Diplomarbeit

3D-Snapping-Techniken in Virtuellen Umgebungen

Sascha Scholz

Vorgelegt am Institut für Informatik II,
Rheinische Friedrich-Wilhelms-Universität Bonn

Gutachter: Prof. Dr. Reinhard Klein

Betreuer: Dipl.-Phys. Thorsten Holtkämper

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, 18. Dezember 2006

Danksagung

Ich möchte mich bei Prof. Dr. Reinhard Klein für seine Bereitschaft bedanken, diese Diplomarbeit anzunehmen und zu betreuen. Mein Dank geht an alle Kollegen des VRGeo-Teams und des Competence Centers Virtual Environments des Fraunhofer Instituts IAIS, insbesondere an Dr. Manfred Bogen, Thorsten Holtkämper, Klaus-Günter Rautenberg, Armin Dressler und Kai Riege. Desweiteren möchte ich mich bei meiner Freundin Anita Sosnecki bedanken, die mir nicht nur stetigen Rückhalt gegeben hat, sondern sich zudem aktiv an Diskussionen beteiligt und sich für die während dieser Arbeit entstandenen Aufnahmen zur Verfügung gestellt hat. Ebenso gilt mein Dank meinen Eltern für die finanzielle Unterstützung meines Studiums.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieses Kapitel gibt einen Überblick über den Inhalt der Arbeit. Dazu wird nach einer kurzen Einführung und Motivation auf das Ziel und ihren Aufbau eingegangen.

1.1 Motivation

Präzise Objektmanipulation mit 3D-Interaktionsgeräten in Virtuellen Umgebungen ist meist schwieriger als an einem Desktop-System. Einerseits kann die Interaktion aufgrund einer größeren Anzahl von Freiheitsgraden natürlicher und direkter sein als mit einem 2D-Eingabegerät wie einer Maus. Auf der anderen Seite leidet die Präzision von Manipulationen nicht nur unter praktischen Problemen wie ungenauem Tracking, sondern schon aufgrund der meist freien Arm- und Handbewegung des Benutzers.

In vielen Fällen wollen Benutzer Objekte unter bestimmten Randbedingungen manipulieren. Ein einfaches aber häufig auftretendes Beispiel hierfür ist die Herstellung bestimmter Anordnungen durch eine Ausrichtung an Hilfselementen wie zum Beispiel einem Gitter. Hierbei können Snapping-Techniken die Benutzer unterstützen.

Im 2D-Bereich ist Snapping ein etabliertes Hilfsmittel. Populäre Beispiele auf dem Desktop sind das Snapping von Fenstern aneinander bzw. an den Rändern des Desktops und Snapping von Kontrollelementen auf bestimmte Wertebereiche (z. B. bei Schiebereglern). Anwendungsbeispiele sind Design-, Konstruktions- und Geometrieprogramme, bei denen es möglich ist, Objekte aneinander auszurichten bzw. Hilfselemente wie Gitter oder Linien zu definieren, an denen Objekte bei Manipulation snappen.

Auch bei der Interaktion in 3D-Szenen gibt es ein breites Anwendungsgebiet für Snapping-Techniken, allerdings beschränken sich existierende Anwendungen wie 3D-Modellierungs- oder Konstruktionsprogramme und die Forschung weitgehend auf nicht-immersive Desktop-Anwendungen, d.h. insbesondere auf die Maus als Eingabegerät und die Arbeit vor einem Monitor. Für tatsächliche 3D-Interaktion kommen wenn überhaupt höchstens indirekte 3D-Eingabegeräte wie eine SpaceMouse [3Dconnexion] zum Einsatz. Gerade aber bei der Verwendung von Eingabegeräten mit mehr Freiheitsgraden werden Snapping-Techniken eher selten verwendet.

Auf die besonderen Herausforderungen bei der Verwendung von Snapping im Bereich der Virtuellen Realität (VR) wurde bislang wenig eingegangen. Die Betrachtungen beschränken sich weitgehend auf die Unterstützung von Selektionen durch Snapping.

1.2 Ziel

In der vorliegenden Diplomarbeit werden 3D-Snapping-Techniken für die Objektmanipulation mit absolut im Raum positionierten und orientierten 3D-Eingabegeräten vorgestellt und dabei insbesondere auf Herausforderungen bei der Interaktion in Virtuellen Umgebungen eingegangen. Im Fokus stehen Punkt-Snapping bzw. Snapping des Translations-Anteils von Objekttransformationen sowie Snapping von Rotationen. Geeignete Feedback-Techniken werden entwickelt, um dem Benutzer die Arbeitsweise der verwendeten Verfahren transparent zu machen und eine koordinierte Interaktion zu ermöglichen.

Zur Evaluation der entwickelten Verfahren wurden sie in eine prototypische Anwendung zur Exploration seismischer Daten aus dem VRGeo-Projekt [VRGeo] integriert. Insbesondere ist es für derartige Applikationen notwendig, dass auch beim Snapping auf große Polygonmodelle Interaktivität erhalten bleibt, um z. B. die exakte Platzierung und Manipulation von Markierungen auf detaillierten Oberflächen zu erlauben.

1.3 Aufbau

Zunächst wird in Kapitel 2 eine kurze Einführung in Systeme der Virtuellen Realität gegeben und die benötigten Grundlagen der Interaktion in Virtuellen Umgebungen erläutert.

Kapitel 3 beschreibt die Grundlagen und Entwicklung von Snapping. Zuerst wird auf in 2D arbeitende Snapping-Verfahren eingegangen. Es folgt eine Übersicht über existierende 3D-Snapping-Techniken für die Interaktion mit 2D- und 3D-Eingabegeräten.

In Kapitel 4 werden Techniken für 3D-Snapping von Punkten bzw. des Translationsanteils von Objekttransformationen vorgestellt, die für die Interaktion in Virtuellen Umgebungen geeignet sind. Nach Behandlung der grundlegenden Verfahren wird ein Algorithmus zur schnellen Nächster-Nachbar-Suche vorgestellt, welcher Snapping auf große Polygonmodelle mit interaktiven Bildraten ermöglicht.

Kapitel 5 behandelt Snapping von Rotationen. Die hier vorgestellten Techniken erlauben die direkte Betrachtung des Rotationsanteils von Objekttransformationen, ohne dass eine Indirektion notwendig ist, bei der (möglicherweise gesnappte) Translationswerte auf Rotationen abgebildet werden (wie bei Kontrollelementen in 3D-Modellierungsprogrammen üblich). Dabei wird insbesondere auf den Zusammenhang mit den zugrundeliegenden Interaktionstechniken eingegangen.

Nach Vorstellung der eigentlichen für Interaktion in Virtuellen Umgebungen geeigneten Snapping-Verfahren in den vorangegangen Kapiteln wird in Kapitel 6 auf Visualisierungen für

den Benutzer eingegangen, die Feedback über den jeweils aktuellen Snapping-Status geben und so eine koordinierte Interaktion bei der Objektmanipulation erlauben. Eine Erweiterung der in Kapitel 2 vorgestellten Scaled-Grab-Technik erlaubt zudem, dass die Feedback-Verfahren auch für diese effiziente Manipulationstechnik eingesetzt werden können.

In Kapitel 7 wird das Design der Implementierung der Snapping-Techniken für eine einfache Integration in VR-Anwendungen vorgestellt. Dafür wird zunächst eine kurze Einführung in das verwendete VR-Framework Avango gegeben.

Schließlich stellen wir in Kapitel 8 die Integration verschiedener Snapping-Techniken in eine prototypische Applikation zur Exploration seismischer Daten im Öl- und Gas-Kontext vor. Experten aus verschiedenen Gebieten hatten Gelegenheit, die Anwendung auszuprobieren. Die Rückmeldungen werden ebenfalls in diesem Kapitel präsentiert.

2 Interaktion in Virtuellen Umgebungen

Die in dieser Arbeit vorgestellten 3D-Snapping-Techniken bauen auf etablierten Techniken für die Interaktion in Virtuellen Umgebungen auf, die in diesem Kapitel beschrieben werden. Zunächst betrachten wir die grundlegenden Komponenten und den Aufbau von Systemen der Virtuellen Realität (VR-Systeme), bevor schließlich auf die Techniken zur Objektmanipulation eingegangen wird, die die Basis der später präsentierten Snapping-Verfahren bilden.

Nach [Sherman u. Craig 2003] sind die Schlüsselelemente für die Wahrnehmung einer Virtuellen Realität eine dargestellte virtuelle Welt, Immersion, d. h. das Gefühl des Eintauchens in die virtuelle Welt, sensorisches Feedback, also z. B. die Darstellung in Abhängigkeit von der Position des Benutzers, sowie Interaktivität in dem Sinne, dass der Benutzer die Möglichkeit hat, in gewisser Weise mit der Welt bzw. Elementen in ihr zu interagieren. Das Design von Systemen der Virtuellen Realität korrespondiert in der Regel zu den Anforderungen dieser Schlüsselelemente: Übliche VR-Systeme bestehen aus einem Display-System zur Darstellung stereoskopischer Bilder, einem Tracking-System zur Erfassung der Position und Orientierung des Benutzers und der Eingabegeräte sowie einem Software-Framework zum Design der virtuellen Welt und deren Interaktionsmöglichkeiten. Weitere nicht immer vorhandene Komponenten sind beispielsweise haptische Eingabegeräte oder die Verwendung olfaktorischen Feedbacks [Bowman et al. 2005].

2.1 Displays

Display-Systeme für die Virtuelle Realität müssen zur Erzeugung einer stereoskopischen Darstellung unterschiedliche Bilder für die beiden Augen des Benutzers erzeugen. Darüberhinaus müssen die beiden Bilder separiert werden, so dass die Augen jeweils nur das für sie bestimmte Bild wahrnehmen. Dies kann wie bei *Head-Mounted Displays* [Bowman et al. 2005] durch Verwendung eigener LCDs für jedes Auge erfolgen, wobei die Displays jeweils das entsprechende Bild darstellen.

