VR-Programmierung

OpenInventor & Amira

Technische Fakultät — Universität Bielefeld

Björn Sommer & Sebastian Janowski — 7. Juni 2005

Inhaltsverzeichnis

1 Einführung **3**
- 1.1 Bedeutung . . . 3
- 1.2 Basis . . . 3
- 1.3 Historie . . . 3
  - 1.3.1 Open Inventor . . . 3
  - 1.3.2 Coin 3D . . . 4

2 Struktur **5**
- 2.1 Bestandteile . . . 5
- 2.2 Objekt-Funktionen . . . 5
- 2.3 Architektur . . . 5
- 2.4 SceneDatabase und SceneGraph . . . 6
  - 2.4.1 Node Kits . . . 7
  - 2.4.2 Manipulators . . . 7
  - 2.4.3 SceneGraph-Beispiel . . . 8
- 2.5 Implementierung von OpenGL . . . 9

3 Component Library **11**
- 3.1 Bestandteile . . . 11
- 3.2 Funktionsweise . . . 11
- 3.3 Components . . . 11
- 3.4 ClassTree . . . 12
- 3.5 Kamera . . . 13
- 3.6 Licht . . . 13
- 3.7 Text . . . 15
  - 3.7.1 Einführung . . . 15
  - 3.7.2 2D-Textdarstellung . . . 15
  - 3.7.3 3D-Textdarstellung . . . 16
- 3.8 Formen . . . 16
  - 3.8.1 Erstellen von simplen geometrischen Objekten und Polygonen . . . 16
  - 3.8.2 Kurven . . . 17
  - 3.8.3 Flächen . . . 19
- 3.9 Oberflächenstruktur . . . 19
- 3.10 Aktionen . . . 20
- 3.11 Events und Selection . . . 21
  - 3.11.1 Events . . . 21
  - 3.11.2 Selections . . . 22
- 3.12 Sensoren . . . 23

3.13 Engines 23
3.14 Dragger and Manipulatoren 24
3.15 Dateiformat .iv 25

4 Amira **26**
4.1 Einsatzgebiete 26
4.2 Visualisierungen 26
4.3 Features 26

5 Zukunft **27**

Zusammenfassung

Diese Ausarbeitung entstand im Rahmen des Seminars *VR-Programmierung in der Systembiologie*. Sie behandelt das *Open Inventor Toolkit*, seine Grundidee, seine Entstehungsgeschichte, seine Struktur und seine einzelnen Elemente.
Abschließend wird darüber hinaus *Amira* vorgestellt, eine auf *Open Inventor* basierende Entwicklungsumgebung, die insbesondere den Anforderungen der Systembiologie gerecht wird.

1 Einführung

1.1 Bedeutung

Open Inventor ist ein objektorientiertes 3D-Toolkit für Grafik-Programmierer und Anwendungsentwickler. Dabei handelt es sich nicht um etwa eine Anwendung, sondern um eine Bibliothek von Objekten und Methoden um interaktive 3D Grafik-Anwendungen zu erstellen.

1.2 Basis

Die Basis von *Open Inventor* ist in C++ und teilweise C geschrieben und basiert auf OpenGL. Wie sich aufgrund dieser Konstellation schon vermuten lässt, wurde die Umgebung zunächst unter und für Unix entwickelt.
Dabei wurde bei der Entwicklung von Anfang an darauf Wert gelegt, die hardwareseitige Grafikbeschleunigung auszunutzen.
Ziel war es, mit dieser Bibliothek dem Entwickler eine Möglichkeit zu bieten, bei möglichst geringen Programmieraufwand mächtige Anwendungen zu schaffen, die den o.g. Kriterien gerecht werden.
Auch für die Kommunikation mit anderen, *Open Inventor* fremden Programmen wurde gesorgt: Um mit externen Anwendungen Daten austauschen zu können, wurde ein 3D- interchange file format (.iv) eingeführt, mit welchem der Nutzer 3D-Szenarien zwischen unterschiedlichen Programmen austauschen kann.

1.3 Historie

1.3.1 Open Inventor

Die Wurzeln von *Open Inventor* reichen bis in das Jahr 1988 zurück, als Wei Yen und Rick Carey das *IRIS Inventor* Projekt bei *Silicon Graphics (SGI)* initiierten.

Ziel war es, das Erstellen von 3D-Objekten mit OpenGL zu vereinfachen. Nachdem das Toolkit für DrittUnternehmen lizensierbar gemacht wurde, änderte sich der Name in *Open Inventor*.
Etwa zur gleichen Zeit ereignete sich eine Abspaltung einer Entwicklergruppe, die fortan *Open Performer* entwickelten. Ihnen war die Performance von *Open Inventor* zu gering und sie begannen eine anderen Variante eines 3D-Toolkits zu entwickeln, die ebenfalls heute noch von vielen Entwicklern eingesetzt wird.
Seit dem Jahr 2000 hat *SGI Open Inventor* unter der Open Source License freigegeben, woraufhin bald der erste Release von *Systems in Motions Coin 3D* erfolgte.

1.3.2 Coin 3D

Das erste Grundgerüst von *Coin 3D*, welches zunächst noch nicht auf *Open Inventor* basierte, entstand 1995 bei *Systems in Motion*. Aber mit der Zunahme der Ansprüche an *Coin 3D* seitens der Benutzer begannen die Entwickler, *Coin 3D* von Grund auf neu aufzubauen. Dabei wurde diesmal *Open Inventor* als Grundlage gewählt.
Inzwischen gibt es von *Systems in Motion* verschiedene Lizenzmodelle, unter denen *Coin 3D* verwendet werden kann. Dabei bietet die so genannte Free Edition für Privatanwender eine frei nutzbare Alternative zu den kommerziellen *SGI's* und *TGS' Open Inventor* Bibiotheken.
Darüber hinaus ist die neueste Version fast vollständig kompatibel zur *Open Inventor* 2.1 API.

2 Struktur

2.1 Bestandteile

Das *Open Inventor* Toolkit besteht zunächst aus drei Grundelementen:

- Die database primitives, z.B. shape, property, group, engine objects. Das sind z.B. Kugeln, Quadrate, etc..
- Die Manipulators, z.B. handlebox, trackball. Dabei handelt es sich um Steuerelemente, die on-the-run die Manipulierung der Objekte zulassen, so z.B. das Bewegen einer Kugel in verschiedene Richtungen.
- Die components, z.B. material editor, directional light editor, examiner viewer. Mit ihrer Hilfe lassen sich die einzelnen Objekte überhaupt erst darstellen.

2.2 Objekt-Funktionen

Die verwendeten Objekte bieten die unterschiedlichsten Funktionen:

- Die Auswahl des Objekts.
- Highlighting um z.B. das ausgewählte Objekt hervorzuheben.
- Manipulierung bzw. Bewegen der Objekte.
- Berechnung der Abgrenzungen um z.B. Kollisionsabfrage durchführen zu können.
- Laden eines Objektes im .iv-Format.
- Abspeichern eines Objektes im .iv-Format.
- Die Suche nach einem Objekt oder eines seiner Eigenschaften im SceneGraph.

