilon > 0$ such that for any $0 < r < \varepsilon$ there exists $\delta > 0$ such that for every metric space X with $d_{GH}(M, X) < \delta$ the **Vietoris–Rips complex** of X at scale r is homotopy equivalent to M.
- Given a **hypergraph**, turn hyperedges into cliques, take the 1-skeleton, and take its Vietoris-Rips complex at scales $r = 1, 2, \ldots, \mathrm{diam}(G)$. Note that for $r = 1$ the Vietoris-Rips complex is precisely the **clique complex.**
- **Problem:** the Vietoris–Rips complex suffers from **combinatorial explosion**. One must either truncate high-dimensional simplices or first embed the data and then triangulate.

In[•]:=
```
g = Graph[TorusGraph[{5, 10}], VertexLabels → Automatic];
VisualizeRipsComplex[g, 2]
```

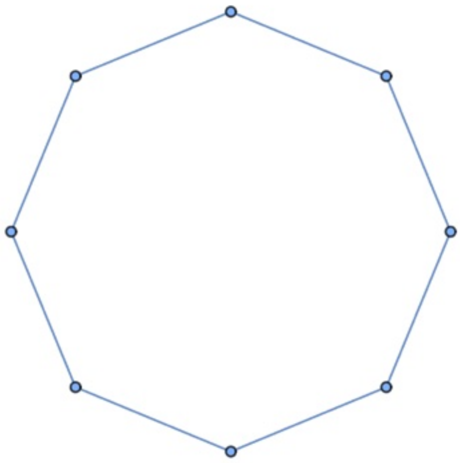
Out[•]=

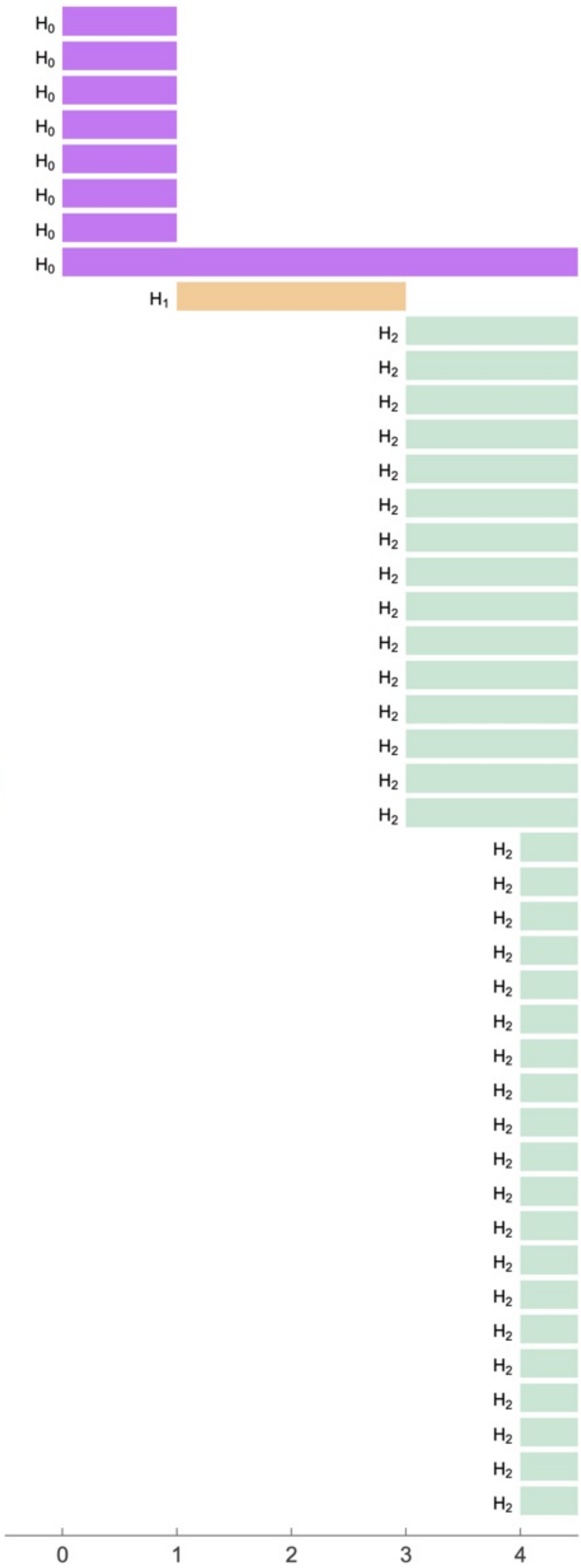


- Given an associated **abstract simplicial complex**, one can compute various topological invariants (Betti numbers, homology, etc.). Having a sequence of ASCs for increasing parameters, one can then compute **persistent homology**.