Projektionsbasierte Systeme wie z. B. die CAVE [Cruz-Neira et al. 1993] und PowerWall [Fakespace] besitzen pro Auge keine einzelnen Bildflächen (*Screens*), haben aber den Vorteil, dass der Benutzer nicht vollständig von der umgebenden Welt abgeschirmt ist und damit u. a. den eigenen Körper auch visuell wahrnehmen kann. Zur Trennung der für die beiden Augen auf denselben Screen projizierten Bilder kommen Stereo-Brillen zum Einsatz, wobei Shutter- oder Polarisations-Systeme am gebräuchlichsten sind. Shutter-Brillen schalten abwechselnd das linke

Abbildung 2.1: Das TwoView-Display in Zwei-Benutzer- und L-Shape-Konfiguration

und rechte Auge undurchsichtig. Die Projektoren zeigen damit synchronisiert nur das Bild für das entsprechende Auge (*Aktiv-Stereo*). Eine andere Technik arbeitet mit polarisiertem Licht: Die dafür eingesetzten Stereo-Brillen enthalten Filter, so dass ein Auge bspw. nur horizontal und das andere nur vertikal polarisierte Lichtwellen erreichen. Die Darstellung der Bilder erfolgt mit zwei aufeinander kalibrierten Projektoren, deren Licht mit entsprechenden Filtern ebenfalls polarisiert wird. Neben linearer Polarisation kommen auch zirkulär polarisierende Filter oder Band-Filter [Infitec] zum Einsatz (*Passiv-Stereo*).

Zur perspektivisch korrekten Darstellung der virtuellen Welt muss die Augposition und -orientierung des Benutzers berücksichtigt werden. Dieses sog. *Head-Tracking* (siehe 2.2) kann durch Bestimmung der Koordinaten der Stereo-Brille erreicht werden. Da bei den üblichen Systemen alle Benutzer dasselbe Bild auf dem Screen sehen, kann nur für einen Benutzer eine perspektivisch korrekte Darstellung erzeugt werden.

Mehrbenutzersysteme wie das TwoView Display [Riege et al. 2006] (siehe Abbildung 2.1) erlauben hingegen die gleichzeitige Darstellung zumindest zweier stereoskopischer Bilder, so dass auch zwei getrackte Benutzer mit jeweils korrekter Perspektive das System benutzen können. Die Trennung der Bilder für die Benutzer erfolgt beim TwoView mit zirkulär polarisierenden Filtern, die Trennung der Augen mit Shuttern. Die Rückprojektion auf den die Polarisation erhaltenden Screen erfolgt durch zwei Aktiv-Stereo-DLP-Projektoren, denen jeweils entgegengesetzt zirkulär polarisierende Filter vorgeschaltet sind. Jeder Projektor erzeugt also ein stereoskopisches Bild für genau einen Benutzer. Darüberhinaus unterstützt das TwoView eine L-Shape-Konfiguration für nur einen Benutzer, bei der zusätzlich eine Bodenprojektion erfolgt. Anderere Multi-Viewer Systeme verwenden modifizierte Shutter-Brillen und ge-shutterte Projektoren zur Trennung der Benutzer und Polarisation zur Links-Rechts-Trennung [Fröhlich et al. 2005].

Bei für ein größeres Publikum geeigneten Panorama-Displays wie der i-Cone [Simon u. Göbel 2002] (siehe Abbildung 2.2) wird auf Head-Tracking eines Benutzers verzichtet, um eine ständig variierende nicht korrekte Perspektive für die aktuell nicht getrackten Betrachter (also die meisten) zu vermeiden. Stattdessen wird das Bild aus der festen Perspektive eines sich in der

Abbildung 2.2: Das i-Cone Panorama-Display. Durch Verwendung von OmniStereo und Multi-Viewpoint Images für die perspektivisch korrekte Darstellung der Interaktions-Elemente wird eine gleichzeitige Interaktion mehrerer Benutzer ermöglicht (sog. *Co-Located Collaboration*)

Mitte des Display-Systems befindlichen fiktiven Betrachters dargestellt. Für nicht zu weit am Rand stehende Betrachter ergibt sich bei den für derartige Displays üblichen wenig interaktiven 3D-Kino-artigen Anwendungen eine gering verzerrte Perspektive. Ein Nachteil dabei ist, dass bei einer festen Blickrichtung die Stereo-Parallaxe über die 240° des i-Cone-Displays variiert und orthogonal zur Blickrichtung des fiktiven Betrachters sogar vollständig verschwindet. Zur Vermeidung dieses Effekts kann die OmniStereo-Technik [Simon et al. 2004] verwendet werden, bei der das Bild in einzelne Slices aufteilt und für jede Slice eine zu ihr gedrehte Blickrichtung verwendet wird, das Gesamtbild also aus mehreren Einzelbildern mit jeweis leicht gedrehter Perspektive zusammengesetzt wird. Das Ergebnis ist eine in jede Richtung annähernd korrekte Perspektive für virtuelle Objekte in nicht zu naher Distanz zum fiktiven Betrachter und für reale Betrachter, die sich in mittlerer Entfernung zur Projektionsfläche befinden.

Aufgrund der besonders für nahe Objekt deutlich sichtbar falschen Perspektive ist die Interaktion in einem solchen Display-System ohne Head-Tracking schwierig, da für einen nicht an der Position der virtuellen Kamera stehenden Benutzer dargestellte Interaktionselemente nicht mit der Ausrichtung seines Eingabegerätes korrespondieren. Eine Möglichkeit zur Erhaltung der Panorama-Perspektive, die aber gleichzeitig präzise Interaktion erlaubt, besteht in der Verwendung von Multi-Viewpoint Images [Simon u. Scholz 2005]. Dabei wird die Position jedes interagierenden Benutzers durch Tracking erfasst und die persönlichen Interaktions-Widgets aus seiner Perspektive dargestellt. Das Gesamtbild ergibt sich somit als Komposition von Bildern unterschiedlicher Perspektive. Durch die Verwendung von OmniStereo kann auf die Bestimmung der Orientierung der Benutzer verzichtet und trotzdem eine annähernd korrekte perspektivische Darstellung der Interaktionselemente in jede Blickrichtung erreicht werden. Bei einem Pick-Ray (siehe 2.4.2) kann die Darstellung sogar durch bloßes Verschieben des Strahl-Startpunktes im Eingabegerät und Korrektur der Zeigerichtung approximiert werden. Durch diese Technik ist gleichzeitige Interaktion mehrerer Benutzer vor einem großen Display möglich [Simon et al. 2005].

2.2 Tracking

Zur Ermittlung der Position und Orientierung des Benutzers (für die perspektivisch korrekte Darstellung der Szene) sowie der Eingabegeräte (für die Interaktion) kommen sog. *Tracking*-Systeme zum Einsatz. Am gebräuchlichsten sind elektromagnetische und optische Systeme. Bei elektromagnetischem Tracking wird durch einen Sender (*Transmitter*), dessen Position kalibriert wird und dann fest ist, ein elektromagnetisches Feld erzeugt. In diesem können sich nun kleinere an den zu trackenden Objekten befindliche Empfänger bewegen. Durch Induktion in darin befindlichen Spulen erzeugter Strom dient dabei zur (relativ störungsanfälligen) Messung der Koordinaten des Empfängers.

Stabiler und in einem größeren Bereich arbeiten optische Tracking-Systeme, die meist mit Hilfe retroreflektiver *Marker* und mehrerer Infrarotkameras arbeiten. Die Position und Orientierung verschiedener Objekte kann eindeutig festgestellt werden, indem dem Tracking-System feste Konfigurationen von Markern (*Targets*) bekannt gemacht werden, deren Koordinaten durch Triangulierung aus den unterschiedlichen Bildern der (zuvor kalibrierten) Kamerapositionen errechnet werden können. Andere optische Tracking-Systeme arbeiten mit emittierenden Leuchtdioden, die über ihre Farbe oder Leuchtfrequenz identifiziert werden können. In speziellen Anwendungsgebieten können auch optische Tracking-Systeme zum Einsatz kommen, die in definierten Umgebungen ohne spezielle Marker auskommen und beispielsweise die Benutzerposition oder bestimmte Gesten direkt erkennen können.

Weitere alternative in bestimmten Anwendungsgebieten zum Einsatz kommende Tracking-Systeme arbeiten mit Ultraschall- oder Beschleunigungs-Sensoren, bieten aber teilweise nur entweder Positions- oder Orientierungs-Tracking.

2.3 Eingabegeräte

Eingabegeräte dienen als Schnittstelle des Benutzers zur Interaktion in der Virtuellen Umgebung. Am häufigsten kommen dabei solche zum Einsatz, deren absolute Koordinaten im Raum (also Position und Orientierung) bestimmt werden können, die dann für die Interaktion ausschlaggebend sind (*Spatial Input*). Eine Klassifizierung der Geräte kann anhand der Anzahl und Art der Freiheitsgrade (*Degree of Freedom*) erfolgen. Bei 3DOF-Eingabegeräten sind entweder Position oder Orientierung im Raum bekannt, bei 6DOF-Eingabegeräte dagegen die vollständigen Koordinaten. Diese häufig mit einem oder mehreren Buttons ausgestatteten Stylus-artigen Geräte werden bei den im nächsten Abschnitt behandelten Interaktionstechniken, welche die Basis der in dieser Arbeit vorgestellten 3D-Snapping-Techniken bilden, vorausgesetzt (siehe Abbildung 2.3). Die Ermittlung der Position und Orientierung eines Eingabegerätes für Spatial Input kann mit Hilfe von Empfängern bei elektromagnetischem Tracking oder Markern bei optischem Tracking erfolgen. Andere Eingabegeräte wie z. B. das hauptsächlich zur Navigation geeignete NOYO [Simon u. Doulis 2004] ermitteln nur die Orientierung im Raum mit

Abbildung 2.3: Optisch getrackte Eingabegeräte für Spatial Input Interaktion. (links) Ein drahtloses leicht modifiziertes Präsentations-Eingabegerät. Mit dem an der Unterseite befindlichen Stift kann ein stationärer oder zusätzlich gehaltener Tablet-PC bedient werden (rechts) Ein getrackter PDA erlaubt beispielsweise kontextbezogene 2D-Interaktion mit zuvor in 3D selektierten Objekten und die Darstellung privater Informationen in Mehrbenutzer-Szenarien.

Hilfe eines kombinierten Gravitations-, Magnetfeld- und Beschleunigungs-Sensors [InterSense], kommen dafür aber auch ohne aufwendige Kalibrierung und zuätzliche Emitter aus. Das NOYO ist auch ein gutes Beispiel für ein Eingabegerät mit mehr als 6 Freiheitsgraden: Zusätzlich zu dem kombinierten Orientierungssensor enthält es einen analogen 6DOF-Kraftsensor wie bei einer SpaceMouse [3Dconnexion]. Die Werte der beiden Sensoren können kombiniert werden, um es als blickrichtungsunabhängiges 6DOF-Navigationsgerät für Panorama-Displays wie die i-Cone benutzen zu können.

