04 3D-Rendering und Szenen-Management

Die grafische Darstellung ist das Kernstück von Nodges. Dieses Kapitel behandelt die Implementierung der 3D-Engine mittels Three.js und die verwendeten Strategien zur effizienten Darstellung grosser Graphen.

04.1 Three.js Integration

Nodges nutzt **Three.js** als WebGL-Abstraktionslayer. Der Renderer wird in der `App`-Klasse initialisiert und verwaltet.

Szenen-Setup (`App.initThreeJS`)

- **Scene**: Der Container für alle 3D-Objekte.
- **Camera**: Eine `PerspectiveCamera` simuliert das menschliche Sehfeld (FOV 75°). Sie wird so positioniert, dass der Graph initial vollständig sichtbar ist.
- **Renderer**: Der `WebGLRenderer` zeichnet die Szene in ein HTML5-Canvas. Anti-Aliasing ist aktiviert, um glatte Kanten zu gewährleisten.
- **Controls**: Die `OrbitControls` ermöglichen dem Benutzer, die Kamera um den Mittelpunkt des Graphen zu drehen, zu zoomen und zu verschieben.

Beleuchtungskonzept

Um Tiefe und Dreidimensionalität zu erzeugen, wird ein kombiniertes Beleuchtungssetup verwendet:

1. **AmbientLight**: Sorgt für eine Grundhelligkeit, damit Schattenbereiche nicht komplett schwarz sind.
2. **DirectionalLight**: Simuliert Sonnenlicht und erzeugt Schatten und Glanzlichter (Specular Highlights) auf den Knotenoberflächen, was deren Form (z.B. Kugelrundung) betont.
3. **PointLights**: Werden dynamisch eingesetzt, z.B. bei Glow-Effekten (optional).

Post-Processing

Die Architektur ist für Post-Processing vorbereitet (z.B. Bloom-Effekte für stärkeres Leuchten), aktuell werden visuelle Effekte jedoch primär über Material-Eigenschaften (`emissive`) gelöst, um die Performance hoch zu halten.

04.2 Knoten-Visualisierung (`NodeObjectsManager`)

Der `NodeObjectsManager` ist für die Erstellung und Verwaltung der Knoten (Nodes) verantwortlich. Da ein Graph tausende Knoten enthalten kann, ist Performance hier der kritische Faktor.

InstancedMesh für maximale Performance

Anstatt für jeden Knoten ein eigenes `THREE.Mesh`-Objekt zu erstellen (was tausende Draw-Calls verursachen würde), nutzt Nodges **InstancedMesh**.

- **Prinzip**: Eine Geometrie (z.B. eine Kugel) und ein Material werden nur *einmal* an die GPU gesendet.
- **Instancing**: Die Position, Skalierung und Rotation für tausende Kopien werden in einem einzelnen Buffer übergeben.

- **Vorteil**: Die Grafikkarte kann 10.000+ Knoten in einem einzigen Draw-Call zeichnen, was die Framerate auch bei grossen Netzwerken stabil bei 60 FPS hält.
- **Herausforderung**: Individuelle Eigenschaften (wie Farbe) müssen über Instanz-Attribute oder Texture-Maps verwaltet werden, nicht über einfache Material-Änderungen.

Geometrie-Typen

Nodges unterstützt verschiedene Formen zur Repräsentation von Knoten-Typen:

- **Sphere**: Standard für generische Knoten.
- **Box**: Oft für Infrastruktur-Komponenten (Server, Datenbanken).
- **Icosahedron / Octahedron**: Für spezielle Entitäten.

Der Manager verfügt über einen Cache für Geometrien und Materialien, um Speicher zu sparen.

04.3 Kanten-Visualisierung (`EdgeObjectsManager`)

Die Darstellung von Verbindungen (Edges) ist komplexer als die von Knoten, da sie zwei Punkte im Raum verbinden und unterschiedliche Formen annehmen können.

Rendering-Strategien

Der `EdgeObjectsManager` entscheidet intelligent, welche Geometrie verwendet wird:

1. **InstancedMesh (Cylinders)**:
    - **Einsatz**: Für einfache, gerade Verbindungen zwischen zwei Knoten.
    - **Technik**: Ein Einheits-Zylinder wird so skaliert und rotiert (mittels Quaternions), dass er von Punkt A nach Punkt B reicht. Auch hier wird Instancing für Performance genutzt.
2. **TubeGeometry (Curved Lines)**:
    - **Einsatz**: Wenn *mehrere* Kanten zwischen denselben zwei Knoten existieren (Multi-Edges).
    - **Problem**: Gerade Linien würden sich exakt überlagern und wären nicht unterscheidbar.
    - **Lösung**: Es werden Bezier-Kurven (`QuadraticBezierCurve3`) berechnet. Jede zusätzliche Kante erhält einen stärkeren "Bauch" (Offset), sodass alle Verbindungen wie Kabelstränge nebeneinander sichtbar sind.
    - **Performance**: Da Tubes komplexe Geometrien sind, werden sie individuell erstellt (kein Instancing), was bei sehr vielen Multi-Edges rechenintensiver ist.
3. **LineSegments (Debugging/Legacy)**:
    - Einfache Linien (1px breit) werden primär für Debugging oder als Fallback auf schwacher Hardware genutzt.

Dynamische Updates

Wenn sich Knoten bewegen (z.B. durch Layout-Algorithmen), müssen die Kanten nachgezogen werden. Der Manager optimiert dies:

- **Gerade Kanten**: Nur die Matrix-Transformation der Instanzen wird aktualisiert (sehr schnell).

- **Kurvige Kanten**: Die Geometrie-Punkte müssen neu berechnet werden.

Ende Kapitel 04