In[∘]:=
```
Row @ { g = CycleGraph[8], PersistenceBarcode[g]}
```

Out[∘]=





- Notice a similar phenomenon as for the dimension:

- At the **quantum scale** $d = 0$, the space is fully discretized.
- At the **mesoscale** $d = 1, 2,$ the homology matches our geometric intuition.
- At the **coarse scale** $d > 3$, the space collapses to a point. The appearance of higher-dimensional homology generators is an artifact of truncating the Vietoris–Rips complex to avoid combinatorial explosion.

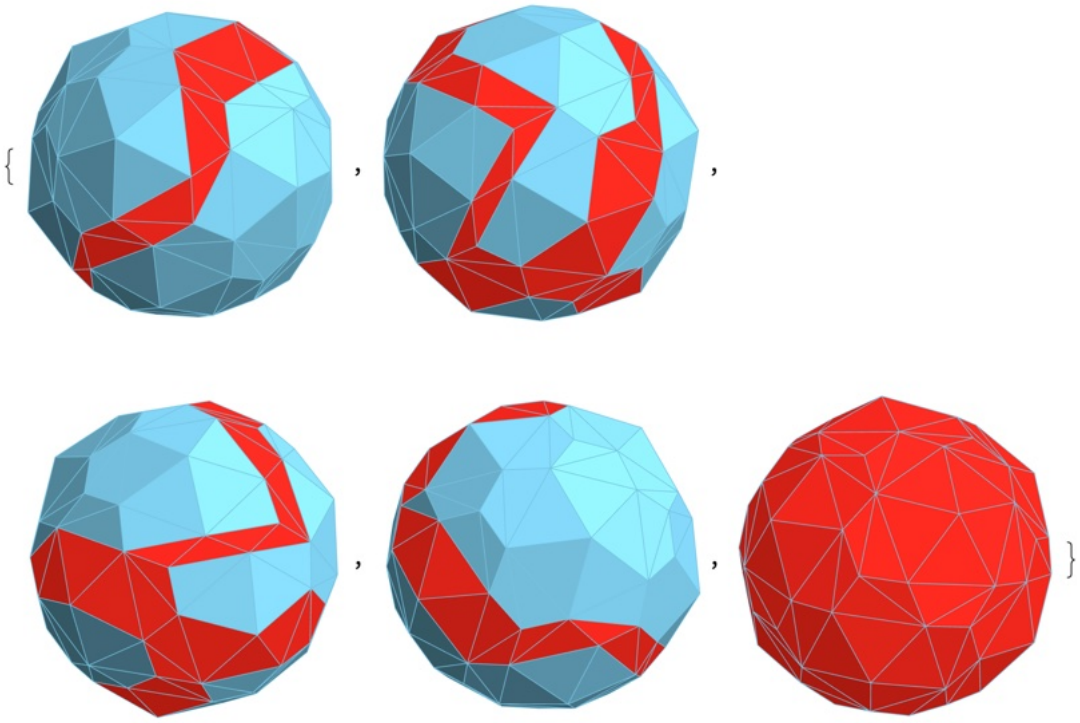
- **Oliver Knill.** https://www.quantumcalculus.org/

  Defines natural differential geometric notions associated with clique complexes - "**quantum geometry**", which is in the spirit of Infrageometry. However, most of his notions require the complex to be a **Dehn-Sommerfeld q-manifold** to be well-defined, which is not exactly "robust" in our sense.

- For example, his notion of a **topological geodesic** requires top-simplices to have the same dimension and to be glued along unique faces. This condition is natural for triangulations but not for emergent structures. But perhaps a **"quantum geodesic"** should branch?

