

01 Einführung und Projektvision

Überblick

Nodges ist eine moderne 3D-Graphvisualisierungs-Applikation, die es ermöglicht, komplexe Netzwerkstrukturen und Beziehungen in einem interaktiven dreidimensionalen Raum darzustellen und zu erkunden. Die Anwendung wurde mit Fokus auf Benutzerfreundlichkeit, visuelle Qualität und Performance entwickelt.

Technologie-Stack

Technologie	Version	Zweck
Three.js	0.161.0	3D-Rendering und WebGL-Abstraction
TypeScript	5.3.3	Typsichere Entwicklung
Vite	5.1.4	Build-Tool und Development Server
Zod	3.22.4	Runtime-Validierung von Datenstrukturen
Tween.js	18.6.4	Animationen und Interpolation
lil-gui	0.19.1	Debug-Panels und UI-Steuerelemente

Projektziele

Primäre Ziele

1. Intuitive Visualisierung

Komplexe Graphenstrukturen verständlich und navigierbar darstellen.

2. Flexibles Datenformat

Unterstützung sowohl eines Legacy-Formats als auch eines erweiterten "Future-Formats" mit Visual Mappings.

3. Interaktive Exploration

Ermöglicht durch Hover, Selektion, Suche und Pfadfindung tiefgehende Analyse der Daten.

4. Dynamische Visualisierung

Daten-Eigenschaften werden auf visuelle Attribute wie Farbe, Größe, Animation gemappt.

Kernfunktionen

Graph-Darstellung

- **Nodes (Knoten):** Repräsentieren Entitäten im Graph
 - Verschiedene Geometrien je nach Typ (Sphere, Box, Dodecahedron, etc.)
 - Farb- und Größen-Mapping basierend auf Dateneigenschaften

- **Edges (Kanten):** Repräsentieren Beziehungen zwischen Entitäten
 - Gekrümmte Darstellung mittels Quadratic Bezier Curves
 - Unterstützung von Mehrfach-Kanten zwischen gleichen Nodes
 - Animations-Möglichkeiten (Pulse, Flow, Sequential)

Interaktion

- Hover-Highlighting mit visueller Feedback
- Einzelselektion und Multi-Selektion (Ctrl+Klick)
- Box-Selektion (Shift+Drag)
- Orbit-Kamerasteuerung (Three.js OrbitControls)

Layout-Algorithmen

- Force-Directed Layout
- Circular Layout
- Grid Layout
- Hierarchical Layout
- Fruchterman-Reingold Layout
- Spring Embedder Layout

Datenformat-Unterstützung

Legacy-Format

```
{
  "nodes": [
    { "id": "n1", "x": 0, "y": 0, "z": 0, "name": "Node 1" }
  ],
  "edges": [
    { "start": "n1", "end": "n2", "name": "Connection" }
  ]
}
```

Future-Format

```
{
  "system": "ExampleSystem",
  "metadata": {
    "version": "1.0",
    "author": "Author Name"
  },
  "dataModel": {
    "entities": { ... },
    "relationships": { ... }
  },
  "visualMappings": {
```

```

    "defaultPresets": { ... }
},
"data": {
  "entities": [ ... ],
  "relationships": [ ... ]
}
}
}

```

Architektur-Philosophie

Nodges folgt einer **komponentenbasierten Architektur** mit klarer Trennung der Verantwortlichkeiten:

- **Manager-Pattern:** Zentrale Manager-Klassen koordinieren spezifische Funktionsbereiche
- **State-Management:** Zentralisierter Anwendungszustand mit Subscriber-Pattern
- **Event-Driven:** Kommunikation zwischen Komponenten über Events
- **Mapping-Engine:** Daten-zu-Visual-Transformation über konfigurierbare Mappings

Projektstruktur

```

Nodges/
└── src/
    ├── App.ts          # Haupteinstiegspunkt
    ├── types.ts        # TypeScript-Typdefinitionen
    ├── core/           # Kern-Manager
    ├── effects/        # Visuelle Effekte
    ├── ui/             # UI-Komponenten
    ├── utils/          # Hilfsklassen
    ├── styles/         # CSS-Styles
    └── workers/        # Web Workers
    └── public/
        └── data/        # JSON-Datendateien
    └── doc/            # Dokumentation
    └── index.html      # HTML-Einstiegspunkt

```

Zielgruppe

Nodges richtet sich an:

- **Datenwissenschaftler:** Zur Exploration von Netzwerkdaten
- **Entwickler:** Die Graph-Visualisierung in eigene Projekte integrieren möchten
- **Forscher:** Zur Analyse komplexer Beziehungsnetzwerke
- **Designer:** Die interaktive Visualisierungen erstellen möchten

Weiterentwicklung

Das Projekt befindet sich in aktiver Entwicklung (Version 0.97.4) mit dem Fokus auf:

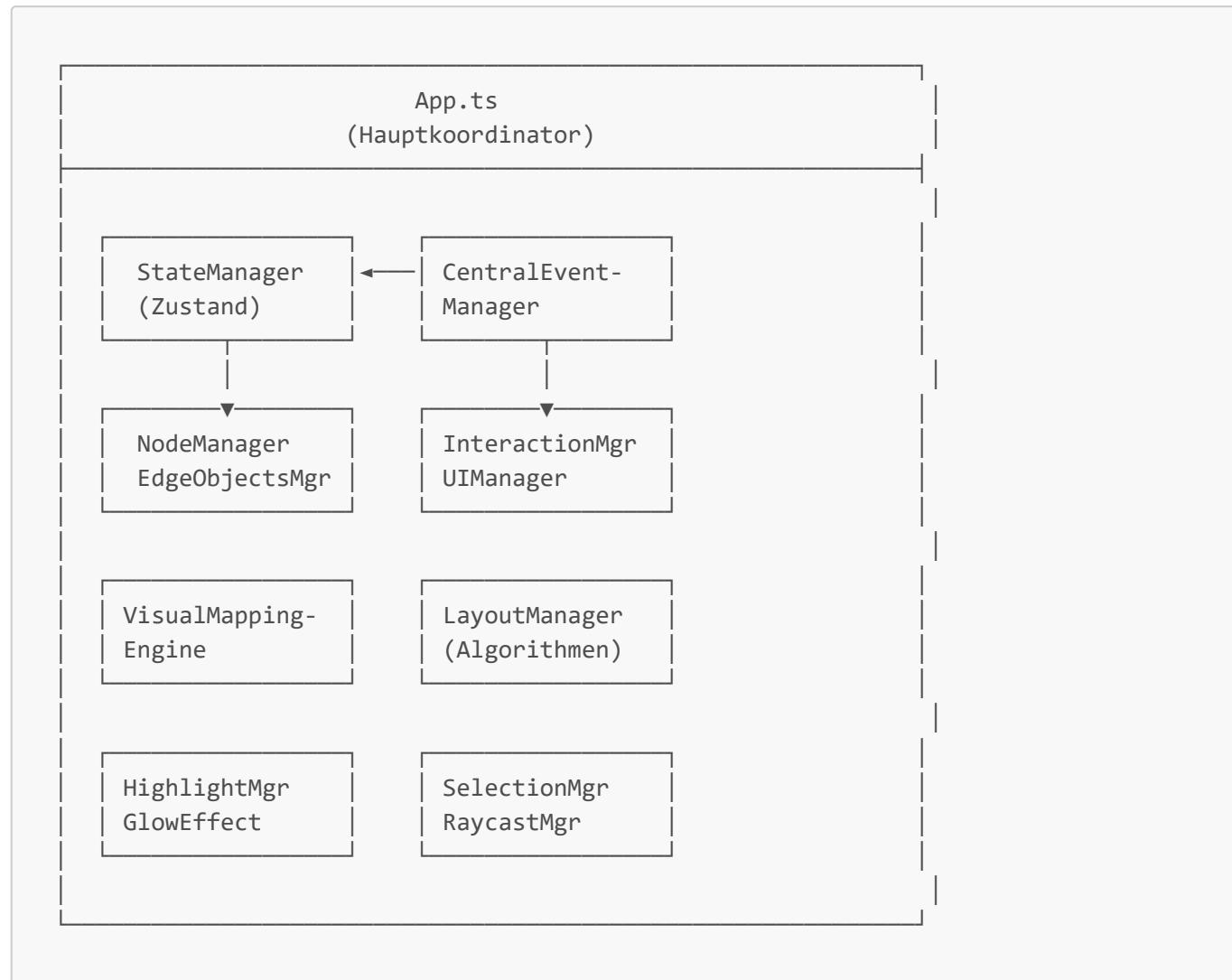
- Erweiterung der Layout-Algorithmen
- Verbesserte Performance bei großen Graphen

- Erweiterte Export-Funktionen
- Kollaborative Features

02 Systemarchitektur und Design-Prinzipien

Architektur-Überblick

Nodges folgt einer modularen Architektur, die Wartbarkeit, Erweiterbarkeit und testbare Komponenten ermöglicht.



Core-Komponenten

App.ts - Haupteinstiegspunkt

Die **App**-Klasse ist der zentrale Koordinator:

Verantwortlichkeit	Beschreibung
Three.js-Setup	Initialisiert Scene, Camera, Renderer, Controls
Manager-Instanziierung	Erstellt und verbindet alle Manager
Datenladung	Lädt und verarbeitet Graph-Daten
Animation-Loop	Steuert den Render-Zyklus

Wichtige Methoden:

- `init()`: Initialisiert die gesamte Anwendung
- `loadGraphData()`: Lädt Graphdaten und erstellt 3D-Objekte
- `animate()`: Der Haupt-Renderingloop
- `fitCameraToScene()`: Passt Kamera an den Graph an

StateManager - Zustandsverwaltung

Der `StateManager` implementiert ein **Subscriber-Pattern** für reaktive Zustandsverwaltung:

```
interface State {
    hoveredObject: THREE.Object3D | null;
    selectedObject: THREE.Object3D | null;
    selectedObjects: Set<THREE.Object3D>;
    isBoxSelecting: boolean;
    highlightedObjects: Set<THREE.Object3D>;
    glowIntensity: number;
    // ... weitere Properties
}
```

Features:

- Kategorisierte Subscriber für selektive Benachrichtigung
- Batch-Updates zur Performance-Optimierung
- Automatische Glow-Animation

DataParser - Datenverarbeitung

Der `DataParser` unterstützt zwei Formate:

1. **Legacy-Format**: Einfache `nodes/edges`-Arrays
2. **Future-Format**: Erweitert mit DataModel und VisualMappings

Datenfluß:

```
JSON-Datei → DataParser.parse() → GraphData → App.loadGraphData()
```

Der Parser führt folgende Schritte durch:

- Format-Erkennung (`isLegacyFormat()`, `isFutureFormat()`)
- Konvertierung (`convertLegacyFormat()`)
- Zod-Validierung (`GraphDataSchema`)
- Wertparsing basierend auf DataModel

Design-Prinzipien

1. Separation of Concerns

Jede Klasse hat eine klar definierte Verantwortlichkeit:

Klasse	Verantwortlichkeit
NodeManager	Node-Erstellung und -Verwaltung
EdgeObjectsManager	Edge-Erstellung und -Animation
LayoutManager	Layout-Algorithmen
VisualMappingEngine	Daten-zu-Visual-Transformation
HighlightManager	Hover/Selection-Feedback
InteractionManager	Input-Verarbeitung
UIManager	UI-Panel-Steuerung

2. Manager-Pattern

Alle Kernfunktionalitäten sind in Manager-Klassen gekapselt:

- Jeder Manager ist eigenständig testbar
- Manager kommunizieren über den StateManager
- Manager haben definierten Lebenszyklus (`destroy()`, `dispose()`)

3. Event-Driven Architecture

Komponenten kommunizieren über Events statt direkter Aufrufe:

```
// StateManager Subscriber-Pattern
stateManager.subscribe((state) => {
    // Reagiere auf Änderungen
}, 'highlight');

// Änderungen lösen Benachrichtigungen aus
stateManager.update({ hoveredObject: object });
```

4. Type-First Development

TypeScript wird konsequent eingesetzt:

- Zod-Schemas für Runtime-Validierung
- Interfaces für alle Datenstrukturen
- Generische Typen wo sinnvoll

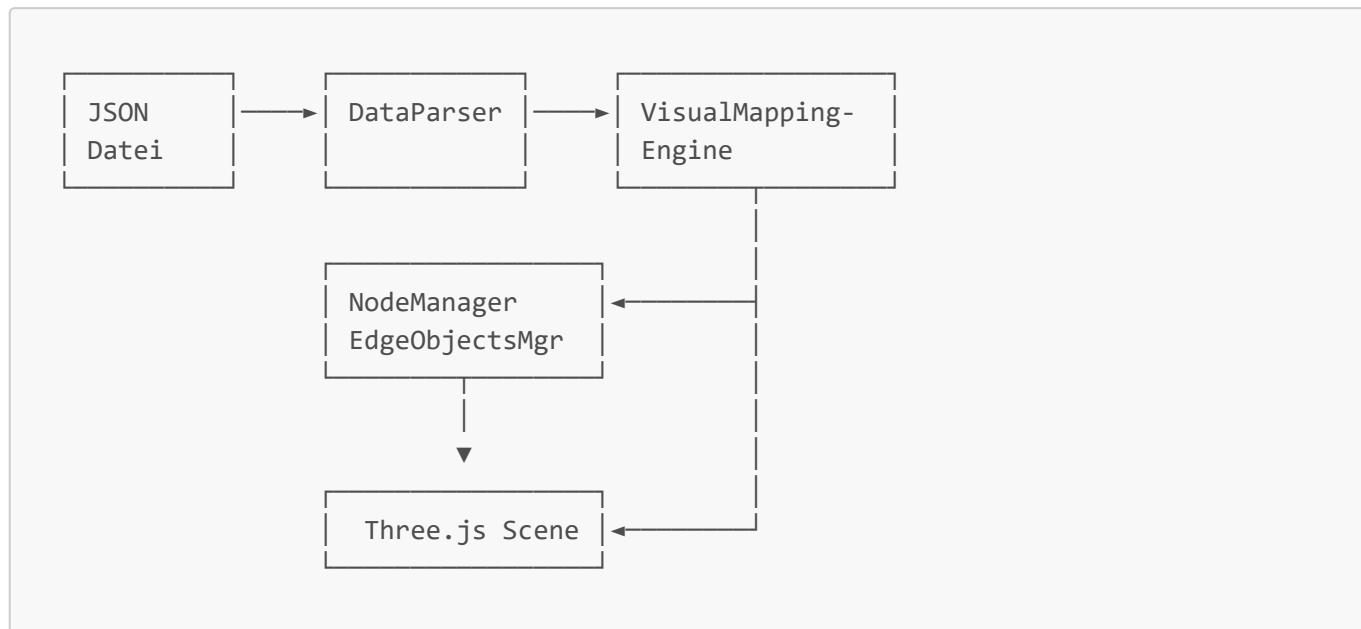
5. Performance-First

Strategien für optimale Performance:

