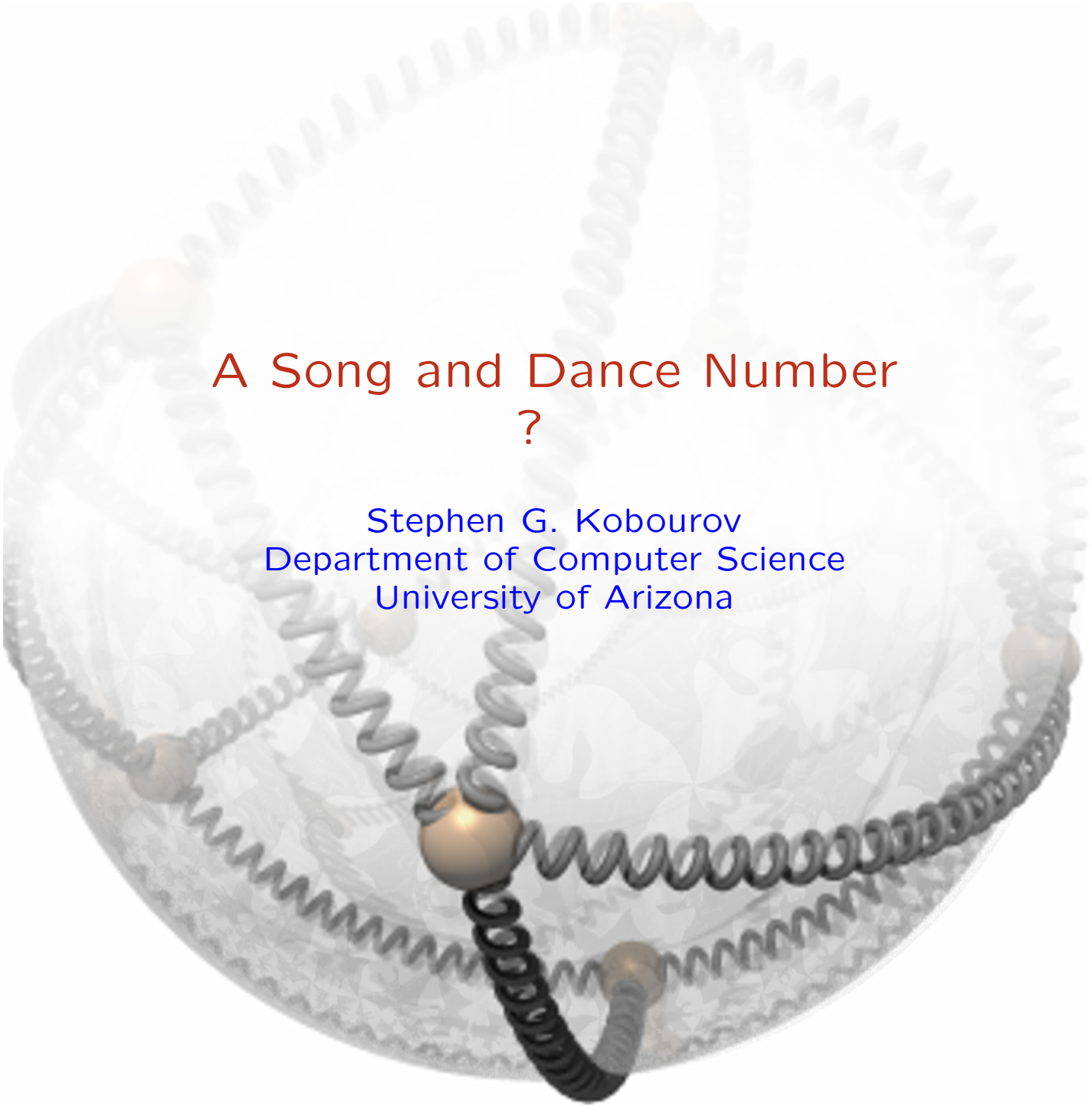
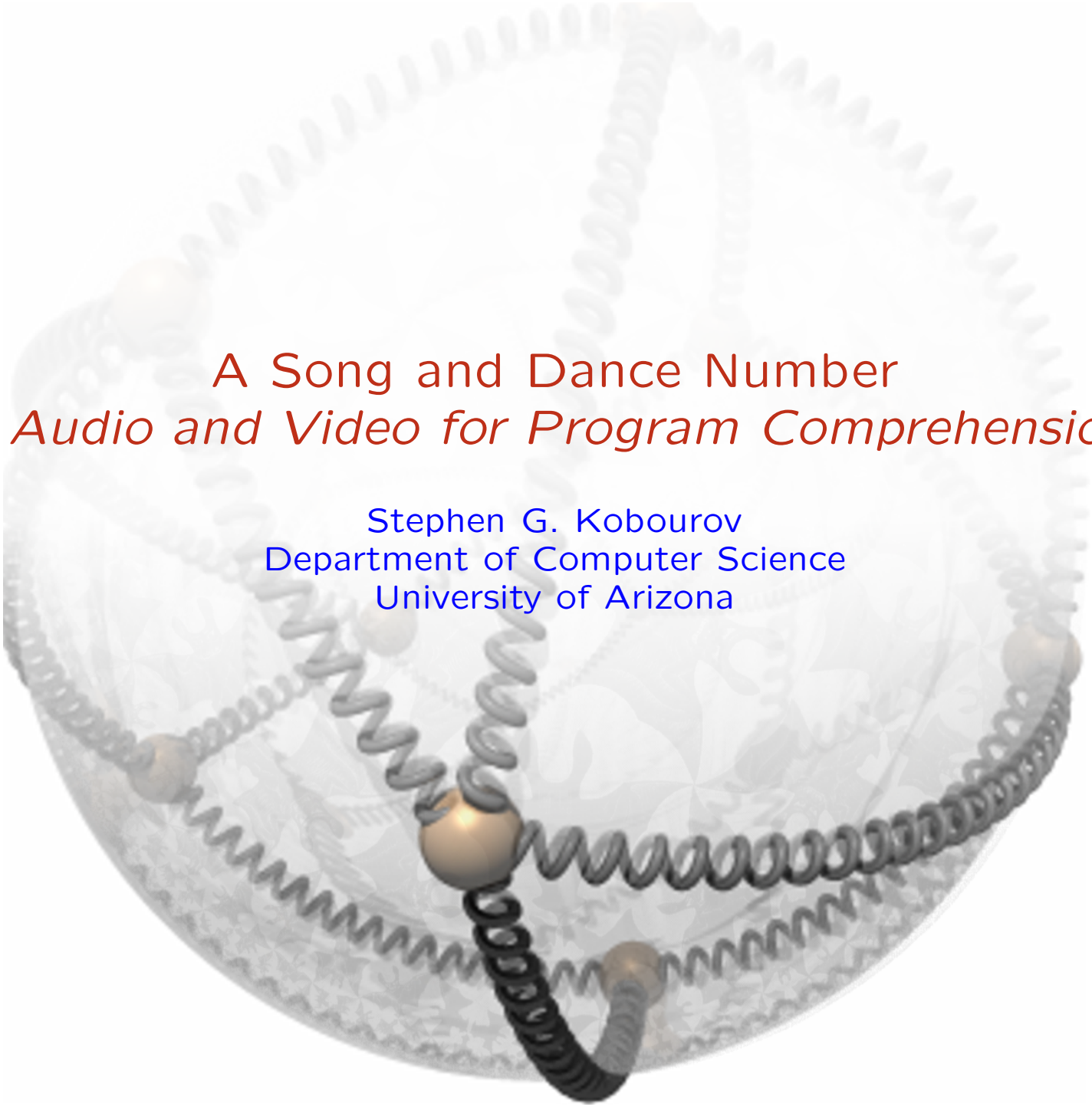
A Song and Dance Number ?

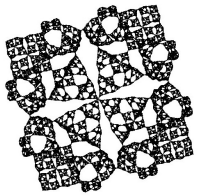
Stephen G. Kobourov
Department of Computer Science
University of Arizona



A Song and Dance Number
(or Audio and Video for Program Comprehension)

Stephen G. Kobourov
Department of Computer Science
University of Arizona

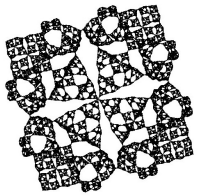




Program Comprehension

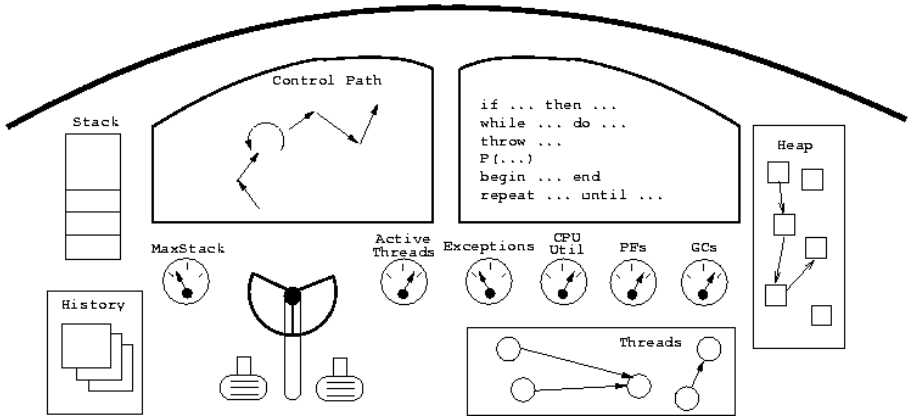
- Real-world programs
  - large, complex, evolving...
  - built by many programmers
  - maintained by other programmers
  - programmers of varying skill levels