In[•]:=
```
Module[{poly, geodesics},
 poly =
  RegionBoundary@PolyhedronData["PentagonalHexecontahedron", "MeshRegion"];
 geodesics = ComplexGeodesics[MeshComplex @ poly];
 HighlightMesh[poly, Simplex /@ #] & /@ RandomSample[geodesics, 5]
]
```

Out[•]=



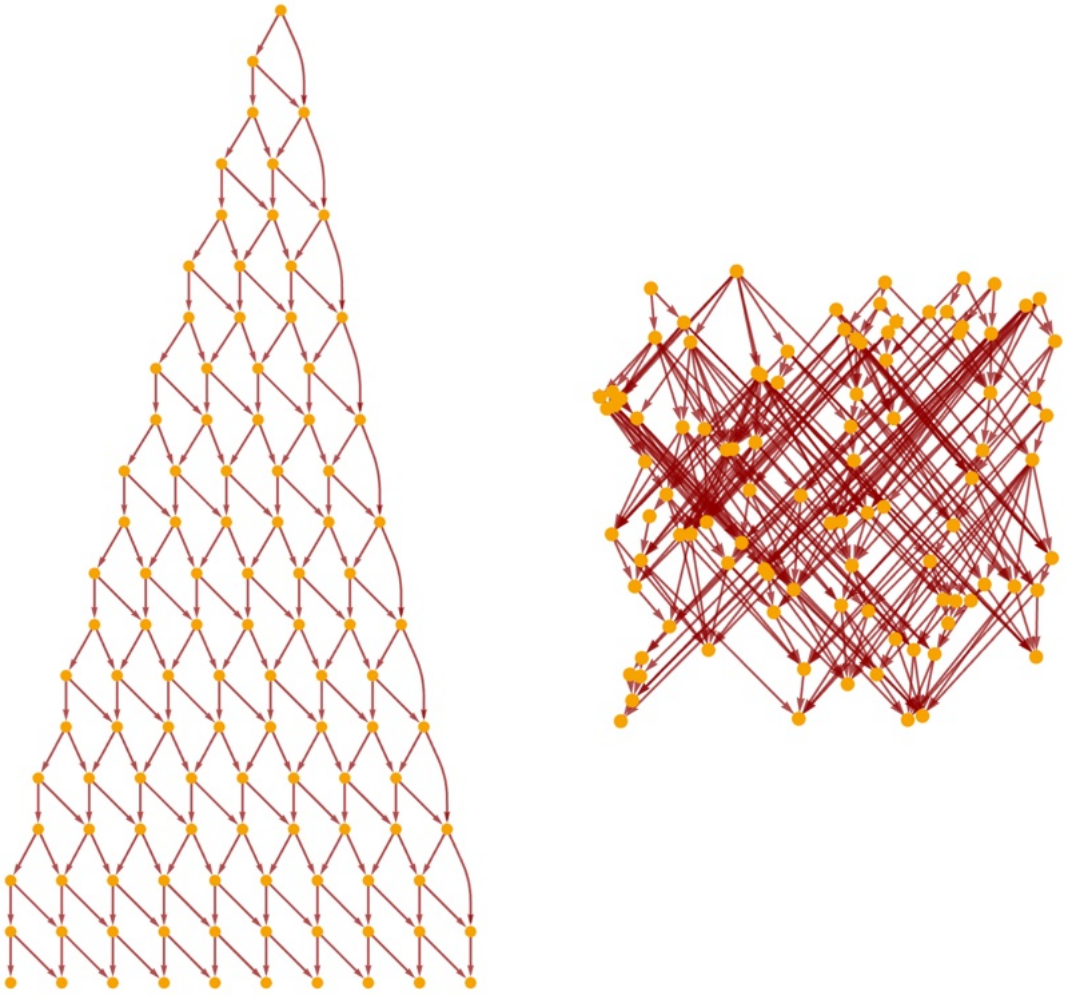
- It is always a question whether to formulate notions at the **level of graphs** or at the **level of simplices** (vertices being points vs midpoints of top-simplices being points). The explicit duality will be realized by the form-integration map mentioned earlier. For connections, for example, we do not yet have an ASC-level analogue. There is a notion of curvature in both pictures, and we should compare them. We should also consider dimension and related structures.

Example 5: Causal Infrageometry

- **Relativity from causal graphs? Emergent? Axiomatic?**

In[•]:=
```
Row @ { g1 = Graph[WolframModel[{{{1, 1, 2}} → {{1, 1, 3}, {3, 2}},
        {{1, 2}, {2, 3}} → {{1, 2}, {2, 3}}}, {{1, 1, 2}}, 20, "CausalGraph"],
    GraphLayout → "LayeredDigraphEmbedding"], Spacer[40],
  g2 = ResourceFunction["FlatSpacetimeSprinkling"][2, 100]["CausalGraph"] }
```