- **InstancedMesh** für gleiche Geometrien (weniger Draw Calls)
- **Web Workers** für rechenintensive Layout-Algorithmen

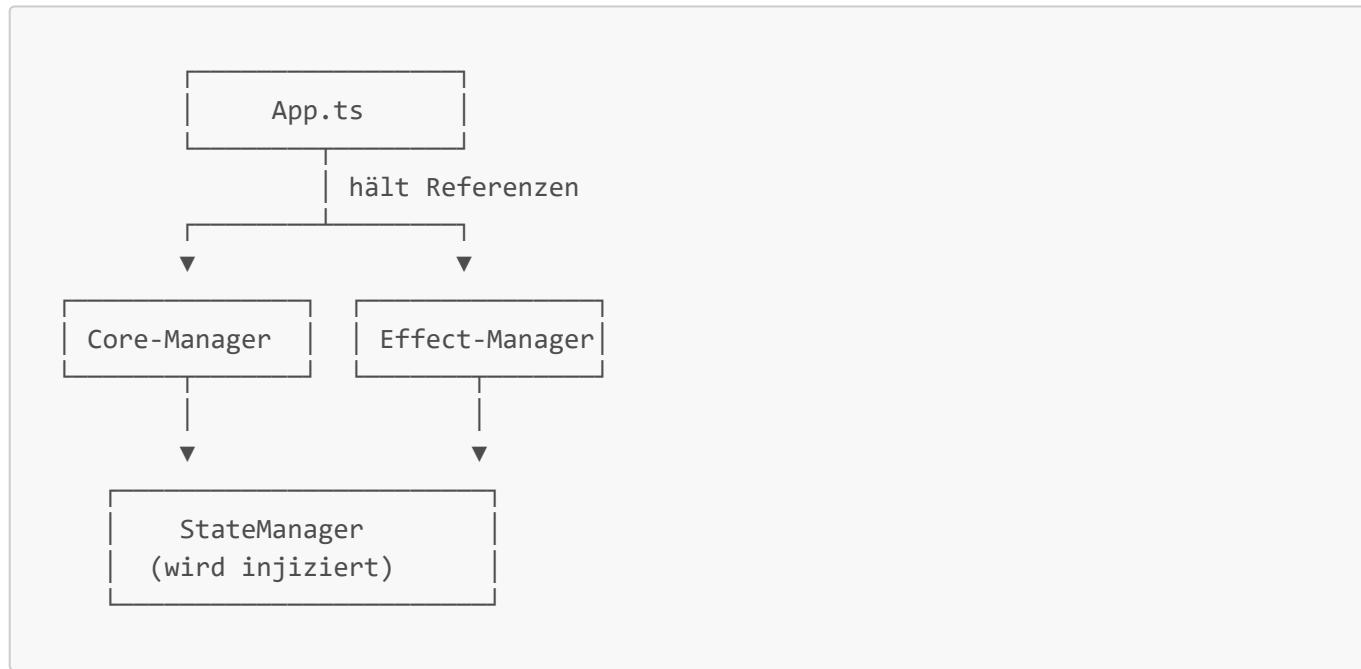
- **Lazy Initialization** bei teuren Operationen
- **Object Pooling** bei häufig erstellten Objekten

Datenfluss



Abhängigkeiten

Die Abhängigkeits-Hierarchie ist strikt definiert:



Erweiterbarkeit

Das System ist auf Erweiterung ausgelegt:

Neue Layouts hinzufügen:

```
layoutManager.registerLayout('custom', {
    name: 'Custom Layout',
    apply: (nodes, edges, options) => { /* ... */ },
    options: { /* defaults */ }
});
```

Neue Visual Mappings:

```
// In VisualMappingEngine erweitern
applyMapping(mapping, data) {
    switch(mapping.function) {
        case 'custom': return this.customMapping(value, mapping);
        // ...
    }
}
```

Neue Geometrien:

```
// In NodeManager.initializeGeometries()
this.geometryCache.set('custom', new CustomBufferGeometry());
```

03 Datenmanagement und Validierung

Überblick

Das Datenmanagement in Nodges basiert auf einem flexiblen, erweiterbaren Daten-Schema, das sowohl einfache Legacy-Formate als auch komplexe Future-Formate mit Visual Mappings unterstützt.

Unterstützte Datenformate

Legacy-Format

Das Legacy-Format ist einfach strukturiert und für schnelle Prototypen geeignet:

```
{  
  "nodes": [  
    {  
      "id": "node1",  
      "name": "Knoten 1",  
      "x": 0,  
      "y": 0,  
      "z": 0,  
      "type": "default"  
    }  
  ],  
  "edges": [  
    {  
      "start": "node1",  
      "end": "node2",  
      "name": "Verbindung"  
    }  
  ]  
}
```

Eigenschaften:

- **id**: Eindeutige Kennung (string oder number)
- **x, y, z**: Position im 3D-Raum
- **name**: Anzeigename
- **type**: Optionaler Typ für Geometrie-Mapping

Future-Format

Das Future-Format ermöglicht komplexe Visual Mappings und strikte Typisierung:

```
{  
  "system": "Systemname",  
  "metadata": {
```

```
"created": "2024-01-01",
"version": "1.0",
"author": "Autor",
"description": "Beschreibung"
},
"dataModel": {
"entities": {
"person": {
"properties": {
"age": { "type": "continuous", "range": [0, 100] },
"role": { "type": "categorical", "values": ["admin", "user"] }
}
}
},
"relationships": {
"knows": {
"properties": {
"strength": { "type": "continuous", "range": [0, 1] }
}
}
}
},
"visualMappings": {
"defaultPresets": {
"node": {
"size": {
"source": "age",
"function": "linear",
"range": [0.3, 1.5]
},
"color": {
"source": "role",
"function": "heatmap"
}
}
}
},
"data": {
"entities": [
{
"id": "p1",
"type": "person",
"label": "Alice",
"position": { "x": 0, "y": 0, "z": 0 },
"age": 30,
"role": "admin"
}
],
"relationships": [
{
"id": "r1",
"type": "knows",
"source": "p1",
"target": "p2",
"strength": 0.5
}
]
}
```

```

        "strength": 0.8
    }
}
}
}

```

DataModel-Spezifikation

Property-Typen

Typ	Beschreibung	Beispiel
continuous	Numerische Werte	{ "type": "continuous", "range": [0, 100] }
categorical	Kategorische Werte	{ "type": "categorical", "values": ["A", "B"] }
vector	Mehrdimensionale Werte	{ "type": "vector", "dimensions": ["x", "y", "z"] }
spatial	Koordinaten	{ "type": "spatial", "coordinates": ["lat", "lon"] }
temporal	Zeitliche Werte	{ "type": "temporal" }

PropertySchema-Definition

```

interface PropertySchema {
  type: 'continuous' | 'categorical' | 'vector' | 'spatial' | 'temporal';
  range?: [number, number];           // für continuous
  unit?: string;                    // Maßeinheit
  dimensions?: string[];           // für vector
  values?: string[];                // für categorical
  coordinates?: string[];           // für spatial
  default?: any;                   // Standardwert
}

```

Zod-Validierung

Nodges verwendet Zod für Runtime-Validierung:

Haupt-Schema

```

export const GraphDataSchema = z.object({
  system: z.string(),
  metadata: z.object({
    created: z.string().optional(),
    version: z.string().optional(),
    author: z.string().optional(),
    description: z.string().optional(),
  })
})

```

```

}).passthrough(),
dataModel: DataModelSchema.optional(),
visualMappings: VisualMappingsSchema.optional(),
data: z.object({
    entities: z.array(EntityDataSchema),
    relationships: z.array(RelationshipDataSchema),
}),
});

```

Entity-Schema

```

export const EntityDataSchema = z.object({
    id: z.string(),
    type: z.string(),
    label: z.string().optional(),
    position: z.object({
        x: z.number(),
        y: z.number(),
        z: z.number()
    }).optional(),
}).passthrough(); // Erlaubt zusätzliche Properties

```

Relationship-Schema

```

export const RelationshipDataSchema = z.object({
    id: z.string().optional(),
    type: z.string(),
    source: z.string(),
    target: z.string(),
    label: z.string().optional(),
}).passthrough();

```

DataParser

Der **DataParser** ist für die Datenverarbeitung zuständig:

Hauptmethoden

Methode	Beschreibung
<code>parse(rawData)</code>	Parst beliebige Daten und gibt GraphData zurück
<code>isLegacyFormat(data)</code>	Prüft auf Legacy-Format
<code>isFutureFormat(data)</code>	Prüft auf Future-Format
<code>convertLegacyFormat(data)</code>	Konvertiert Legacy zu Future

Methode	Beschreibung
<code>normalizeFutureFormat(data)</code>	Normalisiert Future-Format

Format-Erkennung

```
isLegacyFormat(data: any): boolean {
    return data && (
        Array.isArray(data.nodes) ||
        Array.isArray(data.edges) ||
        data.Nodes ||
        data.Edges
    );
}

isFutureFormat(data: any): boolean {
    return data &&
        data.data &&
        Array.isArray(data.data.entities) &&
        Array.isArray(data.data.relationships);
}
```

Wertparsing

Basierend auf dem DataModel werden Werte typisiert geparsst:

```
parseValue(value: any, schema: PropertySchema): any {
    switch (schema.type) {
        case 'continuous': return this.parseContinuous(value, schema);
        case 'categorical': return this.parseCategorical(value, schema);
        case 'vector': return this.parseVector(value, schema);
        case 'spatial': return this.parseSpatial(value, schema);
        case 'temporal': return this.parseTemporal(value, schema);
        default: return value;
    }
}
```

Hilfsmethoden

Entity-Abfragen

```
// Alle Entitäten eines Typs
getEntities(graphData: GraphData, type?: string): EntityData[]

// Alle Typen im Graph
getEntityTypes(graphData: GraphData): string[]
```

```
// Entity nach ID finden
findEntity(graphData: GraphData, id: string): EntityData | undefined
```

Relationship-Abfragen

```
// Alle Beziehungen eines Typs
getRelationships(graphData: GraphData, type?: string): RelationshipData[]

// Alle Beziehungstypen
getRelationshipTypes(graphData: GraphData): string[]

// Beziehungen einer Entity
findRelationshipsForEntity(graphData: GraphData, entityId: string):
RelationshipData[]
```

Import/Export

ImportManager

Der [ImportManager](#) unterstützt:

- JSON-Dateien (per File-Dialog)
- Drag & Drop
- URL-basiertes Laden

ExportManager

Der [ExportManager](#) ermöglicht:

- JSON-Export der aktuellen Graphdaten
- PNG-Screenshots
- Positions-Export

Fehlerbehandlung

Bei Validierungsfehlern werden klare Fehlermeldungen generiert:

```
try {
    const parsed = GraphDataSchema.parse(data);
} catch (error) {
    if (error instanceof z.ZodError) {
        console.error('Validierungsfehler:', error.errors);
        // Fehler enthalten: path, message, code
    }
}
```

Best Practices

1. **Immer IDs verwenden:** Statt Array-Indizes immer String-IDs für Referenzen
2. **DataModel definieren:** Für strikte Typisierung DataModel angeben
3. **Positionen vorgeben:** Wenn keine Positionen, wird ein Layout-Algorithmus benötigt
4. **Zod-Validierung nutzen:** Vor dem Laden immer validieren

04 3D-Rendering und Szenen-Management

Three.js-Integration

Nodges verwendet Three.js als WebGL-Abstraktionsschicht für Hardware-beschleunigtes 3D-Rendering.

Scene-Setup

Grundkonfiguration

Die Szene wird in `App.initThreeJS()` initialisiert:

```
// Scene erstellen
this.scene = new THREE.Scene();
this.scene.background = new THREE.Color(0x1a1a2e);

// Kamera konfigurieren
this.camera = new THREE.PerspectiveCamera(60, aspect, 0.1, 2000);
this.camera.position.set(30, 30, 30);

// Renderer mit Antialiasing
this.renderer = new THREE.WebGLRenderer({ antialias: true });
this.renderer.setSize(window.innerWidth, window.innerHeight);
this.renderer.setPixelRatio(window.devicePixelRatio);

// OrbitControls für Kamerasteuerung
this.controls = new OrbitControls(this.camera, this.renderer.domElement);
this.controls.enableDamping = true;
this.controls.dampingFactor = 0.25;
```

Beleuchtung

```
// Ambient Light für Grundbeleuchtung
const ambientLight = new THREE.AmbientLight(0x404040, 0.5);
this.scene.add(ambientLight);

// Directional Light für Schattierungen
const directionalLight = new THREE.DirectionalLight(0xffffff, 0.8);
directionalLight.position.set(20, 30, 20);
this.scene.add(directionalLight);
```

NodeManager

Der `NodeManager` verwaltet alle Node-Objekte in der Szene.

Geometrie-Typen

Typ	Geometrie	Verwendung
sphere	SphereGeometry(0.5, 32, 32)	Standard-Nodes
box	BoxGeometry(0.8, 0.8, 0.8)	Strukturelle Elemente
dodecahedron	DodecahedronGeometry(0.5)	Wichtige Nodes
icosahedron	IcosahedronGeometry(0.5)	Verbindungsknoten
octahedron	OctahedronGeometry(0.5)	Reguläre Nodes
tetrahedron	TetrahedronGeometry(0.5)	Einfache Nodes

InstancedMesh-Optimierung

Für Performance werden gleichartige Nodes als InstancedMesh gerendert:

```
// Pro Geometrie-Typ ein InstancedMesh
const mesh = new THREE.InstancedMesh(
    geometry,
    material,
    nodeCount
);

// Transformation pro Instanz
const matrix = new THREE.Matrix4();
matrix.setPosition(x, y, z);
matrix.scale(new THREE.Vector3(size, size, size));
mesh.setMatrixAt(instanceIndex, matrix);

// Farbe pro Instanz
mesh.setColorAt(instanceIndex, color);
```

Node-Erstellung

```
updateNodes(entities: EntityData[]) {
    // Gruppierung nach Geometrie-Typ
    const grouped = this.groupByGeometry(entities);

    for (const [type, nodes] of grouped) {
        // InstancedMesh erstellen/aktualisieren
        const mesh = this.getOrCreateMesh(type, nodes.length);

        nodes.forEach((node, i) => {
            // Visual Properties anwenden
            const visuals = this.visualMappingEngine.applyToEntity(node);
            this.setInstanceTransform(mesh, i, node.position, visuals.size);
            this.setInstanceColor(mesh, i, visuals.color);
        });
    }
}
```

```

    }
}
```

EdgeObjectsManager

Der [EdgeObjectsManager](#) erstellt und animiert alle Kanten.