2.3 Architektur

Wie oben bereits erwähnt, bilden OpenGL und Unix die Basis für *Open Inventor*.
Auf diese baut der komplette *Open Inventor* 3D Toolkit auf (sh. Abb. 1). Dieser wiederum besteht aus folgenden drei Elementen:

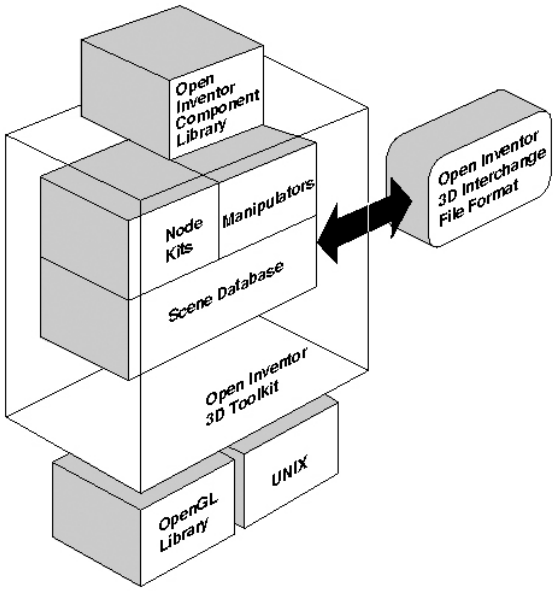

Abbildung 1: Die Architektur des *Open Inventor* Toolkits [1]

- Die Scene Database, welche Informationen zu allen Objekten wie z.B. Umriß, Größe, Texturen, Farben oder Position enthält (sh. Kap. 2.4).
- Die Node Kits, welche komfortable Mechanismen zum Gruppieren von nodes bieten und das Grundgerüst für den SceneGraph bilden (sh. Kap. 2.4).
- Die manipulators, welche Komponenten zum Verändern von Objekten bieten (sh. Kap. 2.1).

Auf die einzelnen Elemente, die das Grundgerüst für die *Open Inventor* Component Library bilden, wird nun im Folgenden genauer eingegangen.

2.4 SceneDatabase und SceneGraph

Die Basiseinheit für die Erzeugung von SceneGraphs ist ein node. D.h., ein SceneGraph besteht aus mindestens einem node, dem sogenannten root. Alle folgenden Knoten sind Kinder dieses Knotens, die wiederum Kinder haben können. Auf diese Weise entsteht ein Baum bzw. ein Graph.
Dabei werden alle 3D-Formen, Attribute, Kameras und Lichtquellen als nodes repräsentiert. Diese nodes verweisen wiederum auf die in der SceneDatabase enthaltenen Objekte. D.h. also, die im SceneGraph enthaltenen nodes sind

Verweise auf die in der SceneDatabase enthaltenen Objekte.
Die Grundmodelle dieser nodes, die sog. database primitives, sind folgende:

- shape nodes (z.B. Kugel)
- property nodes (z.B. Material)
- group nodes (z.B. seperator)
- engines (z.B. Rotation)
- sensors (z.B. Ereignisregistrierung)

2.4.1 Node Kits

Die Node Kits ermöglichen den Aufbau einer strukturierten, konsistenten Datenbank aus den nodes. Sie enthalten Regeln, welche die Art und die Platzierung der nodes über ein Template regulieren. Der Vorteil liegt darin, dass sich der Anwender nicht mehr mit dem Verwalten der nodes auseinandersetzen muss. Der SceneGraph wird im Hintergrund automatisch erzeugt, während sich der Anwender nur auf das Programmieren seiner Szene konzentriert. Somit ermöglichen Node Kits das Erschaffen von anwendungsorientierten Objekten und Semantiken.

2.4.2 Manipulators

Bei manipulators handelt es sich um nodes, die vom Benutzer on-the-run verändert werden können und Auswirkungen auf die mit ihnen verknüpften Objekte haben. So transformiert z.B. eine Bounding Box ein Objekt per Mouse-Drag (sh. auch Kap. 2.1, 3.14).

2.4.3 SceneGraph-Beispiel

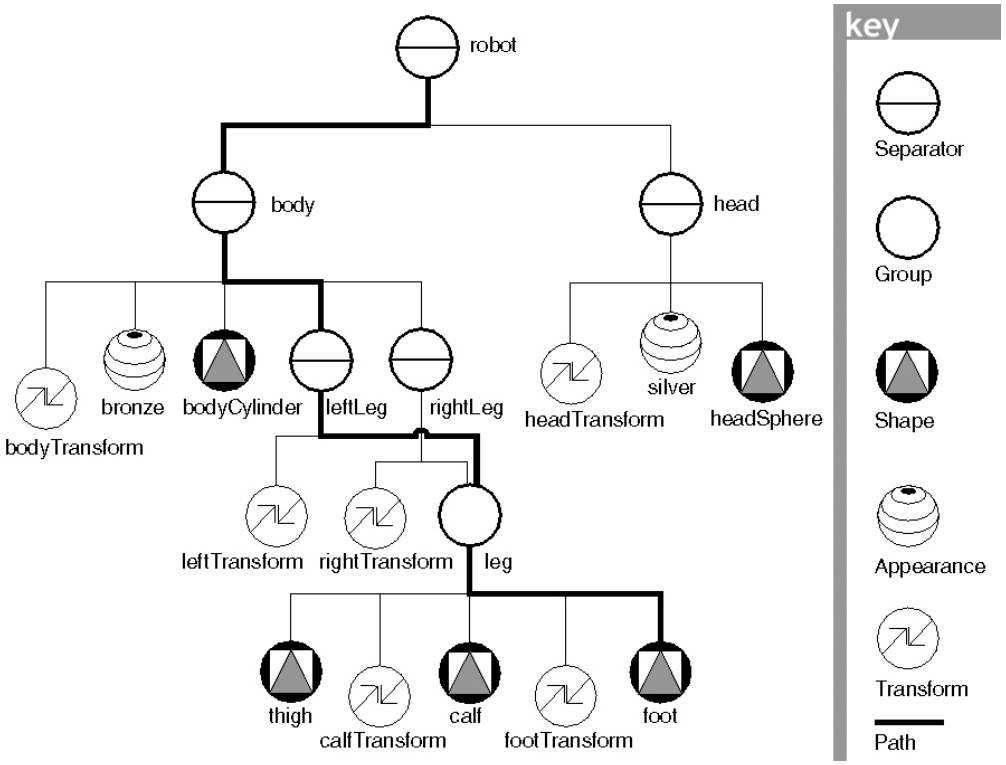

Abbildung 2: Der SceneGraph-Pfad des linken Robot-Fußes [1]

Das nun folgende Beispiel behandelt den Pfad, der den linken Fuß des Robots repräsentiert (sh. Abb. 3).
Der root des SceneGraphs ist der oberste node „robot“, für welchen das Seperator-Objekt benutzt wird, welches die unterschiedlichsten Eigenschaften, Funktionalitäten und Formen miteinander gruppieren kann. Dieser Umstand wird im weiteren Verlauf des Pfades ersichtlich werden.
Die folgenden nodes sind zum Einen der „body“, zum Anderen der „head“, beides ebenfalls Seperators. Im Folgenden wird der rechte Pfad vernachlässigt werden und wir folden dem linken Pfad des „bodies“.
Auf „body“ folgen nun der „bodyCylinder“, das eigentliche formgebende, Shape-Objekt, zu welchem zusätzlich die Appearance-Eigenschaft, also die Farbe „bronze“ gehört, und die Transformation „bodyTransform“, welche die Positionierung im Raum festlegt.
Außer diesen nodes befinden sich zwei weitere Seperators auf dieser Ebene des SceneGraphs: „left-“ und „rightLeg“.
Verfolgen wir den Pfad nun weiter, stellt sich heraus, dass beide nodes von

dem Group-node „leg“ abgeleitet sind. Dieses Vorgehen wird als shared instancing bezeichnet und vermeidet Redundanz; typisch für die Objektorientierte Programmierung. Um zu vermeiden, dass sowohl „left-“ als auch „rightLeg“ sich auf der selben Position befinden, gibt es zwei unterschiedliche Transform-nodes, welche die beiden Objekte an der korrekten Stelle platzieren.
Folgt man nun dem Pfad in die letzte Ebene, wird ersichtlich, aus welchen Shape- und Transform-nodes der Group-node „leg“ zusammengesetzt ist.