2.4 Interaktionstechniken

Durch Interaktionstechniken werden die Rohwerte der verwendeten Eingabegeräte geeignet auf Objekte in der virtuellen Welt abgebildet. [Mine 1995] unterscheidet fünf fundamentale Interaktionsformen in Virtuellen Umgebungen: Navigation, Selektion, Manipulation, Skalierung und Applikationssteuerung. Davon relevant im Kontext der in dieser Arbeit präsentierten Snapping-Techniken sind hauptsächlich Manipulationstechniken, also Verfahren, um die Position und Orientierung von Objekten in der virtuellen Welt zu verändern. Methoden zur Selektion sind damit allerdings eng verknüpft. Im Folgenden geben wir einen kurzen Überblick über die gebräuchlichsten Manipulationstechniken für Spatial Input. Insbesondere wird dabei auf die Scaled-Grab-Technik eingegangen, die eine effektive Objektkontrolle im Nah- und Fernbereich erlaubt und daher die von uns bevorzugte und in den entwickelten Anwendungen verwendete Interaktionstechnik ist.

2.4.1 Virtual Hand

Bei den sog. *Virtual-Hand*-Interaktionstechniken werden eine virtuelle Hand oder ein anderes 3D-Cursor-Widget an der Position des Eingabegerätes in der virtuellen Welt angezeigt. Objektselektion erfolgt durch Schnitt der Cursor-Geometrie mit einem Objekt. Bei der durch anschließende Betätigung eines Buttons ausgelösten Bewegung des Objekts werden die Positions- und Orientierungswerte des Eingabegerätes direkt darauf übertragen. Diese als *Simple Virtual Hand* bezeichnete Interaktionstechnik [Bowman et al. 2005] erlaubt nur eine Manipulation innerhalb der Reichweite des Benutzers. Bei der Go-Go-Technik [Poupyrev et al. 1996] wird die Reichweite des Benutzers durch Skalierung erweitert. Dabei ist der 3D-Cursor bis zu einer bestimmten Distanz zum Benutzer direkt mit dem Eingabegerät verbunden. Darüber hinausgehend wird die Bewegung durch eine nicht-lineare Übertragungsfunktion hochskaliert, so dass auch Objekte außerhalb der eigentlichen Reichweite selektiert oder Objekte an weiter entfernte Positionen bewegt werden können.

2.4.2 Simple Dragging

Eine weitere Möglichkeit zur Interaktion mit auch außerhalb der Reichweite des Benutzers liegenden Objekten ist eine Selektion durch Zeigen. Um die Zeigeoperation zu vereinfachen, wird der aus dem Eingabegerät in die Zeigerichtung entspringende sog. *Pick-Ray* als eine Art Laser-Strahl visualisiert. Das Objekt mit dem zum Benutzer nächsten Schnittpunkt mit dem Strahl wird bei Betätigung des entsprechenden Buttons auf dem Eingabegerät selektiert.

Bei der *Simple-Dragging*-Manipulationstechnik „hängt“ das Objekt bei Bewegung immer an der bei der Selektion festgelegten Schnittposition am Pick-Ray. Der Pick-Ray bewegt sich wie bei der Selektion, entspringt also in einer festgelegten Richtung aus dem Eingabegerät. Der Rotationspunkt der Objektmanipulation ist somit die Hand des Benutzers bzw. das Eingabegerät. Um die Entfernung eines Objektes zum Benutzer zu ändern, ist u. U. mehrfache Neuselektion nötig, da Translationen des Eingabegerätes in Strahlrichtung direkt auf das Objekt übertragen werden. Einfach ist durch den im Eingabegerät befindlichen Rotationspunkt dagegen eine Rotation des bewegten Objekts in konstanter Entfernung um den Benutzer.

2.4.3 Scaled Grab

Die Scaled-Grab-Technik [Simon et al. 2005] kombiniert die Selektion durch Zeigen mit einem Pick-Ray, direkte Übertragung der Rotation des Eingabegerätes auf das manipulierte Objekt und beschleunigte Translation, um eine präzise Kontrolle über nahe und ferne Objekte zu ermöglichen.

Bei der Pick-Ray-Selektion wird der Schnittpunkt des Strahls mit dem ausgewählten Objekts als der neue Rotationspunkt (*Pivot-Punkt*) festgelegt. Die Rotation des Eingabegerätes wird dann bei der Manipulation auf die Rotation des Objekts um den Pivot-Punkt übertragen. Währenddessen wird die Darstellung des aus dem Eingabegerät entspringenden Pick-Rays

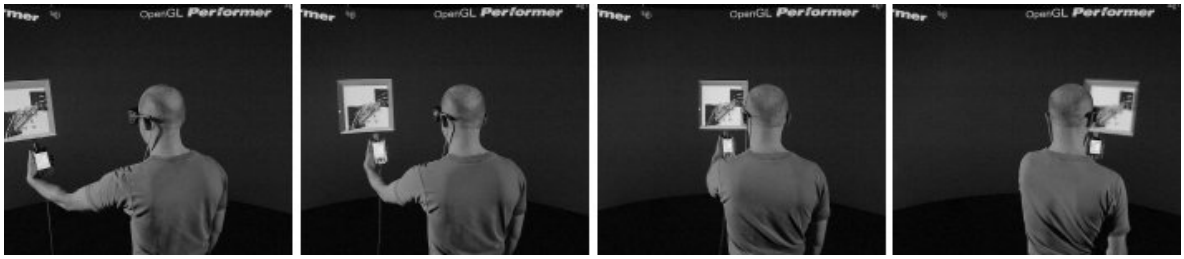

Abbildung 2.4: Der Scaled Grab erlaubt präzise Nah- und Ferninteraktion sowie eine effektive Rotationskontrolle. Ähnlich wie beim Simple Dragging bleibt bei Bewegung um den Benutzer die Ausrichtung zwischen dem Eingabegerät und dem bewegten Objekt in der Bildebene erhalten. Die Interaktion hier erfolgt mit Hilfe eines getrackten PDAs.

deaktiviert, da der Strahl im Gegensatz zum Simple Dragging nicht mehr unbedingt auf den zuvor selektierten Punkt auf dem Objekt zeigt und so den Benutzer irritieren könnte.

Um effektive Nah- und Ferninteraktion zu ermöglichen, wird die vom Eingabegerät auf das Objekt übertragene Translation skaliert. Der für jede Manipulation neu berechnete Skalierungsfaktor bestimmt sich dabei aus dem Verhältnis zwischen Benutzer-Pivot-Punkt- und Benutzer-Eingabgerät-Entfernung. Dadurch können Objekte unabhängig von ihrer Entfernung über die komplette Breite des Screens bewegt und auch die Entfernung vom Benutzer schnell verändert werden (siehe Abbildung 2.4).

2.5 Software

Ein VR-Software-Framework integriert die im VR-System benötigten Technologien wie eine Rendering- und Szenengraph-API, die Anbindung der Eingabegeräte, bietet Möglichkeiten zum Rapid-Application-Development und vieles Weitere. Auf das im Rahmen dieser Diplomarbeit verwendete Framework Avango [Avango; Tramberend 1999, 2003] wird in Kapitel 7 im Rahmen der Vorstellung des Implementierungs-Konzepts eingegangen.

3 Snapping-Grundlagen

Snapping (deutsch: Einrasten, Einschnappen) ist eine Technik, mit deren Hilfe präzise Interaktion mit Benutzeroberflächen und präzise geometrische Konstruktionen oder Transformationen durchgeführt werden können. Durch den Einsatz von Snapping können die menschlichen Leistungsbeschränkungen bei der Selektion kleiner Ziele umgangen werden, welche durch *Fitts' Law* [Fitts 1954] modelliert werden. Dieses besagt, dass die durchschnittlich benötigte Zeit, um auf ein Ziel zu zeigen, von dessen Entfernung und Größe abhängt. Durch Verwendung von Snapping-Techniken kann die visuelle Repräsentation eines Ziels verkleinert werden, ohne dass diese Zeit beeinflusst wird [Hudson 1990].

Dieses Kapitel gibt einen Überblick über die Grundlagen und Entwicklung von Snapping-Techniken. Teile der vorgestellten Verfahren sind heute fester Bestandteil von Anwendungssoftware. Gleichwohl ist Snapping aktuelles Forschungsgebiet, in dem immer wieder neue Ansätze entwickelt und die Anwendung in weiteren Einsatzbereichen untersucht werden. Der Überblick beginnt mit 2D-Snapping-Techniken und fährt fort mit Snapping-Techniken, die im dreidimensionalen Raum mit 2D- oder 3D-Eingabegeräten arbeiten.

3.1 2D-Snapping

Ein Anwendungsgebiet für Snapping-Techniken in 2D ist der Computer-Desktop mit seinen Bedienelementen. So erlauben Desktopsysteme wie [KDE] oder [Gnome] die präzise Ausrichtung von Fenstern aneinander und an den Rändern des Desktops. Auch bei vielen Komponenten von Benutzeroberflächen (*Widgets*) ist Snapping-Funktionalität integriert (z. B. bei Schiebereglern).

Insbesondere haben sich Snapping-Techniken mit ihrer Einführung in Grafiksoftware etabliert. Vor allem in aktuellen Vektorgrafikprogrammen wie z. B. Adobe Illustrator [Adobe], Corel Draw [Corel] und [Inkscape] sind Techniken wie Snapping auf Punkte in einem Gitter (*Grid-Snapping*) oder auf Hilfslinien (*Guide-Snapping*) sowie die Ausrichtung von Objekten aneinander unentbehrlich. Auch in Bildbearbeitungsprogrammen (z. B. Adobe Photoshop [Adobe], Corel Photopaint [Corel], [Gimp]) gehören Grid- und Guide-Snapping zum Standardrepertoire der Interaktionstechniken. Darüberhinaus wird präzise Selektion anhand von Bildmerkmalen wie beispielsweise Kanten oder Ecken unterstützt (siehe 3.1.4).

Weitere Verwendung finden Snapping-Techniken in Software zur Erstellung von Präsentationen (z. B. Microsoft PowerPoint [Microsoft]) und in Geometrieprogrammen (z. B. Geometer's Sketchpad [KeyPress], Kig [KDE]) bei denen es darauf ankommt, präzise Verbindungen

Abbildung 3.1: Benutzung des Sketchpad-Systems. Die Bedienung erfolgt mit Hilfe eines Lightpens und verschiedenen Knöpfen, Schaltern und Drehreglern. [Sutherland 1964]

zwischen geometrischen Komponenten herzustellen oder sie exakt aneinander auszurichten. Häufig ist es sogar möglich, durch Snapping entstandene Verbindungen zwischen mehreren Elementen auch bei nachfolgenden Manipulationen dauerhaft zu erhalten. Dadurch können in Präsentationsprogrammen z. B. einfacher Flussdiagramme oder Mindmaps erstellt werden oder in Geometrieprogrammen der Einfluss von Änderungen auf abhängige Konstruktionselemente visualisiert werden.

