Katharina Greiner Matrikel-Nr. 943471
Carsten Arnold Matrikel-Nr. 232237
Fachhochschule Wiesbaden
Fachbereich Informatik

05.02.2006



# Phantom Graph Demo – Projektbeschreibung

# 1 Aufgabenstellung

Die Aufgabe des Projekts bestand in der Umsetzung einiger Funnktionen, wie sie im Dokument „HinweiseAufgabe5“ erwähnt werden.

So sollte im Allgemeinen das Phantom Device dazu genutzt werden, um eine Navigation in einem Graphen zu ermöglichen.

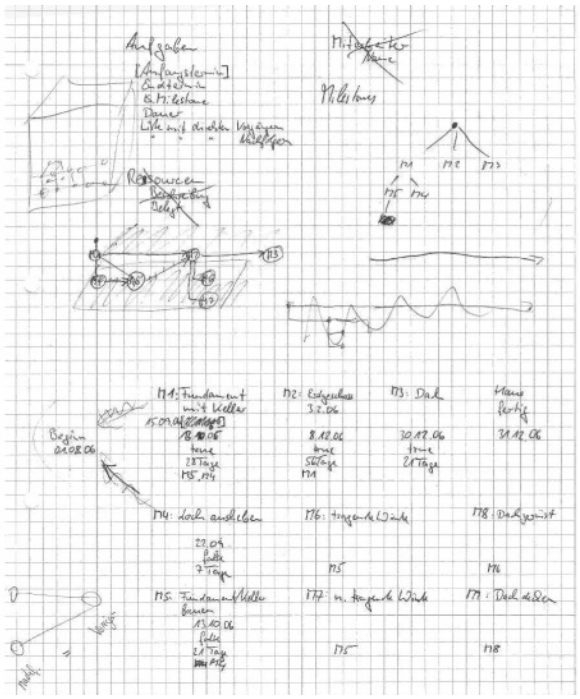
Dazu sollte eine Auswahl verschiedener Möglichkeiten des Phantom Device genutzt werden um eine alternative Bedienungsmöglichkeit zur bisherigen Maussteuerung zu bieten.

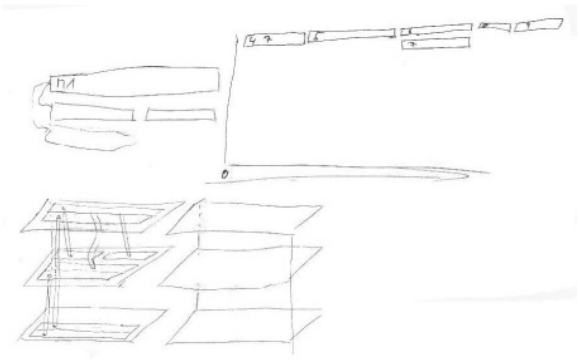
Wir haben uns dafür entschieden für die Navigation und Manipulation folgende Punkte zu berücksichtigen:

- Navigation im 2D Raum
- Abhängigkeit der Knoten untereinander durch "Kräfte" visualisieren
  Knoten können sich nur frei vor und zurück bewegen lassen, wenn keine anderen Knoten dadurch beeinflusst werden.
  Werden andere Knoten beeinflusst, wird das Verschieben schwieriger und unter Umständen unmöglich

# 2 Herangehensweise

Brainstormin … war angesagt ;) … viele Verworfene Ideen und einige Ansätze mit der Aufgabe umzugehen, sind anhand dieser Notizen entstanden:





Da die Funktionsweise des Phantom völlig neu und anders gegenüber bisherigen, herkömmlichen Zeige- und Navigationsgeräten ist, war eine Einarbeitung in die Thematik erforderlich.

In der Anfangsphase war deshalb eine umfangreiche Analyse der zur Verfügung gestellten Software nötig. Unter anderem wurden bereits existierende Beispiele genutzt um die Möglichkeiten des Phantom zu erproben.

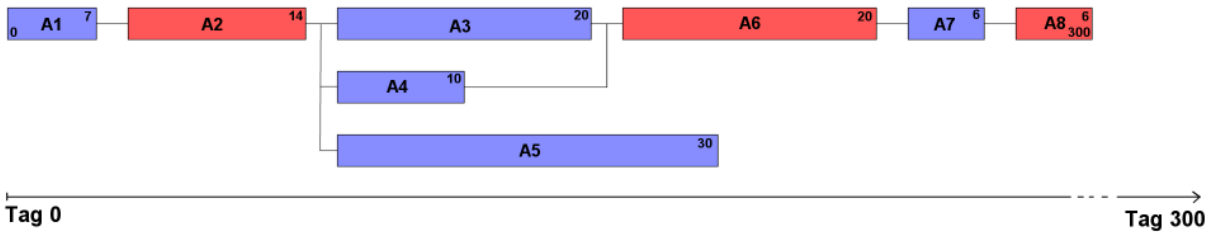
Entsprechend dieser Analyse wurde ene Umsetzung für die Graphenmanipulation erarbeitet.