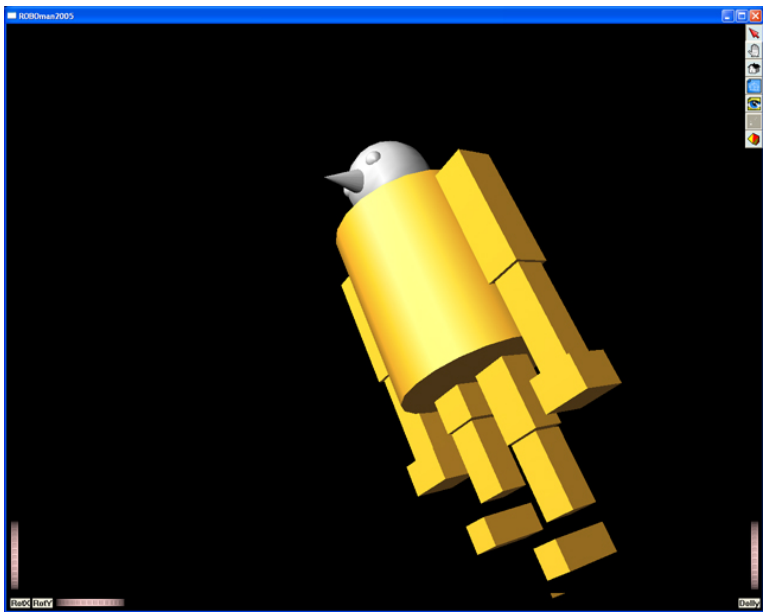

Abbildung 3: Der Robot

2.5 Implementierung von OpenGL

Wie bereits erwähnt, basiert *Open Inventor* auf OpenGL, welches zum Rendern benutzt wird.
Der große Unterschied zum direkten Einsatz von OpenGL liegt in der Initialisierung des Rendervorganges: Anwender von OpenGL müssen den Rendervorgang direkt anstoßen. Er ist damit explizit bzw. framebasiert.
In *Open Inventor* hingegen ist das Rendern Bestandteil der Objekte, die eine seperate Renderfunktion haben, die während des Programm-Ablaufs automatisch ausgeführt werden. Der Nachteil ist allerdings, dass kein direkter Zugriff auf den Frame-Buffer möglich ist.

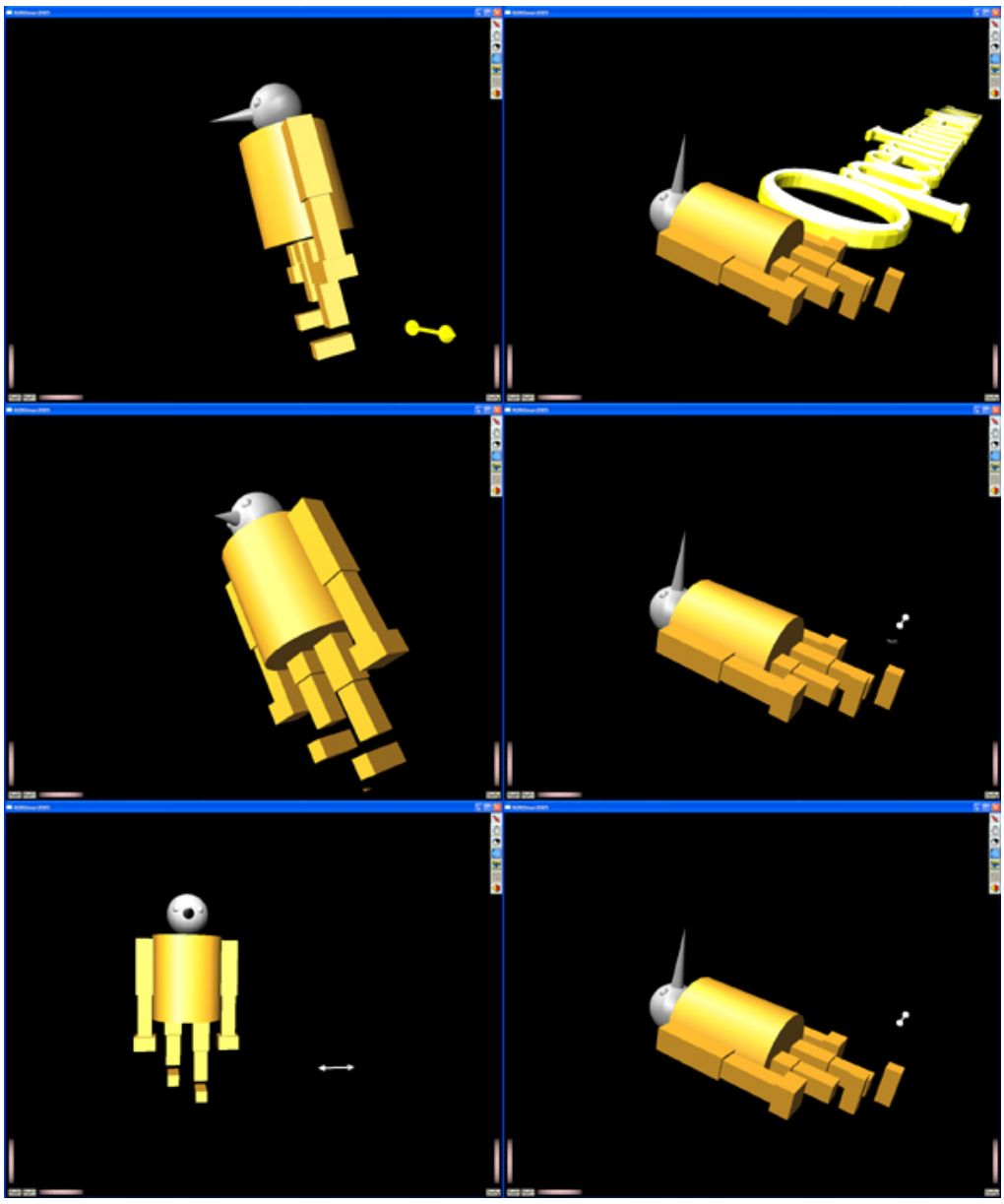

Abbildung 4: Der Robot in 6 Ansichten (Rechte Seite: Mit dem Pfeil-Manipulator kann die Größe der Nase reguliert werden, Linke Seite: 3D-Text, 2D-Text, kein Text)

3 Component Library

3.1 Bestandteile

Die Component Library stellt die eigentliche *Open Inventor* Bibliothek da, die, wie der Name schon sagt, alle Komponenten bietet, um eine 3D-Szene aufzubauen.
Ursprünglich für Unix entwickelt, integriert sie das X window system.
Inzwischen unterstützt sie unterschiedliche Window-Systeme, so natürlich auch das gängige Microsoft Windows System.
Zu den Schnittstellen zu diesem Window-Systemen gehören:

- viewers, in denen die 3D-Szene dargestellt wird, also das Grafikfenster.
- render area (window), das Fenster, das sich auf dem Desktop bewegen lässt und die viewers enhält.
- main loop, die den eigentlichen Ablauf eines Programmes und weitere, das Initialisieren vereinfachende Methoden enthält.
- event translator utility, welches die Ereignisse weiterverarbeitet (sh. Kap. 3.2).
- Editoren, mit denen Objekte on-the-run manipuliert werden können (sh. Kap. 3.3).