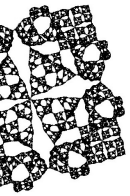
- Program comprehension
  - gather static, dynamic, historical data
  - present data in simple, coherent way
  - development, debugging, maintenance, tuning



Total Program Awareness

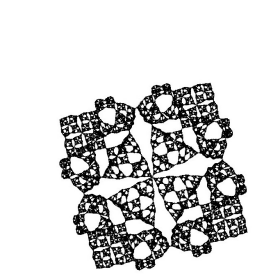
- Visualize and auralize everything: *Programmer's Cockpit*
  - garbage collection activity, exception activity
  - number of page faults, active threads
  - maximum stack depth, execution stack behavior
  - heap graph, thread wait-graphs, call graphs, pdg's
  - control flow graphs, inheritance graphs, package graphs
  - version history, CPU utilization



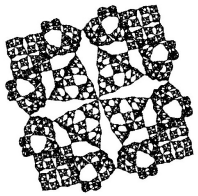


Total Program Awareness

Visualization

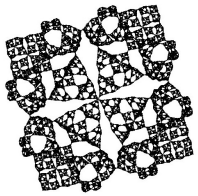


Visualization



Visualization

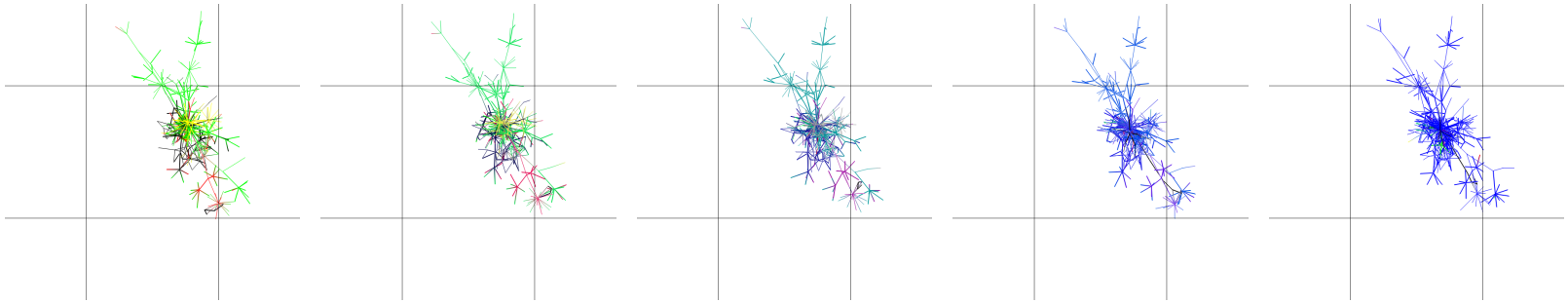
- Why visualize?
  - when we do not know what we are looking for
  - to leverage the visual processing machinery of the brain

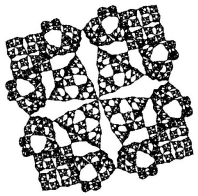


Visualization

- Graphs, graphs, and more graphs
  - static graphs
  - evolving graphs
  - dynamic graphs

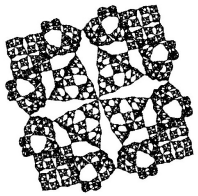
- Evolution of software
  - why is the program structured that way?
  - who worked on which part and when?
  - which parts of the code are unstable?



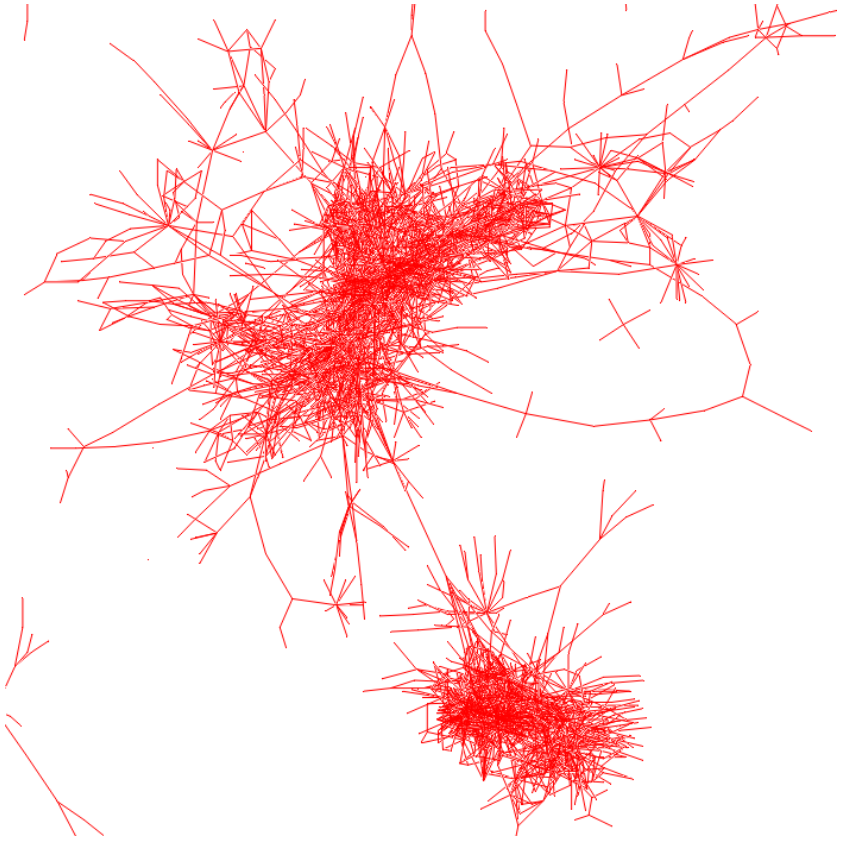


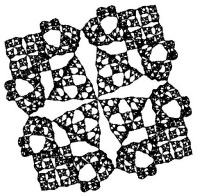
SandMark

- The SandMark system (Christian Collberg)
  - software watermarking
  - tamper-proofing
  - code obfuscation
- SandMark history
  - 26 developers
  - 3+ years of CVS data
  - 100,000+ lines of Java code