Um im Rahmen der Praktikumsveranstaltung gewisse Möglichkeiten des Phantom in Verbindung mit der Graphennavigation vorführen zu können, wurde ein kleines Szenario erstellt.

Dieses sah folgende Grundstruktur vor:

## 2.1 Demograph

Der Demograph stellt ein kleines Bauprojekt vor, in dem verschieden Aufgaben erledigt werden müssen.




Diese Aufgaben wurden erzeugt und dienten später als Grundlage für die Modellierung.

### 2.1.1 Definition der Aufgaben

A1: Baugrube ausheben
10
7
FALSE
NULL

A2: Fundament / Keller bauen
24
14
TRUE
A1

A3: Erdgeschoss Aussenwände
44
20
FALSE
A2

A4: Erdgeschoss tragenden Wände
34
10
FASLE
A2

A5: Erdgeschoss nichttragende Wände
56
30
FALSE
A2

A6: Zwischendecke Ergeschoss
70
20
TRUE
A3, A4

A7: Dachgerüst aufbauen
80
6
FALSE
A6

A8: Dach decken
300
6
TRUE
A7

### 2.1.2 Grobe Struktur einer Aufgabe

| | |
|---|---|
| Endtermin | # definiert den spätesten Projekttag an dem die Aufgabe abgeschlossen werden muss |
| Dauer | # Dauer der Aufgabe in Tagen |
| Milestone | # Legt eine Aufgabe als Milestone fest |
| Liste direkter Vorgänger | # Vorgänger die abgeschlossen sein müssen, bevor die Aufgabe beginnen kann |
| Deadline | # Termin an dem die Aufgabe fertig gestellt sein muss. |

# 3 Anwendungsarchitektur

## 3.1 Ein Framework für Haptische Objekte

### 3.1.1 Programmierschnittstelle HLAPI / OpenGL

Das verwendete haptische Ein-/Ausgabegerät, das Phantom-Device, besitzt eine Programmierschnittstelle, die ähnlich wie die 3D-Graphik-API OpenGL aufgebaut ist und teilweise auch OpenGL-Aufrufe verarbeitet. Da HLAPI (Haptics Library API) und OpenGL C-Bibliotheken sind, wurden sie für dieses Projekt mit C++ gekapselt.

### 3.1.2 GraphScene – Verwaltung des gesamten Graphen

Um die Komponenten eines Graphen darzustellen und für das Phantom-Device fühlbar zu machen, wurde eine kleine Klassenbibliothek für haptische Objekte erstellt, die im Wesentlichen HLAPI-Aufrufe kapselt und koordiniert. Sie ist so angelegt, dass sie sehr einfach verwendet werden kann. Das HapticFramework stellen wir gerne zur Verwendung und Weiterentwicklung zur Verfügung. Eine Umfangreiche Dokumentation befindet sich im Ordner blabla.