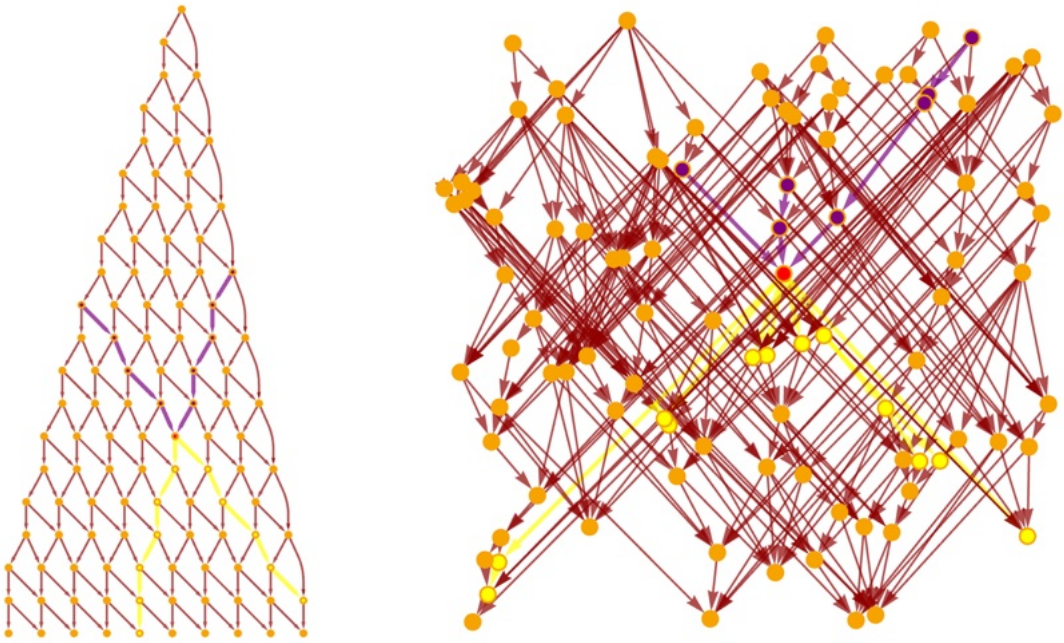
Out[•]=



- **Light ray:** degenerate causal interval

In[•]:=
```
VisualizeLightRays[Graph[#, ImageSize → {Automatic, 300}]] & /@ {g1, g2} //
 Row[Riffle[#, Spacer[50]]] &
```