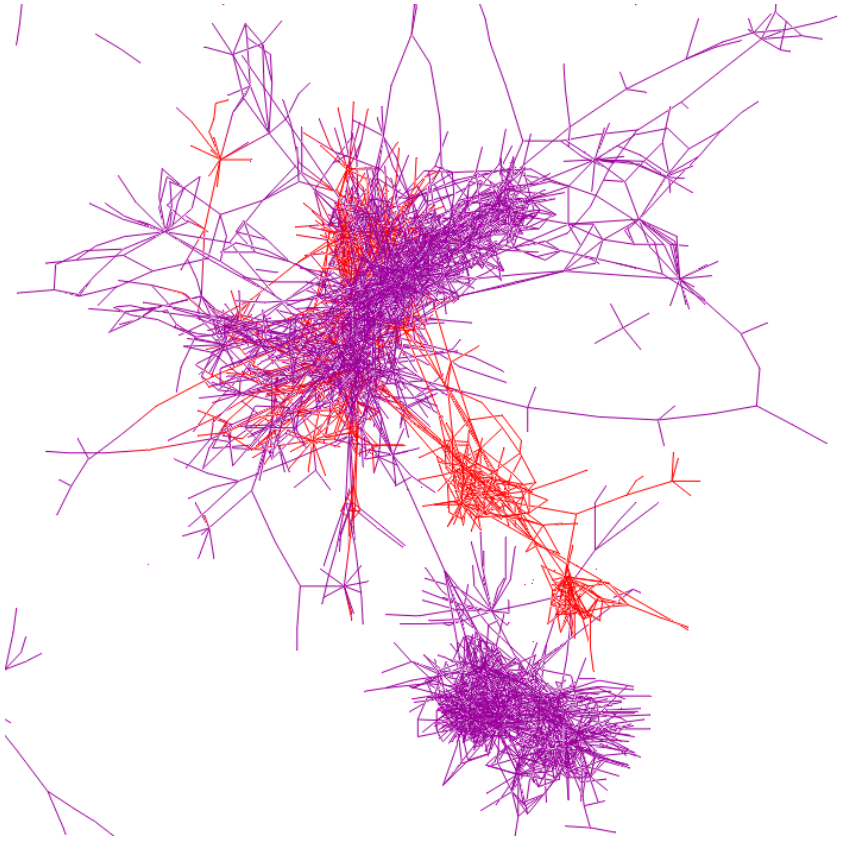


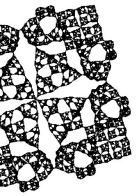
SandMark Call Graph



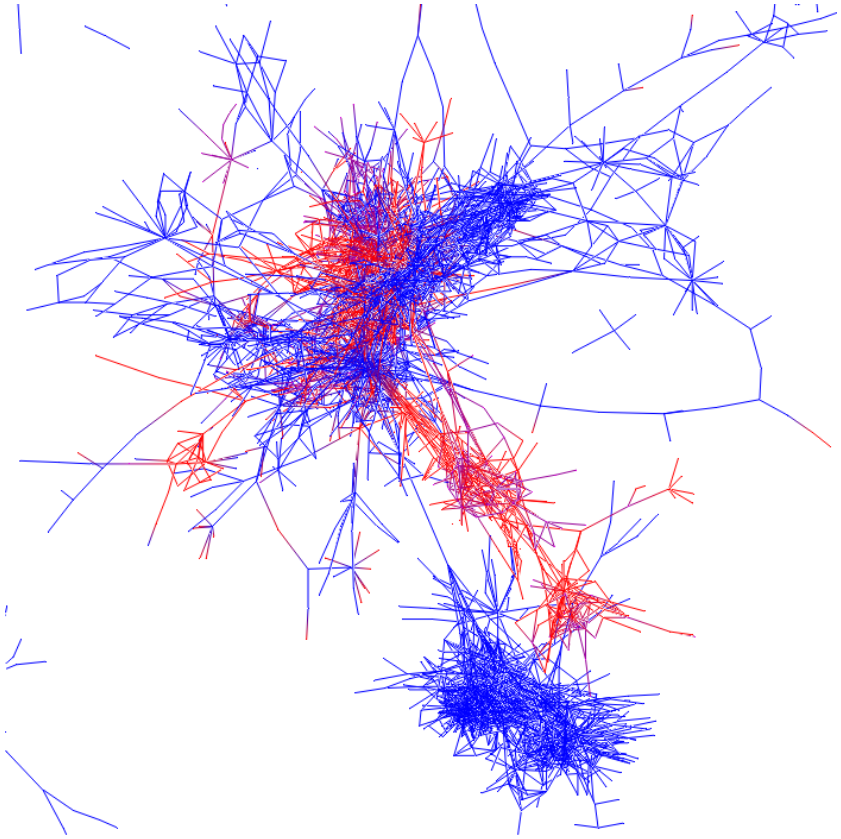


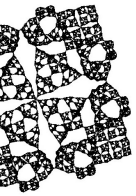
SandMark Call Graph



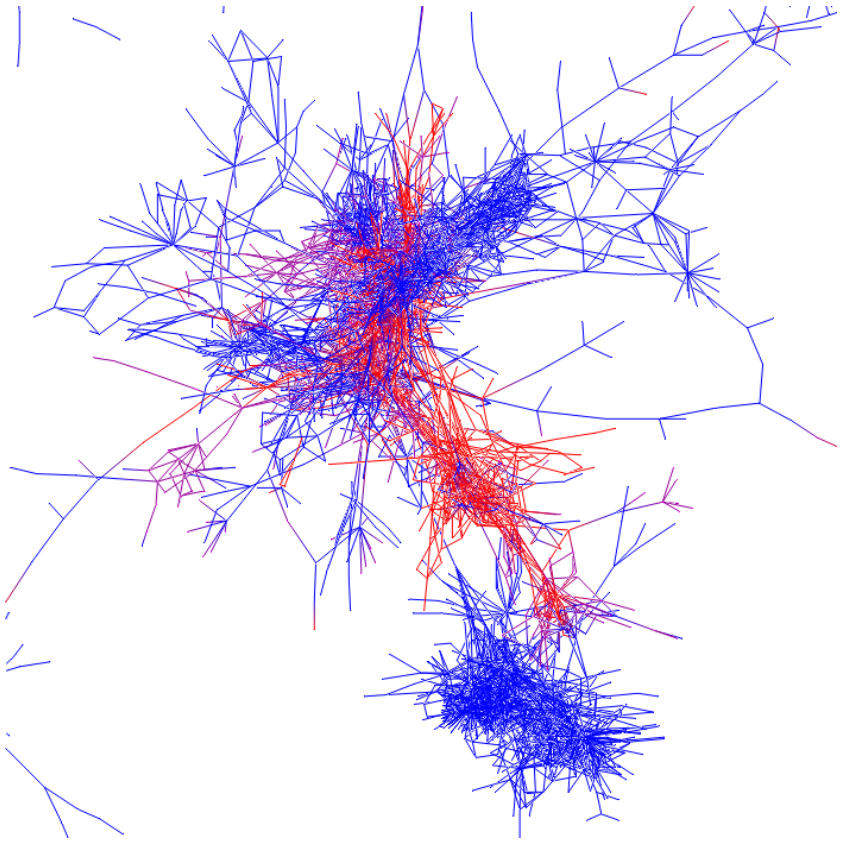


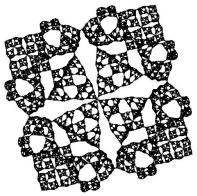
SandMark Call Graph



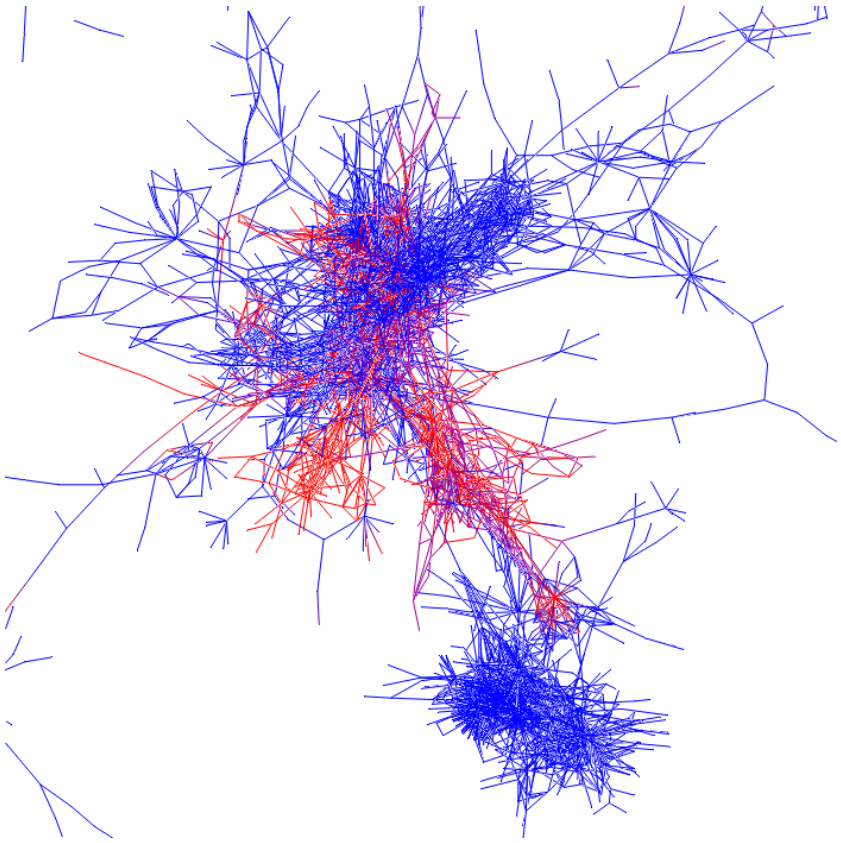


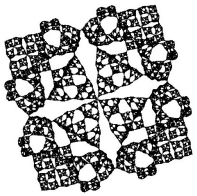
SandMark Call Graph



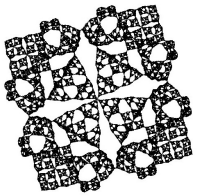


SandMark Call Graph

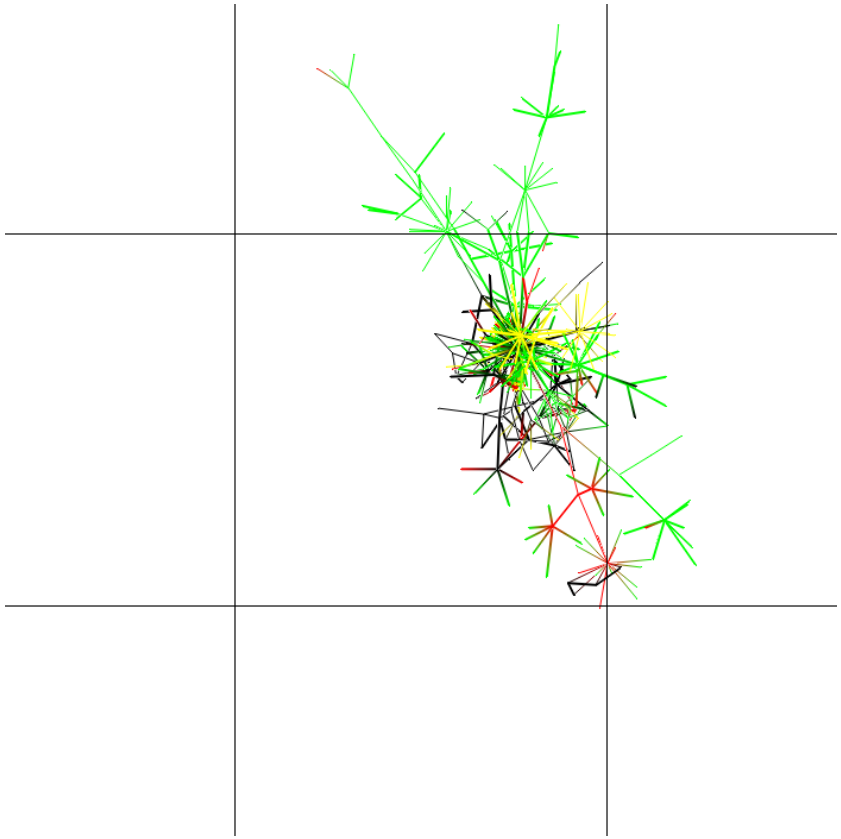


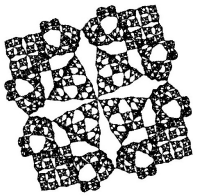


SandMark Call Graph

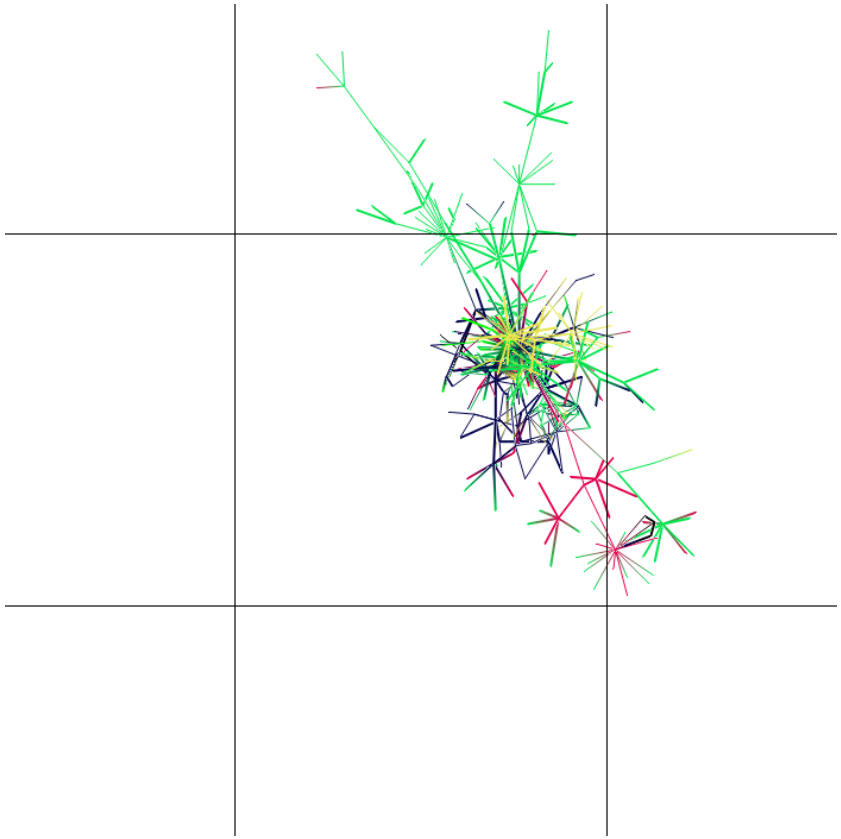


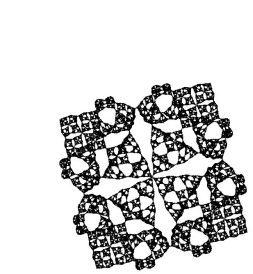
SandMark Inheritance Graph