3.2 Funktionsweise

Die Funktionsweise der render area ist die folgende: Zunächst empfängt sie ein X event, also ein Ereignis, welches vom X Window-System generiert wurde. So dann wird es in ein Inventor Event übersetzt, an *Open Inventor* weitergeleitet und anschließend an die Unterobjekte wie z.B. manipulators weitergeleitet und von diesen ausgewertet (sh. Kap. 3.11.1).

3.3 Components

Die Component Library enthält auch viewers und editors, die sowohl eine render area als auch ein user interface enthalten. Ein gutes Beispiel hierfür sind die fly- oder examiner-viewer, die es ermöglichen, sich frei durch den Raum um ein Objekt herum zu bewegen und zu Navigieren. So bieten sie auch Funktionen wie das Zurücksetzen der Ansicht oder das Heranzoomen eines bestimmten, auf der Oberfläche eines Objektes ausgewählten Punktes.

3.4 ClassTree

Ein Blick auf den Ausschnitt des *Open Inventor* ClassTrees zeigt, wie komplex die einzelnen Komponenten-Klassen untereinander verschachtelt sind. So hat z.B. der SoSeperator, der den root der meisten SceneGraphs repräsentiert, vier Oberklassen, von denen er abgeleitet ist: SoGroup, SoNode, SoFieldContatiner und das Grundobjekt, die SoBase.

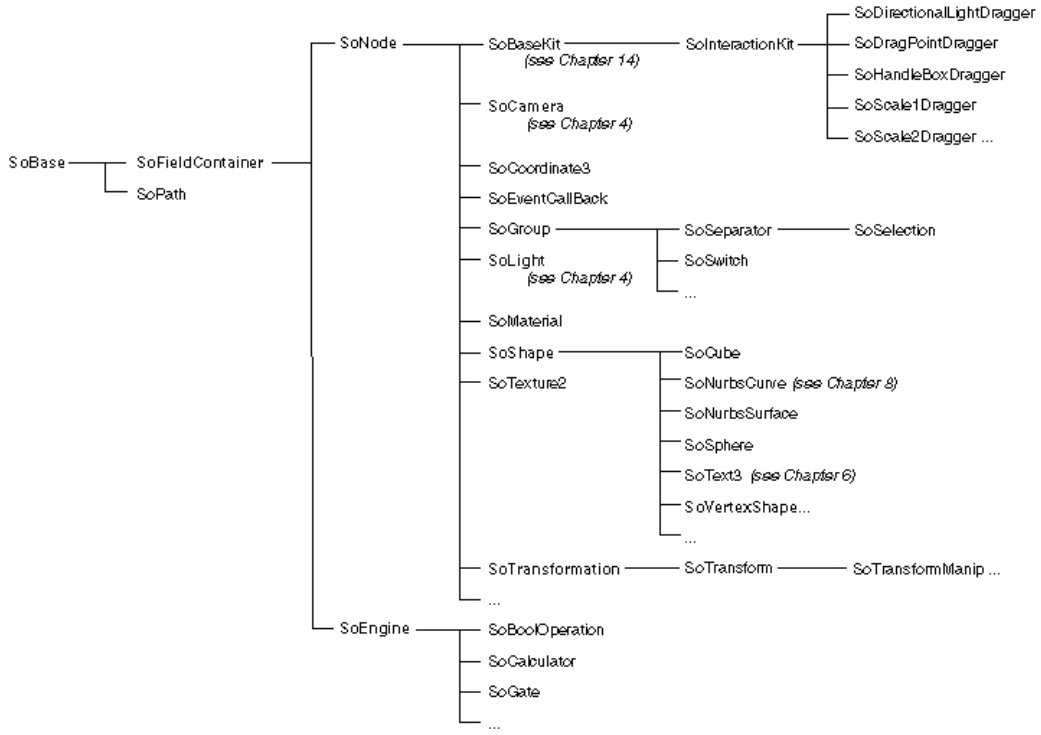

Abbildung 5: Ein ClassTree [1]

3.5 Kamera

Die Kamera ist das wichtigste Element bei der 3D-Programmierung. Sie generiert aus dem Szenengraphen ein sichtbares Bild für den Benutzer. Ohne aktive Kamera werden die generierten Elemente nie sichtbar sein. Es darf immer nur eine Kamera aktiv sein, aber durch einen „Blinker“ kann zwischen verschiedenen Kameras im Szenengraphen gewechselt werden.

Beim „rendering“ wird in Abhängigkeit von den Kameraeinstellungen die Kamera im Raum positioniert. Alle Elemente die im Kamerasichtfeld liegen, werden dargestellt. Elemente außerhalb dieses Sichtfelds werden nicht dargestellt. Der Vorteil liegt in der *Performance* der Generierung. Wenn alle Elemente im Sichtfeld generiert wurden, erzeugt die Kamera aus der 3D-Darstellung ein 2D-Bild, welches auf der graphischen Oberfläche dargestellt wird.

Open Inventor stellt dem Benutzer drei verschiedene Kameratypen zur Verfügung.

- **SoCamera:** Dieser Typ von Kamera ist die Standartkamera, die keine Besonderheiten aufweist. Lediglich das Sichtfeld der Kamera kann verändert werden.

- **SoPerspectiveCamera:** Die SoPerspektiveCamera simuliert das menschliche Auge. Dementsprechend erscheinen Objekte je nach Distanz größer oder kleiner. Die möglichen Einstellungen zur Veränderung des Sichtfelds sind der Abbildung 6 a) zu entnehmen.

- **SoOrthographicCamera:** Bei der SoOrthographicCamera wird eine parallele Projektion des Objektes generiert. Eine Distanzfunktion ist nicht vorhanden. Das Objekt bleibt in Form und Größe konstant. Die möglichen Einstellungen zur Veränderung des Sichtfelds sind der Abbildung 6 b) zu entnehmen.

Für nähere Informationen und Programmbeispiele zum Thema Kamera wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Kamera.