Kurven-Berechnung

Kanten werden als Quadratic Bezier Curves dargestellt:

```

// Kontrollpunkt für Krümmung berechnen
const mid = startPos.clone().add(endPos).multiplyScalar(0.5);
const normal = new THREE.Vector3()
  .crossVectors(
    endPos.clone().sub(startPos).normalize(),
    new THREE.Vector3(0, 1, 0)
  )
  .normalize();

// Offset basierend auf Kantenindex (für Mehrfach-Kanten)
const offsetScale = (index - (totalEdges - 1) / 2) * curveFactor;
mid.add(normal.multiplyScalar(offsetScale));

// Bezier-Kurve
const curve = new THREE.QuadraticBezierCurve3(startPos, mid, endPos);
```

TubeGeometry

Kanten werden als Röhren gerendert:

```

const tubeGeometry = new THREE.TubeGeometry(
  curve,                      // Pfad
  this.tubularSegments,        // Segmente entlang der Kurve
  this.edgeThickness,          // Radius
  this.radialSegments,         // Segmente um den Umfang
  false                        // geschlossen
);
```

Edge-Animation

Vier Animationsmodi werden unterstützt:

```

animate() {
  const time = performance.now() * 0.001 * this.pulseSpeed;

  switch(this.animationMode) {
```

```

        case 'pulse':
            // Globale Pulse-Animation
            this.animatePulse(time);
            break;
        case 'sequential':
            // Wellen-Animation
            this.animateSequential(time);
            break;
        case 'flow':
            // Bewegender Punkt
            this.animateFlow(time);
            break;
        case 'segments':
            // Segment-Farben
            this.animateSegments(time);
            break;
    }
}

```

Kamera-Management

OrbitControls

Die OrbitControls ermöglichen:

- Rotation um den Fokuspunkt (linke Maustaste)
- Zoom (Mausrad)
- Pan (rechte Maustaste)

Automatisches Framing

```

fitCameraToScene() {
    const box = new THREE.Box3().setFromObject(this.scene);
    const size = box.getSize(new THREE.Vector3());
    const center = box.getCenter(new THREE.Vector3());

    // Distanz berechnen
    const maxDim = Math.max(size.x, size.y, size.z);
    const fov = this.camera.fov * (Math.PI / 180);
    const distance = maxDim / (2 * Math.tan(fov / 2)) * 1.5;

    // Kamera positionieren
    this.camera.position.copy(center);
    this.camera.position.z += distance;
    this.controls.target.copy(center);
}

```

Render-Loop

Animation Loop

```

animate() {
    requestAnimationFrame(() => this.animate());

    // Zeitdelta für Animationen
    const now = performance.now();
    const deltaTime = (now - this.lastTime) / 1000;
    this.lastTime = now;

    // Manager-Updates
    this.stateManager.animate();
    this.edgeObjectsManager.animate();

    // Controls aktualisieren (für Damping)
    this.controls.update();

    // Rendern
    this.renderer.render(this.scene, this.camera);
}

```

Performance-Metriken

```

// FPS-Tracking
let frameCount = 0;
let lastFpsTime = performance.now();

animate() {
    frameCount++;
    const now = performance.now();
    if (now - lastFpsTime >= 1000) {
        const fps = frameCount;
        document.getElementById('fileFPS').textContent = fps.toString();
        frameCount = 0;
        lastFpsTime = now;
    }
    // ...
}

```

Materialien

Node-Material

```

const nodeMaterial = new THREE.MeshStandardMaterial({
    color: 0x4fc3f7,
    metalness: 0.1,
    roughness: 0.6,
}

```

```
    transparent: true,  
    opacity: 1.0  
});
```

Edge-Material

```
const edgeMaterial = new THREE.MeshBasicMaterial({  
    color: visualProps.color,  
    transparent: true,  
    opacity: visualProps.opacity || 1.0  
});  
  
// Bei Animationen: VertexColors  
edgeMaterial.vertexColors = true;
```

Szenen-Bereinigung

```
clearScene() {  
    // Alle Meshes entfernen  
    this.nodeManager.clear();  
    this.edgeObjectsManager.dispose();  
  
    // Highlight-Manager zurücksetzen  
    this.highlightManager.clearAllHighlights();  
  
    // StateManager zurücksetzen  
    this.stateManager.update({  
        hoveredObject: null,  
        selectedObject: null,  
        selectedObjects: new Set()  
    });  
}
```

GPU-Optimierungen

1. **InstancedMesh**: Reduziert Draw Calls bei vielen gleichen Objekten
2. **BufferGeometry**: Effiziente GPU-Speicherung
3. **Material-Sharing**: Mehrere Objekte teilen ein Material
4. **Frustum Culling**: Automatisch durch Three.js
5. **LOD**: Level-of-Detail (geplant für zukünftige Versionen)

05 Visuelle Effekte und Feedback-Systeme

Überblick

Nodges bietet ein umfangreiches System für visuelle Effekte, die dem Benutzer direktes Feedback über Interaktionen und Zustände geben.

HighlightManager

Der **HighlightManager** ist die zentrale Komponente für visuelles Feedback.

Highlight-Typen

Typ	Verwendung	Visuelle Darstellung
hover	Maus über Objekt	Kontur + Emissive-Glow
selection	Angeklicktes Objekt	Stärkere Kontur
search	Suchergebnis	Markierungsfarbe
path	Pfad-Highlighting	Pfadfarbe
group	Gruppenmarkierung	Gruppenfarbe

Highlight-Datenstruktur

```
interface HighlightData {
    type: string;                                // Typ des Highlights
    object: THREE.Object3D;                        // Zielobjekt
    originalMaterial: MaterialBackup | null;      // Backup für Reset
    options: any;                                  // Zusätzliche Optionen
    timestamp: number;                            // Erstellungszeit
}
```

Material-Backup

Vor dem Anwenden von Highlights wird das Original-Material gesichert:

```
backupMaterial(object: THREE.Object3D): MaterialBackup | null {
    const mesh = object as THREE.Mesh;
    const material = mesh.material as THREE.MeshStandardMaterial;

    return {
        color: material.color.clone(),
        emissive: material.emissive?.clone() ?? null,
        emissiveIntensity: material.emissiveIntensity,
        opacity: material.opacity,
```

```
        transparent: material.transparent,
        wasShared: this.isMaterialShared(object)
    };
}
```

Node-Highlighting

```
addNodeOutline(object: THREE.Object3D) {
    // Outline-Geometrie erstellen
    const outlineGeometry = (object as THREE.Mesh).geometry.clone();
    outlineGeometry.scale(1.15, 1.15, 1.15); // Leicht größer

    const outlineMaterial = new THREE.MeshBasicMaterial({
        color: 0x00ff00,
        side: THREE.BackSide, // Nur Rückseite rendern
        transparent: true,
        opacity: 0.5
    });

    const outline = new THREE.Mesh(outlineGeometry, outlineMaterial);
    object.add(outline);
}
```

Edge-Highlighting

```
addEdgeOutline(edge: THREE.Object3D, options = {}) {
    const userData = edge.userData;
    const curve = userData.curve;

    // Dickere Röhre für Outline
    const highlightThickness = this.stateManager.state.highlightThickness;
    const outlineGeometry = new THREE.TubeGeometry(
        curve,
        20, // Segmente
        highlightThickness,
        8,
        false
    );

    const outlineMaterial = new THREE.MeshBasicMaterial({
        color: options.color || userData.color || 0x00ff00,
        transparent: true,
        opacity: 0.7
    });

    const outline = new THREE.Mesh(outlineGeometry, outlineMaterial);
    edge.add(outline);
}
```

GlowEffect

Der **GlowEffect** erzeugt einen pulsierenden Leuchteffekt:

```
class GlowEffect {
    private glowIntensity: number = 0;
    private glowDirection: number = 1;

    update(deltaTime: number) {
        // Pulsieren zwischen 0.3 und 1.0
        this.glowIntensity += this.glowDirection * deltaTime * 2;

        if (this.glowIntensity >= 1.0) {
            this.glowIntensity = 1.0;
            this.glowDirection = -1;
        } else if (this.glowIntensity <= 0.3) {
            this.glowIntensity = 0.3;
            this.glowDirection = 1;
        }
    }

    applyToMaterial(material: THREE.MeshStandardMaterial) {
        material.emissiveIntensity = this.glowIntensity;
    }
}
```

Edge-Animationen

Pulse-Animation

Globale Helligkeits-Pulsation:

```
animatePulse(time: number) {
    const intensity = 0.5 + 0.5 * Math.sin(time * Math.PI * 2);

    this.animatedEdges.forEach(edge => {
        const color = edge.baseColor.clone();
        color.lerp(new THREE.Color(0xffffffff), intensity * 0.3);
        (edge.tube.material as THREE.MeshBasicMaterial).color = color;
    });
}
```

Sequential-Animation

Wellen-Effekt entlang der Kanten:

```
animateSequential(time: number) {
    this.animatedEdges.forEach((edge, index) => {
```

```

    const offset = index * 0.1; // Phasenverschiebung
    const intensity = 0.5 + 0.5 * Math.sin((time - offset) * Math.PI * 2);
    // ... Farbe anwenden
  });
}

```

Flow-Animation

Bewegender Punkt entlang der Kante:

```

animateFlow(time: number) {
  const positions = geometry.attributes.position;
  const colors = geometry.attributes.color;

  // Position auf der Kurve (0-1)
  const t = (time % 1);

  for (let i = 0; i < segments; i++) {
    const segmentT = i / segments;
    const distance = Math.abs(segmentT - t);
    const intensity = Math.max(0, 1 - distance * 10);

    // Highlight-Farbe an der Position
    colors.setXYZ(i,
      baseColor.r + intensity,
      baseColor.g + intensity,
      baseColor.b + intensity
    );
  }
}

```

Segments-Animation

Farbige Segmente:

```

animateSegments(time: number) {
  const colors = geometry.attributes.color;

  for (let i = 0; i < segments; i++) {
    const hue = (i / segments + time * 0.1) % 1;
    const color = new THREE.Color().setHSL(hue, 0.8, 0.5);
    colors.setXYZ(i, color.r, color.g, color.b);
  }
}

```

Visual Mapping Engine

Die [VisualMappingEngine](#) transformiert Datenwerte zu visuellen Eigenschaften.

Mapping-Funktionen

Funktion	Beschreibung	Formel
linear	Lineare Interpolation	$a + t * (b - a)$
exponential	Exponentielle Skalierung	$a * (b/a)^t$
logarithmic	Logarithmische Skalierung	$\log(\text{value}) / \log(\text{max})$
heatmap	Farb-Mapping	Wert zu Farbpalette
bipolar	Bipolare Skala	-1 bis +1 Mapping
pulse	Animations-Config	Gibt Animation-Object zurück

Heatmap-Paletten

```
getHeatmapColor(value: number, palette = 'viridis'): THREE.Color {
    const palettes = {
        viridis: [
            [0.267, 0.004, 0.329], // Violett
            [0.282, 0.140, 0.458],
            [0.253, 0.265, 0.530],
            [0.206, 0.372, 0.553],
            [0.163, 0.471, 0.558],
            [0.127, 0.566, 0.551],
            [0.134, 0.658, 0.517],
            [0.267, 0.749, 0.441],
            [0.477, 0.821, 0.318],
            [0.741, 0.873, 0.150],
            [0.993, 0.906, 0.144] // Gelb
        ],
        // ... weitere Paletten
    };
}

// Interpolation zwischen Farbstops
const index = value * (palette.length - 1);
const low = Math.floor(index);
const high = Math.ceil(index);
const t = index - low;

return new THREE.Color().lerpColors(
    new THREE.Color(...palette[low]),
    new THREE.Color(...palette[high]),
    t
);
}
```

State-basiertes Feedback

Das Feedback-System reagiert auf State-Änderungen:

```
// StateManager Subscriber
stateManager.subscribe((state) => {
    // Hover-Feedback
    if (state.hoveredObject !== previousHovered) {
        highlightManager.updateHighlights(state);
    }

    // Selection-Feedback
    if (state.selectedObject !== previousSelected) {
        highlightManager.updateHighlights(state);
    }
}, 'highlight');
```

Dev-Panel Steuerung

Über das Dev-Panel können Effekte angepasst werden:

Einstellung	Beschreibung	Standardwert
Edge Thickness	Kantendicke	0.10
Highlight Effects	Effekte an/aus	aktiviert
Highlight Size	Highlight-Größe	30%
Selection Bonus	Selektions-Verstärkung	20%
Curve Segments	Kurven-Auflösung	20
Tube Facets	Röhren-Facetten	8
Pulse Speed	Animations-Geschwindigkeit	1.00
Anim Mode	Animations-Modus	Pulse
Opacity	Kanten-Transparenz	1.00

06 Interaktions-Design und Input-Processing

Überblick

Das Interaktionssystem von Nodges ermöglicht intuitive Navigation und Manipulation von Graph-Daten durch verschiedene Input-Methoden.