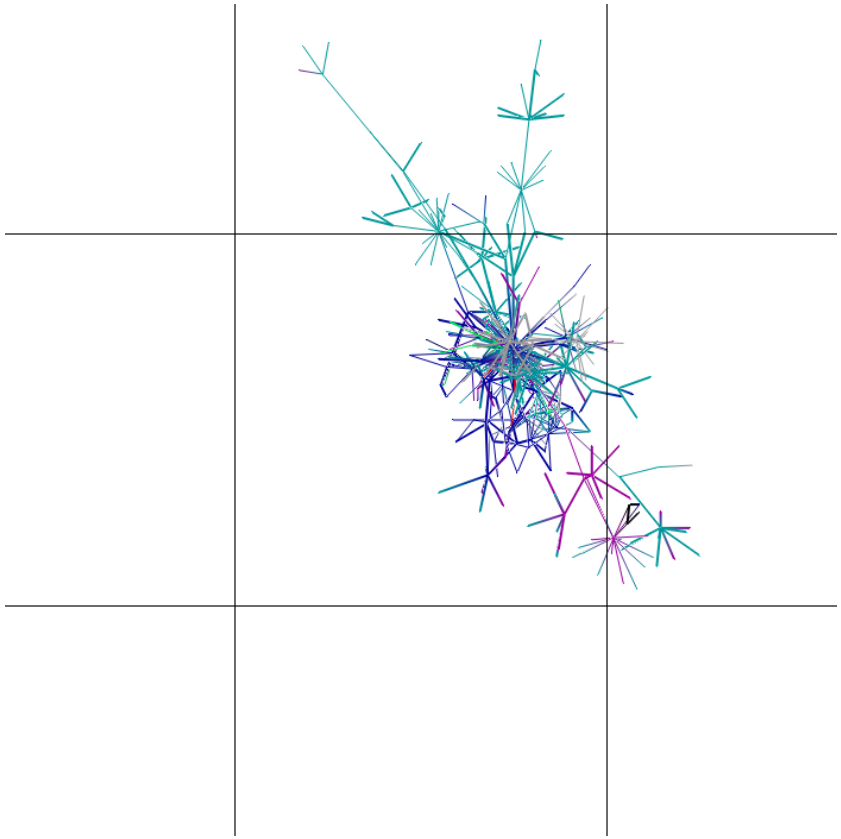


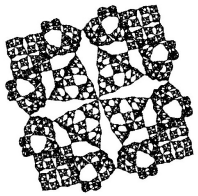
SandMark Inheritance Graph



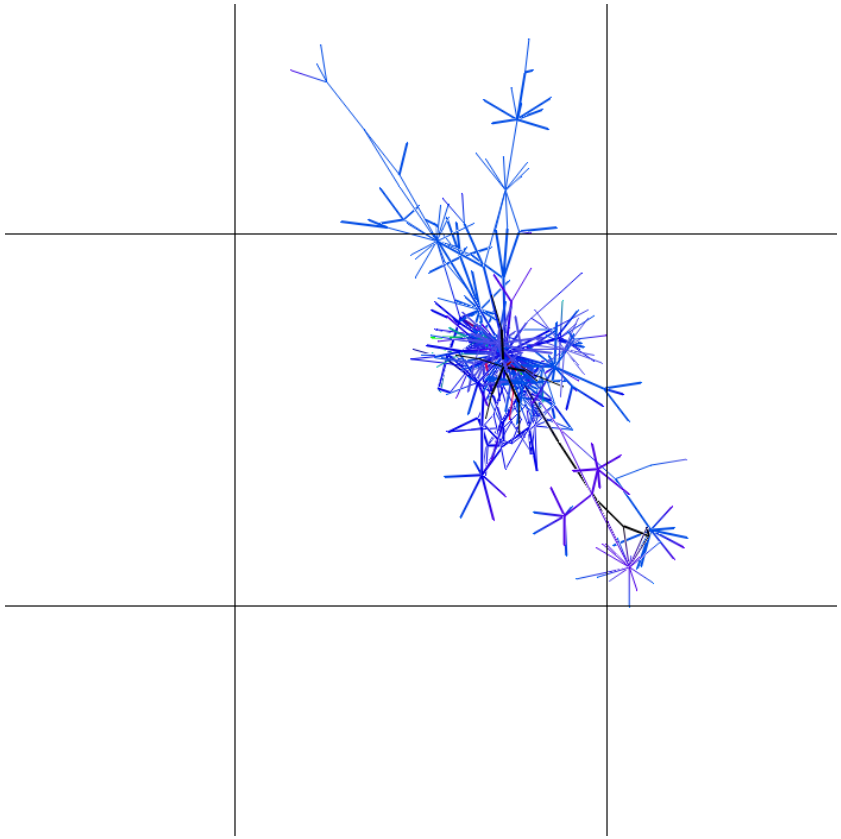


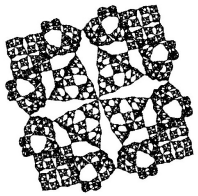
SandMark Inheritance Graph



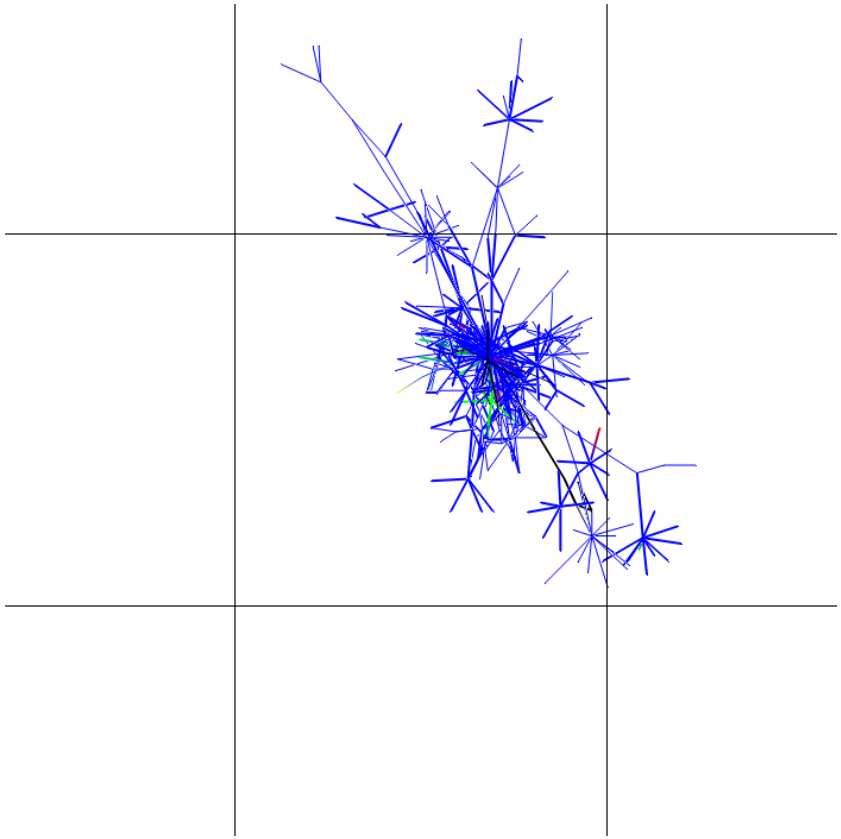


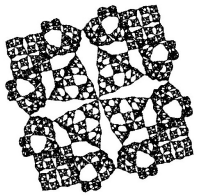
SandMark Inheritance Graph



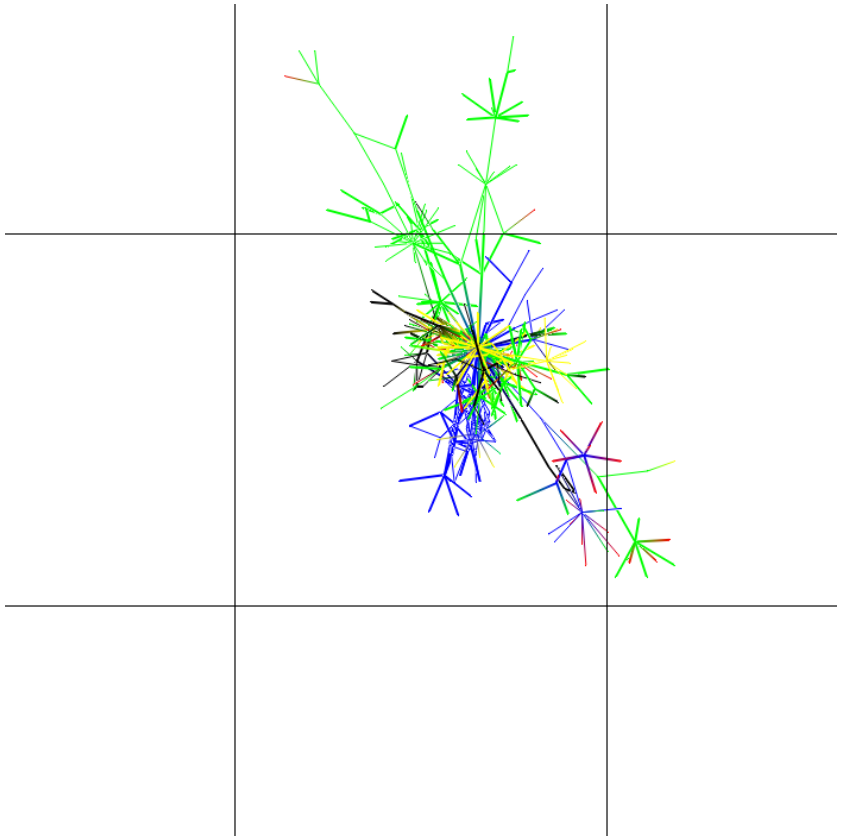


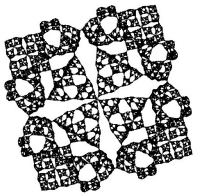
SandMark Inheritance Graph



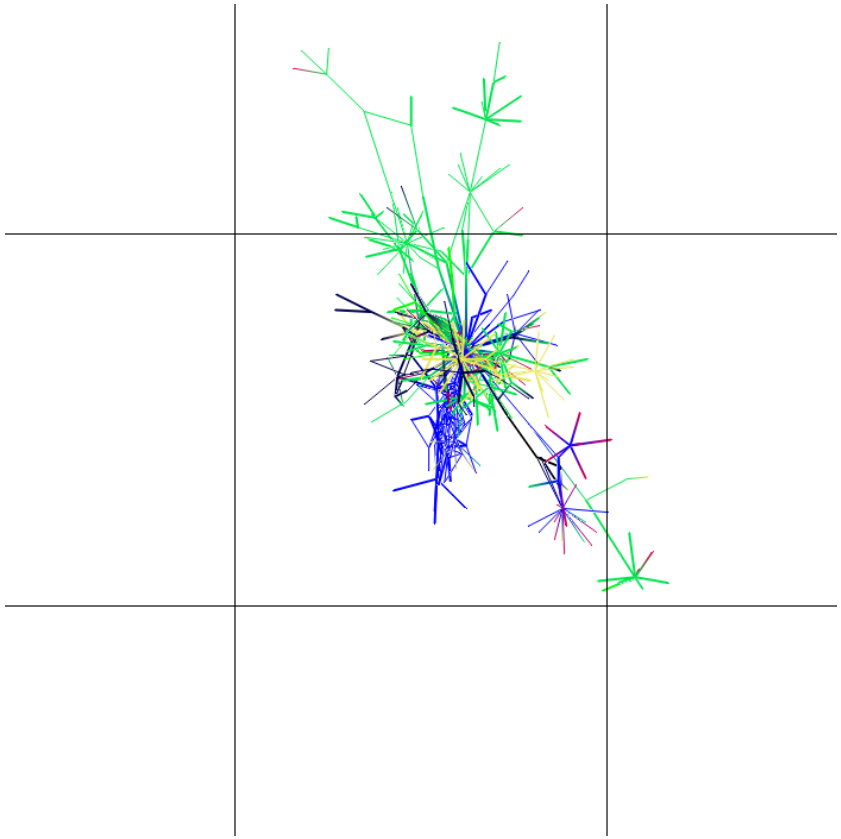


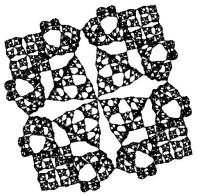
SandMark Inheritance Graph



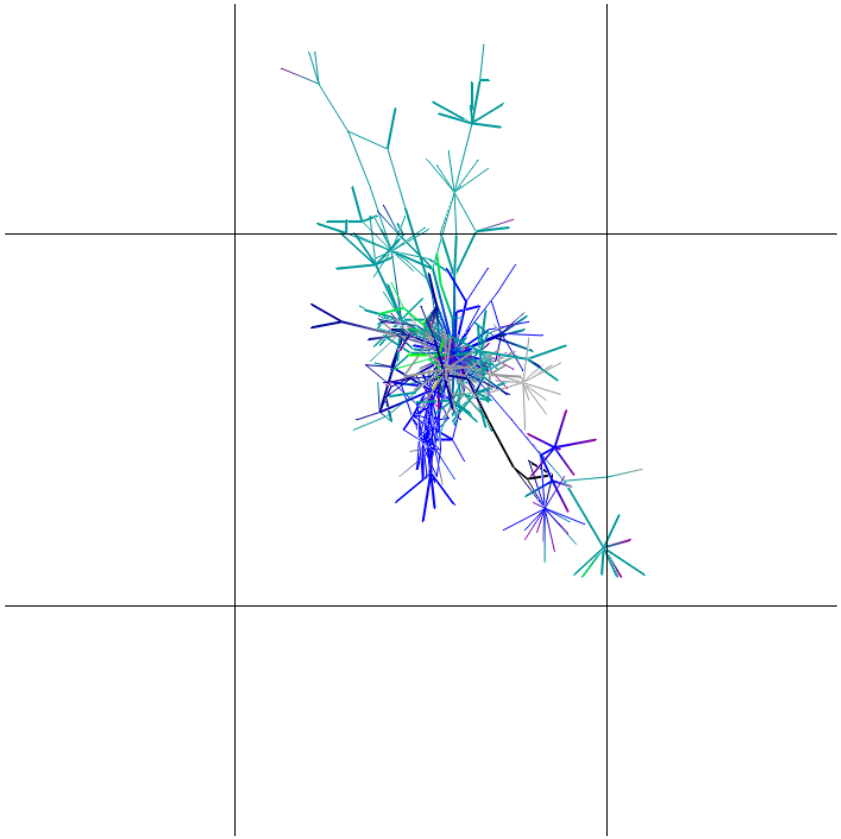


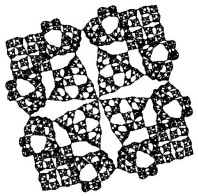
SandMark Inheritance Graph



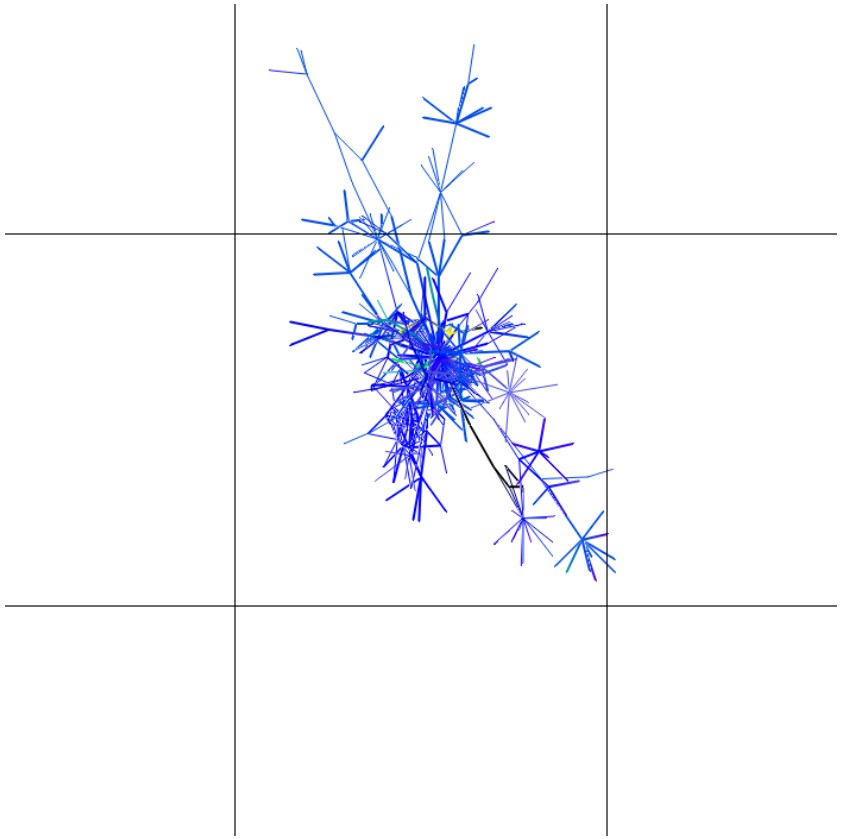