3.6 Licht

Das Licht ist neben der Kamera eines der wichtigsten Objekte beim 3D-Programmieren. Wenn kein Licht vorhanden ist, sind die Objekte für den Benutzer nicht sichtbar. Dementsprechend muss mindestens eine Lichtquelle dem Szenengraphen zugeordnet sein. Es ist möglich, mehrere Lichtquellen zu

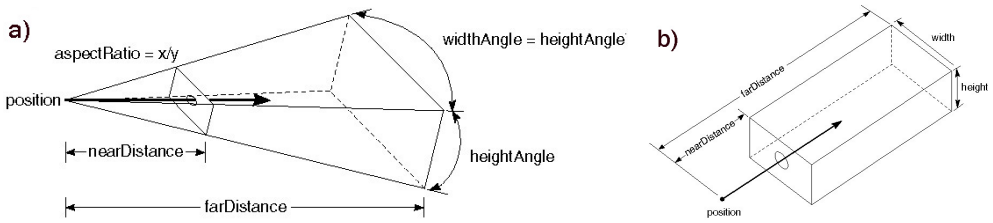

Abbildung 6: Bei diesen zwei Abbildungen zur (a) SoPerspectiveCamera und zur (b) SoOrthographicCamera sind die Parameter dargestellt, die bei den zwei Kameratypen verändert werden können, um das Sichtfeld nach eigenen Vorstellungen zu verändern. [1]

definieren. Jedoch ist die Anzahl von aktiven Lichtquellen durch die OpenGL-Programmierung beschränkt. Zudem ist anzumerken, dass mit der Anzahl der Lichtquellen die *Performance* beeinträchtigt wird. *Open Inventor* muss für jedes Objekt den Lichteinfall, die Reflektion, wie auch die Lichteinflüsse untereinander berechnen.

Open Inventor bietetet neben der Standartklasse SoLight drei weitere Lichttypen an.

- **PointLight:** Beim PointLight wird eine Koordinate für den Ausgangspunkt festgelegt, von dem an in alle Richtungen in gleicher Intensität gestrahlt wird (sh. Abb. 7 b).
- **DirectionalLight:** Beim DirectionalLight wird ein Lichtstrahl erzeugt. Alle Objekte, die in diesem Lichtstrahl liegen, werden mit gleicher Lichtintensität bestrahlt (sh. Abb. 7 a).

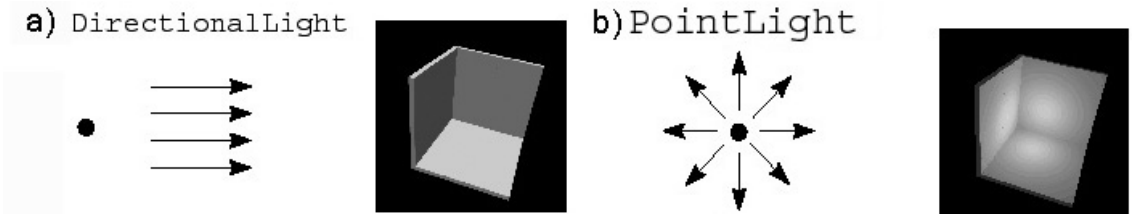

Abbildung 7: Diese zwei Abbildungen vom (a) DirectionalLight und vom (b) Pointlight zeigen den Lichteinfall im Raum. [1]

- **SpotLight:** Diese Lichtart simuliert einen Scheinwerfer, bei dem der Strahl einem Kegel gleicht. Im Innersten des Kegels ist die Lichtintensität am höchsten. Je weiter nach außen man sich in diesem Lichtkegel bewegt, um so schwächer wird die Lichtintensität. Die Richtung und

der Auffallwinkel des Lichtstrahls können beliebig verändert werden (sh. Abb. 8).

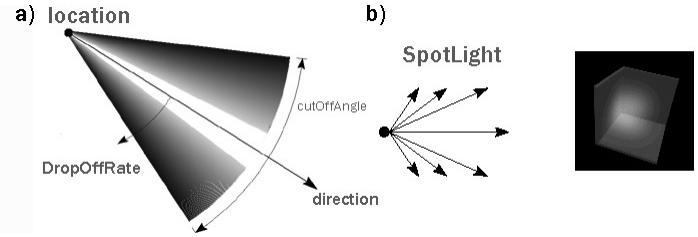

Abbildung 8: Bei (a) wird der SpotLight-Lichtkegel mit all seinen veränderbaren Parametern dargestellt. Bei (b) wird der Lichteinfall vom SpotLight im Raum dargestellt. [1]

Für nähere Informationen und Programmbeispiele zum Thema Licht wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Licht.

3.7 Text

3.7.1 Einführung

Zur Darstellung von Text gibt es zwei verschiedene Arten. Entweder es wird die 2D- oder die 3D-Textdarstellung benutzt.

Beide Textarten können durch die SoFont-Klasse in Form, Größe oder Schriftart verändert werden. Folgend zeigt ein kleines Beispiel die nötigen Aufrufe zum Verändern der Schriftart und der Größe.

```
SoFont *schriftart = new SoFont;
schriftart->name.setValue('TimesNewRoman');
schriftart->size.setValue(140);
root->addChild(schriftart);
```

Um an mehr Informationen zu der SoFont-Klasse zu gelangen, wird hier auf das *Manual* von *Open Inventor* [2] verwiesen.

3.7.2 2D-Textdarstellung

Der 2D-Text bleibt immer parallel zur Kamera stehen, d.h. der Text dreht sich bei Kamerabewegungen nicht mit. Außerdem ist eine Distanzfunktion für diese Textart nicht gegeben. Die Größe und die Textausrichtungen bleiben immer konstant. Vorteilhaft ist die schnelle Generierung und die immer

lesbare Darstellung. Folgendes Beispiel zeigt, wie 2D-Texte initialisiert werden:

```
SoText2 *2D_text = new SoText2;
2D_text->string = 'OpenInventor';
root->addChild(2D_text);
```

Für weitere Programmbeispiele wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Textdarstellung.

3.7.3 3D-Textdarstellung

Der 3D-Text dreht sich im Gegensatz zum 2D-Text mit der Kamera mit. Diese Textart bietet Disatanzfunktionen und Transformationen an. Transformationen können an der Höhe, Breite, Tiefe, Gestalt, Oberflächenstruktur etc. vorgenommen werden.

Für Programmbeispiele und nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Textdarstellung.

3.8 Formen

3.8.1 Erstellen von simplen geometrischen Objekten und Polygonen

Open Inventor stellt eine Reihe von simplen Formen zur Verfügung. Darunter stehen Rechtecke, Kugeln, Kegel und Zylinder. Diese Formen sind von der SoShape-Klasse abgeleitet und können anhand ihrere Parameter in Form und Größe verändert werden.

Jedoch reicht es bei der 3D-Programmierung nicht aus, nur mit simplen Formen zu arbeiten. Viele Darstellungen von Objekten können mit simplen Formen nicht erstellt werden. Für dieses Problem stellt *Open Inventor* Polygone zur Verfügung, die anhand von Koordinaten generiert werden.

Open Inventor bietet vier Möglichkeiten zur Darstellung von Polygonen. Bei allen vier Möglichkeiten wird zumindest ein Set von Koordinaten benötigt, welche den x,y und z-Wert im Raum bestimmen.

- **SoFaceSet:** Das SoFaceSet erstellt Polygone anhand von gegebenen Koordinaten. Hierbei spielt die Reihenfolge der Koordinaten eine wich-

tige Rolle, da diese für die Verbindungen der einzelnen Koordinaten ausschlaggebend sind.

- **IndexedFaceSet:** Das IndexedFaceSet ist vom Prinzip her aufgebaut wie das SoFaceSet. Der einzige Unterschied liegt an der Reihenfolge der Koordinaten. Mit dieser Klasse können die Koordinaten in beliebiger Reihenfolge vorliegen, da sie indexiert werden.