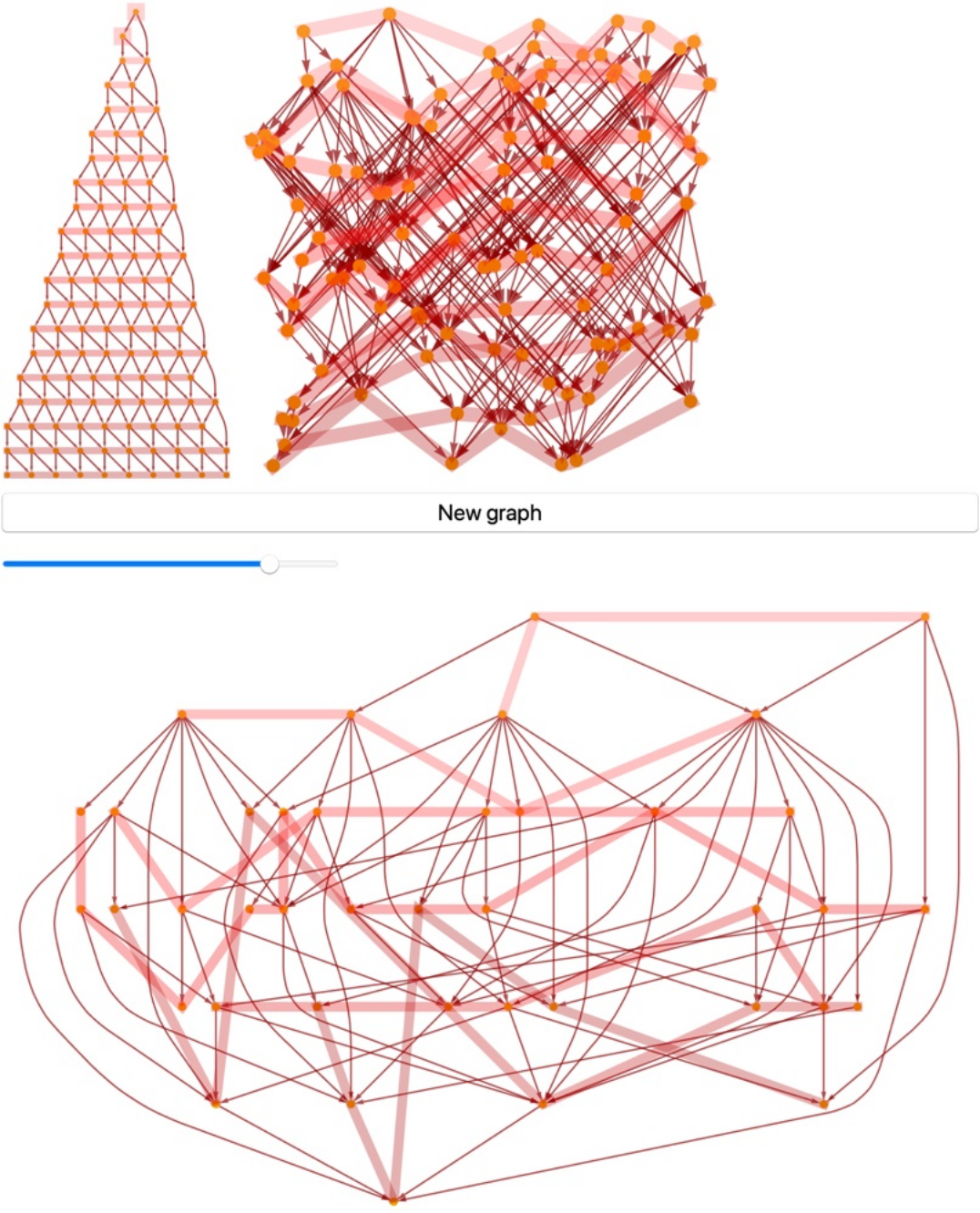
Out[•]=



- **Black hole:** event horizon? White hole?
- **Mass and energy?:** Observer dependent
- **Observer:** Induces foliation (simultaneity), longest path foliation?

In[•]:=
```
Column@ { Row @ {
     HighlightFibers[g1, ProperTime[g1, LongestPathFoliation[g1]]],
     HighlightFibers[g2,
      ProperTime[g2, RandomChoice[ ResourceFunction["GraphFoliations"][g2] ] ] ]
    },
  DynamicModule[{g, f, i = 1}, g = RandomCausalGraph[{20, 40}];
   f = ResourceFunction["GraphFoliations"][g];
   Dynamic@Column[{Row[{Button["New graph",
          g = RandomCausalGraph[{40, 80}];
          f = ResourceFunction["GraphFoliations"][g];
          i = 1;, Method → "Queued"]}], Slider[Dynamic[i], {1, Length@f, 1}],
      Dynamic@Show[HighlightFibers[g, ProperTime[g, f[[Round[i]]]],
         "FiberThickness" → 0.01], ImageSize → Large]}]]
}
```