3.1.1 Sketchpad

Der Vorgänger aller modernen Vektorgrafik-, Modellierungs- und CAD-Programme ist Sketchpad [Sutherland 1963]. Dort finden erstmals Snapping-Techniken Erwähnung. Sketchpad wird mit Hilfe eines Lightpens auf einem CRT-Bildschirm bedient, einer Vorgänger-Technik der heutigen Touchscreens (siehe Abbildung 3.1).

Mit dem Lightpen wird ein Cursor auf der Zeichenebene bewegt. Befindet sich dieser in der Nähe eines Objekts wie z. B. einem Kreisbogen, einem Liniensegment oder auch dessen Endpunkten, wird er durch die sog. *Gravity*-Funktion auf den nächstliegenden Punkt verschoben (bzw. gesnappt). So können existierende Elemente leicht gegriffen und als Basis neuer Konstruktionen genutzt werden. Sutherland nutzt die Information über das aktuell gesnappte Objekt darüberhinaus, um dauerhafte Beziehungen (*Constraints*) zwischen den Objekten herzustellen, die auch bei künftigen Manipulationen dieser oder anderer Elemente in der Zeichnung erhalten bleiben.

3.1.2 Snap-Dragging

In [Bier u. Stone 1986; Bier 1988] wird das sog. *Snap-Dragging* in Zusammenhang mit 2D-Linien-Zeichenprogrammen vorgestellt. Die bis dato etablierten Snapping-Verfahren erlaubten eine Ausrichtung an Gittern, bereits erzeugten Punkten oder Linien oder an manuell erzeugten Ausrichtungsobjekten. Beim Snap-Dragging werden hingegen sog. Ausrichtungsobjekte

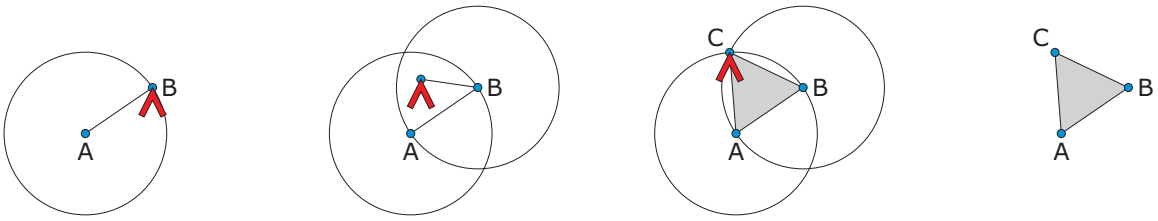

Abbildung 3.2: Konstruktion eines gleichseiten Dreieckes mit Snap-Dragging. Die Punkte A und B erzeugen Gravity-aktive Kreise mit der Länge der ersten Seite als Radius. Mit dem rot dargestellten Caret wird Punkt C bewegt, bis er auf den Schnittpunkt der beiden Kreise snappt. Nach Vollendung der Aktion werden die Ausrichtungselemente wieder deaktiviert.

dynamisch abhängig von Kontext und Benutzereinstellungen erstellt, die nur für die Dauer einer Aktion existieren.

Der Benutzer bewegt im vorgestellten System Gargoyle mit der Maus einen Software-Cursor, das sog. *Caret*. Ähnlich zu [Sutherland 1963] wird dessen Position durch eine Gravity-Funktion beeinflusst. Befindet sich das Caret in der Nähe eines Gravity-aktiven Objekts, wird es darauf gesnappt. Alle durch den Benutzer initiierten Operationen beziehen sich auf die durch die Gravity-Funktion bestimmte Caret-Position. Das Caret folgt dem Hardware-Cursor allerdings nur während bestimmter Operationen wie z. B. bei der Erzeugung neuer Objekte, damit der Benutzer nach der Platzierung des Cursors auf einem Objekt davon unabhängig mit der Maus Einstellungen für dieses im Menü vornehmen kann.

Der aktuelle Kontext wird benutzt, um dynamisch Ausrichtungselemente zu erzeugen, auf die das Caret snappt. Bewegt der Benutzer z. B. einen Kontrollpunkt, werden automatisch für in der Nähe befindliche Objekte horizontale Ausrichtungselemente erzeugt, falls die entsprechende Option im Menü aktiviert ist. Im Gegensatz zu Constraint-basierten Ansätzen wie in [Sutherland 1963; Nelson 1985] bestehen die Beziehungen zu diesen Gravity-aktiven Elementen nur temporär, wodurch die Komplexität der Benutzerschnittstelle und des Zeichensystems reduziert wird. Die Kombination der Snap-Dragging-Technik mit einem solchen Ansatz ist jedoch möglich, um z. B. auf expliziten Wunsch bestimmte Beziehungen dauerhaft zu erhalten [Gleicher u. Witkin 1994; Masui 2001].

Das System bietet dem Benutzer durchgehendes Feedback während der Durchführung von Aktionen. Dazu zählt ein sog. *Gummiband* beim Erzeugen oder Verschieben von Linien [Newman u. Sproull 1979; Foley et al. 1995], welches dem Benutzer schon vor Bestätigung einer Aktion ihr Ergebnis präsentiert. Auch für die Ausrichtungselemente werden Visualisierungen erzeugt, damit der Benutzer stets über den aktuellen Snapping-Status informiert ist.

Die Benutzung bestimmter Klassen von Ausrichtungselementen kann im Menü aktiviert und angepasst werden. So können unter anderem Prioritäten vergeben werden, so dass beispielsweise, anders als bei der Standardeinstellung, Linien-Snapping gegenüber Punkt-Snapping bevorzugt wird. Konfigurierbarkeit ist auch deswegen notwendig, um zu verhindern, dass die

Interaktion durch eine zu hohe Zahl gleichzeitig aktiver Ausrichtungselemente in Bereichen großer Objektdichte gestört wird. Als Beispiele für Klassen von Ausrichtungselementen seien vertikale und horizontale Linien an bereits konstruierten Punkten, Parallelen zu Linien oder fest einzuhaltende Abstände für Translations- und Skalierungsoperationen genannt (siehe Abbildung 3.2). Bei Rotationsoperationen ist es unter anderem möglich, feste Winkelschritte zu definieren. Herkömmliches Grid-Snapping wird ebenfalls unterstützt.

3.1.3 Semantic Snapping

Um eine semantische Komponente wird Snapping in [Hudson 1990] erweitert und am Beispiel eines Editors zur visuellen Programmierung erläutert. Der Benutzer hat hierbei die Möglichkeit, Verbindungen zwischen einzelnen Komponenten herzustellen. Dabei snappt die verbindende Linie auf Rechtecke, die jeweils eine Komponente repräsentieren, sobald sie in den Einflussbereich der Gravity-Funktion kommen.

Allerdings wird zusätzlich die semantische Korrektheit überprüft und die Verbindung bei negativem Ergebnis nicht hergestellt. Um die Direktheit der Benutzerschnittstelle zu erhöhen, wird für alle Aktionen Feedback gegeben. Zusätzlich zu dem Feedback bei erfolgtem Snapping wird dem Benutzer bei Verweigerung der Verbindung sog. *Anti-Gravity-Feedback* präsentiert, welches die fehlerhafte Semantik anzeigt.

3.1.4 Image Snapping

Für die interaktive Bildbearbeitung wird in [Gleicher 1995] das sog. *Image Snapping* vorgestellt, mit dem es möglich ist, auf aus dem Bild extrahierte Merkmale (*Features*) zu snappen.

Zunächst wird das betrachtete Bild weichgezeichnet und ein Höhenfeld bestimmt. Der Cursor wird nun als Kugel interpretiert, die sich auf den Unebenheiten bewegt und durch die Gravitation zu lokalen Minima hingezogen wird. Die Weichzeichnung hat den Vorteil, das die mittleren Bereiche von Punkten, Linien und Schnittpunkten hervorgehoben werden, hohe Frequenzen aus dem Bild herausgefiltert werden und der Cursor daher nicht zu sehr umherspringt. Jedoch behindert die Weichzeichnung auch die Bewegung z. B. innerhalb einer breiten Linie, weil immer auf mittlere Punkte gesnappt wird, welche durch die Weichzeichnung verstärkt werden. Das gewünschte Ergebnis hängt vom Anwendungsgebiet ab und kann durch Feinjustierung der Weichzeichnung beeinflusst werden.

Ein weiterer Ansatz für Snapping im Bereich der Bildbearbeitung ist das sog. *Lazy Snapping* [Li et al. 2004]. Nachdem der Benutzer ungefähr den Vorder- und Hintergrund des Bildes skizziert hat, wird dabei versucht, die genaue Begrenzungslinie zum Freistellen des Vordergrundes zu bestimmen. Dem Benutzer wird daraufhin Gelegenheit gegeben, diese nach seinen Wünschen zu editieren.

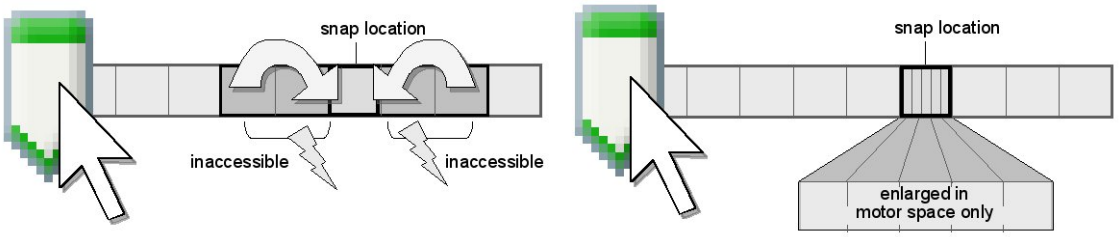

Abbildung 3.3: Snapping am Beispiel eines Schieberegler-Widgets. (a) Traditionelles Snapping transformiert alle Positionen auf das Snap-Target, sobald sich der Cursor innerhalb der Snap-Range befindet, wodurch ein Bereich um dieses herum nicht erreichbar ist. (b) Snap-and-go fügt am Snap-Target zusätzlichen Bewegungsraum ein und hält den Cursor innerhalb dessen fest. [Baudisch et al. 2005]

3.1.5 Snap-and-go

Alle bislang vorgestellten Snapping-Verfahren haben gemeinsam, dass ein gewisser Bereich um die Gravity-aktiven Objekte, die sog. *Snap-Range*, für den Benutzer nicht erreichbar ist, ohne die Snapping-Funktionalität zu deaktivieren. Stattdessen springt der Cursor auf das Ziel-Objekt (*Snap-Target)*, sobald dieser Bereich betreten wird. Es wird also eine Unstetigkeit erzeugt, die die Direktheit der Interaktion negativ beeinflusst. Dies tritt insbesondere bei Fällen mit sehr vielen Gravity-aktiven Objekten in einem kleinem Bereich auf, wenn kaum Raum existiert, in dem kein Snapping stattfindet.