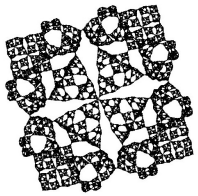
SandMark Inheritance Graph





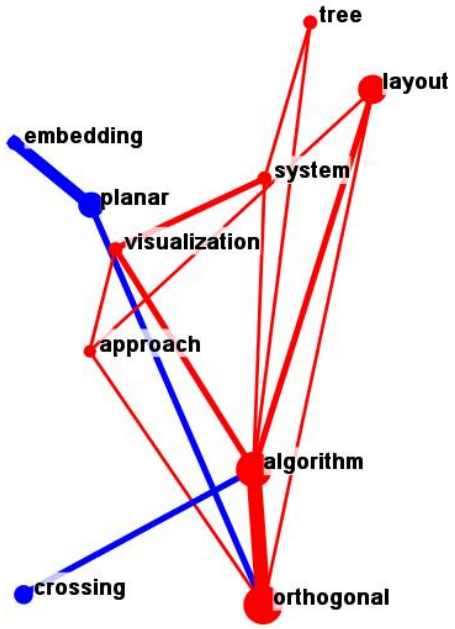
SandMark Inheritance Graph

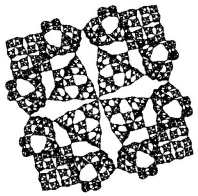




Graph Visualization

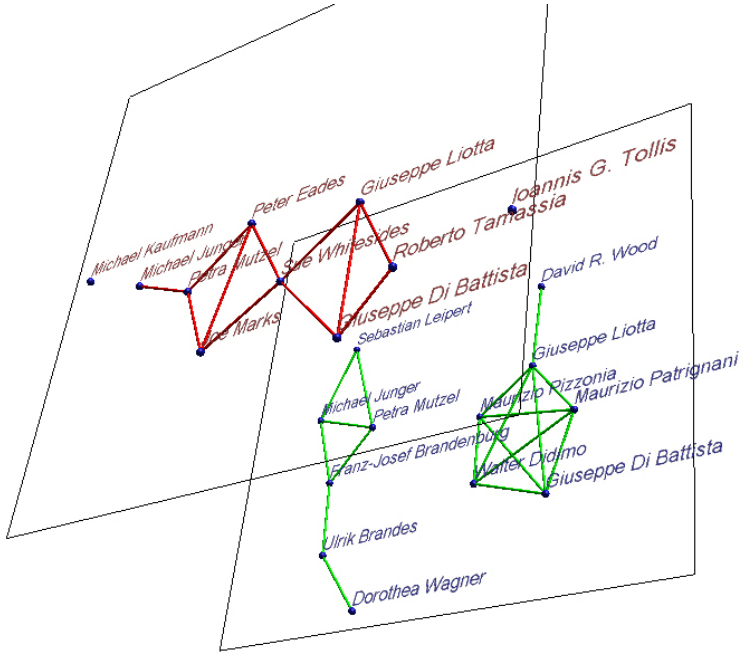
- In theory
  - static unlabeled graph
  - vertices (objects)
  - edges (relationships)
  - goal is *nice* drawing
- In practice
  - dynamic or evolving
  - weights on vertices and edges
  - uncertainty
  - meaningful placement
  - spatial locality
  - temporal locality

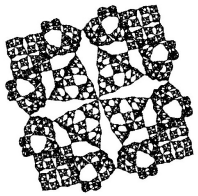




Visualization Wish-list

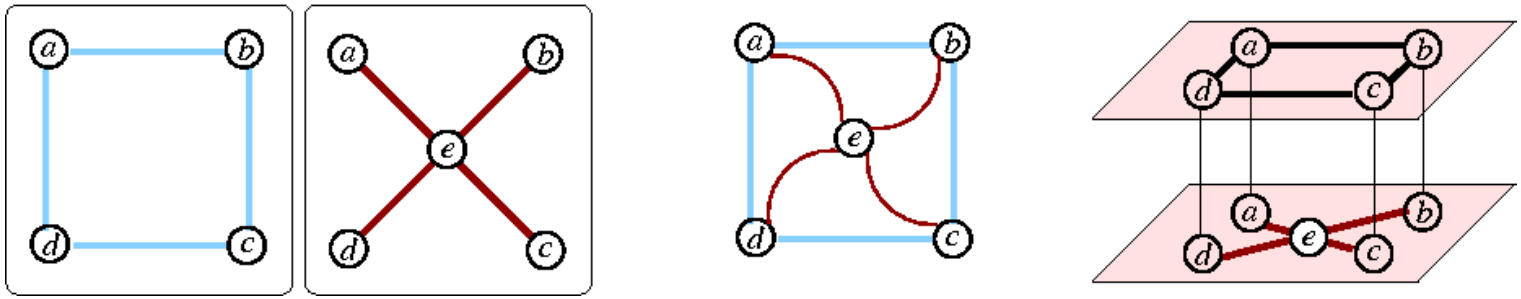
- Each drawing should be readable
  - individually "nice" layouts
  - no crossings, symmetries, etc.
