Nodges 2.0: Projektbericht & Entwicklungsstatus

Datum: 25. November 2025 **Projekt:** Nodges (Network 3D Graph Visualization System) **Version:** 2.0.0 (Migration von 0.92.18)

1. Executive Summary

Nodges ist eine hochperformante, webbasierte Anwendung zur 3D-Visualisierung komplexer Netzwerkgraphen. Das Projekt hat kürzlich eine signifikante Transformation ("Phase 2: Core Migration") durchlaufen, bei der eine monolithische JavaScript-Codebasis in eine modulare, typensichere TypeScript-Architektur überführt wurde. Ziel war es, die Wartbarkeit zu erhöhen, die Rendering-Performance für große Datensätze zu verbessern und eine solide Basis für zukünftige Features zu schaffen.

Dieser Bericht dient als Grundlage für eine Audio-Diskussion über die technischen Herausforderungen und Erfolge dieser Transformation.

2. Technische Architektur

Core Stack

- **Rendering Engine:** Three.js (WebGL)
- **Sprache:** TypeScript (migriert von ES6 JavaScript)
- **Build Tool:** Vite
- **State Management:** Zentralisierter `StateManager` (Observer Pattern)

Architektur-Design

Die Anwendung folgt nun einer strikten Trennung der Verantwortlichkeiten (Separation of Concerns):

1. **App.ts**: Der Einstiegspunkt, der die Initialisierung der 3D-Szene und der Manager koordiniert.
2. **Manager-Layer**: Spezialisierte Module für verschiedene Aspekte:
    - `LayoutManager`: Berechnet Node-Positionen (Force-Directed, Circular, etc.), teilweise ausgelagert in Web Workers für Performance.
    - `InteractionManager`: Handhabt User-Input (Maus, Tastatur) und Raycasting.
    - `Node/EdgeObjectsManager`: Verwaltet die Three.js-Geometrien.
    - `UIManager`: Steuert die HTML-Overlays und Panels.
3. **Daten-Layer**: Typisierte Interfaces (`NodeData`, `EdgeData`) ersetzen lose Objekte.

3. Die "Core Migration" (Herausforderungen & Lösungen)

Vom Chaos zur Struktur

Ausgangslage: Eine `main.js` mit über 2000 Zeilen Code, globalen Variablen und vermischter Logik für UI, Rendering und Daten. **Lösung:** Zerlegung in Klassen. Der `InteractionManager` isoliert nun komplexe Logik wie "Drag & Drop" oder "Hover-Effekte" vom Rest der App. Dies macht den Code testbar und verständlich.

Performance-Optimierung (The "Rendering Bottleneck")

Ein Hauptfokus lag auf der Darstellung tausender Knoten und Kanten.

- **Problem:** Das Erstellen eines eigenen Meshs für jeden Knoten (z.B. `THREE.SphereGeometry`) bringt den Browser bei >1000 Knoten zum Absturz (zu viele Draw Calls).
- **Lösung:** Implementierung von `THREE.InstancedMesh`.
  - *Ergebnis:* Alle Knoten werden in einem einzigen Draw Call gerendert. Die GPU übernimmt die Positionierung.
  - *Status:* Erfolgreich implementiert für Knoten.

Edge Rendering

- **Optimierung:** Kanten nutzen nun optimierte Geometrien. Volumetrische Darstellung wurde für bessere Sichtbarkeit und Performance untersucht.

4. Aktueller Status (Verifikation)

Das Projekt befindet sich in **Phase 5: Verifikation**.

- **Build:** Der TypeScript-Compiler (`tsc`) läuft fehlerfrei. Strenge Typenprüfung hat diverse "Unused Variables" und potenzielle Bugs aufgedeckt und eliminiert.
- **Runtime:** Die Anwendung startet erfolgreich im Browser (`npm run dev`). Die UI-Interaktion (Panels öffnen/schließen) wurde verifiziert.
- **Visuelle Prüfung:** Steht als nächster Schritt an, um sicherzustellen, dass die neuen Shader und Geometrien korrekt aussehen.

5. Ausblick & Nächste Schritte

Obwohl die Migration technisch abgeschlossen ist, gibt es klare Ziele für die Zukunft:

1. **Entfernung von Global State:** Es existieren noch Reste von globalen Zugriffen, die vollständig in den `StateManager` überführt werden müssen.
2. **Erweiterte Layouts:** Integration komplexerer Algorithmen für spezifische Datensätze (z.B. hierarchische Cluster).
3. **Testing:** Einführung von automatisierten E2E-Tests (z.B. mit Cypress oder Playwright), um Regressionen bei zukünftigen Updates zu verhindern.

6. Fazit für die Diskussion

Für die Audio-Generierung sind folgende Punkte besonders spannend:

- Der **Kontrast** zwischen der alten "Hacker"-Codebasis und der neuen "Enterprise"-Architektur.
- Die **technische Tiefe** der 3D-Optimierungen (Instancing vs. einzelne Meshes).
- Die **Bedeutung von TypeScript** für die Stabilität großer Frontend-Projekte.