- **TriangleStripSet:** Mit Hilfe der TriangleStripSet-Klasse lassen sich Polygone am schnellsten erzeugen. Die Klasse generiert aus den Eckpunkten Dreiecke, die folgend alle miteinader verbunden werden.

- **QuadMesh:** Die Quad Mesh-Klasse ermöglicht die schnelle und einfache Darstellung von Rechtecken. Dabei erzeugt die Klasse aus den Eckpunkten die Form.

Bei der Darstellung von Polygonen müssen vom Benutzer die Verbindungen der einzelnen Punkte und die Oberflächeneigenschaften definiert werden. Dieses geschieht mit den Klassen SoNormal, SoNormalBinding und SoMaterial.

Für Programmbeispiele und nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Formen.

3.8.2 Kurven

Für die Darstellung von Kurven benutzt *Open Inventor NURBs (Non Uniform Rational B-Splines). Nurbs* sind Kurven im 3D-Raum, die stückweise durch rationale Funktionen dargestellt werden. Jede Kurve in einem 3D-Raum wird durch die Funktionen x = f (u), y = g (u) und z = h (u) dargestellt. Die Funktionen definieren für jede Achse die Position der Kurve im Raum.

Da wahrscheinlich die meisten Benutzer ihre Kurve nicht anhand von Funktionen darstellen können, bietet *Open Inventor* die Möglichkeit, die Kurve in ihrem Verlauf durch ControlPoints zu beeinflussen. ControlPoints verhalten sich wie Magnete, welche die Kurve entweder anziehen oder abstossen. Die ControlPoints haben dabei einen bestimmten Wirkungsgrad, der durch durch den Benutzer beliebig eingestellt werden kann (sh. Abb. 9 a).

Jedoch kann es vorkommen, dass ControlPoints in ihren Einflüssen überschneiden. Um diese Fälle abfangen zu können, muss der Benutzer KnotSequences definieren, um ein definiertes Verhalten zu erlangen. KnotSequences

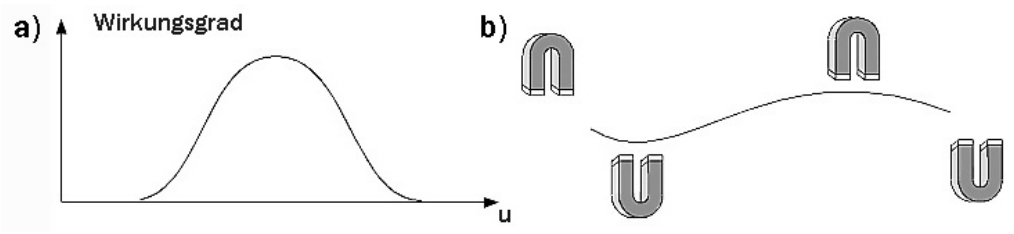

Abbildung 9: Diese Abbildung zeigt bei (a) den Wirkungsgrad mit gegebenem Parameter u von einem ControlPoint und bei (b) den Verlauf einer Kurve mit mehreren aktiven ControlPoints. [1]

benutzen die Klassen SoNurbsCurve und SoIndexedNurbsCurve. Diese berechnen mit Hilfe von mathematischen Funktionen die einzelnen Wirkungsgrade zu einander und erstellen für ein Intervall einen gewissen Kurveneinfluss (sh. Abb. 10).

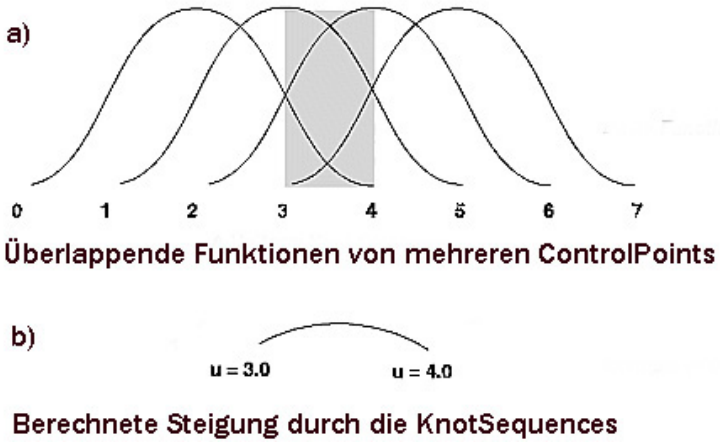

Abbildung 10: Die Abbildung (a) zeigt mehrere ControlPoints, die in ihren Einflüssen überlappen und (b) zeigt den errechneten Einfluss durch die KnotSequences auf die Kurve. [1]

Open Inventor bietet dem Benutzer auch die Möglichkeit, verschiedene Kurven miteinander zu verbinden. Somit kann der Benutzer verschiedene Kurven, die eventuell schon in seiner Datenbank vorhanden sind, miteinander verbinden. Damit dieser Vorgang gelingt, werden Breakpoints an den Anfang und an das Ende der Kurve gelegt. Diese Breakpoints können dann durch verschiedene Wege miteinander verbunden werden. Hierbei muss der Benutzer durch geforderte Parameter selbst definieren, wie diese miteinander verknüpft werden sollen. Der Benutzer muss die Orientierung und die

Transformationen der Kurven passend wählen, weil sonst *Open Inventor* die Kurven soweit biegt, bis sich die Breakpoints berühren.

Für Programmbeispiele und nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Kurven.

3.8.3 Flächen

Damit der Benutzer nicht mit viel Mühe Flächen aus Polygonen und simplen Formen zusammen setzten muss, hat *Open Inventor* ihm die Möglichkeit gegeben, komplexe Flächen selbst zu gestalten. Für diesen Vorgang werden zumindest zwei Kurven benötigt, die im Raum miteinander verbunden sind. Entweder, man verbindet diese direkt miteinander oder es werden zu jeder einzelnen Kurve eine gleiche parallele Kurve an das Ende der anknüpfenden Kurve gelegt. Die Fläche wird dann später mit einer Oberflächentextur in Form von Kleinen Polygonen überzogen. Diese Funktionen können mit Hilfe der *NURBS Surfaces* durchgeführt werden.

Für Programmbeispiele und nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Open Inventor*-Flächen.

3.9 Oberflächenstruktur

Für alle Formen ist die Möglichkeit gegeben, ihre Fläche nach eigenen Vorstellungen zu gestalten. Zur Darstellung von Flächen bietet *Open Inventor* eine Reihe von Möglichkeiten, die folgend aufgelistet sind.

- **SoMaterial:** Diese Klasse wird zur Definition der Farbe benutzt. Hierbei kann der Benutzer verschiedene Eigenschaften für die Farbe auswählen, wie z.B. das Erscheinungsbild der Farbe bei Lichteinstrahlung, Transparenz und vieles mehr.
- **SoDrawStyle:** Diese Klasse definiert die Darstellungstechnik für Objekte.
- **SoLightModel:** Diese Klasse definiert die Art der Lichtwiedergabe bei Objekten.
- **SoEnviroment:** Diese Klasse ermöglicht es, atmosphärische Effekte, wie Nebel, Rauch etc. zu simulieren.

- **SoShapeHints:** Mit dieser Klasse kann die 3D-Darstellung optimiert werden, indem bestimmte Funktionen an der Darstellung von Formen verändert werden.
- **SoComplexity:** Mit dieser Klasse kann die Komplexität der Oberflächenstruktur verändert werden.
- **SoUnits:** Diese Klasse erlaubt es, ein Standartmaß für Objekte zu definieren.