- Mental map preservation
  - structures, relative locations
  - animation, morphing
- In theory ... but in practice...
  - conflicting goals
  - domain knowledge
  - issues of scale

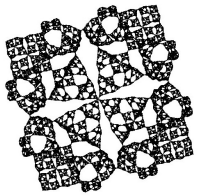




Visualizing Evolving Graphs

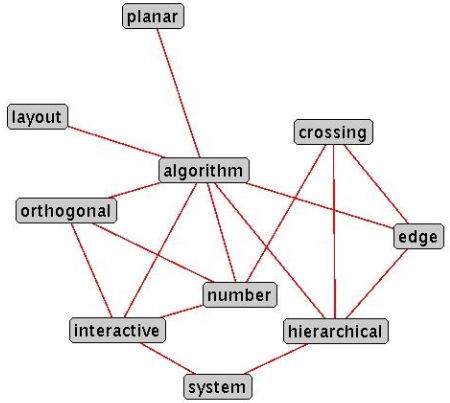
- Time model
  - a *timeslice* for each unit of time
  - a snapshot of the graph
- Viewing the evolving graph
  - different views
  - morphing between timeslices
- Beautiful theoretical problems

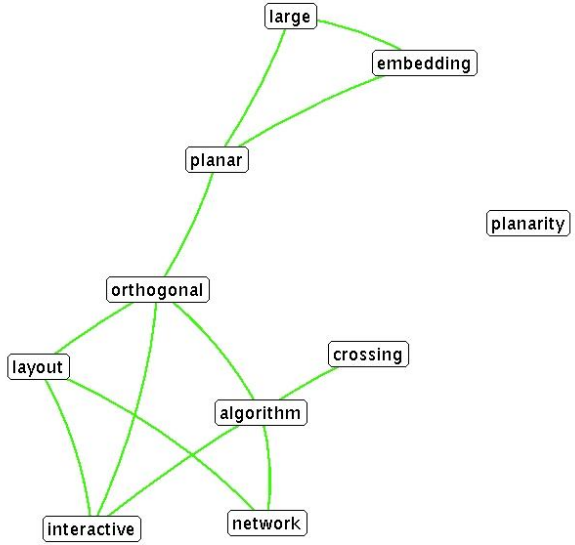


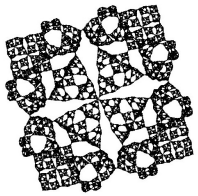


Readability and Mental Map

- Individual layouts for each graph
  - maximizes individual readability
  - no mental map preservation