CentralEventManager

Der **CentralEventManager** koordiniert alle Event-Handler:

Registrierte Events

```
class CentralEventManager {
    private events = [
        'mousedown', 'mouseup', 'mousemove',
        'click', 'dblclick',
        'keydown', 'keyup',
        'wheel',
        'touchstart', 'touchmove', 'touchend'
    ];

    constructor() {
        this.events.forEach(event => {
            window.addEventListener(event, this.dispatch.bind(this));
        });
    }
}
```

Event-Dispatch

```
dispatch(event: Event) {
    const handlers = this.handlers.get(event.type);
    if (handlers) {
        handlers.forEach(handler => handler(event));
    }
}
```

InteractionManager

Der **InteractionManager** verarbeitet Benutzerinteraktionen:

Maus-Pointer-Tracking

```
onMouseMove(event: MouseEvent) {
    // Normalisierte Device Coordinates (-1 bis +1)
    this.mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
    this.mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

    // Raycasting für Hover
    this.updateHover();
}
```

Click-Handling

```
onClick(event: MouseEvent) {
    const intersected = this.raycastManager.findIntersectedObject();

    if (event.ctrlKey) {
        // Multi-Selection
        this.toggleSelection(intersected);
    } else {
        // Einfache Selektion
        this.setSelection(intersected);
    }
}
```

RaycastManager

Der **RaycastManager** ermittelt 3D-Objekte unter dem Mauszeiger:

Raycaster-Setup

```
class RaycastManager {
    private raycaster = new THREE.Raycaster();

    findIntersectedObject(): THREE.Object3D | null {
        this.raycaster.setFromCamera(this.mouse, this.camera);

        // Prüfe alle interaktiven Objekte
        const objects = [
            ...this.nodeManager.getMeshes(),
            ...this.edgeObjectsManager.getMeshes()
        ];

        const intersects = this.raycaster.intersectObjects(objects, true);
        return intersects.length > 0 ? intersects[0].object : null;
    }
}
```

InstancedMesh-Handling

Bei InstancedMesh wird zusätzlich die Instanz-ID ermittelt:

```
if (intersected.object instanceof THREE.InstancedMesh) {
    const instanceId = intersected.instanceId;
    return {
        object: intersected.object,
        instanceId,
        userData: this.nodeManager.getNodeAt(geometryType, instanceId)
    };
}
```

SelectionManager

Der **SelectionManager** verwaltet Selektionszustände:

Selektions-Modi

Modus	Tastenkombination	Beschreibung
Einfach	Klick	Ersetzt Selektion
Additiv	Ctrl+Klick	Fügt zur Selektion hinzu
Subtraktiv	Ctrl+Klick (auf selektiert)	Entfernt von Selektion
Box	Shift+Drag	Rechteck-Selektion

Box-Selektion

```
startBoxSelection(event: MouseEvent) {
    this.boxStart = { x: event.clientX, y: event.clientY };
    this.isBoxSelecting = true;
    this.createSelectionBox();
}

updateBoxSelection(event: MouseEvent) {
    const box = this.selectionBox;
    box.style.left = Math.min(this.boxStart.x, event.clientX) + 'px';
    box.style.top = Math.min(this.boxStart.y, event.clientY) + 'px';
    box.style.width = Math.abs(event.clientX - this.boxStart.x) + 'px';
    box.style.height = Math.abs(event.clientY - this.boxStart.y) + 'px';
}

endBoxSelection() {
    const objectsInBox = this.findObjectsInBox();
    this.stateManager.setSelectedObjects(new Set(objectsInBox));
    this.isBoxSelecting = false;
}
```

Kamera-Steuerung

OrbitControls

```
// Konfiguration
this.controls = new OrbitControls(this.camera, this.renderer.domElement);
this.controls.enableDamping = true;          // Trägheit
this.controls.dampingFactor = 0.25;
this.controls.enableZoom = true;
this.controls.enablePan = true;
this.controls.enableRotate = true;

// Shift deaktiviert Pan für Box-Selektion
document.addEventListener('keydown', (e) => {
  if (e.key === 'Shift') {
    this.controls.enablePan = false;
  }
});

document.addEventListener('keyup', (e) => {
  if (e.key === 'Shift') {
    this.controls.enablePan = true;
  }
});
```

Kamera-Limits

```
this.controls.minDistance = 1;
this.controls.maxDistance = 500;
this.controls.maxPolarAngle = Math.PI; // Vollständige Rotation
```

Keyboard-Shortcuts

Der **KeyboardShortcuts**-Handler stellt Tastaturkürzel bereit:

Taste	Aktion
Escape	Selektion aufheben
Delete	Selektierte Objekte löschen (wenn editierbar)
Ctrl+A	Alle auswählen
Ctrl+F	Suche öffnen
F	Kamera auf Selektion fokussieren
R	Layout zurücksetzen
1-9	Layout-Presets

Implementation

```
class KeyboardShortcuts {
    private shortcuts: Map<string, () => void> = new Map();

    constructor() {
        this.registerDefaults();
        document.addEventListener('keydown', this.handleKeyDown.bind(this));
    }

    handleKeyDown(event: KeyboardEvent) {
        const key = this.getKeyString(event);
        const action = this.shortcuts.get(key);
        if (action) {
            event.preventDefault();
            action();
        }
    }

    getKeyString(event: KeyboardEvent): string {
        const parts = [];
        if (event.ctrlKey) parts.push('Ctrl');
        if (event.shiftKey) parts.push('Shift');
        if (event.altKey) parts.push('Alt');
        parts.push(event.key.toUpperCase());
        return parts.join('+');
    }
}
```

Touch-Support

Grundlegende Touch-Unterstützung:

```
onTouchStart(event: TouchEvent) {
    if (event.touches.length === 1) {
        // Single Touch = Maus-Simulation
        this.simulateMouseEvent('mousedown', event.touches[0]);
    }
}

onTouchMove(event: TouchEvent) {
    if (event.touches.length === 1) {
        this.simulateMouseEvent('mousemove', event.touches[0]);
    } else if (event.touches.length === 2) {
        // Pinch-to-Zoom
        this.handlePinch(event);
    }
}
```

Hover-Info-Panel

Der `HoverInfoPanel` zeigt Details zum gehovered Objekt:

```
updateHoverInfo(object: THREE.Object3D | null) {
    if (!object) {
        this.hidePanel();
        return;
    }

    const userData = object.userData;
    const content = this.formatInfo(userData);

    this.panel.innerHTML = content;
    this.panel.style.display = 'block';
    this.positionNearMouse();
}

formatInfo(data: any): string {
    return `
        <div class="hover-info">
            <h4>${data.label || data.id}</h4>
            <p>Type: ${data.type}</p>
            ${this.formatProperties(data)}
        </div>
    `;
}
```

Context-Menu

Das `ContextMenu` erscheint bei Rechtsklick:

```
class ContextMenu {
    show(event: MouseEvent, object: THREE.Object3D | null) {
        event.preventDefault();

        const menu = this.buildMenu(object);
        this.element.innerHTML = menu;
        this.element.style.left = event.clientX + 'px';
        this.element.style.top = event.clientY + 'px';
        this.element.style.display = 'block';
    }

    buildMenu(object: THREE.Object3D | null): string {
        const items = [
            { label: 'Eigenschaften', action: 'properties' },
            { label: 'Fokussieren', action: 'focus' },
            { separator: true },
            { label: 'Pfad finden...', action: 'pathfind' }
        ];
    }
}
```

```
    if (object) {
        items.push({ label: 'Nachbarn hervorheben', action: 'neighbors' });
    }

    return items.map(this.renderItem).join('');
}
}
```

Drag & Drop

Datei-Import per Drag & Drop

```
document.body.addEventListener('dragover', (e) => {
    e.preventDefault();
    document.body.classList.add('drag-over');
});

document.body.addEventListener('drop', (e) => {
    e.preventDefault();
    document.body.classList.remove('drag-over');

    const files = e.dataTransfer?.files;
    if (files?.length) {
        const file = files[0];
        if (file.name.endsWith('.json')) {
            this.importManager.importFile(file);
        }
    }
});
```

07 Algorithmen und Layout-Engine

Überblick

Der **LayoutManager** bietet verschiedene Algorithmen zur automatischen Anordnung von Graphen im 3D-Raum.

Verfügbare Layouts

Layout	Beschreibung	Komplexität
Force-Directed	Physik-basiert	$O(n^2)$
Fruchterman-Reingold	Optimiertes Force-Layout	$O(n^2)$
Spring Embedder	Federkraft-Simulation	$O(n * e)$
Circular	Kreisanordnung	$O(n)$
Grid	Rasteranordnung	$O(n)$
Hierarchical	Baumstruktur	$O(n + e)$
Random	Zufällige Platzierung	$O(n)$

LayoutManager-Architektur

```
class LayoutManager {
    private layouts: Map<string, LayoutDefinition>;
    private currentLayout: string | null;
    public isAnimating: boolean;
    public animationSpeed: number;

    registerLayout(id: string, layout: LayoutDefinition) {
        this.layouts.set(id, layout);
    }

    async applyLayout(
        layoutId: string,
        nodes: EntityData[],
        edges: RelationshipData[],
        options: LayoutOptions = {}
    ): Promise<boolean> {
        const layout = this.layouts.get(layoutId);
        if (!layout) return false;

        const mergedOptions = { ...layout.options, ...options };
        await layout.apply(nodes, edges, mergedOptions);
        this.currentLayout = layoutId;
        return true;
    }
}
```

```

    }
}
```

Force-Directed Layout

Das Force-Directed Layout simuliert physikalische Kräfte:

Kräfte

1. **Abstoßung** zwischen allen Knoten (Coulomb)
2. **Anziehung** entlang der Kanten (Feder)

Implementation

```

applyForceLayout(nodes: EntityData[], edges: RelationshipData[], options = {}) {
  const iterations = options.iterations || 100;
  const repulsion = options.repulsion || 1000;
  const attraction = options.attraction || 0.01;
  const damping = options.damping || 0.9;

  // Initialisierung
  nodes.forEach(node => {
    node.vx = 0; node.vy = 0; node.vz = 0;
  });

  for (let i = 0; i < iterations; i++) {
    // Abstoßung berechnen
    for (let j = 0; j < nodes.length; j++) {
      for (let k = j + 1; k < nodes.length; k++) {
        const dx = nodes[k].x - nodes[j].x;
        const dy = nodes[k].y - nodes[j].y;
        const dz = nodes[k].z - nodes[j].z;
        const dist = Math.sqrt(dx*dx + dy*dy + dz*dz) || 0.01;

        const force = repulsion / (dist * dist);
        const fx = (dx / dist) * force;
        const fy = (dy / dist) * force;
        const fz = (dz / dist) * force;

        nodes[j].vx -= fx; nodes[j].vy -= fy; nodes[j].vz -= fz;
        nodes[k].vx += fx; nodes[k].vy += fy; nodes[k].vz += fz;
      }
    }
  }

  // Anziehung entlang Kanten
  edges.forEach(edge => {
    const source = nodes.find(n => n.id === edge.source);
    const target = nodes.find(n => n.id === edge.target);
    if (!source || !target) return;

    const dx = target.x - source.x;
```

```

        const dy = target.y - source.y;
        const dz = target.z - source.z;
        const dist = Math.sqrt(dx*dx + dy*dy + dz*dz);

        const force = dist * attraction;
        source.vx += (dx / dist) * force;
        source.vy += (dy / dist) * force;
        source.vz += (dz / dist) * force;
        target.vx -= (dx / dist) * force;
        target.vy -= (dy / dist) * force;
        target.vz -= (dz / dist) * force;
    });

    // Positionen aktualisieren
    nodes.forEach(node => {
        node.x += node.vx; node.y += node.vy; node.z += node.vz;
        node.vx *= damping; node.vy *= damping; node.vz *= damping;
    });
}
}

```

Fruchterman-Reingold Layout

Optimierte Variante des Force-Directed Layouts:

```

applyFruchtermanReingoldLayout(nodes, edges, options = {}) {
    const area = options.area || 100;
    const iterations = options.iterations || 50;
    const k = Math.sqrt(area / nodes.length); // Optimale Distanz

    let temperature = area * 0.1; // "Cooling" Schedule

    for (let i = 0; i < iterations; i++) {
        // Initialisiere Displacement
        nodes.forEach(node => {
            node.disp = { x: 0, y: 0, z: 0 };
        });

        // Abstoßung
        for (let j = 0; j < nodes.length; j++) {
            for (let l = j + 1; l < nodes.length; l++) {
                const delta = subtract(nodes[j], nodes[l]);
                const dist = length(delta) || 0.01;
                const force = (k * k) / dist; // Fruchterman-Reingold Formel

                const disp = multiply(normalize(delta), force);
                nodes[j].disp = add(nodes[j].disp, disp);
                nodes[l].disp = subtract(nodes[l].disp, disp);
            }
        }
    }
}

```

```

// Anziehung
edges.forEach(edge => {
    const u = getNode(nodes, edge.source);
    const v = getNode(nodes, edge.target);
    const delta = subtract(v, u);
    const dist = length(delta);
    const force = (dist * dist) / k; // Federkraft

    const disp = multiply(normalize(delta), force);
    u.disp = add(u.disp, disp);
    v.disp = subtract(v.disp, disp);
});

// Positionen anwenden mit Temperature-Limit
nodes.forEach(node => {
    const dispLength = length(node.disp);
    if (dispLength > 0) {
        const capped = Math.min(dispLength, temperature);
        node.x += (node.disp.x / dispLength) * capped;
        node.y += (node.disp.y / dispLength) * capped;
        node.z += (node.disp.z / dispLength) * capped;
    }
});

// Cool down
temperature *= 0.95;
}
}

```

Circular Layout

Ordnet Nodes kreisförmig an:

```

applyCircularLayout(nodes, edges, options = {}) {
    const radius = options.radius || 10;
    const center = options.center || { x: 0, y: 0, z: 0 };

    nodes.forEach((node, i) => {
        const angle = (i / nodes.length) * Math.PI * 2;
        node.x = center.x + Math.cos(angle) * radius;
        node.y = center.y; // Flach in XZ-Ebene
        node.z = center.z + Math.sin(angle) * radius;
    });
}

```

Grid Layout

Rasteranordnung:

```
applyGridLayout(nodes, edges, options = {}) {
    const spacing = options.spacing || 2;
    const cols = Math.ceil(Math.sqrt(nodes.length));

    nodes.forEach((node, i) => {
        const row = Math.floor(i / cols);
        const col = i % cols;

        node.x = col * spacing - (cols * spacing) / 2;
        node.y = 0;
        node.z = row * spacing - (Math.ceil(nodes.length / cols) * spacing) / 2;
    });
}
```