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link *Inventor Mentor* → Flächen.

Eine weitere Möglichkeit die Oberfläche von Objekten zu gestalten, ist die, indem man das Objekt mit einem Bild umhüllt. Diese Methode wird durch die Texture-Klasse ermöglicht. *Open Inventor* kann ein Bild über jedes beliebige Objekt legen. Wenn nötig, führt *Open Inventor* Transformationen durch, damit das ganze Objekt bedeckt ist. Transformationen können Streckungen, Stauchungen, Erweiterung des Bildes um gewählte Pixel oder ähnliches sein. Die Umhüllung des Objekts wird durch interne mathematische Funktionen realisiert. Zudem kann der Benutzer noch Einfluss auf die Art der Umhüllung nehmen, indem er die Deckungskraft definiert.

Für Programmbeispiele und nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Texturen.

3.10 Aktionen

Sobald ein Szenengraph vorhanden ist, können auf diesem Aktionen durchgeführt werden. Aktionen ermöglichen dem Benutzer eine Menge von Operationen, die folgend aufgelistet sind.

- Es ist möglich, Teile oder den ganzen Graphen zu generieren.
- Den Graphen als Datei zu speichern.
- Nach bestimmten Elementen im Graphen zu suchen.
- Informationen über bestimmte Objekte zu holen.
- Die Generierung von Manipulatoren für Objekte.

- Eigene Aktionen können mit Hilfe von CallBack-Funktionen erstellt werden.
- ...

Das Prinzip der Initialisierung von Aktionen ist immer gleich. Als erstes muss die ausgewählte Action-Klasse importiert und initialisiert werden. Nach der Initialisierung müssen nötige Parameter gesetzt werden, die den Verlauf bestimmen. Darauf muss die Action-Klasse einem gewählten Knoten zugeordnet werden, auf dem diese arbeiten soll. Folgendes Beispiel soll diesen Verlauf anhand der SoWriteAction, die einen Graphen als Datei speichert, gezeigt werden:

```
SoWriteAction myAction;
myAction.getOutput()-> openFile('myFile.iv');
myAction.getOutput()-> setBinary(FALSE);
myAction.apply(root);
myAction.getOutput()-> closeFile();
```

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Aktionen.

3.11 Events und Selection

3.11.1 Events

Events sind vom Benutzer ausgelöste Ereignisse, die auf der graphischen Oberfläche stattfinden. *Open Inventor* ermöglicht es dem Benutzer, durch eine X-Event-Schnittstelle den Szenengraphen direkt zu beeinflussen. Das ausgelöste Event, sei es ein Mausklick oder ein Tastenschlag wird auf der RenderArea ausgeführt und als solches erkannt. Der Eventtranslator übersezt das Ereignis in ein für *Open Inventor* verständliches Ereignis um. SoEvent erkennt die Art des Events und leitet dieses an den SoSceneManager weiter. Der SoSceneManager übernimmt den Zugriff auf den Szenengraphen, wie auch die angestrebten Transformationen. Bei erfolgreicher Ausführung wird die 3D-Darstellung neu generiert (sh. Abb. 11).

Vordefinierte Events können bei Manipulatoren oder Draggern stattfinden. *Open Inventor* bietet aber auch die Möglichkeit, eigene Events zu gestalten. Dies geschieht mit so genannten CallBack-Funktionen. Diese CallBackfunktionen warten auf ein bestimmtes Ereignis und lösen selbstgeschriebene Transformationen oder Funktionen aus.

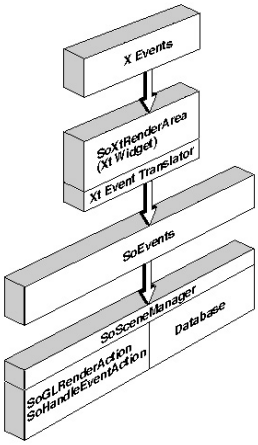

Abbildung 11: Diese Abbildung zeigt den Verlauf von einer Eventauslösung bis zur Veränderung des Szenengraphen. [1]

Das folgende Beispiel zeigt den Aufruf eines selbst geschriebenen Events. Hierbei ist der Auslöser ein Tastenschlag auf der Tastatur. Wenn dieser stattfindet, erfolgt die selbst geschriebene Funktion (myCallbackFunc) die auf dem Szenengraphen arbeitet.

```
SoEventCallback *eventCB = new SoEventCallback;
eventCB-> addEventCallback ( SoKeyboardEvent::getClassTypeId(),
myCallbackFunc, userData);
```

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Events und Selections.

3.11.2 Selections

Mit Selections können bestimmte Objekte auf der graphischen Oberfäche ausgewählt oder markiert werden. Die Klasse SoSelection erstellt eine Liste von Pfaden zu Objekten, die durch Events angesprochen werden. Per Mausklick oder Tastatur können aus dieser Liste Pfade zu Objekten entfernt oder hinzugefügt werden, indem direkt auf das Objekt gedrückt wird. Alle Objekte, die in dieser Liste auftauchen, können nach eigenen Vorstellungen per Events transformiert oder sogar gelöscht werden.

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Events und Selections.

3.12 Sensoren

Sensoren sind eine spezielle Klasse von Knoten, die dem Szenengraphen zugeordnet werden. Sensoren reagieren bei Änderungen im Szenengraphen oder bei gesetzten Zeitfunktionen. Die am häufigsten benutzten Sensoren sind DataSensors und TimerSensors. Bei den DataSensors unterscheidet man drei verschiedene Arten.

- **SoFieldSensor:** Dieser Sensor wird einem ganzem Feld zugeordnet, wie z.B. dem Sichtfeld der Kamera. Bei Veränderung des Sichtfelds springt der Sensor an und löst bestimmte Events aus.
- **SoNodeSensor:** Dieser Sensor wird einem Knoten zugeordnet und reagiert bei Veränderungen oder bei Aufruf des Knotens.
- **SoPathSensor:**Dieser Sensor wird einer Verbindung im Szenengraphen zugeordnet. Der Sensor überwacht alle Objekte, die an diesem Pfad gebunden sind. Sobald eines der Objekte am Pfad angesprochen wird, reagiert der Sensor.

TimeSensors werden eingesetzt, wenn ein bestimmtes Event in bestimmten Zeiteinheiten ablaufen soll. So ist es möglich, Countdowns oder ähnliches zu programmieren, die nach dem Ablauf des Zeitintervalls ausgewählte Events auslösen.

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Sensoren.

3.13 Engines

Engines sind Motoren die bei 3D-Darstellungen Bewegungen und Animationen ermöglichen. Die Engine-Klasse kann Rotationen, geometrische Veränderungen und Bewegungen generieren. Die nötigen Informationen für die Transformationen vom Szenengraphen werden im Engine-Knoten gespeichert. Zudem ist es auch möglich, verschiedene Engines miteinander zu verbinden und in Relation zu setzen.

Open Inventor unterscheidet folgende Engine-Arten.

- **Arithmetische Engines:** Diese Engines werden eingesetzt, um Manipultionen, Interpolationen, Entwürfe und Boolsche Operationen zu bearbeiten.

- **Animations Engines:** Diese Engines werden eingesetzt, um zeitabhängige Bewegungen zu generieren. Hierbei werden Sensoren und Engines miteinander kombiniert.