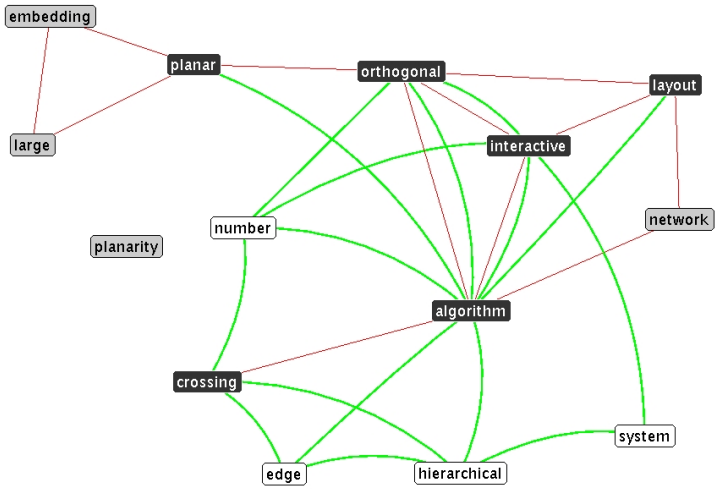


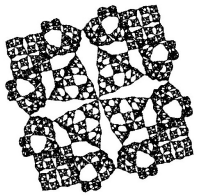




Readability and Mental Map

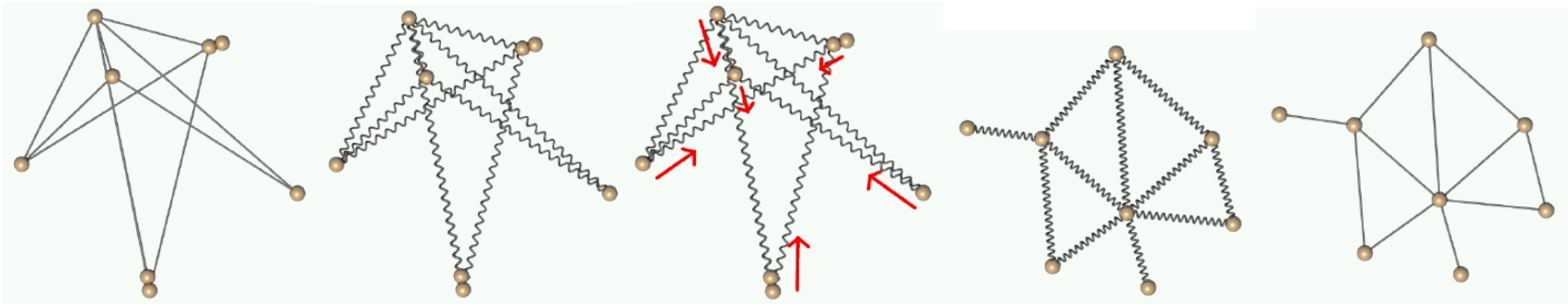
- Individual layouts for each graph
  - maximizes individual readability
  - no mental map preservation
- Simultaneous embedding
  - no individual readability
  - maximizes mental map preservation

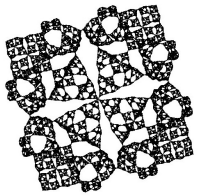




Force-Directed Methods

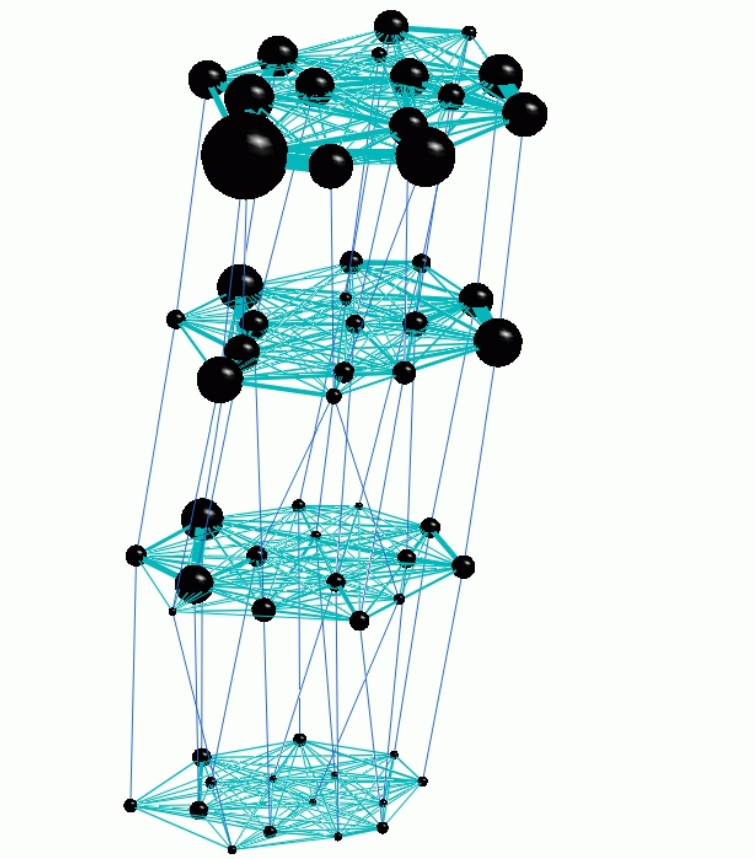
- Assign energy function to the current layout
  - based on graph distances [KK]
  - based on attractive/repulsive forces [FR]
- Energy model
  - characterizes stability
  - iterative improvement
  - minimal energy $\Rightarrow$ good layout





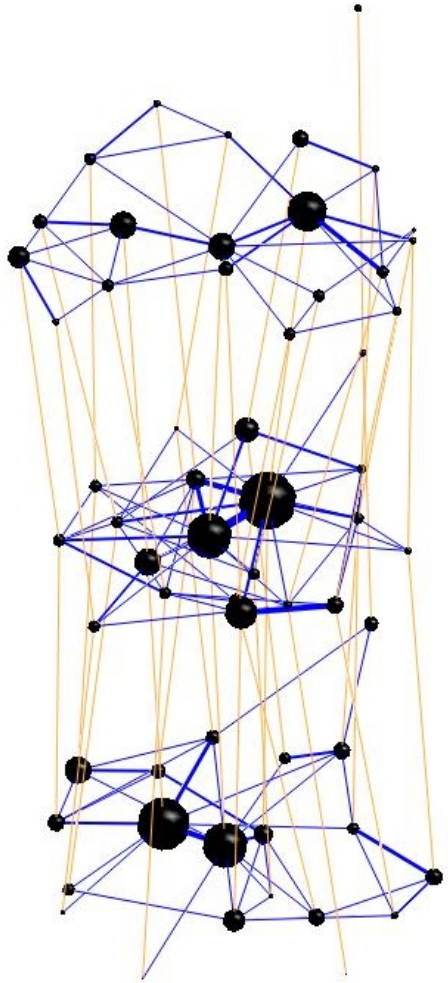
Making It Work

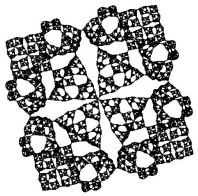
- Force-directed methods
  - modification of Kamada-Kawai
  - modification of Fruchterman-Reingold
- Large graphs become even larger
  - multi-scale methods
  - non-random initial placement
  - high-dimensional embedding
- Readability and mental map
  - enforced via inter-timeslice edges
  - using vertex-weights and edge-weights



Merged Graph

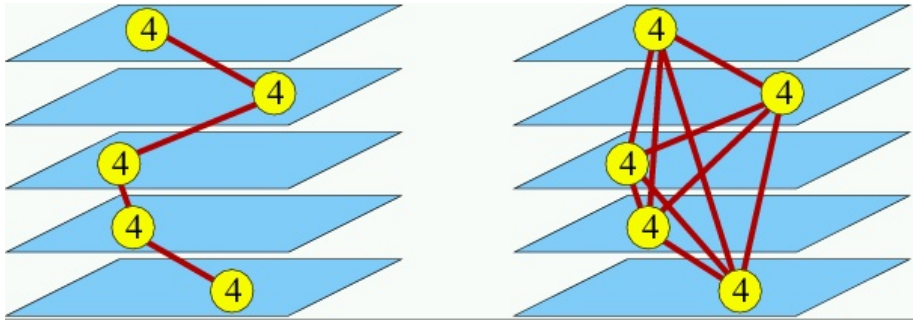
- $G_1, G_2, \ldots, G_k \Rightarrow$ merged graph $\mathcal{G} = (V, E)$
  - $V = V_1 \cup V_2 \cup \ldots \cup V_k$
  - $E = E_1 \cup E_2 \cup \ldots \cup E_k \cup E^*$
  - E^* contains the inter-timeslice edges
- Weights: $w(v)$ and $w(e)$
  - sum of weights in each timeslice
  - or cumulative (collaboration graph)
  - $w(e) = \beta(w(u) + w(v)), e = (u, v) \in E^*$
  - $0 \leq \beta < \infty$: *balance control*



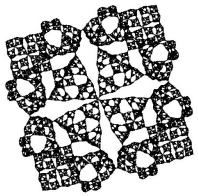


Modified Layout Algorithm

- Inter-timeslice options
  - weight of inter-timeslice edges in E^*
  - connectivity of inter-timeslice edges

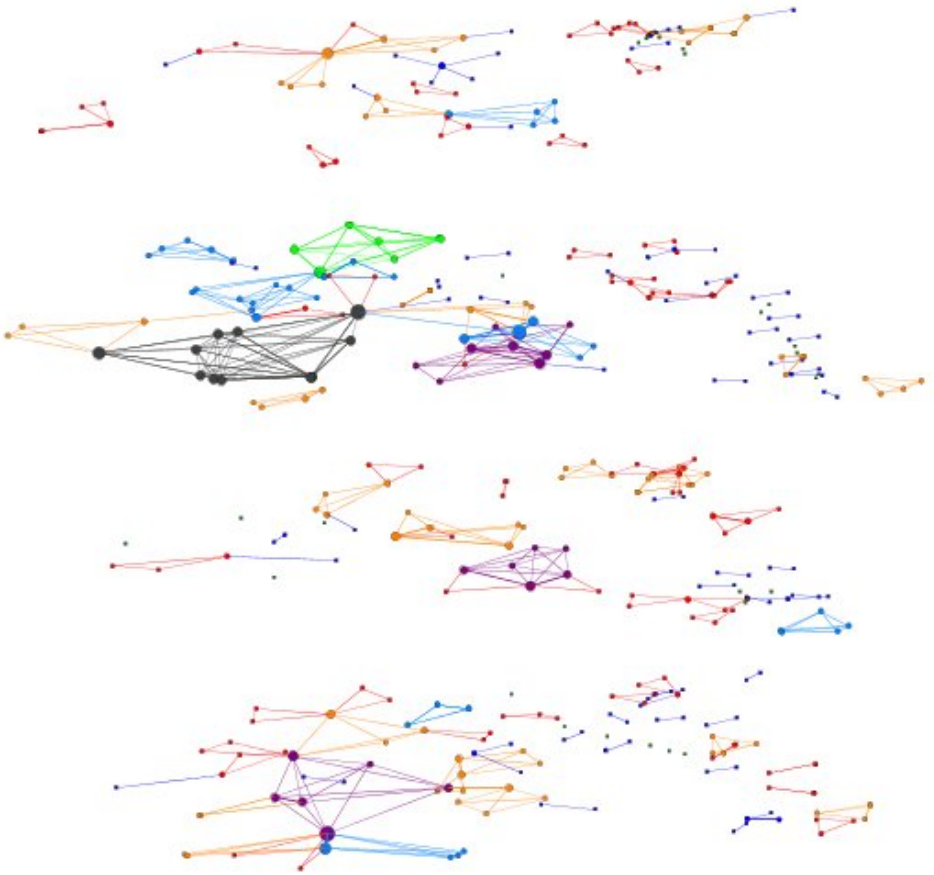


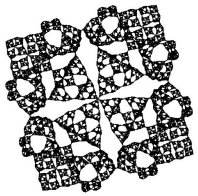
- Role of inter-timeslice edges
  - $w(e) = \beta(w(u) + w(v)), e = (u, v) \in E^*$
  - $\beta \rightarrow 0 \Rightarrow$ good individual layouts
  - $\beta \rightarrow \infty \Rightarrow$ good mental map preservation
- Need to deal with weighted graphs