In [Baudisch et al. 2005] wird die *Snap-and-go*-Technik vorgestellt, die sich dieser Herausforderung im Kontext der Ausrichtung von Objekten in graphischen Benutzeroberflächen widmet. Die Idee dabei ist, in der Nähe des Snap-Targets den Bewegungsraum der Maus bezogen auf die Cursor-Bewegung auf dem Desktop zu vergrößern. Sobald das Snap-Target erreicht ist, wird der Cursor einen Moment angehalten, um dem Benutzer Gelegenheit zu geben, die Aktion zu beenden. Durch diesen „Widerstand“ beim Durchdringen eines Snap-Targets wird ein pseudo-haptischer Eindruck erzeugt [Lecuyer et al. 2000]. Abbildung 3.3 stellt die Arbeitsweise von traditionellem Snapping und Snap-and-go bei der Interaktion mit einem Schieberegler-Widget gegenüber.

Ein Nachteil von Snap-and-go ist, dass die Interaktion nur mit indirekten Eingabegeräten wie z. B. einer Maus erfolgen kann. Bei einer direkten Eingabe z. B. mit einem Stift auf einem Touchpad geht die Korrespondenz zwischen dem Eingabegerät und dem Cursor dauerhaft verloren.

Eine Herausforderung ist das Snapping auf punktförmige Ziele. Bei eindimensionaler Interaktion (wie beim Schieberegler-Beispiel) wird das Snap-Target unweigerlich berührt bzw. überquert. Dies gilt im 2D nicht mehr, da die Bewegung daran vorbeiführen kann. Baudisch et al. definieren für diesen Fall ein (unsichtbares) sog. *Cross-Widget*, welches zusätzlichen

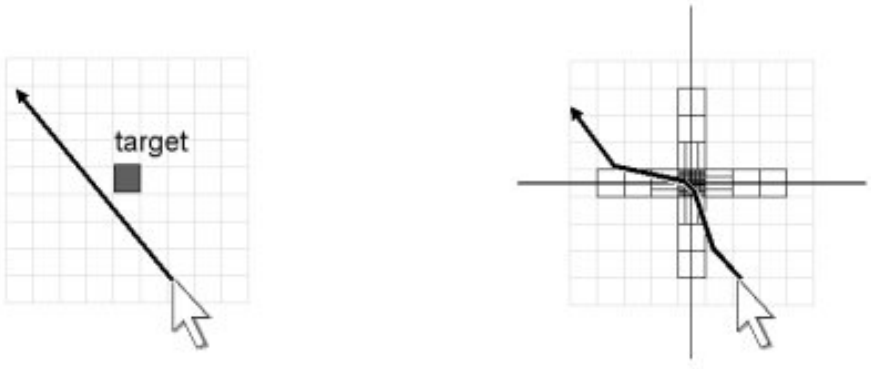

Abbildung 3.4: (a) Bewegungen im 2D können an punktförmigen Snap-Targets vorbeiführen, ohne diese zu berühren. (b) Ein Cross-Widget leitet den Cursor während der Bewegung zum Snap-Target, so dass auch im 2D Snap-and-go möglich ist. [Baudisch et al. 2005]

Bewegungsraum in Richtung orthogonal zu seinen Kanten einfügt. Auf diese Weise wird der Cursor stetig zum Ziel hingeleitet (*Guiding*, siehe Abbildung 3.4).

Eine Benutzerstudie belegt, dass Snap-and-go aufgrund der zusätzlich nötigen Bewegung etwas langsamer ist als traditionelles Snapping. Diese Differenz wird jedoch bei höherem Schwierigkeitsgrad (bei der Untersuchung gemessen durch *Fitts' Index of Difficulty* [Fitts 1954]) kleiner. Im Übrigen vereinfacht Snap-and-go die Benutzerschnittstelle dadurch, dass Snapping dauerhaft eingeschaltet bleiben kann.

3.2 3D-Snapping mit 2D-Eingabegeräten

Populäres Anwendgunsgebiet für Snapping in drei Dimensionen sind vor allem Sketching- und Modellierungsprogramme sowie CAD-Software, die mit 2D-Eingabegeräten bedient werden. So können z. B. Kontrollpunkte in 3D-Modellen oder die Modelle selbst präzise platziert werden. Dieser Abschnitt gibt einen Überblick über existierende 3D-Snapping-Techniken, die für Interaktion in 3D mit 2D-Eingabegeräten geeignet sind.

3.2.1 Snap-Dragging in 3D

Ähnlich wie Snap-Dragging in 2D (siehe 3.1.2) erlaubt 3D-Snap-Dragging die präzise Selektion von Kontrollpunkten, Kurven und Flächen, sowie außerdem präzise Translation, Rotation und Skalierung von Objekten in dreidimensionalen Szenen. Die Bedienung des in [Bier 1990] vorgestellten 3D-Modellierungs-Systems Gargolye3D erfolgt mit Maus und Tastatur. Die größten Unterschiede des eigentlichen Snapping-Algorithmus zum 2D-Snap-Dragging bestehen bei der Bestimmung der Cursor-Position und der Auswahl der Ausrichtungsobjekte.

Das Äquivalent zum Caret, dem Software-Cursor beim 2D-Snap-Dragging, ist ein 3D-Cursor, der sog. *Skitter*. Dessen Position wird bestimmt, in dem ausgehend vom Augpunkt durch

die 2D-Cursor-Position in der Kameraebene ein Strahl in die Szene geschickt wird. Für alle Flächen, Linien, Kontrollpunkte und Ausrichtungsobjekte wird der nächste Punkt, die Normale an diesem und der Abstand zum Strahl bestimmt. Dabei werden die Abstände bei der Bestimmung des nächsten Punktes durch die jeweilige Tiefe (also die z-Komponente des betrachteten Punktes im Kamerakoordinatensystem) dividiert und nach diesem Wert sortiert, damit auch Snapping auf weiter vom Benutzer entfernte Punkte möglich ist. Vom Benutzerstandpunkt aus arbeitet der Snapping-Algorithmus bzgl. der Abstandsbestimmungen somit zweidimensional in der Bildschirmebene. Der Skitter wird zusätzlich zur Positionierung abhängig von der Normale an der gefundenen nächstliegenden Position gedreht, wodurch Ausrichtungsobjekte für Rotationsoperationen bestimmt werden.

Je nach aktiviertem Modus snappt der Skitter bevorzugt auf Punkte, Linien oder Flächen. Falls der Benutzer mit der Auswahl unzufrieden ist, kann er den Cursor so lange bewegen, bis das gewünschte Objekt selektiert wird oder mit Tastaturkommandos eine nach den Abständen sortierte Liste durchlaufen. Als Ausrichtungsobjekte werden Linien, Ebenen und Kugeln unterstützt, an denen der Skitter bei Selektion und Objekttransformation snappt.

Snap-Dragging in 3D kann vielseitig eingesetzt werden. [Conner et al. 1992; Hsu et al. 1997] erlauben mit Hilfe von Snap-Dragging, Ebenen von Objekten in der Szene bzw. im CAD-Modell zu wählen, die aneinander ausgerichtet werden sollen. [Igarashi u. Hughes 2001] benutzen Snap-Dragging auf Punkte und Linien, um dem Benutzer Vorschläge für weitere Konstruktionsschritte ausgehend von der zugehörigen Ebene in einem 3D-Sketching-System zu präsentieren. Ausserdem werden dort zusätzliche Klassen von Ausrichtungsobjekten vorgestellt. Wenn der Benutzer beispielsweise die Mitte einer Linie selektiert, wird eine Orthogonale ausgehend vom Mittelpunkt der gewählten Linie erzeugt.

3.2.2 Mesh Snapping

Haupteinsatzgebiet von Snapping-Verfahren, die auf Dreiecksnetzen (*Meshes*) arbeiten, sind CAD- und 3D-Modellierungs-Anwendungen, die mit einem 2D-Eingabegerät wie einer Maus bedient werden. Dabei wird zur Cursor-Positionierung bzw. Objektselektion ähnlich wie beim Snap-Dragging ein sog. *Mouse Ray* [Oh u. Stuerzlinger 2005] benutzt.

Geometric Snapping [Yoo u. Ha 2004] erlaubt Snapping des Cursors auf lokale Features in Dreiecksnetzen. Einsatzgebiete sind unter anderem die Mesh-Simplifizierung oder Mesh-Kompression, bei der markierte Features bevorzugt behandelt werden können. Deformations-Verfahren erlauben interaktive Manipulation des Meshes anhand von Features. Auch beim Mesh-Morphing werden korrespondierende Features in zwei Meshes genutzt, um eines flüssig in das andere zu transformieren.

Beim Geometric Snapping wird mit Hilfe der Oberflächennormalen ein Maß für die Krümmung der Oberfläche bestimmt. Ähnlich wie in [Gleicher 1995] arbeitet der Algorithmus dabei mit Glättung der Oberflächenkrümmung und einer Funktion, welche die (positiven

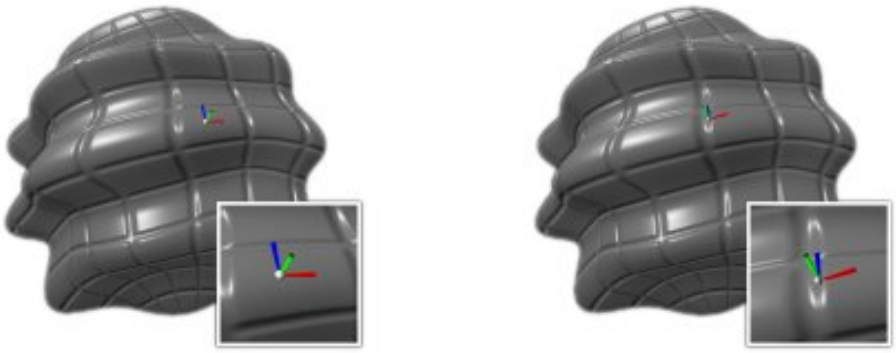

Abbildung 3.5: Snapping auf eine GPU-transformierte Kugel. Der 3D-Cursor bewegt sich auf der Oberfläche und wird mit der durch Bump-Mapping variierten Normalen gedreht. [Batagelo u. Wu 2005]

und negativen) Kosten der Cursor-Bewegung zwischen zwei Vertices anhand der Krümmung modelliert. Wenn der Benutzer einen beliebigen Vertex oder Punkt des Meshes selektiert, wird der Cursor auf ein lokales geometrisches Feature in der Nähe verschoben. Dazu wird er iterativ zum benachbarten Vertex mit maximalen Funktionswert transformiert, bis die Bewegung zu keinem benachbarten Vertex positive Kosten verursacht.

