Code Explorer Mission

Vision for evolving the Code Explorer from a playground into a compelling showcase that demonstrates real-world application potential for the PureScript D3 library.

Goals

- **Showcase, not playground** - Curated scenes that tell a story, while retaining key controls (force toggles) that illuminate what's happening
- **4-7 scenes** with smooth, meaningful transitions
- **Genuinely informative** - Views that would be useful for understanding a real codebase
- **Point to potential** - Demonstrate that this library can build serious applications

---

Scene Concepts

1. Galaxy Overview (Force layout)

All modules floating freely, colored by package/layer, sized by LOC. The "here's everything" starting point.

2. Dependency Flow (Layered DAG)

Same nodes arranged to show dependency direction - sources at top, sinks at bottom. Reveals the architecture's shape.

3. Package Bubbles (Force + circle packing)

Modules gather into their packages as packed circles. Shows organizational structure and relative package sizes. Great transition from flow - nodes coalesce into clusters.

4. Expansion Tree (Radial with slider)

Starting from Main, progressively reveal:

- Direct dependencies
- Transitive dependencies
- Package dependencies
- Transitive package dependencies
- node_modules (the JS iceberg)
- What esbuild bundles
- What PureScript compiler prunes

A slider controls depth - slide right to reveal more of the dependency iceberg. Could also work in reverse: "if I change this, what's affected?"

5. Activity Overlay

Git activity as a mode/overlay on any layout - recent commits, churn, age. Colored/pulsing nodes. Toggle rather than separate scene.

6. Neighborhood Focus

Click a module → everything else fades, its dependencies and dependents highlight and pull closer. Interactive exploration mode.

7. Complexity Landscape (Scatter)

x = fan-in (dependents), y = fan-out (dependencies), size = LOC. Reveals architectural hotspots. **Novel aspect**: nodes remain interactive - hover for deps, click to focus. Bridges analytical and exploratory modes.

Key Visualization Ideas

Link Strength

Represent connection weight visually (as in LesMis). Could encode:

- Call frequency (static analysis)
- Import count
- Architectural coupling (do they change together in git?)

AI/LLM Code Tracking

Scatter plot showing AI vs human contribution. Addresses the real concern about reviewing AI code less carefully due to the "firehose" effect.

Possible metrics:

- % of file touched by AI commits
- Ratio of AI additions to human modifications
- Churn rate of AI vs human code
- Test coverage of AI-written vs human code

Test Coverage Shadow

Isometric paired trees - code tree on one side, test tree mirrored. Where the shadow is thin, coverage is weak. Immediately shows structural gaps.

Git Blame Ribbon

For any selected file, show a colored ribbon representing authorship over the file's length.

Transition Flow

Suggested demo sequence for maximum impact:

```
Galaxy → Dependency DAG → Expansion Slider → AI Contribution Scatter
```

Story: "Here's your code → how it connects → the full dependency depth → where AI helped"

Alternative narrative flow:

```
Galaxy → Dependency Flow → Package Bubbles → Radial Tree → Galaxy
```

What Makes Transitions Work

- **Same nodes throughout** - smooth morphs, no teleporting
- **Each reveals different insight** - structure, flow, organization, activity
- **Natural narrative** - progressive revelation

Architecture Evolution

Repository Split

The repo should be split into distinct concerns:

```
purescript-d3/           # Library only
  src/
  examples/              # Simple, focused examples

code-explorer/           # Separate app
  frontend/              # Halogen UI
  backend/               # Data server
  scripts/               # Git analysis, static analysis tools
```

Data Layer

GraphQL feels right for the data server. Schema could define:

- Modules (with metrics, git data)
- Dependencies (with strength/type)
- Packages
- Git history

This would let the explorer work on *any* codebase with appropriate analyzers, not just PureScript.

Current Technical Foundation

Completed

- Path-based links (can morph between diagonal/bezier)
- Depth stretching for trees (2x multiplier)
- Smooth link transitions during scene changes
- Layer scale presets (linear, sqrt, log, exponential)
- Separation function integration in Tree4

Next Steps (Transitions)

- Bezier link morphing during tree transitions
- Link style matching tree orientation (BezierVertical, BezierHorizontal, BezierRadial)

Open Questions

- How to handle the node_modules scale? Thousands of packages - need aggregation/sampling
- Static analysis tooling for call frequency / link strength
- Git data extraction pipeline (currently have some, needs expansion)
- How deep to go on AI attribution? Commit message parsing vs more sophisticated analysis