Hierarchical Layout

Für baumähnliche Strukturen:

```
applyHierarchicalLayout(nodes, edges, options = {}) {
    const levelSpacing = options.levelSpacing || 5;
    const nodeSpacing = options.nodeSpacing || 3;

    // Level berechnen (BFS von Wurzeln)
    const levels = this.calculateNodeLevels(nodes, edges);

    // Nodes pro Level gruppieren
    const byLevel = new Map<number, EntityData[]>();
    nodes.forEach(node => {
        const level = levels.get(node.id) || 0;
        if (!byLevel.has(level)) byLevel.set(level, []);
        byLevel.get(level)!.push(node);
    });

    // Positionieren
    byLevel.forEach((levelNodes, level) => {
        const y = level * levelSpacing;
        const width = levelNodes.length * nodeSpacing;

        levelNodes.forEach((node, i) => {
            node.x = i * nodeSpacing - width / 2;
            node.y = y;
            node.z = 0;
        });
    });
}

calculateNodeLevels(nodes, edges) {
    const levels = new Map<string, number>();
    const inDegree = new Map<string, number>();
```

```
// In-Degree berechnen
nodes.forEach(n => inDegree.set(n.id, 0));
edges.forEach(e => {
    inDegree.set(e.target, (inDegree.get(e.target) || 0) + 1);
});

// Wurzeln finden (In-Degree = 0)
const queue: string[] = [];
nodes.forEach(n => {
    if (inDegree.get(n.id) === 0) {
        queue.push(n.id);
        levels.set(n.id, 0);
    }
});

// BFS
while (queue.length > 0) {
    const current = queue.shift()!;
    const currentLevel = levels.get(current)!;

    edges.filter(e => e.source === current).forEach(e => {
        if (!levels.has(e.target)) {
            levels.set(e.target, currentLevel + 1);
            queue.push(e.target);
        }
    });
}

return levels;
}
```

Web Worker Integration

Für große Graphen werden Layouts in einem Web Worker berechnet:

```
applyLayoutWithWorker(layoutId, nodes, edges, options): Promise<boolean> {
    return new Promise((resolve, reject) => {
        const worker = new LayoutWorker();

        worker.postMessage({
            type: 'layout',
            layoutId,
            nodes: nodes.map(n => ({ id: n.id, x: n.x, y: n.y, z: n.z })),
            edges: edges.map(e => ({ source: e.source, target: e.target })),
            options
        });

        worker.onmessage = (event) => {
            const { positions } = event.data;

            // Positionen übernehmen
        };
    });
}
```

```

        positions.forEach((pos, i) => {
            nodes[i].x = pos.x;
            nodes[i].y = pos.y;
            nodes[i].z = pos.z;
        });

        worker.terminate();
        resolve(true);
    );
};

}

```

Positions-Normalisierung

Nach dem Layout werden Positionen normalisiert:

```

normalizeNodePositions(nodes, maxExtent = 10) {
    // Bounds berechnen
    let min = { x: Infinity, y: Infinity, z: Infinity };
    let max = { x: -Infinity, y: -Infinity, z: -Infinity };

    nodes.forEach(node => {
        min.x = Math.min(min.x, node.x);
        min.y = Math.min(min.y, node.y);
        min.z = Math.min(min.z, node.z);
        max.x = Math.max(max.x, node.x);
        max.y = Math.max(max.y, node.y);
        max.z = Math.max(max.z, node.z);
    });

    // Skalierung berechnen
    const range = {
        x: max.x - min.x || 1,
        y: max.y - min.y || 1,
        z: max.z - min.z || 1
    };
    const scale = maxExtent / Math.max(range.x, range.y, range.z);

    // Normalisieren
    nodes.forEach(node => {
        node.x = (node.x - (min.x + max.x) / 2) * scale;
        node.y = (node.y - (min.y + max.y) / 2) * scale;
        node.z = (node.z - (min.z + max.z) / 2) * scale;
    });
}

```

PathFinder

Der **PathFinder** findet Wege zwischen Nodes:

```
findShortestPath(startId: string, endId: string): string[] | null {
    // Dijkstra's Algorithmus
    const distances = new Map<string, number>();
    const previous = new Map<string, string>();
    const unvisited = new Set<string>();

    // Initialisierung
    this.nodes.forEach(n => {
        distances.set(n.id, n.id === startId ? 0 : Infinity);
        unvisited.add(n.id);
    });

    while (unvisited.size > 0) {
        // Minimum finden
        let current: string | null = null;
        let minDist = Infinity;
        unvisited.forEach(id => {
            if (distances.get(id)! < minDist) {
                minDist = distances.get(id)!;
                current = id;
            }
        });
        if (!current || current === endId) break;
        unvisited.delete(current);

        // Nachbarn aktualisieren
        this.getNeighbors(current).forEach(neighbor => {
            const dist = distances.get(current)! + 1;
            if (dist < distances.get(neighbor)!) {
                distances.set(neighbor, dist);
                previous.set(neighbor, current);
            }
        });
    }

    // Pfad rekonstruieren
    if (!previous.has(endId) && startId !== endId) return null;

    const path: string[] = [];
    let current: string | undefined = endId;
    while (current) {
        path.unshift(current);
        current = previous.get(current);
    }

    return path;
}
```

08 Benutzeroberfläche und UX

Überblick

Die Benutzeroberfläche von Nodges basiert auf einem Panel-System, das flexible Steuerung und Informationsanzeige ermöglicht.

Panel-System

Haupt-Panels

Panel	Position	Zweck
File Info Panel	Links oben	Zeigt Datei- und Graph-Statistiken
Files Panel	Links	Dateiauswahl und -verwaltung
Visual Mapping Panel	Links	Konfiguration der visuellen Mappings
Environment Panel	Links	Umgebungseinstellungen
Info Panel	Links unten	Details zum selektierten Objekt
Dev Panel	Rechts	Entwickler-Optionen

Panel-Struktur

```
<div id="fileInfoPanel" class="ui-panel collapsed">
    <div class="ui-panel-header">
        <h3>Info</h3>
        <button id="fileInfoToggle" class="ui-panel-toggle">></button>
    </div>
    <div id="fileInfoContent" class="ui-panel-content">
        <!-- Dynamischer Inhalt -->
    </div>
</div>
```

Toggle-Funktionalität

```
class UIManager {
    setupPanelToggle(panelId: string, toggleId: string) {
        const panel = document.getElementById(panelId);
        const toggle = document.getElementById(toggleId);

        toggle?.addEventListener('click', () => {
            panel?.classList.toggle('collapsed');
            toggle.textContent = panel?.classList.contains('collapsed') ? '>' :
            '<';
        });
    }
}
```

```

        });
    }
}

```

File Info Panel

Zeigt Statistiken zur aktuellen Visualisierung:

```

updateFileInfo(graphData: GraphData) {
    document.getElementById('fileFilename')!.textContent = sourceName;
    document.getElementById('fileNodeCount')!.textContent =
        graphData.data.entities.length.toString();
    document.getElementById('fileEdgeCount')!.textContent =
        graphData.data.relationships.length.toString();

    // Bounds anzeigen
    const bounds = this.calculateBounds(graphData.data.entities);
    document.getElementById('fileXAxis')!.textContent =
        `${bounds.min.x.toFixed(1)} - ${bounds.max.x.toFixed(1)}`;
    // ... Y und Z analog
}

```

Files Panel

Listet verfügbare JSON-Dateien:

```

async load fileList() {
    const response = await fetch('/api/files'); // oder statische Liste
    const files = await response.json();

    const container = document.getElementById('filePanelContent');
    container.innerHTML = files.map(file => `
        <div class="file-item" data-file="${file}">
            <span class="file-icon">📄</span>
            <span class="file-name">${file}</span>
        </div>
    `).join('');

    // Click-Handler
    container.querySelectorAll('.file-item').forEach(item => {
        item.addEventListener('click', () => {
            this.loadFile(item.dataset.file);
        });
    });
}

```

Visual Mapping Panel

Der **VisualMappingPanel** ermöglicht interaktive Mapping-Konfiguration:

```
class VisualMappingPanel {
    buildPanel(visualMappings: VisualMappings) {
        const container = document.getElementById('visualMappingContent');

        // Node Mappings
        if (visualMappings.defaultPresets.node) {
            container.innerHTML += this.createMappingSection(
                'Node',
                visualMappings.defaultPresets.node
            );
        }

        // Edge Mappings
        if (visualMappings.defaultPresets.edge) {
            container.innerHTML += this.createMappingSection(
                'Edge',
                visualMappings.defaultPresets.edge
            );
        }
    }

    createMappingSection(type: string, preset: any): string {
        return `
            <div class="mapping-section">
                <h4>${type} Mappings</h4>
                ${Object.entries(preset).map(([prop, mapping]) => `
                    <div class="mapping-row">
                        <label>${prop}</label>
                        <select data-type="${type}" data-prop="${prop}">
                            ${this.createSourceOptions(mapping)}
                        </select>
                    </div>
                `).join('')}
            </div>
        `;
    }
}
```

Environment Panel

Der **EnvironmentPanel** steuert Umgebungseinstellungen:

```
class EnvironmentPanel {
    buildPanel() {
        const container = document.getElementById('environmentContent');

        container.innerHTML =
            <div class="control-group">
```

```

        <label>Hintergrundfarbe</label>
        <input type="color" id="bgColorPicker" value="#1a1a2e">
    </div>
    <div class="control-group">
        <label>Ambient Light</label>
        <input type="range" id="ambientSlider" min="0" max="1" step="0.1"
value="0.5">
    </div>
    <div class="control-group">
        <label>Directional Light</label>
        <input type="range" id="directionalSlider" min="0" max="2"
step="0.1" value="0.8">
    </div>
`;

    this.setupEventListeners();
}

setupEventListeners() {
    document.getElementById('bgColorPicker')?.addEventListener('change', (e)
=> {
        const color = (e.target as HTMLInputElement).value;
        this.stateManager.update({ backgroundColor: color });
    });
}
}

```

Dev Options Panel

Das Dev-Panel bietet erweiterte Steuerelemente:

Edge-Controls

```

<!-- Edge Thickness -->
<div style="margin-bottom: 12px;">
    <label>Edge Thickness:</label>
    <input type="range" id="edgeThicknessSlider" min="0.01" max="0.5" value="0.1">
    <span id="edgeThicknessValue">0.10</span>
</div>

<!-- Highlight Effects Toggle -->
<label>Highlight Effects</label>
<input type="checkbox" id="highlightToggleInput" checked>

<!-- Animation Mode -->
<select id="edgeAnimModeSelect">
    <option value="pulse">Pulse (Global)</option>
    <option value="sequential">Sequential (Wave)</option>
    <option value="flow">Flow (Moving Point)</option>
    <option value="segments">Segments (Colors)</option>
</select>

```

Event-Handler

```

setupDevControls() {
    // Edge Thickness
    const thicknessSlider = document.getElementById('edgeThicknessSlider') as HTMLInputElement;
    thicknessSlider.addEventListener('input', () => {
        const value = parseFloat(thicknessSlider.value);
        this.stateManager.update({ edgeThickness: value });
        document.getElementById('edgeThicknessValue')!.textContent =
value.toFixed(2);
    });

    // Animation Mode
    const animSelect = document.getElementById('edgeAnimModeSelect') as HTMLSelectElement;
    animSelect.addEventListener('change', () => {
        this.stateManager.update({ edgeAnimationMode: animSelect.value });
    });

    // Reset Button
    document.getElementById('resetEdgeControls')?.addEventListener('click', () =>
{
    this.resetToDefaults();
});
}

```

LayoutGUI

Der [LayoutGUI](#) bietet eine lil-gui-basierte Oberfläche für Layout-Einstellungen:

```

class LayoutGUI {
    private gui: GUI;

    constructor(layoutManager: LayoutManager) {
        this.gui = new GUI({ title: 'Layout Controls' });

        // Layout-Auswahl
        const params = {
            layout: 'force',
            iterations: 100,
            repulsion: 1000,
            attraction: 0.01
        };

        this.gui.add(params, 'layout', layoutManager.getAvailableLayouts())
            .onChange(value => this.applyLayout(value));
    }
}

```

```
// Parameter
const folder = this.gui.addFolder('Parameters');
folder.add(params, 'iterations', 10, 500);
folder.add(params, 'repulsion', 100, 5000);
folder.add(params, 'attraction', 0.001, 0.1);

// Apply Button
this.gui.add({ apply: () => this.applyLayout(params.layout) }, 'apply')
.name('Apply Layout');
}

}
```

CSS-Styling

Panel-Grundstil

```
.ui-panel {
  position: fixed;
  background: rgba(30, 30, 40, 0.95);
  border: 1px solid #444;
  border-radius: 8px;
  color: #fff;
  font-family: 'Inter', sans-serif;
  font-size: 12px;
  min-width: 200px;
  max-width: 300px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.3);
  transition: all 0.3s ease;
}

.ui-panel.collapsed .ui-panel-content {
  display: none;
}

.ui-panel-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px 12px;
  border-bottom: 1px solid #444;
  cursor: pointer;
}

.ui-panel-toggle {
  background: transparent;
  border: none;
  color: #888;
  cursor: pointer;
  font-size: 14px;
}
```

Info-Row-Styling

```
.info-row {  
    display: flex;  
    justify-content: space-between;  
    padding: 4px 12px;  
    border-bottom: 1px solid #333;  
}  
  
.info-label {  
    color: #888;  
}  
  
.info-value {  
    color: #4fc3f7;  
    font-family: monospace;  
}
```

Slider-Styling

```
input[type="range"] {  
    -webkit-appearance: none;  
    width: 100%;  
    height: 4px;  
    background: #444;  
    border-radius: 2px;  
}  
  
input[type="range"]::-webkit-slider-thumb {  
    -webkit-appearance: none;  
    width: 14px;  
    height: 14px;  
    background: #4fc3f7;  
    border-radius: 50%;  
    cursor: pointer;  
}
```

Responsive Design

```
@media (max-width: 768px) {  
    .ui-panel {  
        min-width: 150px;  
        max-width: 250px;  
        font-size: 11px;  
    }  
  
    #devPanel {
```

```
        display: none; /* Dev-Panel auf kleinen Screens ausblenden */
    }
}
```

Accessibility

- Keyboard-Navigation für Panels
- ARIA-Labels für interaktive Elemente
- Ausreichender Farbkontrast
- Focus-Indikatoren

```
<button
  id=" fileInfoToggle"
  class="ui-panel-toggle"
  aria-label="Panel ein-/ausklappen"
  aria-expanded="false"
>
</button>
```

09 Utilities und Hilfssysteme

Überblick

Das `/src/utils`-Verzeichnis enthält spezialisierte Hilfsklassen, die Kernfunktionalitäten erweitern.