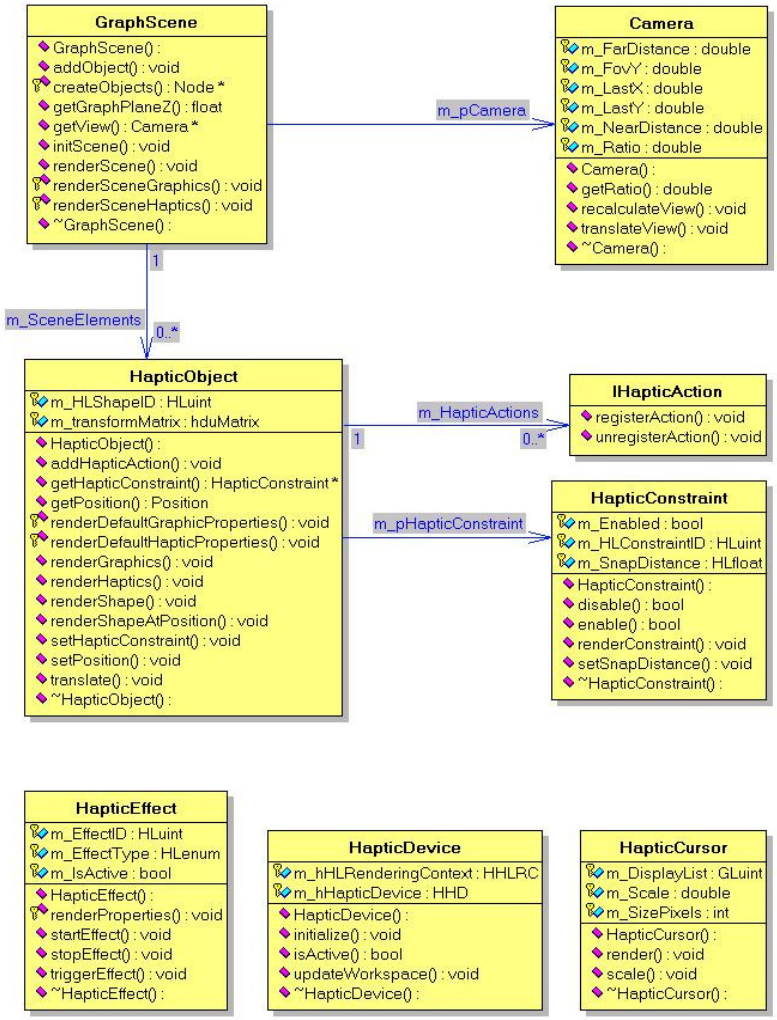


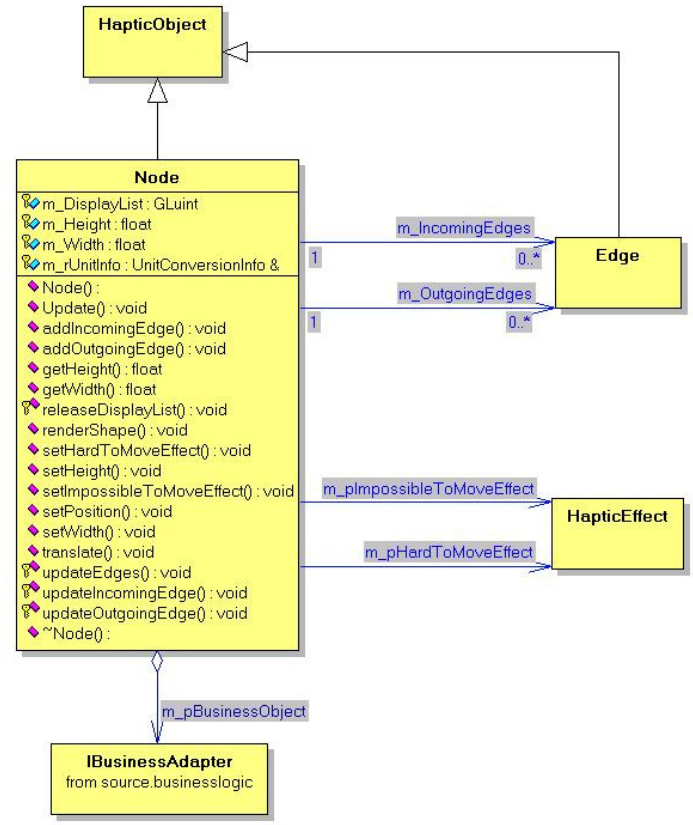

Abb. 3.1.2 Übersicht über das HapticFramework

Die Hauptaufgaben werden von den Klassen GraphScene und HapticObject übernommen. GraphScene ist für die Verwaltung des gesamten Graphen und evtl. Hilfsobjekte (z.B. das Grid) zuständig, sie veranlasst die Objekte, sich zur richtigen Zeit zu rendern, sorgt dafür, dass registrierte Eventhandler aufgerufen werden und hat Einfluss auf die Kameraführung. Damit ist sie für die Steuerung des Frontend verantwortlich.
Außerdem ist sie in der Lage, aus einem Graphen, der als Verkettung von Objekten im Speicher vorhanden ist, die entsprechenden Darstellungsobjekte zu erzeugen. Alternativ können die haptischen Objekte auch extern erzeugt und der Szene hinzugefügt werden.