Modified Layout Algorithm

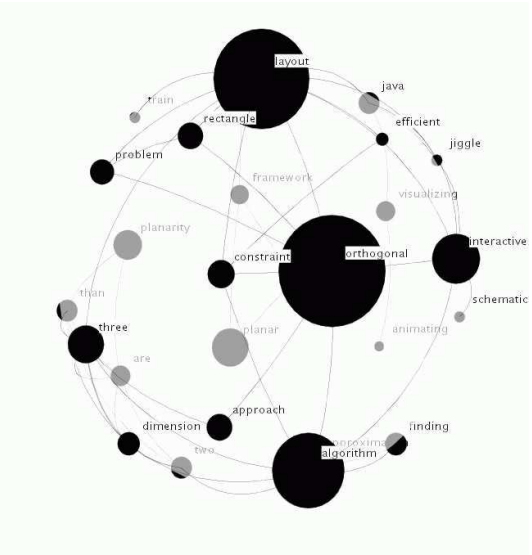
- Heavy vertices: persistent
  - further apart
  - closer to center
  - move little
- Light vertices: transient
  - (dis)appear on the periphery
  - move more
- Heavy edges: persistent
  - shorter in length
- Light edges: transient
  - less important

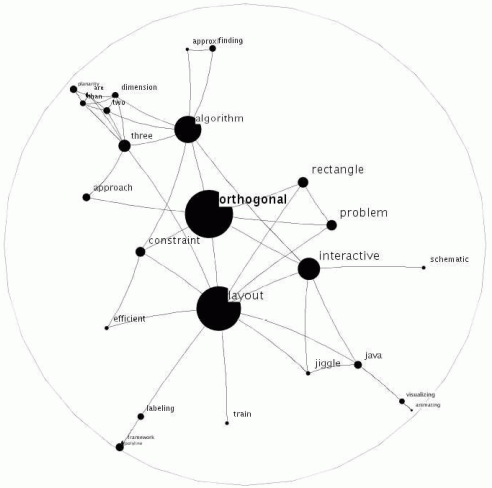


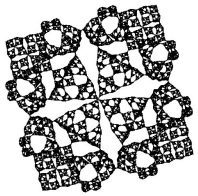


The graphael System

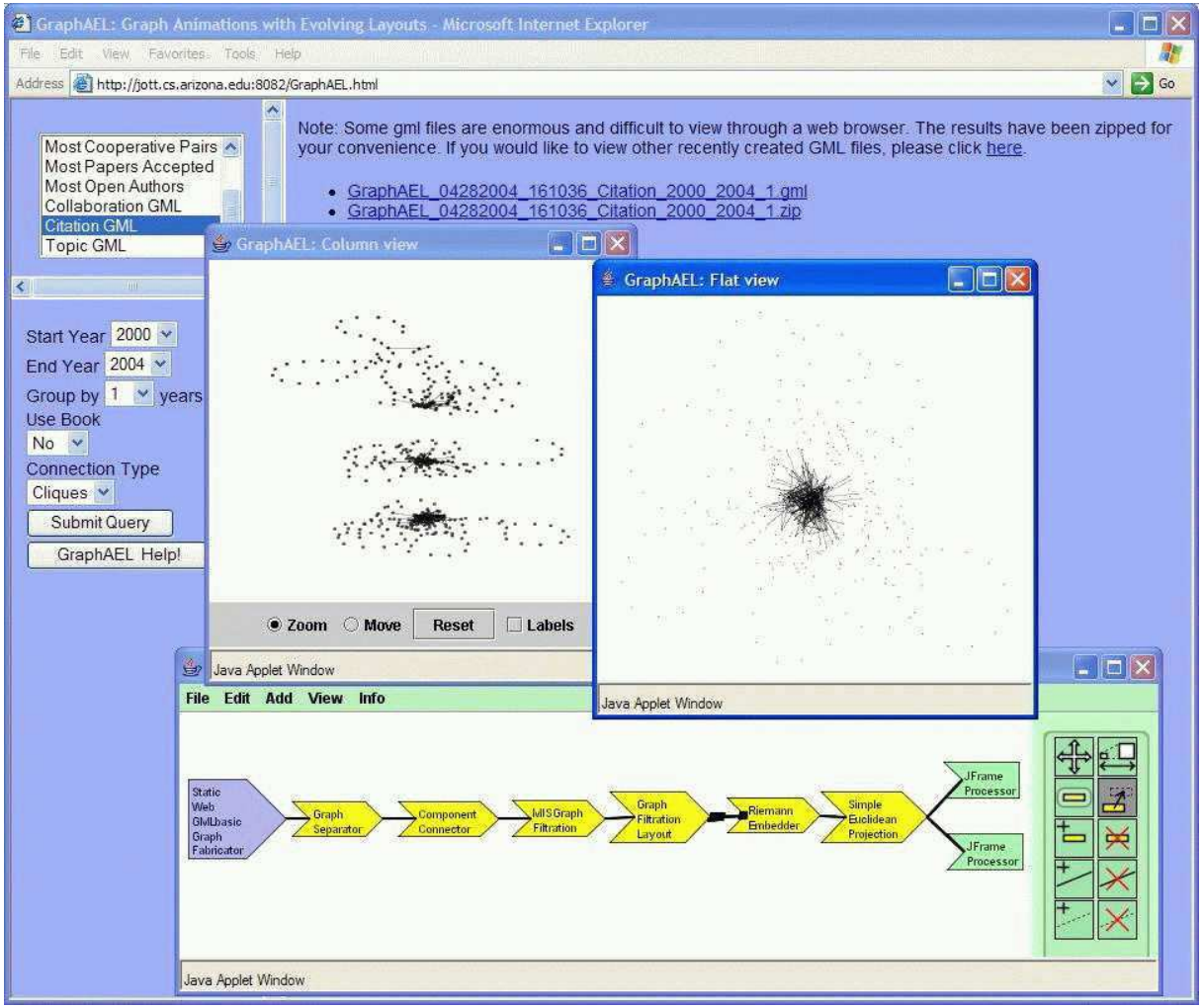
- Yet another graph drawing system
- Features
  - static, evolving graphs, morphing
  - hyperbolic, spherical embedding
  - as a database plugin: ACM, InfoVis, GD
  - user-control over CFG
- Availability
  - `http://graphael.cs.arizona.edu`
  - demo

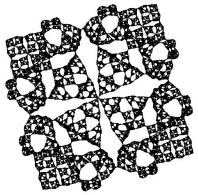






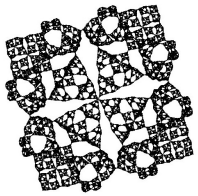
The graphael System





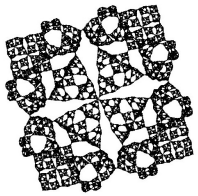
Auralization

- Sonification examples
  - Geiger counter
  - stick-shift cars
  - spinning hard disks
- Human hearing
  - sensitive to (a)periodic sounds
  - detects small changes in frequency
- Suitable for program analysis?
  - rapidly changing data can be missed visually
  - good sound cues: repeated patterns
  - augment visual cues: doesn't require user focus
- How does a good program sound?



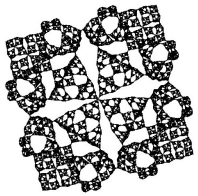
JMusic

- Auralization for Java programs:
  - anotation script
  - auralization script
  - jar file
- Anotation
  - number of method calls
  - number of array manipulations
  - track loops
- Auralization
  - change tempo, key, pitch, instrument
  - change melody, flip major/minor key



Future Work

- Visual
  - which graphs? (inheritance, pdg's,...)
  - where? (locality)
  - what? (changes)
- Audio
  - when? (development, debugging, tuning,...)
  - how? (mapping between events and sounds)
- Cockpit
  - computer game metaphor
  - charts, dials, controls



Acknowledgments

- Christian Collberg
- Saumya Debray
- John Hartman
- Students
  - Cesim Erten (PhD)
  - Justin Cappos (PhD)
  - Kevin Wampler (MS)
  - Kelly Hefner (BS)
  - Ed Carter (BS)
  - Gary Yee (BS)
  - Phil Harding (BS)
  - Armand Navabi (BS)
  - David Forrester (BS)