[Batagelo u. Wu 2005] erlauben Snapping auf Oberflächen, die durch Vertex- und Fragment-Shader auf der GPU transformiert werden. Der vorgestellte Algorithmus führt eine Rückprojektion der 2D-Cursor-Koordinaten in den 3D-Raum durch. Dazu wird in einem Multi-Pass-Rendering-Verfahren aus dem Bildspeicher die Normale und die Tiefe des Pixels unter dem 2D-Cursor bestimmt, wodurch die Position und Orientierung in 3D berechnet werden kann (siehe Abbildung 3.5).

Um eine Identifikation des gesnappten Dreieckes zu ermöglichen, bekommt jedes Dreieck einen eindeutigen Bezeichner zugewiesen, der in der Farbinformation des ersten Vertex kodiert wird. Nach einem weiteren Render-Pass mit Flat-Shading kann durch Auslesen der Farbe unter der 2D-Cursor-Position das zugehörige Dreieck identifiziert werden. Eine ähnliche Methode kann angewendet werden, um die baryzentrischen Koordinaten des gesnappten Dreiecks zu berechnen. Dabei werden den drei Vertices unterschiedliche Farbkomponenten zugewiesen, die bei einem weiteren Render-Pass mit Gouraud-Shading über das Dreieck interpoliert werden. Durch Auslesen des entsprechenden Farbwertes ergeben sich die baryzentrischen Koordinaten unmittelbar.

Zur Reduktion der aufwändigen Multi-Pass-Rendering-Schritte können Kohärenzen in der Bewegung ausgenutzt werden, so dass eine komplette Neuberechnung der Cursor-Position nicht in jedem Frame nötig ist. Stattdessen wird die aktuelle Bewegung parallel zur aktuellen 3D-Cursor-Normalen auf die gesnappte Oberfläche projiziert [Wu et al. 2003] (siehe 3.3.2).

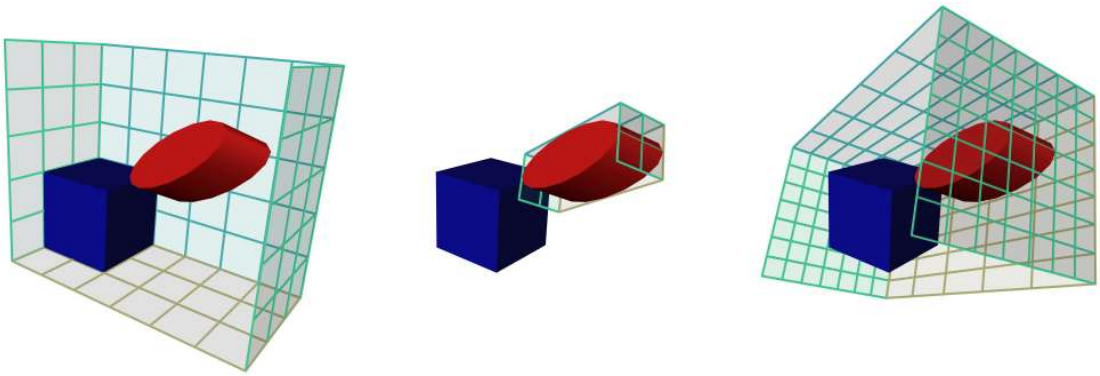

Abbildung 3.6: Interaktion mit der Cage. (a) Die Cage umschliesst die Szene mit dem Würfel als Referenzsystem. (b) Die Situation nach einer *Set-System*-Operation auf dem deformierten Zylinder. (c) *Surround*-Operation auf der gesamten Szene nun mit dem deformierten Zylinder als Referenzsystem. [Baudisch 1996]

3.2.3 The Cage

Nicht unerwähnt bleiben sollte eine interessante Idee zur Erweiterung des Grid-Snapping. Die *Cage* (Cubic Adaptive Grid) [Baudisch 1996] ist ein adaptives Gitter zur Objektausrichtung in 3D. Mit verschiedenen Kommandos kann die Cage um die aktuell selektierten Objekte ausgerichtet werden, die Unterteilungen in den Hauptachsen festgelegt werden und Objekte als das Gitter bestimmende Referenz-System gewählt werden (siehe Abbildung 3.6).

3.3 3D-Snapping mit 3D-Eingabegeräten

Die bislang vorgestellten 3D-Snapping-Verfahren arbeiten zwar im dreidimensionalen Raum, erwarten die Benutzereingaben jedoch über ein 2D-Eingabegerät wie beispielsweise eine Maus. Die Verwendung von 3D-Eingabegeräten erfordert neue Lösungen, da z. B. die Cursor-Positionierung und Selektion anders erfolgen muss als mit einfacher Projektion der 2D-Cursor-Position oder Ray-Casting-Techniken. Auf der anderen Seite ergeben sich daraus aber auch neue Möglichkeiten, da die Interaktion mit Eingabegeräten mit mehr Freiheitsgraden gerade bei der Erstellung von 3D-Modellen direkter und effizienter sein kann als Maus-basierte Interaktion [Butterworth et al. 1992].

3.3.1 Snap-to

Die *Snap-to*-Technik [Venolia 1993] erlaubt die präzise Ausrichtung von Objekten in 3D mit Hilfe eines 3D-Cursors, der mit 3D-Eingabegeräten (z.B. einer SpaceMouse [3Dconnexion]) gesteuert werden kann. Im Gegensatz zu Snap-Dragging ist Snap-to nicht auf Punkte beschränkt, sondern arbeitet auf Objekt-Ebene bzw. auf deren Oberflächen. Dem Verfahren liegt ein auf „magnetischer“ Anziehung zwischen Objekten beruhendes Modell zugrunde. Sobald

ein Objekt auf ein anderes zubewegt wird, wird es vom 3D-Cursor weg in dessen Richtung gezogen. Sobald die Objekte nahe genug beieinander liegen, wird das bewegte Objekt am anderen exakt ausgerichtet.

Ähnlich wie bei Snap-and-go mit direkten Eingabegeräten (siehe 3.1.5) erzeugt Snap-to bei Bedienung mit 3D-Eingabegeräten, die die Bewegung des Gerätes direkt übertragen, einen dauerhaften Versatz zwischen dem 3D-Cursor und dem bewegten Objekt, was für bestimmte Interaktionsformen problematisch sein kann [Dumas et al. 1998]. In Virtuellen Umgebungen führt ein Versatz sogar dazu, dass bei Spatial-Input-Interaktionstechniken wie z. B. Simple Dragging die Korrespondenz zwischen dem Eingabegerät und dem manipulierten Objekt verloren geht, wodurch ein indirekter Eindruck über die Objektbewegung beim Benutzer entstehen kann.

Da die Details des Verfahrens nicht erläutert werden, bleibt unklar, wie es sich in Regionen mit vielen Objekten bedienen läßt oder wie die Form der Objekte in die Berechnungen eingeht. Genauer geht Venolia allerdings auf Feedback während der Bewegung und des Snappings ein. Die Arbeit des Algorithmus wird dem Benutzer einerseits durch visuelle Markierungen wie eine Feder bei Objektanziehung und ein Gummiband beim Wegbewegen eines gesnappten Objekts verständlich gemacht. Zusätzlich wird akkustisches Feedback geboten, wenn Kräfte auf das bewegte Objekt wirken oder es gesnappt bzw. wieder losgelassen wird.

3.3.2 Snapping auf Konvexe Oberflächen

Ein Algorithmus zur präzisen Bewegung eines 3D-Cursors auf konvexen, geschlossenen und glatten Oberflächen wird in [Wu et al. 2003] vorgestellt. Das vorgestellte System läßt sich sowohl mit 2D- als auch mit 3D-Eingabegeräten bedienen, wobei die 2D-Interaktion auch hier mit Strahlen ausgehend von der Cursor-Position in der Kameraebene arbeitet, um einen 3D-Cursor in der Szene zu platzieren.

Sobald ein Objekt selektiert wurde und der Abstand zwischen dem 3D-Cursor und dem Objekt nicht zu groß ist, bleibt der 3D-Cursor auf der Oberfläche des Objekts gesnappt. Um aufwendige Nächster-Nachbar-Berechnungen für die Bestimmung des nächsten Dreieckes zum 3D-Cursor zu vermeiden, nutzt der Algorithmus die Normale an der letzten Cursor-Position auf der Oberfläche und Kohärenzen in der Bewegung aus, um den neuen Punkt auf dem Objekt zu bestimmen (siehe Abbildung 3.7).

Der Algorithmus erlaubt Snapping ausschliesslich auf konvexe, geschlossene und glatte Oberflächen. Sogar einfache geometrische Formen wie ein Würfel erfordern eine Sonderbehandlung der Kanten.

3.3.3 Snapping für die CAD-Modellierung

Für das Anwendungsfeld der CAD-Modellierung wird in [Stork u. Maidhof 1997; Stork 2000] ein Picking- und Snapping-Algorithmus vorgestellt, der mit einem 3D-Cursor und 3D-Eingabe-

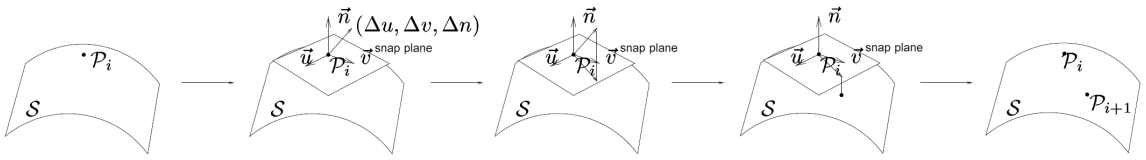

Abbildung 3.7: Snapping auf konvexe Oberflächen. Der Bewegungsvektor des 3D-Cursors $(\Delta u, \Delta v, \Delta n)$ im Snap-Referenz-Koordinatensystem des Punktes $\mathcal{P}_i$ wird parallel zur Normalen $\overrightarrow{n}$ an $\mathcal{P}_i$ auf $\mathcal{S}$ projiziert, um $\mathcal{P}_{i+1}$ zu bestimmen. [Wu et al. 2003]

geräten arbeitet. Der Cursor besteht aus einer halbtransparenten Kugel. Objekte werden selektiert und der Cursor auf Objekte gesnappt, wenn sie sich ganz oder teilweise innerhalb der Cursor-Geometrie befinden. Bei Mehrdeutigkeit wird nicht einfach das nächste Objekt ausgewählt, sondern wie in [Bier 1990] Vertices gegenüber Kanten und diese gegenüber Flächen bevorzugt. Eine eigene Priorisierung kann bei Bedarf vorgenommen werden.