Die abstrakte Klasse HapticObject dient als Basisklasse aller Objekte, die vom Phantom ertastet werden sollen. Abgeleitete Klassen müssen lediglich die Geometrie des darzustellenden Objekts mittels OpenGL-Aufrufen spezifizieren. Dem haptischen Objekt können dynamisch beliebig viele Eventhandler (über die Schnittstelle IHapticAction) und ein Constraint zugeordnet werden. Constraints (repräsentiert durch Instanzen des Typs HapticConstraint) bewirken, dass sich Objekte je nach Parametrisierung mehr oder weniger magnetisch für das Phantom anfühlen.

## 3.1.3 Knoten und Kanten

Abb. 3.1.3 Details zur Node-Klasse




Da ein Graph aus Knoten und Kanten besteht, wurden für unsere Demo-Anwendung einfache Node- und Edge-Klassen implementiert. Die Knoten werden als Rechtecke, die Kanten als Linien, deren Anfangs- und Endpunkte an den Knoten befestigt sind dargestellt. Der Node sorgt dafür, dass die ihm zugeordneten Kanten sich mit ihm bewegen, wenn er seine Position verändert.
Für die hier demonstrierte Funktionalität ist gefordert, dass sich der Knoten je nach Kontext im Graphen leicht, schwer oder gar nicht bewegen lässt. Dies wurde über so genannte ForceEffects realisiert, die in Punkt blabla näher beschrieben werden.
Der Node ist mit einem Objekt verbunden, dass die Businesslogik für Knoten enthält. Dieses entscheidet, wie leicht oder schwer der Knoten bewegt werden kann, woraufhin der Node einen seiner zugeordneten ForceEffects aktiviert.

## 3.1.4 Das Grid – Positionierung der Knoten

Das Grid, auch von HapticObject abgeleitet, stellt ein Hilfsobjekt dar, das das Layout des Graphen erleichtert. Jeder der roten Punkte steht für eine Einheit in horizontaler beziehungsweise vertikaler Richtung. Das Grid stellt eine Methode zur Verfügung, die zu

jedem beliebigen Punkt im Raum den nächstgelegenen Gitterpunkt berechnet. Diese kann dazu verwendet werden, Knoten nur an gültigen Gitterpunkten zu platzieren.

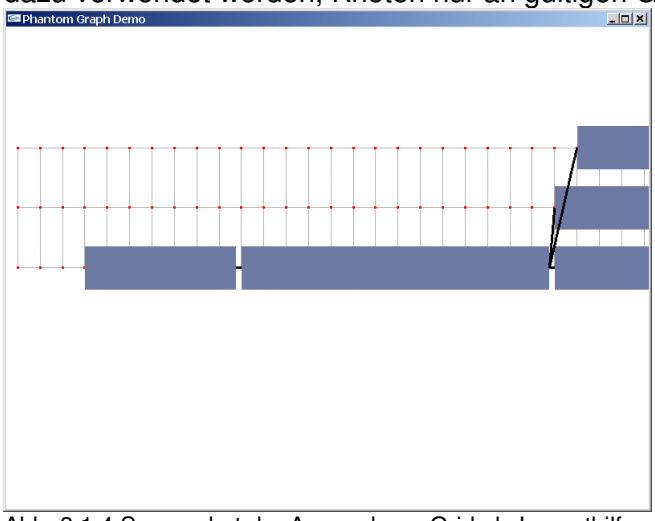



Abb. 3.1.4 Screenshot der Anwendung: Grid als Layouthilfe

## 3.1.5 Interaktion

Um den Graphen nicht nur befühlen, sondern auch manipulieren zu können, bietet das HapticFramework eine Schnittstelle für Eventhandler. Diese können einem HapticObject zugeordnet werden und registrieren für dieses Objekt HLAPI-Callback-Funktionen für ein oder mehrere HLAPI-Events (z.B. vorderer Phantombutton gedrückt). Die Eventhandler-Objekte können dynamisch weitere Events registrieren und deregistrieren, was ihnen große Flexibilität hinsichtlich ihrer Funktionalität erlaubt. All dies bleibt vor dem HapticObject (und dem Anwender) verborgen.

