Projektbericht: Nodges 2.0

1. Projektübersicht

Nodges 2.0 ist eine webbasierte 3D-Netzwerkvisualisierungsanwendung, die auf **Three.js** basiert. Sie ermöglicht die interaktive Darstellung, Analyse und Manipulation von komplexen Graphen (Knoten und Kanten). Das Projekt nutzt moderne Webtechnologien wie **TypeScript**, **Vite** und **CSS** für das Styling.

2. Softwarearchitektur

Die Architektur von Nodges 2.0 folgt einem modularen Ansatz, bei dem die Verantwortlichkeiten klar getrennt sind. `App.ts` fungiert als zentraler Einstiegspunkt und Koordinator.

Kernkomponenten (Core)

- **App.ts**: Die Hauptklasse, die die Three.js-Szene, Kamera und Renderer initialisiert. Sie instanziiert alle Manager und steuert den Haupt-Render-Loop (`animate`).
- **StateManager.ts**: Implementiert ein zentrales State-Management basierend auf dem **Observer-Pattern**. Es hält den globalen Anwendungszustand (z.B. selektierte Knoten, Hover-Status, UI-Sichtbarkeit) und benachrichtigt abonnierte Komponenten bei Änderungen.
- **UIManager.ts**: Verwaltet die HTML-Benutzeroberfläche (Panels, Buttons). Es reagiert auf State-Änderungen, um UI-Elemente zu aktualisieren (z.B. Info-Panel anzeigen).
- **CentralEventManager.ts**: (Erwähnt in Imports) Wahrscheinlich für die Koordination von Events zuständig.
- **InteractionManager.ts**: Handhabt Benutzerinteraktionen wie Klicks und Mausbewegungen.
- **LayoutManager.ts**: Zuständig für die Berechnung von Knotenpositionen (z.B. Force-Directed Layout).

Daten & Objekte

- **DataParser.ts**: Ein flexibler Parser, der sowohl das "Legacy"-Format (direkte Knoten/Kanten-Listen) als auch ein "Future"-Format (Entities/Relationships) unterstützt und normalisiert.
- **NodeObjectsManager / EdgeObjectsManager**: Verwalten die 3D-Repräsentation der Daten. Sie kümmern sich um die Erstellung, Aktualisierung und das Löschen von Three.js-Meshes (z.B. `InstancedMesh` für Performance).
- **VisualMappingEngine.ts**: Mappt Datenattribute auf visuelle Eigenschaften (Farbe, Größe).

Effekte

- **HighlightManager.js**: Steuert alle Hervorhebungen (Hover, Selektion, Suche).
- **GlowEffect.js**: Implementiert den visuellen "Glow"-Effekt durch Manipulation von Material-Eigenschaften (Emissive).

---

3. Detaillierte Abläufe

3.1 Laden von JSON-Daten

Der Prozess des Datenladens ist robust gestaltet und unterstützt verschiedene Formate.

(Siehe `architecture_diagrams.md` für das Sequenzdiagramm)

3.2 State Management Fluss

Der `StateManager` agiert als Single Source of Truth.

(Siehe `architecture_diagrams.md` für das Flussdiagramm)

---

4. Deep Dive: Highlighting & Visual Effects

Das Highlighting-System ist hochentwickelt und zentralisiert im `HighlightManager`.

Funktionsweise des HighlightManagers

1. **Registry**: Der Manager führt eine `highlightRegistry` (Map), die speichert, welches Objekt warum gehighlighted ist (Hover, Selection, Search, Path, Group).
2. **Material Backup**: Bevor ein Highlight angewendet wird, erstellt der Manager ein Backup des originalen Materials (`backupMaterial`). Dies ist kritisch, um den Ursprungszustand exakt wiederherzustellen.
    - *Besonderheit*: Wenn ein Material von mehreren Objekten geteilt wird (Instancing), wird es geklont, um Seiteneffekte zu vermeiden.
3. **Priorisierung**: Es gibt eine implizite Priorisierung. Selektion überschreibt Hover.
4. **Cleanup**: Die Methode `cleanupUnusedHighlights` entfernt Highlights, die im aktuellen State nicht mehr gültig sind.

GlowEffect & Visuelle Umsetzung

- **Knoten (Nodes)**:
    - Nutzt die `emissive` Eigenschaft des Three.js Materials.
    - Hover: Erhöht Helligkeit (HSL L-Value) + leichter Cyan Glow.
    - Selektion: Grüner Glow (`emissive: 0x00ff00`).
- **Kanten (Edges)**:
    - Da Kanten oft `InstancedMesh` oder einfache Linien sind, ist die Manipulation komplexer.
    - **Outline**: Für Hover wird eine `TubeGeometry` als Umriss um die Kante generiert (`addEdgeOutline`). Dies erzeugt einen volumetrischen Highlight-Effekt.
    - Farbe: Die Farbe wird temporär geändert (z.B. zu Hellblau oder Grün).

Wann wird was gehighlighted?

- **Hover**: Wenn der Mauszeiger über einem Objekt schwebt (gesteuert durch Raycaster -> InteractionManager -> StateManager).
- **Selektion**: Beim Klick auf ein Objekt.
- **Suche**: Wenn ein Knoten über die Suchleiste gefunden wird (Gelb).
- **Pfad/Gruppe**: Bei Analyse-Funktionen (Cyan/Magenta).

---

5. Deep Dive: State Manager

Der `StateManager` ist das Herzstück der Reaktivität.

- **Struktur**: Er hält ein einfaches JavaScript-Objekt (`State`), das alle relevanten Daten enthält:
  - `hoveredObject`, `selectedObject` (Interaktion)
  - `highlightedObjects` (Set von Objekten)
  - `infoPanelVisible`, `tooltipContent` (UI)
- **Abonnement (Subscription)**:
  - Komponenten können sich mit `subscribe(callback, category)` anmelden.
  - Kategorien ('ui', 'highlight', 'default') helfen beim Debugging und der Organisation.
- **Updates**:
  - `update(partialState)`: Nimmt ein Teil-Objekt, merged es mit dem aktuellen State und prüft auf Änderungen (Shallow Comparison).
  - Nur bei tatsächlichen Änderungen werden die Subscriber benachrichtigt.
- **Batch Updates**: `batchUpdate` ermöglicht mehrere Änderungen auf einmal, um unnötige Re-Renders zu vermeiden.

Diese Architektur ermöglicht eine sehr lose Kopplung. Der `UIManager` muss nicht wissen, *wie* ein Objekt selektiert wurde, nur *dass* es selektiert ist. Ebenso muss der `InteractionManager` nicht wissen, wie das UI aktualisiert wird.