Datei-Übersicht

Datei	Zweck
<code>FileHandler.ts</code>	Dateioperationen
<code>ImportManager.ts</code>	Daten-Import
<code>ExportManager.ts</code>	Daten-Export
<code>BatchOperations.ts</code>	Massenoperationen
<code>PathFinder.ts</code>	Pfadfindung
<code>NetworkAnalyzer.ts</code>	Netzwerkanalyse
<code>NeighborhoodHighlighter.ts</code>	Nachbar-Hervorhebung
<code>NodeGroupManager.ts</code>	Gruppenverwaltung
<code>SelectionManager.ts</code>	Selektionslogik
<code>RaycastManager.ts</code>	3D-Objekterkennung
<code>LayoutGUI.ts</code>	Layout-Interface
<code>KeyboardShortcuts.ts</code>	Tastaturkürzel
<code>HoverInfoPanel.ts</code>	Hover-Informationen
<code>EdgeLabelManager.ts</code>	Kantenbeschriftungen
<code>DataEditor.ts</code>	Datenbearbeitung
<code>ContextMenu.ts</code>	Kontextmenü
<code>PerformanceOptimizer.ts</code>	Performance-Optimierung
<code>FutureDataParser.ts</code>	Erweiterter Parser

FileHandler

Verwaltet Dateioperationen:

```
class FileHandler {
    async loadFromUrl(url: string): Promise<any> {
        const response = await fetch(url);
        if (!response.ok) throw new Error(`HTTP ${response.status}`);
    }
}
```

```

        return response.json();
    }

    loadFromFile(file: File): Promise<any> {
        return new Promise((resolve, reject) => {
            const reader = new FileReader();
            reader.onload = () => resolve(JSON.parse(reader.result as string));
            reader.onerror = reject;
            reader.readAsText(file);
        });
    }

    downloadAsJson(data: any, filename: string) {
        const blob = new Blob([JSON.stringify(data, null, 2)], { type: 'application/json' });
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = filename;
        a.click();
        URL.revokeObjectURL(url);
    }
}

```

ImportManager

Erweiterte Import-Funktionalität:

```

class ImportManager {
    private supportedFormats = ['json', 'csv', 'graphml'];

    async import(source: string | File): Promise<GraphData> {
        const raw = typeof source === 'string'
            ? await this.loadUrl(source)
            : await this.loadFile(source);

        const format = this.detectFormat(source);
        return this.parse(raw, format);
    }

    detectFormat(source: string | File): string {
        const name = typeof source === 'string' ? source : source.name;
        const ext = name.split('.').pop()?.toLowerCase();
        return this.supportedFormats.includes(ext!) ? ext! : 'json';
    }

    private parse(raw: any, format: string): GraphData {
        switch(format) {
            case 'csv': return this.parseCSV(raw);
            case 'graphml': return this.parseGraphML(raw);
            default: return DataParser.parse(raw);
        }
    }
}

```

```

        }
    }
}
```

ExportManager

Unterstützt verschiedene Export-Formate:

```

class ExportManager {
    exportJSON(graphData: GraphData, filename = 'graph.json') {
        const data = JSON.stringify(graphData, null, 2);
        this.download(data, filename, 'application/json');
    }

    exportPositions(entities: EntityData[], filename = 'positions.json') {
        const positions = entities.map(e => ({
            id: e.id,
            x: e.position?.x ?? e.x,
            y: e.position?.y ?? e.y,
            z: e.position?.z ?? e.z
        }));
        this.download(JSON.stringify(positions, null, 2), filename,
        'application/json');
    }

    exportScreenshot(renderer: THREE.WebGLRenderer, filename = 'screenshot.png') {
        renderer.render(scene, camera);
        const dataUrl = renderer.domElement.toDataURL('image/png');
        const link = document.createElement('a');
        link.download = filename;
        link.href = dataUrl;
        link.click();
    }
}
```

BatchOperations

Effiziente Massenoperationen:

```

class BatchOperations {
    updateMultiple<T>(items: T[], updates: Partial<T>[]) {
        items.forEach((item, i) => Object.assign(item, updates[i]));
    }

    filterVisible(entities: EntityData[], camera: THREE.Camera): EntityData[] {
        const frustum = new THREE.Frustum();
        frustum.setFromProjectionMatrix(
            new THREE.Matrix4().multiplyMatrices(
                camera.projectionMatrix,
```

```

        camera.matrixWorldInverse
    )
);

return entities.filter(e => {
    const pos = new THREE.Vector3(e.position?.x ?? 0, e.position?.y ?? 0,
e.position?.z ?? 0);
    return frustum.containsPoint(pos);
});
}

groupByProperty<T>(items: T[], property: keyof T): Map<any, T[]> {
    const groups = new Map<any, T[]>();
    items.forEach(item => {
        const key = item[property];
        if (!groups.has(key)) groups.set(key, []);
        groups.get(key)!.push(item);
    });
    return groups;
}
}

```

NetworkAnalyzer

Netzwerk-Metriken berechnen:

```

class NetworkAnalyzer {
    calculateDegree(entityId: string, relationships: RelationshipData[]): number {
        return relationships.filter(r =>
            r.source === entityId || r.target === entityId
        ).length;
    }

    calculateBetweennessCentrality(entities: EntityData[], relationships:
RelationshipData[]) {
        // Vereinfachte Berechnung
        const betweenness = new Map<string, number>();
        entities.forEach(e => betweenness.set(e.id, 0));

        entities.forEach(s => {
            entities.forEach(t => {
                if (s.id !== t.id) {
                    const path = this.findShortestPath(s.id, t.id, relationships);
                    path?.forEach(nodeId => {
                        if (nodeId !== s.id && nodeId !== t.id) {
                            betweenness.set(nodeId, (betweenness.get(nodeId) || 0)
+ 1);
                        }
                    });
                }
            });
        });
    }
}

```

```

    });

    return betweenness;
}

findClusters(entities: EntityData[], relationships: RelationshipData[]): Set<string>[] {
    const visited = new Set<string>();
    const clusters: Set<string>[] = [];

    entities.forEach(entity => {
        if (!visited.has(entity.id)) {
            const cluster = this.bfs(entity.id, relationships, visited);
            clusters.push(cluster);
        }
    });
}

return clusters;
}
}

```

NeighborhoodHighlighter

Hebt Nachbar-Nodes hervor:

```

class NeighborhoodHighlighter {
    highlightNeighbors(entityId: string, depth: number = 1) {
        const neighbors = this.getNeighborsAtDepth(entityId, depth);

        neighbors.forEach((ids, level) => {
            const intensity = 1 - (level / (depth + 1));
            ids.forEach(id => {
                this.highlightManager.applyHighlight(
                    this.getObjectById(id),
                    'neighbor',
                    { intensity }
                );
            });
        });
    }

    getNeighborsAtDepth(startId: string, maxDepth: number): Map<number, Set<string>> {
        const result = new Map<number, Set<string>>();
        const visited = new Set<string>([startId]);
        let currentLevel = new Set<string>([startId]);

        for (let depth = 1; depth <= maxDepth; depth++) {
            const nextLevel = new Set<string>();
            currentLevel.forEach(id => {
                this.getDirectNeighbors(id).forEach(neighbor => {

```

```

        if (!visited.has(neighbor)) {
            nextLevel.add(neighbor);
            visited.add(neighbor);
        }
    });
});
result.set(depth, nextLevel);
currentLevel = nextLevel;
}

return result;
}
}

```

NodeGroupManager

Verwaltet Node-Gruppen:

```

class NodeGroupManager {
    private groups: Map<string, Set<string>> = new Map();

    createGroup(name: string, nodeIds: string[]) {
        this.groups.set(name, new Set(nodeIds));
    }

    addToGroup(groupName: string, nodeId: string) {
        this.groups.get(groupName)?.add(nodeId);
    }

    removeFromGroup(groupName: string, nodeId: string) {
        this.groups.get(groupName)?.delete(nodeId);
    }

    getGroupMembers(groupName: string): string[] {
        return [...(this.groups.get(groupName) || [])];
    }

    findGroupsForNode(nodeId: string): string[] {
        return [...this.groups.entries()]
            .filter(([_, members]) => members.has(nodeId))
            .map(([name]) => name);
    }
}

```

PerformanceOptimizer

Performance-Optimierungen:

```
class PerformanceOptimizer {
    private frameCount = 0;
    private lastFpsTime = performance.now();
    private currentFps = 60;

    measureFps() {
        this.frameCount++;
        const now = performance.now();
        if (now - this.lastFpsTime >= 1000) {
            this.currentFps = this.frameCount;
            this.frameCount = 0;
            this.lastFpsTime = now;
        }
        return this.currentFps;
    }

    shouldReduceQuality(): boolean {
        return this.currentFps < 30;
    }

    optimizeForPerformance(nodeManager: NodeManager, edgeManager: EdgeObjectsManager) {
        if (this.shouldReduceQuality()) {
            // Reduziere Geometrie-Komplexität
            // Deaktiviere Animationen
            // Reduziere Schatten
        }
    }

    throttle<T extends (...args: any[]) => any>(fn: T, delay: number): T {
        let lastCall = 0;
        return (...args) => {
            const now = Date.now();
            if (now - lastCall >= delay) {
                lastCall = now;
                return fn(...args);
            }
        } as T;
    }

    debounce<T extends (...args: any[]) => any>(fn: T, delay: number): T {
        let timeoutId: NodeJS.Timeout;
        return (...args) => {
            clearTimeout(timeoutId);
            timeoutId = setTimeout(() => fn(...args), delay);
        } as T;
    }
}
```

EdgeLabelManager

Verwaltet Kantenbeschriftungen:

```
class EdgeLabelManager {  
    private labels: Map<string, THREE.Sprite> = new Map();  
  
    createLabel(edge: RelationshipData, position: THREE.Vector3): THREE.Sprite {  
        const canvas = document.createElement('canvas');  
        const ctx = canvas.getContext('2d')!;  
        ctx.font = '16px Arial';  
        ctx.fillStyle = 'white';  
        ctx.fillText(edge.label || '', 0, 16);  
  
        const texture = new THREE.CanvasTexture(canvas);  
        const material = new THREE.SpriteMaterial({ map: texture });  
        const sprite = new THREE.Sprite(material);  
        sprite.position.copy(position);  
        sprite.scale.set(2, 1, 1);  
  
        this.labels.set(edge.id || `${edge.source}-${edge.target}`, sprite);  
        return sprite;  
    }  
  
    updateLabelPositions(edges: RelationshipData[], nodes: Map<string, THREE.Vector3>) {  
        edges.forEach(edge => {  
            const start = nodes.get(edge.source);  
            const end = nodes.get(edge.target);  
            if (start && end) {  
                const mid = start.clone().add(end).multiplyScalar(0.5);  
                this.labels.get(edge.id ||  
` ${edge.source}-${edge.target}`)?.position.copy(mid);  
            }  
        });  
    }  
}
```

10 Entwicklungs-Guide und Deployment

Entwicklungsumgebung einrichten

Voraussetzungen

- Node.js 18+
- npm oder yarn
- Moderner Browser mit WebGL-Unterstützung

Installation

```
# Repository klonen
git clone [repository-url]
cd Nodges

# Abhängigkeiten installieren
npm install
```

Development Server starten

```
npm run dev
```

Der Vite-Dev-Server startet auf <http://localhost:5173> mit Hot Module Replacement.

Projektstruktur

```
Nodges/
├── src/
│   ├── App.ts          # Haupteinstieg
│   ├── types.ts        # TypeScript-Definitionen
│   └── core/
│       ├── StateManager.ts
│       ├── DataParser.ts
│       ├── NodeManager.ts
│       ├── EdgeObjectsManager.ts
│       ├── LayoutManager.ts
│       ├── VisualMappingEngine.ts
│       ├── CentralEventManager.ts
│       ├── InteractionManager.ts
│       └── UIManager.ts
│   └── effects/         # Visuelle Effekte
│       ├── HighlightManager.ts
│       └── GlowEffect.ts
└── ui/                # UI-Panels
```

```
    |   └── EnvironmentPanel.ts
    |       └── VisualMappingPanel.ts
    ├── utils/           # Hilfsfunktionen
    ├── styles/          # CSS-Dateien
    └── workers/         # Web Workers
  └── public/
      └── data/          # JSON-Daten
  └── doc/              # Dokumentation
  └── index.html
  └── package.json
  └── tsconfig.json
  └── vite.config.ts
```

TypeScript-Konfiguration

`tsconfig.json`:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "module": "ESNext",
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "skipLibCheck": true,
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["src"]
}
```

Vite-Konfiguration

`vite.config.ts`:

```
import { defineConfig } from 'vite';

export default defineConfig({
  build: {
    outDir: 'dist',
    sourcemap: true,
  },
  server: {
```

```
    port: 5173,
    open: true,
},
// Web Worker Support
worker: {
  format: 'es',
},
});
});
```

Build-Prozess

Typ-Überprüfung

```
npm run type-check
```

Produktion-Build

```
npm run build
```

Erstellt optimierte Assets in `/dist`:

- Minifiziertes JavaScript
- Optimierte Chunks
- Asset-Hashing für Caching

Preview

```
npm run preview
```

Startet einen lokalen Server für den Produktion-Build.

Code-Konventionen

Datei-Benennung

- Klassen: `PascalCase.ts` (z.B. `StateManager.ts`)
- Utilities: `camelCase.ts` (z.B. `helpers.ts`)
- Konstanten: `UPPER_SNAKE_CASE`

TypeScript-Richtlinien

```
// Interfaces für Datenstrukturen
interface NodeData {
```

```

        id: string;
        position: Vector3;
    }

// Type für einfache Typen
type NodeId = string;

// Enum für feste Werte
enum LayoutType {
    Force = 'force',
    Grid = 'grid',
}

// Generics für wiederverwendbare Typen
function findById<T extends { id: string }>(items: T[], id: string): T | undefined
{
    return items.find(item => item.id === id);
}

```

Kommentare

```

/**
 * Berechnet die optimale Position basierend auf Nachbarn
 * @param nodeId - ID des Zielknotens
 * @param neighbors - Liste der Nachbar-IDs
 * @returns Neue Position als Vector3
 */
calculateOptimalPosition(nodeId: string, neighbors: string[]): THREE.Vector3 {
    // Implementation...
}

```

Debugging

Console-Logging

Kategorisierte Log-Ausgaben:

```

// Debug-Flags in StateManager
const DEBUG = {
    RAYCAST: false,
    HIGHLIGHT: false,
    LAYOUT: false,
};

if (DEBUG.RAYCAST) {
    console.log('[RAYCAST]', intersectedObject);
}

```

Chrome DevTools

1. **Performance Tab:** FPS und Render-Performance
2. **Memory Tab:** Speicherlecks identifizieren
3. **Console:** Logging und Errors
4. **Sources:** Breakpoints im TypeScript-Code

Three.js-Debugging