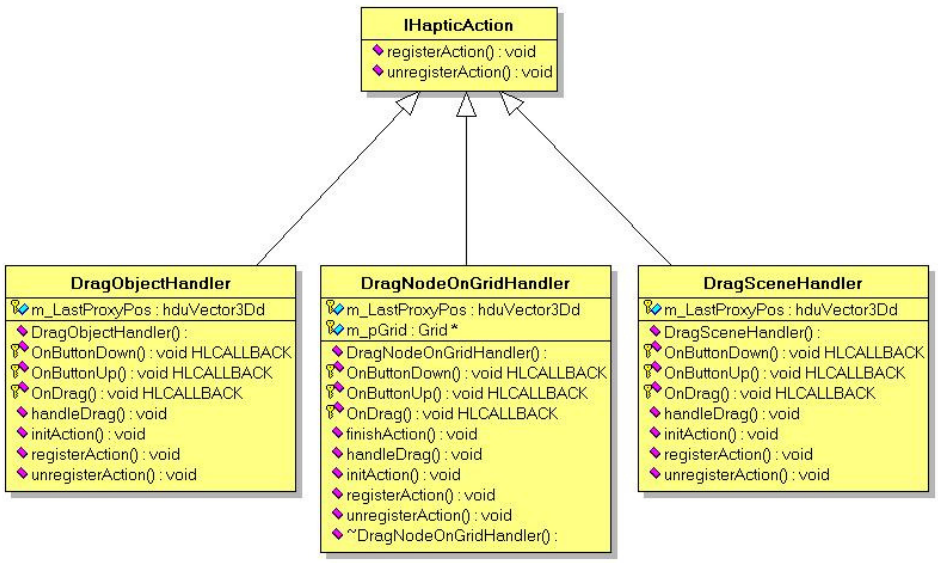
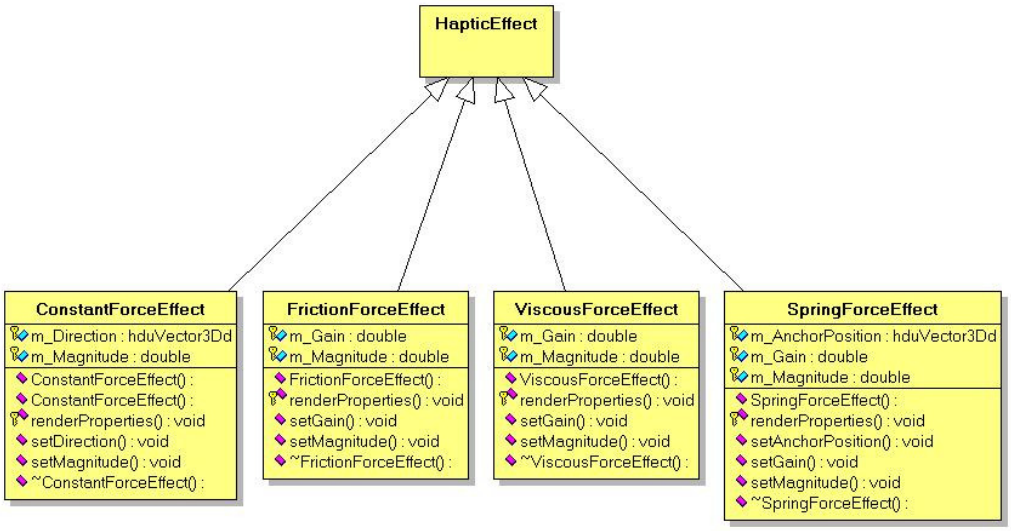



Abb. 3.1.4 Die Schnittstelle IHapticAction und implementierende Eventhandlerklassen

## 3.1.6 Kräfte

Bei der Haptikprogrammierung spielen Kräfte eine zentrale Rolle. Bei den hier implementierten ForceEffects handelt es sich um „ambiente“ Kräfte, die auf das Phantom wirken, aber von keinem bestimmten Objekt ausgehen. Die verschiedenen Effekttypen erzeugen Kräfte, die sich für das Phantom unterschiedlich anfühlen. In unserer

Beispielanwendung werden sie verwendet, um den Eindruck zu vermitteln, dass sich ein Knoten nur schwer oder gar nicht bewegen lässt. Wir haben uns hierbei für den FrictionForceEffect, eine Reibungskraft entschieden, da er die Bewegung des Phantom zwar behindert, aber nicht seine Richtung beeinflusst.



## 3.2 Die Businesslogik

Die Businesslogik erstellt und verwaltet die Objekte der Aufgaben.
Sie dient der Datenhaltung Des Graphen und stellt der Darstellungsebene diese Daten über ein Interface zur Verfügung.