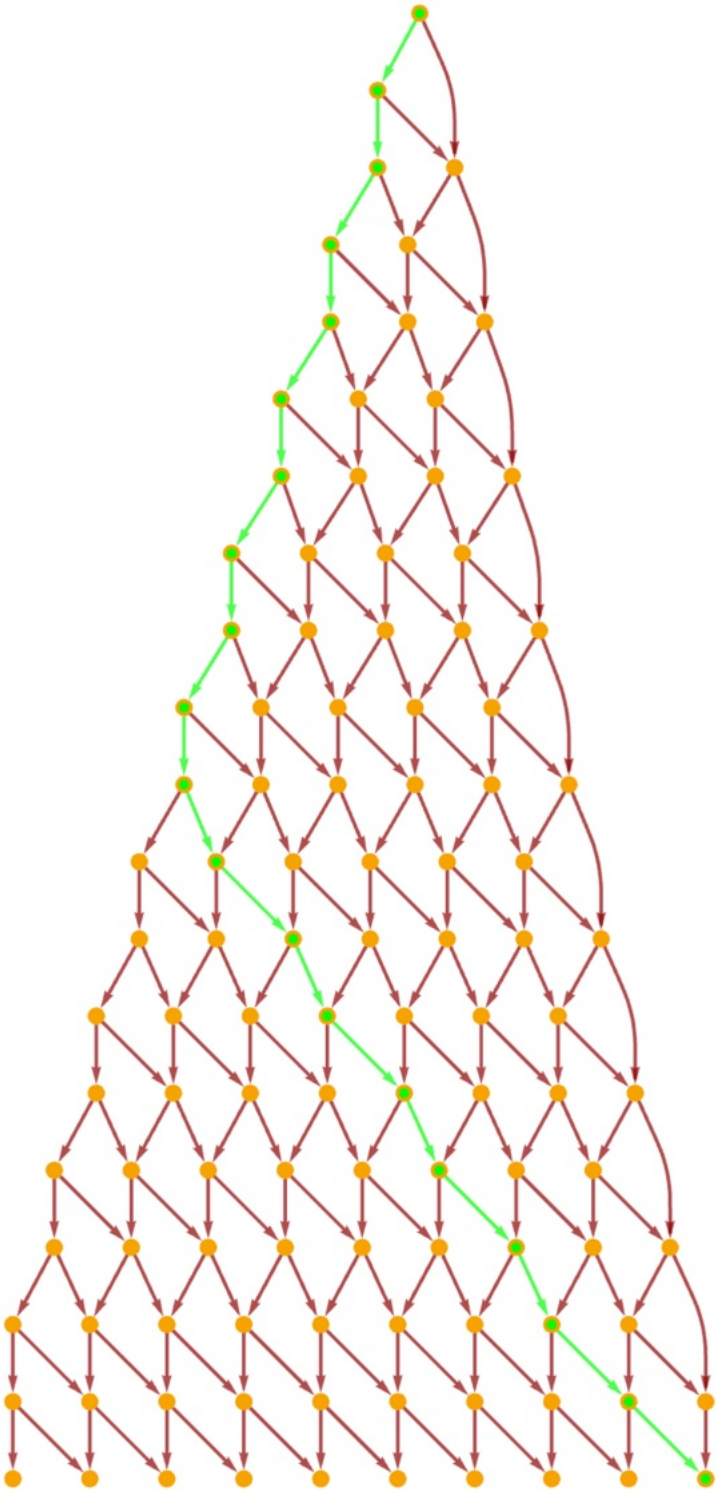
Out[ ]=



- **Minimal model of observer?** Chain? **Clock?**

In[ ]:=
```
HighlightGraph[g1, Style[Subgraph[g1, FindLongestChain[g1]], Green]]
```

Out[ ]=



Synthesis: Coordinatization pipeline

Reconstruction of spatial geometry from a causal graph using:

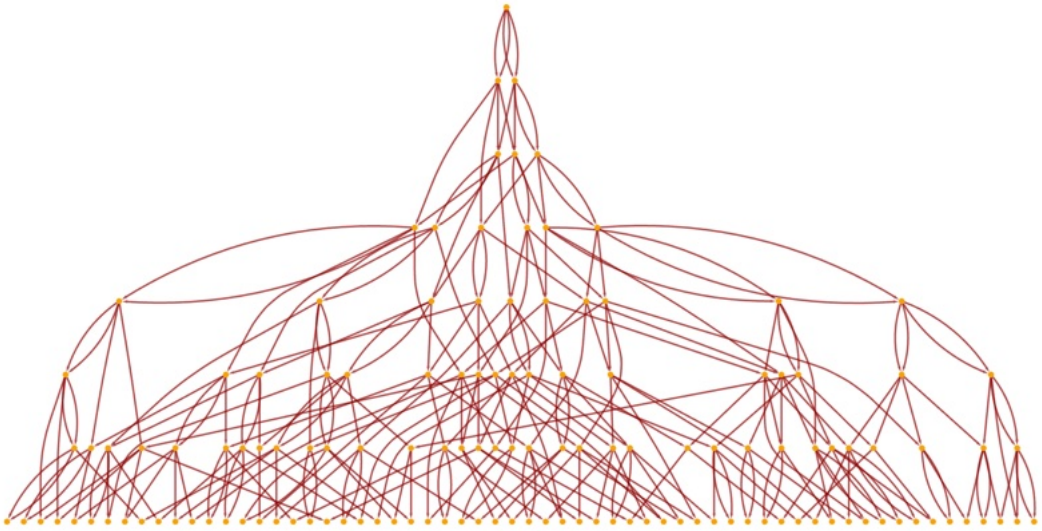
- **minimal model of an observer as a causal edge**
- **minimal model of spatial separation as the existence of an immediate common-ancestor event.**

The following is the pipeline:

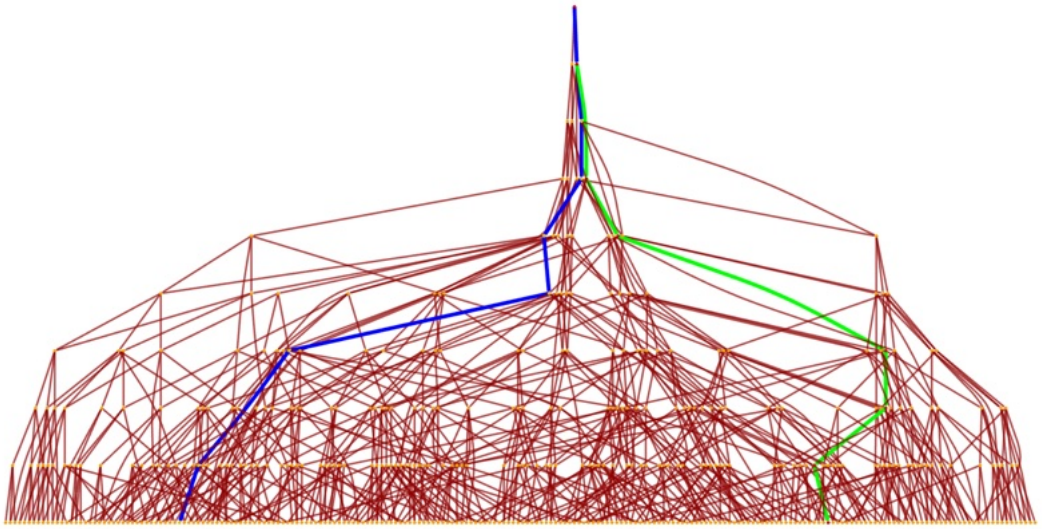
- Take a causal graph of hypergraph rewriting:

In[•]:=
```
g = ResourceFunction["WolframModel"][
  {{x, y}, {x, z}} → {{x, y}, {x, w}, {y, w}, {z, w}}, {{0, 0}, {0, 0}}, 8][
 "LayeredCausalGraph", AspectRatio → 1 / 2]
```