```
// Scene-Inhalt ausgeben
console.log(scene.children);

// Frustum visualisieren
const helper = new THREE.CameraHelper(camera);
scene.add(helper);

// Bounding-Boxes anzeigen
const box = new THREE.Box3().setFromObject(mesh);
const helper = new THREE.Box3Helper(box, 0xff0000);
scene.add(helper);
```

Testing

Manuelles Testing

1. Lade verschiedene JSON-Dateien
2. Teste alle Interaktionen (Hover, Click, Multi-Select)
3. Wechsle zwischen Layout-Algorithmen
4. Überprüfe Performance bei großen Graphen

Empfohlene Test-Dateien

- `small.json`: Basis-Test (< 50 Nodes)
- `medium.json`: Standard-Test (50-500 Nodes)
- `large.json`: Performance-Test (> 500 Nodes)
- `multi_edge_test.json`: Mehrfach-Kanten
- `complex_types.json`: Verschiedene Node-Typen

Deployment

Statisches Hosting

Nach `npm run build`:

1. Kopiere `/dist` auf den Webserver
2. Konfiguriere Server für SPA-Routing (falls nötig)

Nginx-Konfiguration

```

server {
    listen 80;
    root /var/www/nodges/dist;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    # Caching für Assets
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}

```

GitHub Pages

1. Build erstellen
2. `/dist` in `gh-pages` Branch pushen
3. GitHub Pages aktivieren

Erweiterung

Neuen Manager hinzufügen

1. Erstelle Datei in `/src/core/` oder `/src/utils/`
2. Implementiere Konstruktor mit Abhängigkeiten
3. Registriere in `App.ts`:

```
// In App.initManagers()
this.myManager = new MyManager(this.stateManager);
```

Neues Layout hinzufügen

```

// In LayoutManager.registerDefaultLayouts()
this.registerLayout('custom', {
    name: 'Custom Layout',
    apply: (nodes, edges, options) => {
        this.applyCustomLayout(nodes, edges, options);
    },
    options: {
        spacing: 5,
    }
});

applyCustomLayout(nodes, edges, options) {

```

```
// Implementation  
}
```

Bekannte Probleme

Problem	Workaround
Performance bei > 1000 Nodes	InstancedMesh verwenden
Edge-Labels überlappen	Dynamische Positionierung
Touch-Support eingeschränkt	Nur Basis-Gesten

11 Node-Edge-Mesh Creation Report

Überblick

Dieses Kapitel dokumentiert die technische Implementierung der 3D-Mesh-Erstellung für Nodes und Edges in Nodges.

Node-Mesh-Erstellung

Geometrie-Initialisierung

Der **NodeManager** erstellt bei Initialisierung einen Cache verschiedener Geometrien:

```
initializeGeometries() {
    this.geometryCache.set('sphere', new THREE.SphereGeometry(0.5, 32, 32));
    this.geometryCache.set('box', new THREE.BoxGeometry(0.8, 0.8, 0.8));
    this.geometryCache.set('dodecahedron', new THREE.DodecahedronGeometry(0.5));
    this.geometryCache.set('icosahedron', new THREE.IcosahedronGeometry(0.5));
    this.geometryCache.set('octahedron', new THREE.OctahedronGeometry(0.5));
    this.geometryCache.set('tetrahedron', new THREE.TetrahedronGeometry(0.5));
}
```

InstancedMesh-Strategie

Für optimale Performance werden Nodes des gleichen Typs als InstancedMesh gerendert:

```
updateNodes(entities: EntityData[]) {
    // Gruppieren nach Geometrie-Typ
    const grouped = new Map<string, EntityData[]>();
    entities.forEach(entity => {
        const visuals = this.visualMappingEngine.applyToEntity(entity);
        const geoType = visuals.geometry || 'sphere';
        if (!grouped.has(geoType)) grouped.set(geoType, []);
        grouped.get(geoType)!.push(entity);
    });

    // Erstelle InstancedMesh pro Typ
    grouped.forEach((typeEntities, geometryType) => {
        const geometry = this.geometryCache.get(geometryType);
        const material = this.sharedMaterial.clone();

        const mesh = new THREE.InstancedMesh(
            geometry!,
            material,
            typeEntities.length
        );

        // Transformationen setzen
    });
}
```

```

typeEntities.forEach((entity, index) => {
    const matrix = new THREE.Matrix4();
    const pos = entity.position || { x: 0, y: 0, z: 0 };
    const visuals = this.visualMappingEngine.applyToEntity(entity);
    const size = visuals.size || 1;

    matrix.compose(
        new THREE.Vector3(pos.x, pos.y, pos.z),
        new THREE.Quaternion(),
        new THREE.Vector3(size, size, size)
    );

    mesh.setMatrixAt(index, matrix);
    mesh.setColorAt(index, new THREE.Color(visuals.color || 0x4fc3f7));

    // UserData für Raycast
    this.entityIndex.set(entity.id, { geometryType, instanceId: index });
});

mesh.instanceMatrix.needsUpdate = true;
if (mesh.instanceColor) mesh.instanceColor.needsUpdate = true;

this.scene.add(mesh);
this.meshes.set(geometryType, mesh);
});
}

```

Positions-Updates

Bei Layout-Änderungen werden nur die Transformationsmatrizen aktualisiert:

```

updateNodePositions(entities: EntityData[]) {
    entities.forEach(entity => {
        const indexData = this.entityIndex.get(entity.id);
        if (!indexData) return;

        const mesh = this.meshes.get(indexData.geometryType);
        if (!mesh) return;

        const matrix = new THREE.Matrix4();
        mesh.getMatrixAt(indexData.instanceId, matrix);

        // Position extrahieren und aktualisieren
        const position = new THREE.Vector3();
        const quaternion = new THREE.Quaternion();
        const scale = new THREE.Vector3();
        matrix.decompose(position, quaternion, scale);

        position.set(
            entity.position?.x ?? entity.x ?? 0,
            entity.position?.y ?? entity.y ?? 0,
            entity.position?.z ?? entity.z ?? 0
        );
    });
}

```

```

        entity.position?.z ?? entity.z ?? 0
    );

    matrix.compose(position, quaternion, scale);
    mesh.setMatrixAt(indexData.instanceId, matrix);
});

this.meshes.forEach(mesh => {
    mesh.instanceMatrix.needsUpdate = true;
});
}

```

Edge-Mesh-Erstellung

Kurven-Berechnung

Edges werden als Quadratic Bezier Curves dargestellt:

```

createEdgeMesh(startPos: THREE.Vector3, endPos: THREE.Vector3, options) {
    // Mittelpunkt berechnen
    const mid = startPos.clone().add(endPos).multiplyScalar(0.5);

    // Normale zur Verbindungsgeraden (für Krümmung)
    const direction = endPos.clone().sub(startPos).normalize();
    const up = new THREE.Vector3(0, 1, 0);
    let normal = new THREE.Vector3().crossVectors(direction, up).normalize();

    // Fallback wenn parallel zu up
    if (normal.length() < 0.01) {
        normal = new THREE.Vector3().crossVectors(direction, new THREE.Vector3(1,
0, 0));
    }

    // Offset für Mehrfach-Kanten
    const curveFactor = this.stateManager.state.edgeCurveFactor;
    const edgeOffset = (options.index - (options.totalEdges - 1) / 2);
    const offsetMagnitude = edgeOffset * curveFactor * startPos.distanceTo(endPos)
* 0.3;

    // Kontrollpunkt
    const controlPoint = mid.clone().add(normal.multiplyScalar(offsetMagnitude));

    // Bezier-Kurve erstellen
    const curve = new THREE.QuadraticBezierCurve3(startPos, controlPoint, endPos);

    return { curve, controlPoint };
}

```

TubeGeometry-Erstellung

```
createTubeMesh(curve: THREE.QuadraticBezierCurve3, options) {  
    const thickness = this.stateManager.state.edgeThickness;  
    const tubularSegments = this.stateManager.state.tubularSegments;  
    const radialSegments = this.stateManager.state.radialSegments;  
  
    const geometry = new THREE.TubeGeometry(  
        curve,  
        tubularSegments,  
        thickness,  
        radialSegments,  
        false // nicht geschlossen  
    );  
  
    // Material mit Farbe  
    const material = new THREE.MeshBasicMaterial({  
        color: options.color || 0x888888,  
        transparent: true,  
        opacity: this.stateManager.state.edgeOpacity  
    });  
  
    const mesh = new THREE.Mesh(geometry, material);  
  
    // UserData für Raycast und Animation  
    mesh.userData = {  
        type: 'edge',  
        curve: curve,  
        sourceId: options.start,  
        targetId: options.end,  
        color: options.color,  
        index: options.index,  
        totalEdges: options.totalEdges  
    };  
  
    return mesh;  
}
```

Dynamische Updates

```
updatePositions(newStartPos: THREE.Vector3, newEndPos: THREE.Vector3) {  
    // Neue Kurve berechnen  
    const { curve, controlPoint } = this.calculateCurve(newStartPos, newEndPos);  
  
    // Bestehende Geometrie aktualisieren  
    const newGeometry = new THREE.TubeGeometry(  
        curve,  
        this.tubularSegments,  
        this.thickness,  
        this.radialSegments,  
        false  
    );
```

```
// Alte Geometrie entsorgen
this.mesh.geometry.dispose();
this.mesh.geometry = newGeometry;

// UserData aktualisieren
this.mesh.userData.curve = curve;
}
```

Animation-Integration

Vertex-Colors für Animationen

```
setupVertexColors(geometry: THREE.TubeGeometry, baseColor: THREE.Color) {
    const positionCount = geometry.attributes.position.count;
    const colors = new Float32Array(positionCount * 3);

    for (let i = 0; i < positionCount; i++) {
        colors[i * 3] = baseColor.r;
        colors[i * 3 + 1] = baseColor.g;
        colors[i * 3 + 2] = baseColor.b;
    }

    geometry.setAttribute('color', new THREE.BufferAttribute(colors, 3));
}
```

Pulse-Animation Update

```
animatePulse(time: number) {
    const intensity = 0.5 + 0.5 * Math.sin(time * Math.PI * 2 * this.pulseSpeed);

    this.animatedEdges.forEach(({ mesh, baseColor }) => {
        const colors = mesh.geometry.attributes.color;
        const positionCount = colors.count;

        for (let i = 0; i < positionCount; i++) {
            const color = baseColor.clone().lerp(new THREE.Color(1, 1, 1),
intensity * 0.4);
            colors.setXYZ(i, color.r, color.g, color.b);
        }

        colors.needsUpdate = true;
    });
}
```

Performance-Betrachtungen

Mesh-Wiederverwendung

```
// Cache für häufig verwendete Geometrien
private tubeGeometryCache = new Map<string, THREE.TubeGeometry>();

getCachedGeometry(key: string, curve: THREE.Curve<THREE.Vector3>):
THREE.TubeGeometry {
    if (!this.tubeGeometryCache.has(key)) {
        this.tubeGeometryCache.set(key, new THREE.TubeGeometry(curve, 20, 0.1, 8,
false));
    }
    return this.tubeGeometryCache.get(key)!.clone();
}
```

Dispose-Strategie

```
dispose() {
    this.edges.forEach(edge => {
        edge.mesh.geometry.dispose();
        (edge.mesh.material as THREE.Material).dispose();
        this.scene.remove(edge.mesh);
    });
    this.edges = [];

    this.tubeGeometryCache.forEach(geo => geo.dispose());
    this.tubeGeometryCache.clear();
}
```

Statistiken

Messung	Geringe Last	Mittlere Last	Hohe Last
Nodes	< 100	100-1000	> 1000
Edges	< 200	200-2000	> 2000
Geometrie-Speicher	< 10 MB	10-50 MB	> 50 MB
Draw Calls	< 10	10-50	> 50
FPS (Ziel: 60)	60	45-60	< 45

Optimierungsempfehlungen

1. **InstancedMesh** für gleiche Geometrien
2. **LOD** (Level of Detail) für entfernte Objekte
3. **Frustum Culling** automatisch durch Three.js
4. **Geometry Merging** für statische Szenen
5. **Web Workers** für Layout-Berechnungen

12 Ideensammlung und Weiterentwicklung

Geplante Features

Kurzfristig (v0.98 - v1.0)

Performance-Verbesserungen

- LOD (Level of Detail) für Nodes basierend auf Kameraentfernung
- Occlusion Culling für verdeckte Objekte
- Progressive Rendering bei großen Graphen
- Instanced Edge Rendering

UI-Erweiterungen

- Suchfeld mit Auto-Complete
- Filter-Panel für Node-Typen
- Timeline-Slider für temporale Daten
- Minimap-Übersicht

Interaktion

- Verbesserte Touch-Unterstützung
- Gesten für 2-Finger-Rotation
- Verbesserte Multi-Selection
- Drag & Drop für Node-Neupositionierung

Mittelfristig (v1.x)

Datenintegration

- GraphQL-API-Anbindung
- Real-Time-Updates via WebSocket
- CSV/Excel-Import
- Datenbank-Konnektoren (Neo4j, ArangoDB)

Analyse-Features

- Cluster-Erkennung und -Visualisierung
- Zentralitäts-Metriken (Betweenness, PageRank)
- Community-Detection
- Zeitreihen-Analyse

Visualisierung

- 3D-Labels mit Collision-Avoidance
- Animierte Übergänge zwischen Layouts

- Mehrere Ansichten (Split-View)
- VR/AR-Unterstützung

Langfristig (v2.0+)

Kollaboration

- Multi-User-Editing
- Echtzeit-Synchronisation
- Kommentar-System
- Versions-Geschichte

Erweiterbarkeit

- Plugin-System
- Custom Shader Support
- API für externe Tools
- Scripting-Schnittstelle

Architektur-Verbesserungen

Event-System

Aktuell: Callback-basiert

```
// Aktuell
stateManager.subscribe((state) => { ... });
```

Geplant: EventEmitter-Pattern

```
// Geplant
eventBus.on('node:hover', (data) => { ... });
eventBus.on('edge:select', (data) => { ... });
eventBus.emit('layout:apply', { type: 'force' });
```

Undo/Redo-System

```
class CommandHistory {
    private history: Command[] = [];
    private index: number = -1;

    execute(command: Command) {
        command.execute();
        this.history = this.history.slice(0, this.index + 1);
        this.history.push(command);
        this.index++;
    }
}
```