Folgende Aufgaben werden von der Businesslogik durchgeführt:

1. Erzeugung der Objekte für die Aufgaben
2. Berechnung des Anfangstermins Anhand seines geplanten Enddatums und der Dauer, die zur Durchführung benötigt wird.
3. Empfang von Entfernungsdifferenzen in X-Richtung beim Verschieben von korrespondierenden Knoten der View.
4. Berechnung eines neuen Anfangs- und Enddatums einer Aufgabe
5. Überprüfung, ob ein Verschieben möglich ist.
6. Überprüfen, wie weit ein verschieben ohne Beeinflussung von Nachbaraufgaben möglich ist.
7. Überprüfung wie weit verschoben werden kann, unter Berücksichtigung von Nachbarknoten und der eigenen und anderer Deadlines.
8. Berechnung von Kräften:
    - Im freien Bewegungsraum, ohne Beeinflussung von abhängigen Aufgaben
    - Im eingeschränkten Bewegungsraum, unter Beeinflussung von abhängigen Aufgaben
    - Für die Bewegungsschranken bei erreichen von Deadlines oder Projektgrenzen
9. Aufforderung an die View, zum Neuzeichnen, nach Änderung von Businessdaten

## 3.3 Verbindung von Businesslogik und Darstellung

Die Schnittstelle zwischen dem haptischen Frontend und der Businesslogik wurde bewusst sehr schmal gehalten, um prinzipiell ein Austauschen der Businesslogik zu ermöglichen. Die Kommunkiation erfolgt einzig über die Schnittstelle IBusinessAdapter, der allgemeine Methodenschnittstellen zur Abfrage von Größe und Position eines Knotens und seiner Beweglichkeit bereitstellt.

## 3.4 Model-View-Architektur

Durch diese Aufteilung wird ein Aufbau der Anwendung in Model-View-Architektur erreicht, bei dem der View nur eine minimale Logik enthält und die wesentlichen Berechnungen der Anwendungsdomäne von dem Model durchgeführt werden.

## 3.5 Observer-Pattern

Die „Rückkopplung“ des Businessobjekts zu seiner graphischen und haptischen Repräsentation ist durch das Observer-Pattern realisiert worden. Das Businessobjekt meldet eine Änderung seines Zustandes an das ihn beobachtende Darstellungsobjekt, welches sich daraufhin die Informationen an denen es interessiert ist, von dem zugeordneten Businessobjekt besorgt.

# 4 Ergebnisse

Die erzielten Darstellungsmöglichkeiten und Umsetzungen der haptischen Effekte, zeigt deutlich die Möglichkeiten, die erweiterte Eingabegeräte in Bezug auf herkömmliche Darstellungsformen haben können.

Das haptische erfühlen von Daten erweitert die Sinnesaufnahme bei der Interaktion mit Daten. Diese Möglichkeit bietet weitere und Interessante Einsatzmöglichkeiten, die im Rahmen dieses Projektes nur minimal ausgeschöpft werden konnten.

Im Rahmen des Projektverlaufs ergab sich eine Fehleinschätzung des benötigten Zeitaufwandes. Aus diesem Grund wurden einige Umsetzung, die unter Umständen noch zu einer Erweiterung des Demo Projektes und dessen Möglichkeiten geführt hätte, nicht mehr umgesetzt.

Implementiert wurde eine Software, die alle nötigen Berechnungen durchführt, um mit Hilfe des Phantom Eingaben entgegen zu nehmen, Antworten zu berechnen und diese mittels Force Feedback direkt dem Benutzer als Antwort zurück zu liefern.
So können Knoten Erfühlt werden und mittels Constraints dem Benutzer eine vereinfachte Navigation bereit gestellt werden. Snaps erleichtern die Positionierung der Knoten innerhalb des Grids an vordefinierten Punkten.

# 5 Literatur

- 3D TOUCH™ SDK OPENHAPTICS™ TOOLKIT VERSION 1.0 PROGRAMMER'S GUIDE
- 3D TOUCH™ SDK OPENHAPTICS™ TOOLKIT VERSION 1.0 API REFERENCE

# 6 Verwendete Software

Bei der Entwicklung wurden folgende Tools eingesetzt:
- *Microsoft Visual Studio – Visual C++ 6.0* als Entwicklungsumgebung.
- *Subversion* und *TortoiseSVN* zur Verwaltung der Projektdateien.
(Freeware, erhältlich unter http://subversion.tigris.org/ bzw. http://tortoisesvn.tigris.org/ )
- *Jumli* zum Erstellen von Klassendiagrammen und zur Codegenerierung.
(Freeware, erhältlich unter http://www.jumli.de/ )
- *Doxygen* zum Generieren einer HTML-Codedokumentation.
(Freeware, erhältlich unter http://www.doxygen.org )

# 7 Spezielle Hardware

Das Phantom Graph Demo wurde unter Verwendung eines *SensAble Phantom Omni* entwickelt und getestet.