Der Ablauf ist bei allen Engines gleich. Dem Engine-Knoten wird ein Input zugeordnet, welcher in bestimmten Situationen verändert werden soll. Im Engine-Knoten werden auf diesen Werten Funktionen angewendet, die einen bestimmten Output bewirken. Bei verschiedenen Inputs kann *Open Inventor* die Werte für die Operationen eigenständig umwandeln. Der Output wird dem Szenengraphen zugeordnet, der an den betroffenen Stellen neu generiert wird.

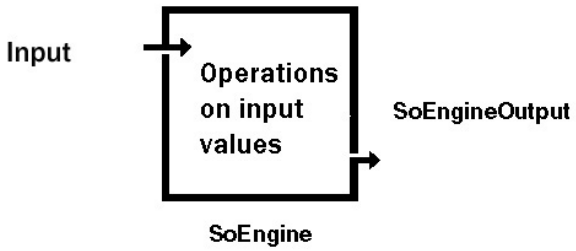

Abbildung 12: Diese Abbildung zeigt den generellen Arbeitsablauf eines Engine-Knotens. [1]

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Engines.

3.14 Dragger and Manipulatoren

Dragger und Manipulatoren ermöglichen die direkte Transformation von Objekten auf der graphischen Oberfläche. WindowEvents sprechen die Dragger oder Manipulatoren an, welche dann bestimmte Transformationen am Objekt ausführen. Die auszuführenden Transformationen müssen mit Hilfe von Events deklariert und am Objekt angewendet werden. Ein Dragger führt eine Transformation an einem Parameter des Objekts aus, wobei ein Manipulator es dem Benutzer ermöglicht, sein Objekt in mehreren Parametern zu ändern.

Die folgende Abbildung 13 zeigt bei a) einen Dragger, der so ausgelegt ist, dass er bei Veränderungen den Radius des Boden verändert und die Abbildung b) zeigt einen Manipulator, welcher das Objekt in alle Richtungen ziehen und stauchen kann.

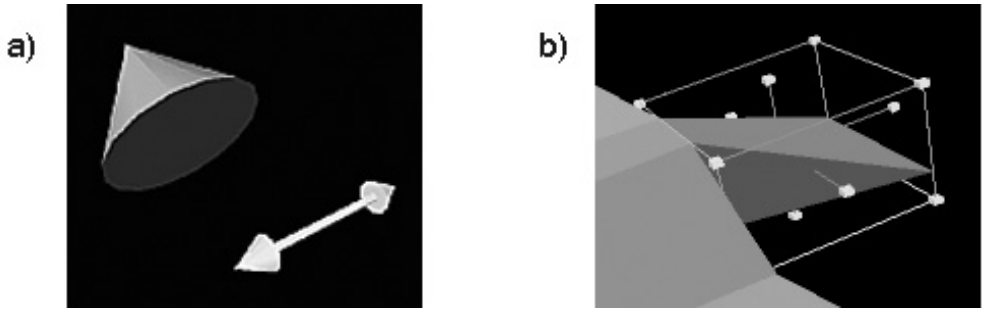

Abbildung 13: (a) Dragger (b) Manipulator [1]

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Dragger und Manipulatoren.

3.15 Dateiformat .iv

Durch die Möglichkeit der SoWriteAction kann der Szenengraph als ASCII oder Binäres File-Format (.iv-Format) gespeichert werden. Theoretisch können alle kompatiblen .iv-Dateien in der *Open Inventor*Database gespeichert und aufgerufen werden. Der Szenengraph wird mit all seinen Objekten nach syntaktischen und semantischen Regeln als Datei gespeichert. Es ist sogar möglich ohne Programmierkentnisse eine .iv-Datei zu schreiben, die beim Generieren eine 3D-Darstellung erzeugt.

Für nähere Informationen wird auf das Literaturverzeichnis verwiesen [1, 2, 6], oder bei Bestehen der Internetseite auf den folgenden Link; *Inventor Mentor* → Exportdatei.

4 Amira

4.1 Einsatzgebiete

Bei Amira handelt es sich um eine modulare, objektorientierte 3D-Visualisierungs- und Modellierungs-Software. Sie basiert auf *TGS' Open Inventor*, welches inzwischen *Open Inventor from Mercury* heißt, seitdem *Mercury* 2004 *TGS* übernommen hat.
Die Einsatzmöglichkeiten dieses Systems erstrecken sich auf die unterschiedlichsten wissenschaftlichen Gebieten wie Biologie, Chemie, Maschinenbau, Medizin und Physik.

4.2 Visualisierungen

Die 3D-Visualisierung unter Amira basiert auf Polygon-Gittermodellen, sowohl in drei- als auch viereckiger Form.
Dabei bietet es echtzeit-nahes Volume-Rendering, welches natürlich immer stark von der Leistungsstärke der verwendeten Hardware abhängt.
Neben dem interaktiven 3D-Viewer bietet Amira eine baumstrukturähnliche Visualisierung der Objektabhängigkeiten, die jederzeit den nötigen Überblick über die Struktur der jeweilgen Szene wahrt.

4.3 Features

Eines der vielen wichtigen Features von Amira ist die 3D-Stereo-Projektion ermöglichende Darstellung der Modelle.
Weitere Features sind der interaktive Editor für 3D-Modelle und die Oberflächen-Reduzierung. Mit deren Hilfe ist es z.B. möglich, ein aus 2000 Polygonen bestehendes Modell in ein auf 200 Polygonen reduziertes Modell zu transformieren.
Zusätzlich können aus Bilderstapeln 3D-Modelle erzeugt werden.
Sämtliche Visualisierungen sind skript- und animierbar und die data sets (3D-Datensätze) können simultan, in verschiedenen Viewern und Koordinatensystem in Abhängigkeit zueinander verwaltet werden.

5 Zukunft

Die Zukunft von *Open Inventor* liegt hauptsächlich in den Händen zweier Firmen, da *SGI* seit dem Release 2.1 keine Weiterentwicklung bzw. Softwarepflege mehr bietet:
Bei *Systems in Motion* ist die Angleichung von *Coin 3D* an den *SGI* Release 2.1 bald abgeschlossen, während der *Open Inventor from Mercury* Release 5.0 einen großen Katalog an Neurerungen gegenüber der inzwischen schon veralteten 2.1 API von *SGI* bietet. Es handelt sich somit um das am weitesten fortgeschrittene *Open Inventor* Projekt, welches allerdings auch leider nicht preiswert erhältlich ist.
Dem Einsteiger sei hier abschließend empfohlen, sich zunächst *Coin 3D* [8] zuzulegen und sich in dieses Toolkit einzuarbeiten. Der Umstieg auf die bessere, kommerzielle Version dürfte später aufgrund der gleichen Basis kein Problem darstellen.

Literatur

[1] Josie Wernecke. *The Inventor Mentor*. Addison Wesley Publishing Company. 2 ed. 1999

[2] TGS. *Open Inventor Manual for Release 2.1*.

[3] *Amira 3.0 User's Guide*. Konrad-Zuse-Zentrum für Inforamtionstechnik Berlin. 2002

[4] *http://www.computerlexikon.com*

[5] *http://www-evasion.imag.fr/Membres/Francois.Faure/doc/inventorMentor/sgi_html/*

[6] *http://oss.sgi.com*

[7] *http://www.tgs.com*

[8] *http://www.coin3d.org*