```

    }

    undo() {
        if (this.index >= 0) {
            this.history[this.index].undo();
            this.index--;
        }
    }

    redo() {
        if (this.index < this.history.length - 1) {
            this.index++;
            this.history[this.index].execute();
        }
    }
}

```

State-Persistence

```

class StatePersistence {
    saveToLocalStorage(key: string, state: any) {
        localStorage.setItem(key, JSON.stringify(state));
    }

    loadFromLocalStorage(key: string): any {
        const data = localStorage.getItem(key);
        return data ? JSON.parse(data) : null;
    }

    saveViewState() {
        return {
            cameraPosition: camera.position.toArray(),
            cameraTarget: controls.target.toArray(),
            selectedNodes: [...stateManager.state.selectedObjects]
        };
    }
}

```

Neue Visualisierungen

Force-Fields

```

class ForceFieldVisualizer {
    visualize(nodes: EntityData[], settings: any) {
        // Vektorfeld basierend auf Kräften
        const grid = this.createGrid(settings.resolution);

        grid.forEach(point => {
            const force = this.calculateForceAt(point, nodes);

```

```
        this.drawArrow(point, force);
    });
}
```

Heat-Maps

```
class HeatmapOverlay {
    create(entities: EntityData[], property: string) {
        const values = entities.map(e => e[property]);
        const texture = this.generateHeatmapTexture(entities, values);

        const plane = new THREE.Mesh(
            new THREE.PlaneGeometry(100, 100),
            new THREE.MeshBasicMaterial({
                map: texture,
                transparent: true,
                opacity: 0.5
            })
        );
        return plane;
    }
}
```

Temporal Animations

```
class TimelinePlayer {
    private currentTime: number = 0;
    private timeRange: [number, number];

    play() {
        requestAnimationFrame(() => this.tick());
    }

    tick() {
        this.currentTime += this.speed;
        this.updateVisualization(this.currentTime);

        if (this.currentTime < this.timeRange[1]) {
            requestAnimationFrame(() => this.tick());
        }
    }

    updateVisualization(time: number) {
        // Filter Entities nach Zeit
        // Animiere Erscheinen/Verschwinden
    }
}
```

Integration-Ideen

KI-Assistenz

- Automatische Layout-Vorschläge basierend auf Graph-Struktur
- Anomalie-Erkennung in Netzwerken
- Natural Language Queries ("Zeige alle Verbindungen von Node A")

Export-Erweiterungen

- SVG-Export für Publikationen
- GLTF-Export für 3D-Programme
- Video-Export von Animationen
- PDF-Reports mit Statistiken

Datenquellen

- Wikipedia/Wikidata-Integration
- Social Media APIs (Twitter Graph, LinkedIn)
- Scientific Paper Networks (Semantic Scholar)
- Code Dependencies (npm, pip)

Community-Beiträge

Bereiche für Open-Source-Beiträge:

1. Neue Layout-Algorithmen

- Sugiyama Hierarchical
- Radial Layout
- Stress Majorization

2. Neue Geometrien

- Custom 3D-Modelle
- Sprites mit Bildern
- Text-Labels

3. Import-Formate

- GML, GEXF, DOT
- Cytoscape JSON
- Gephi-Projekte

4. Themes

- Light Mode
- High Contrast
- Custom Color Schemes

13 Kritik und Limitationen

Bekannte Einschränkungen

Performance

Limitation	Beschreibung	Auswirkung
Node-Anzahl	Performance sinkt ab ca. 2000 Nodes	FPS-Einbrüche
Edge-Anzahl	Viele Edges belasten Rendering	Ruckeln
Animation	Globale Animationen sind CPU-intensiv	Batteriebelastung
Layout	Force-Layouts sind $O(n^2)$	Lange Berechnungszeit

Skalierbarkeit

- **Große Graphen:** Ab 5000+ Nodes unpraktisch ohne Aggregation
- **Dichte Netzwerke:** Viele Edges führen zu visuellem Chaos
- **Echtzeit-Updates:** Keine Streaming-Unterstützung für kontinuierliche Daten

Browser-Kompatibilität

Browser	Status	Anmerkung
Chrome	Vollständig	Empfohlen
Firefox	Vollständig	Leicht langsamer bei Animationen
Safari	Eingeschränkt	WebGL-Unterschiede
Edge	Vollständig	-
Mobile Browser	Eingeschränkt	Touch-Unterstützung begrenzt

Technische Schulden

Code-Qualität

1. Inkonsistente Typisierung

- Einige Module verwenden noch `any`
- Legacy-Code mit weniger strikten Typen

2. Test-Abdeckung

- Keine automatisierten Tests implementiert
- Manuelle Tests für Regression

3. Dokumentation im Code

- Ungleichmäßige JSDoc-Kommentare
- Fehlende Interface-Dokumentation

Architektur

1. Zirkuläre Abhängigkeiten

- Manager-Klassen referenzieren sich teilweise gegenseitig
- Erschwert Modularisierung

2. God-Objects

- `App.ts` übernimmt zu viele Verantwortlichkeiten
- `UIManager` ist sehr groß

3. State-Management

- Kein immutables State-Pattern
- Schwer nachvollziehbare State-Änderungen

UX-Probleme

Navigation

- Keine Rückkehr-Funktion ("Zurück zur vorherigen Ansicht")
- Keine Lesezeichen für Kamera-Positionen
- Keine Pfad-Historie für Exploration

Feedback

- Fehlende Lade-Indikatoren bei langen Operationen
- Keine Fortschrittsanzeige bei Layout-Berechnungen
- Unklare Fehlermeldungen bei ungültigen Daten

Accessibility

- Keine Screenreader-Unterstützung
- Fehlende Keyboard-Only-Navigation
- Kein High-Contrast-Modus
- Keine Untertitel für UI-Elemente

Vergleich mit Alternativen

vs. D3.js

Aspekt	Nodges	D3.js
3D-Unterstützung	Nativ	Erweiterung nötig
Lernkurve	Mittel	Hoch
Flexibilität	Mittel	Sehr hoch

Aspekt	Nodges	D3.js
Performance (2D)	Mittel	Hoch

vs. Cytoscape.js

Aspekt	Nodges	Cytoscape.js
3D-Ansicht	Ja	Nein (nur 2.5D)
Layout-Algorithmen	7	15+
Plugin-Ökosystem	Keines	Umfangreich
Dokumentation	Entwicklung	Ausführlich

vs. Gephi

Aspekt	Nodges	Gephi
Browser-basiert	Ja	Nein (Desktop)
Graph-Größe	< 5K Nodes	> 100K Nodes
Analyse-Tools	Begrenzt	Umfangreich
Export-Optionen	Begrenzt	Umfangreich

Fehlende Features

Kritisch

1. **Undo/Redo** - Keine Möglichkeit, Aktionen rückgängig zu machen
2. **Suche** - Kein globales Suchfeld für Nodes
3. **Filter** - Keine Filter-Optionen nach Eigenschaften
4. **Speichern** - Keine lokale Session-Speicherung

Wünschenswert

1. **Multi-Graph** - Nur ein Graph gleichzeitig
2. **Subgraphs** - Keine Gruppierung/Clustering-Ansicht
3. **Annotations** - Keine Kommentar-Funktion
4. **Vergleich** - Kein Side-by-Side-Vergleich

Nice-to-Have

1. **Themes** - Kein Light-Mode
2. **Lokalisierung** - Nur Deutsch/Englisch-Mix
3. **Tutorials** - Keine interaktive Einführung
4. **Templates** - Keine Vorlagen für häufige Graphtypen

Sicherheitsaspekte

Bedenken

- **Keine Validierung** bei URL-Import (Cross-Site-Risiken)
- **Lokale Daten** im Browser (kein Schutz)
- **Keine Authentifizierung** für sensible Graphen

Empfehlungen

1. URLs vor dem Laden validieren
2. Sensible Daten nicht in Nodges laden
3. Bei Bedarf lokalen Server statt öffentlichem Hosting

Verbesserungsvorschläge

Kurzfristig

1. Implementierung von Error-Boundaries
2. Besseres Logging für Debugging
3. Validierung aller Benutzereingaben
4. Loading-States für async Operationen

Langfristig

1. Vollständige Test-Suite (Unit + Integration)
2. CI/CD-Pipeline für automatische Builds
3. Performance-Monitoring
4. User-Analytics (optional, privacy-respecting)

14 Use Cases und Szenarien

Wissenschaftliche Anwendungen

Neurologie - Nervensystem-Visualisierung

Szenario: Darstellung neuronaler Netzwerke

- **Nodes:** Neuronen, Hirnregionen, Synapsen
- **Edges:** Synaptische Verbindungen, Signalwege
- **Visual Mappings:**
 - Größe = Neuronenaktivität
 - Farbe = Neurotransmitter-Typ
 - Animation = Signalübertragung (Pulse)

Beispiel-Datenstruktur:

```
{
  "system": "Nervensystem",
  "data": {
    "entities": [
      { "id": "n1", "type": "neuron", "region": "cortex", "activity": 0.8 },
      { "id": "n2", "type": "neuron", "region": "hippocampus", "activity": 0.5 }
    ],
    "relationships": [
      { "source": "n1", "target": "n2", "type": "synapse", "strength": 0.7 }
    ]
  }
}
```

Biologie - Proteininteraktionen

Szenario: Protein-Protein-Interaktionsnetzwerke

- **Nodes:** Proteine
- **Edges:** Interaktionen (Bindung, Regulierung)
- **Layout:** Force-Directed für Cluster-Erkennung

Soziologie - Soziale Netzwerke

Szenario: Analyse von Gruppenstrukturen

- **Nodes:** Personen
- **Edges:** Beziehungen (Freundschaft, Kollegen, Familie)
- **Analyse:** Zentralität, Community-Detection

Technische Anwendungen

Software-Architektur

Szenario: Visualisierung von Code-Abhängigkeiten

- **Nodes:** Module, Klassen, Funktionen
- **Edges:** Import-Beziehungen, Aufrufe
- **Hierarchical Layout:** Zeigt Abhängigkeitsebenen

Beispiel-Nutzung:

```
{
  "system": "ProjectDependencies",
  "data": {
    "entities": [
      { "id": "App", "type": "module", "lines": 500 },
      { "id": "StateManager", "type": "module", "lines": 350 },
      { "id": "NodeManager", "type": "module", "lines": 260 }
    ],
    "relationships": [
      { "source": "App", "target": "StateManager", "type": "imports" },
      { "source": "App", "target": "NodeManager", "type": "imports" }
    ]
  },
  "visualMappings": {
    "defaultPresets": {
      "node": {
        "size": { "source": "lines", "function": "logarithmic", "range": [0.5, 2] }
      }
    }
  }
}
```

Netzwerk-Topologie

Szenario: IT-Infrastruktur visualisieren

- **Nodes:** Server, Router, Switches, Clients
- **Edges:** Netzwerkverbindungen
- **Farbe:** Status (online/offline/wartung)
- **Animation:** Datenfluss

Wissensgraphen

Szenario: Ontologien und Taxonomien

- **Nodes:** Konzepte, Entitäten
- **Edges:** is-a, part-of, related-to
- **Hierarchical Layout:** Zeigt Klassifikation

Bildungsanwendungen

Lernpfade

Szenario: Curriculum-Visualisierung

- **Nodes:** Kurse, Themen, Lernziele
- **Edges:** Voraussetzungen, Empfehlungen
- **Pfadfindung:** Optimaler Lernpfad

Geschichts-Netzwerke

Szenario: Historische Zusammenhänge

- **Nodes:** Personen, Ereignisse, Orte
- **Edges:** Beziehungen, Ursache-Wirkung
- **Timeline:** Temporale Visualisierung

Business-Anwendungen

Organisationsstrukturen

Szenario: Unternehmenshierarchien

- **Nodes:** Abteilungen, Teams, Mitarbeiter
- **Edges:** Berichtslinien, Kommunikation
- **Hierarchical Layout:** Organigramm

Supply Chain

Szenario: Lieferketten-Visualisierung

- **Nodes:** Lieferanten, Fabriken, Lager, Kunden
- **Edges:** Materialfluss
- **Animation:** Warenfluss
- **Farbe:** Engpässe (rot), Normal (grün)

Finanz-Netzwerke

Szenario: Transaktionsnetzwerke

- **Nodes:** Konten, Institutionen
- **Edges:** Transaktionen
- **Dicke:** Transaktionsvolumen
- **Analyse:** Verdächtige Muster

Kreative Anwendungen

Story-Mapping

Szenario: Narrative Strukturen

- **Nodes:** Charaktere, Orte, Ereignisse
- **Edges:** Interaktionen, Plotlines
- **Timeline:** Chronologie der Geschichte

Musik-Netzwerke

Szenario: Musiktheorie und Einflüsse

- **Nodes:** Künstler, Genres, Alben
- **Edges:** Einflüsse, Kollaborationen
- **Farbe:** Genre-Zuordnung

Schritt-für-Schritt Anleitung

Use Case: Projektnetzwerk erstellen

1. Daten vorbereiten

```
{
  "system": "MeinProjekt",
  "metadata": { "version": "1.0" },
  "data": {
    "entities": [...],
    "relationships": [...]
  }
}
```

2. Datei laden

- Drag & Drop auf Nodes
- Oder über Files-Panel auswählen

3. Layout anwenden

- Force-Directed für organische Anordnung
- Hierarchical für Strukturen

4. Visualisierung anpassen

- Visual Mapping Panel öffnen
- Eigenschaften zu visuellen Attributen mappen

5. Exploration

- Hover für Details
- Klick für Selektion
- Pfadfindung für Zusammenhänge

6. Export

- Screenshot für Dokumentation

- JSON für Weiterverwendung

Best Practices

Datenaufbereitung

1. **Eindeutige IDs** für alle Entities
2. **Konsistente Typen** für Gruppierung
3. **Normalisierte Werte** für Mappings (0-1 oder bekannte Ranges)
4. **Sinnvolle Labels** für Anzeige

Layout-Wahl

Graph-Typ	Empfohlenes Layout
Hierarchie	Hierarchical, Tree
Cluster	Force-Directed
Zeitlich	Grid nach Zeit sortiert
Vergleich	Circular
Dicht verbunden	Fruchterman-Reingold

Performance-Tipps

1. Bei >500 Nodes: Web Worker aktivieren
2. Bei >1000 Edges: Animations deaktivieren
3. Komplexe Graphen: Aggregation vor dem Laden
4. Mobile: Reduzierte Geometrien verwenden