Zur Vermeidung teurer Nächster-Nachbar-Tests werden hierarchische Inside-Bounding-Box-Tests mit erweiterten Bounding Boxen aller Objekte und, falls erfolgreich, mit erweiterten Bounding Boxen der einzelnen Flächen und Kanten des Objekts durchgeführt. Vertices können direkt getestet werden. Ein Element ist genau dann innerhalb der Kugel, wenn ihr Mittelpunkt innerhalb der Minkowski-Summe von Kugel und Bounding Box des getesteten Elementes ist. Da diese Berechnungen aufwendig sind, wird als Heuristik stattdessen die Bounding Box des Elementes um den Radius der Kugel erweitert und dann komponentenweise gegen die erweiterte Bounding Box getestet.

Um die Anzahl der Tests weiter zu verringen, wird der nächste Punkt nicht ständig neu bestimmt, sondern nur solange sich überhaupt Objektpunkte innerhalb der Cursor-Geometrie befinden. Daher findet kein dauerhaftes Update bei Entfernung vom Objekt statt.

3.3.4 IntenSelect

IntenSelect [de Haan et al. 2005] ist eine auf Ray-Casting basierende Technik für die Objektselektion in Virtuellen Umgebungen. Dabei wird um den vom Eingebegerät des Benutzers ausgehenden Selektionsstrahl ein (unsichtbares) vom Benutzer einstellbares kegelförmiges Volumen gelegt. Von allen darin befindlichen Objekten wird nun laufend mit Hilfe einer Bewertungs-Funktion ein Ranking berechnet. Das Objekt mit der höchsten Bewertung wird selektiert. Um die Auswahl für sich bewegende Objekte zu verbessern, wird die Bewertungsfunktion abhängig von der Bewegung des Benutzers modifiziert.

Um den Benutzer kontinuierliches Feedback über das aktuell aktive Objekt zu geben, wird der Selektionsstrahl zum Objekt hin gebogen (siehe Abbildung 3.8). Der ursprüngliche gerade Strahl wird zu besseren Orientierung weiterhin etwas schwächer in einer anderen Farbe gezeichnet. Falls der Benutzer mit dem Ranking unzufrieden ist, kann er den Strahl bewegen,

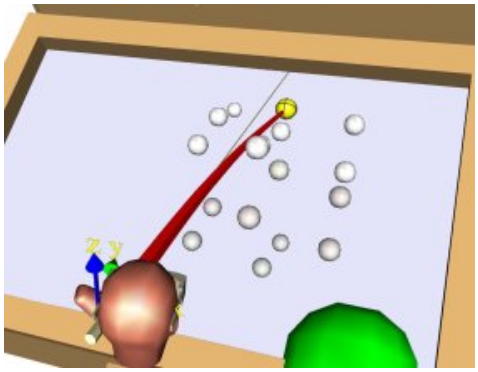

Abbildung 3.8: Selektion eines Objekts mit IntenSelect. Der Selektionsstrahl wird immer zum aktuell aktiven Objekt gebogen. Der eigentliche (gerade) Strahl wird zur Orientierung für den Benutzer weiterhin etwas schwächer dargestellt. [de Haan et al. 2005]

bis ein anderes Objekt selektiert wird.

IntenSelect ist ein reines Selektionsverfahren. Für die Manipulation des selektierten Objekts müssen andere Techniken benutzt werden. Insbesondere muss dabei auf das Benutzer-Feedback bzgl. des gebogenen Strahls eingegangen werden.

[Andujar et al. 2006] greifen die Idee des gebogenen Strahls von IntenSelect auf und verwenden einen solchen zur Unterstützung von Selektionen bei der Interaktion mit im 3D platzierten 2D-Widgets. Dabei snappt der Strahl auf das nächstliegende der aktiven Interaktionselemente (z. B. Buttons).

4 Translations-Snapping in Virtuellen Umgebungen

In diesem Kapitel werden Techniken zum Translations-Snapping für die Interaktion in Virtuellen Umgebungen vorgestellt. Ein Einsatzszenario ist Snapping eines 3D-Cursors auf relevante Punkte oder Objekte zur Vereinfachung von Selektionen. Vor allem aber bei Manipulationen in Virtuellen Umgebungen kann Snapping den Benutzer unterstützen, um Objekte präzise an bestimmten Positionen zu platzieren.

Die vorgestellten Techniken behandeln Snapping von Punkten bzw. des Translationsanteils von Objekttransformationen auf verschiedene Snapping-Ziele. Zunächst werden als Ziele Primitive wie Punkte, Geraden und Liniensegmente, Ebenen und konvexe Polygone betrachtet. Später wird ein Ansatz vorgestellt, der effizientes Snapping auch auf große Polygonmodelle erlaubt.

Bei der Entwicklung von Snapping-Techniken für den Einsatz in Virtuellen Umgebungen müssen die zugrundeliegenden Interaktionsparadigmen berücksichtigt werden. Der Fokus der vorgestellten Verfahren liegt auf direkter Interaktion mit absolut im Raum positionierten und orientierten 6DOF-Eingabegeräten. Dafür geeignete Interaktionstechniken sind die Virtual-Hand-Interaktion mittels eines 3D-Cusors, welcher die Position des Eingabegerätes direkt in die virtuelle Welt überträgt, wie auch insbesondere eine auf Ray-Casting-Selektion basierende Pick-Ray-Interaktion (z. B. Simple Dragging, Scaled Grab).

4.1 Standard-Snapping

Als Standard-Snapping-Techniken werden im Folgenden die vom Desktop bekannten und in Kapitel 3 vorgestellten traditionellen Snapping-Verfahren bezeichnet, welche ein bewegtes Objekt bzw. den Cursor innerhalb der Snap-Range stets auf einen Ziel-Punkt transformieren, wodurch ein Sprung beim Betreten und Verlassen dieses Bereichs erzeugt wird. Diese Unstetigkeit in der Interaktion kann schon am Desktop mit indirekten Eingabegeräten wie einer Maus störend wirken, deutlicher noch bei der Interaktion in Virtuellen Umgebungen, da hier der Benutzer wesentlich stärker in die dreidimensionale Welt eingebunden ist. Dieser als Immersion bezeichnete Effekt kann durch die Sprünge beim Standard-Snapping beeinträchtigt werden, da u. a. bei der stereoskopischen Darstellung der Fokus des Benutzers nicht mit der aufgrund der Stereo-Parallaxe wahrgenommenen Tiefe der Objekte übereinstimmt. Wird diese nun

sprunghaft verändert, kann der Benutzer kurzfristig irritiert werden und ggf. sogar schneller ermüden [Sherman u. Craig 2003; Kim 2005].

Ohne Snapping besteht sowohl bei vollkommen direkter als auch bei strahlbasierter Interaktion eine Korrespondenz zwischen dem Eingabegerät und dem bewegten Objekt, die verloren geht, wenn das Objekt innerhalb der Snap-Range auf das Ziel transformiert wird. Für den Benutzer in der Virtuellen Umgebungen kann dadruch ein indirekter Eindruck der Objektkontrolle während der ansonst direkten Interaktion entstehen. In Kapitel 6 wird genauer auf diese Herausforderung eingegangen und Lösungsansätze werden vorgestellt.

Beim Standard-Snapping ergibt sich mit einer Snap-Range $r \in \mathbb{R}$ und einer von dem jeweiligen Snapping-Ziel abhängigen Funktion $\nu : \mathbb{R}^3 \to \mathbb{R}^3$ zur Bestimmung des nächstliegenden Punktes auf dem Ziel-Objekt die Snap-Funktion (bzw. Gravity-Funktion [Sutherland 1963]) $\sigma : \mathbb{R}^3 \to \mathbb{R}^3$ für einen Punkt $\mathbf{q} \in \mathbb{R}^3$ zu

$$\sigma(\mathbf{q}) := \begin{cases} \nu(\mathbf{q}) & : \quad \delta\left(\mathbf{q}, \nu(\mathbf{q})\right) \leq r \\ \mathbf{q} & : \quad \text{sonst} \end{cases},$$

wobei $\delta : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^+$ die Funktion zur Bestimmung des euklidischen Abstands im $\mathbb{R}^3$ ist.

Bei der Definition von σ wurde von einem symmetrischen Verhalten des Snappings ausgegangen. Es ist aber ebenso möglich, die Snap-Ranges für den Eintritt und den Austritt beim Snapping unterschiedlich groß zu gestalten. So kann z. B. der Bereich, in dem Snapping begonnen wird, kleiner sein als der Bereich, der verlassen werden muss, um das Snapping wieder aufzuheben. Der umgekehrte Fall ist nicht ohne weitere Maßnahmen möglich, da man sich dann beim Verlassen eines kleineren Austrittsbereichs immer noch innerhalb des Eintrittsbereichs befindet und daher sofort wieder snappen würde. Eine Möglichkeit hierbei ist, dass man nach erfolgtem Snapping auch den Eintrittsbereich wieder verlassen muss, um erneut zu snappen. Alternativ kann auch gemessen werden, ob nach dem Verlassen des Austrittsbereichs die Bewegung wieder signifikant in Richtung des Ziel-Objekts geführt wird.

4.1.1 Primitive

In diesem Abschnitt werden 3D-Snapping-Ziel-Primitive vorgestellt, auf die der Cursor bzw. allgemein ein bewegtes Objekt snappen kann. Die vorgestellten Primitive sind ausreichend für viele Anwendungen und erlauben zudem, dass weitere Snapping-Ziel-Objekte aus ihnen zusammengesetzt werden können. Weitere Ziele können bei Bedarf analog definiert werden.

Als Primitive werden hier Punkte, Geraden und Liniensegmente, Ebenen und konvexe Polygone eingeführt, die beliebig in 3D platziert werden können. Die Snap-Ranges der Primitive sind in Abbildung 4.1 für den zweidimensionalen Fall skizziert. Sobald der betrachtete Punkt einer Objekttransformation sich innerhalb dieses Bereichs befindet, wird er auf den nächstliegenden Punkt auf dem Ziel-Primitiv transformiert. Die Skizze deutet wie schon oben erwähnt an, dass es natürlich möglich ist, unterschiedliche Entfernungen für das Betreten und das Verlassen des Snapping-Bereichs zu benutzen.