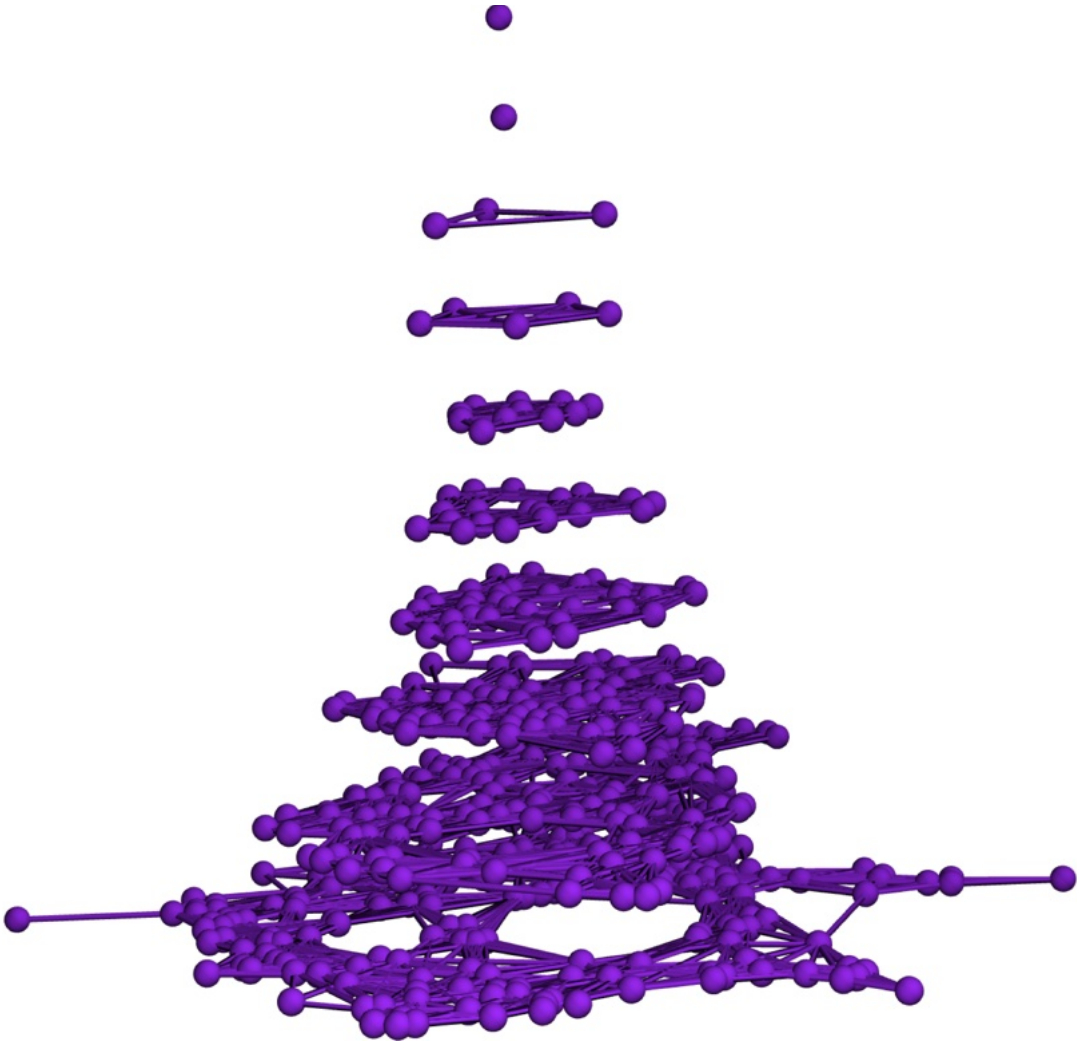
Out[•]=



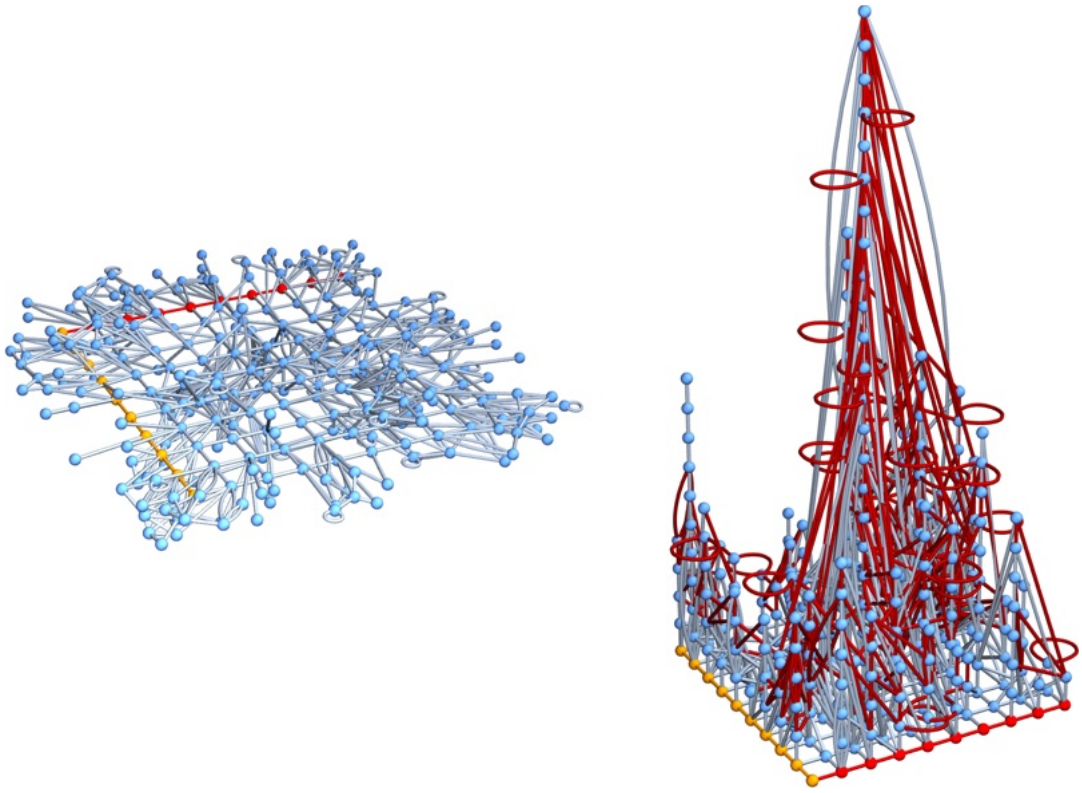
- Pick observer chains (2 observer chains = 1+1 theory):



- Coordinatize the causal graph using light-cone coordinates and reconstruct geometry of constant-time spatial slices via the immediate common-ancestor relations (could be upgraded to irreducible n-ary relations):



- Take the **induced coordinatization of each spatial slice** and regard vertices with the same coordinate as a fiber over that coordinate point (**internal degrees of freedom** not visible to the relativistic observer).



- Do **gauge theory**: enumerate connections, compute Wilson loops, and analyze associated observables.

Setting

In[13]:=
```
<< WolframInstitute`Hypergraph`
<< Wolfram`Multicomputation`

WolframModel = ResourceFunction["WolframModel"];

With[{nb = NotebookDirectory[]}, Get[nb <> #] & /@
    {
     "ExponentialMap.wl",
     "Fibration.wl",
     "LightRays.wl",
     "Coordinatization.wl",
     "Topology.wl"
    }
 ];
```