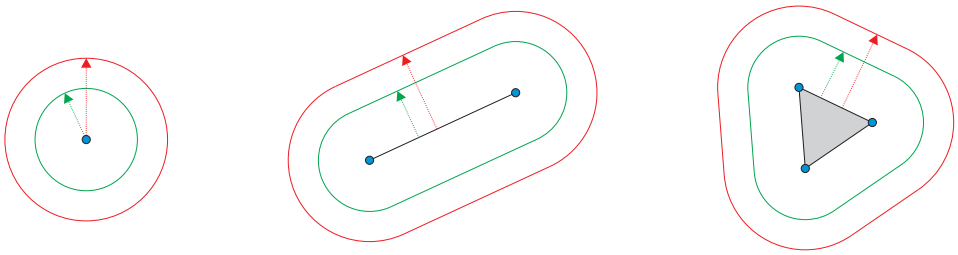

Abbildung 4.1: Snap-Ranges bei Snapping-Ziel-Primitiven. Grün umrandet ist hierbei der Eintrittsbereich, innerhalb dessen auf den nächsten Punkt auf dem Ziel gesnappt wird, rot umrandet ist der Austrittsbereich, außerhalb dessen bestehendes Snapping aufgehoben wird.

Punkte

Bei punktförmigen Snapping-Ziel-Primitiven ist der nächstliegende Punkt auf dem Ziel natürlich der Snapping-Punkt selbst. Daher ergibt sich für ein derartiges Ziel-Objekt $\mathbf{p} \in \mathbb{R}^3$:

$$\nu_p(\mathbf{q}) := \mathbf{p}$$

Geraden und Liniensegmente

Der nächste Punkt auf einer Gerade zu einem gegebenen Punkt $\mathbf{q}$ ist der Fußpunkt $\mathbf{x}_0$ des Lotes von $\mathbf{q}$ auf die Gerade, also die orthogonale Projektion. Für eine Gerade $g(t) = \mathbf{a} + t\mathbf{b}$ gilt somit (siehe auch Abbildung 4.2):

$$\nu_g(\mathbf{q}) := \mathbf{x}_0 = \mathbf{a} + t_0\mathbf{b}, \quad t_0 = \frac{(\mathbf{q} - \mathbf{a}) \cdot \mathbf{b}}{\mathbf{b}^2}$$

Der Parameter t_0 ergibt sich aus zwei Eigenschaften: Einerseits liegt der Punkt $\mathbf{x}_0$ auf der Geraden, d.h. es existiert ein Parameterwert t_0, so dass $\mathbf{x}_0 = \mathbf{a} + t_0\mathbf{b}$ gilt. Zum anderen steht das Lot $\mathbf{x}_0 - \mathbf{q}$ senkrecht auf dem Richtungsvektor $\mathbf{b}$, also gilt:

$$(\mathbf{x}_0 - \mathbf{q}) \cdot \mathbf{b} = 0 \quad \Leftrightarrow \quad \mathbf{x}_0 \cdot \mathbf{b} = \mathbf{q} \cdot \mathbf{b}$$

Nach Multiplikation von $\mathbf{x}_0 = \mathbf{a} + t_0\mathbf{b}$ mit $\mathbf{b}$ können die beiden Angaben wie folgt gleichgesetzt werden und t_0 anschließend durch Auflösen bestimmt werden:

$$\mathbf{x}_0 \cdot \mathbf{b} = \mathbf{q} \cdot \mathbf{b} = \mathbf{a} \cdot \mathbf{b} + t_0\mathbf{b}^2$$

Bei einer endlichen Geraden, also einem Liniensegment s zwischen zwei Punkten $\mathbf{a}_1$ und $\mathbf{a}_2$ kann der nächste Punkt entwededer einer der Randpunkte sein oder auf dem Geradenstück dazwischen liegen. Die Berechnung kann erfolgen, indem man zunächst eine auf dem Segment verlaufende Gerade g betrachtet und derart parametrisiert, dass $g(0) = \mathbf{a}_1$ und $g(1) = \mathbf{a}_2$ gilt:

$$g(\mathbf{x}) = \mathbf{a}_1 + t(\mathbf{a}_2 - \mathbf{a}_1)$$

Abbildung 4.2: Bestimmung des nächsten Punktes zu $\mathbf{q}$ auf Geraden und Ebenen.

Nach Berechnung des Lot-Fußpunktes von $\mathbf{q}$ auf g und des entsprechenden Geradenparameters t_0 ergibt sich leicht:

$$\nu_s(\mathbf{q}) := \begin{cases} \mathbf{a}_1 & : \quad t_0 \leq 0 \\ \mathbf{a}_2 & : \quad t_0 \geq 1 \\ \mathbf{a}_1 + t_0(\mathbf{a}_2 - \mathbf{a}_1) = \nu_g(\mathbf{q}) & : \quad \text{sonst} \end{cases}$$

Alternativ kann bei Liniensegmenten auch das Snapping aufgehoben werden, sobald $\mathbf{x}_0$ nicht mehr auf dem Liniensegment liegt. Dann gilt für die Bestimmung des nächsten Punktes auf dem Liniensegment:

$$\nu_s^*(\mathbf{q}) := \begin{cases} \nu_g(\mathbf{q}) & : \quad 0 \leq t_0 \leq 1 \\ \mathbf{q} & : \quad \text{sonst} \end{cases}$$

Ebenen und Konvexe Polygone

Die Bestimmung des nächsten Punktes auf einer Ebene verläuft analog zu den Berechnungen, die bei Geraden durchgeführt werden. Sei $e : \mathbf{n} \cdot \mathbf{x} = d$ eine in Hesse-Form gegebene Ebene in $\mathbb{R}^3$. Der zu einem Punkt $\mathbf{q}$ nächste Punkt auf der Ebene $\mathbf{x}_0$ ist der Fußpunkt des Lotes von $\mathbf{q}$ auf e. Das Lot verläuft bei einer Ebene natürlich in Richtung der Normalen. Insgesamt ergibt sich damit:

$$\nu_e(\mathbf{q}) := \mathbf{x}_0 = \mathbf{q} + t_0\mathbf{n}, \quad t_0 = \frac{d - \mathbf{n} \cdot \mathbf{q}}{\mathbf{n}^2}$$

Auch hier wird der Parameter t_0 der Lotgeraden durch zwei Angaben bestimmt: Der Endpunktes des Lotes liegt auf der Ebene, das heißt, es gilt $\mathbf{n} \cdot \mathbf{x}_0 = d$. Gleichzeitig liegt er natürlich auf der Lotgeraden, somit existiert ein Parameterwert t_0, für den $\mathbf{x}_0 = \mathbf{q} + t_0\mathbf{n}$ erfüllt ist. Nach Multiplikation mit $\mathbf{n}$ können die beiden Eigenschaften gleichgesetzt werden und t_0 anschließend durch Auflösen bestimmt werden:

$$\mathbf{n} \cdot \mathbf{x}_0 = d = \mathbf{n} \cdot \mathbf{q} + t_0\mathbf{n}^2$$

Zur Berechnung des nächsten Punktes auf einem konvexen Polygon können die entsprechenden Methoden für Ebenen und Segmente benutzt werden. Zunächst wird der nächste Punkt auf der durch das Polygon P definierten Ebene e bestimmt. Falls sich der resultierende Punkt

nicht bereits innerhalb des Polygons befindet, muss nun der nächste Punkt auf einer der Seiten berechnet werden. Diese Schritte können zusammengefasst werden, indem die zu den Seiten korrespondierenden Geraden analog zu oben parametrisiert werden und für jede Gerade der nächstliegende Punkt bestimmt wird. Falls die jeweiligen Geradenparameter für die nächsten Punkte alle zwischen 0 und 1 liegen, lag der Punkt bereits innerhalb des Polygons. Ansonsten ist das Ergebnis der nächste der bestimmten nächstliegenden Punkte auf den Seiten. Insgesamt ergibt sich

$$\nu_P(\mathbf{q}) := \begin{cases} \nu_e(\mathbf{q}) & : \quad \nu_e(\mathbf{q}) \text{ liegt innerhalb von } P \\ \arg\min_{\mathbf{n} \in \{\nu_s(\nu_e(\mathbf{q}));\, s \in \text{sides}(P)\}} \delta\left(\nu_e(\mathbf{q}), \mathbf{n}\right) & : \quad \text{sonst} \end{cases},$$

wobei arg min das Argument $\mathbf{n}$ aus der angegebenen Menge zurückgibt, das den kleinsten Wert bzgl. der angegebenen Funktion $\delta\left(\nu_e(\mathbf{q}), \mathbf{n}\right)$ hat.

4.1.2 Prioritäten

Bei der Auswahl, auf welches von mehreren aktiven Ziel-Objekten gesnappt werden soll, ist die einfachste Vorgehensweise, den minimalen Abstand zu allen Zielen zu bestimmen und dann auf dasjenige mit dem nächstliegenden Punkt zu snappen. Dies ist für viele Fälle ausreichend, es gibt aber Anwendungen, bei denen eine intelligentere Auswahl nötig ist.

Wie schon beim Snap-Dragging (siehe 3.1.2) kann es wünschenswert sein, bestimmten Klassen oder konkreten Instanzen von Ziel-Objekten unterschiedliche Prioritäten zuzuweisen. Dadurch kann Snapping auf höher priorisierte Ziele gegenüber dem auf Ziele mit geringerer Priorität bevorzugt werden, auch wenn erstere weiter entfernt sind. So können z. B. Objekte exakt entlang einer Linie bewegt und zusätzlich bestimmte Punkte auf der Linie besonders ausgezeichnet werden, indem mit höherer Priorität auf sie gesnappt wird (z. B. die Endpunkte bei einem Liniensegment oder Punkte für bestimmte Werte bei einem Schieberegler in 3D-Menüs).

Während auf ein konkretes Ziel-Objekt gesnappt wird, bestehen mehrere Möglichkeiten bei der Interaktion mit anderen Zielen. So kann das aktuelle Snapping-Ziel verlassen werden, sobald ein anderes gleichwertig priorisiertes Ziel näher liegt oder man in die Snap-Range eines Ziels mit höherer Priorität eintritt. Eine feste Bevorzugung des aktuellen Ziels entspricht einer temporären Erhöhung seiner Priorität auf den Maximalwert. Dadurch wird das Snapping in jedem Fall erst bei Verlassen siner Snap-Range gelöst.

Bei den bisherigen Überlegungen wurde stets davon ausgegangen, dass gleichzeitig nur auf ein Ziel-Objekt gesnappt wird. Dabei ist es möglicherweise wünschenswert, zur selben Zeit auf mehrere Ziele zu snappen, wenn sie eine nichtleere Schnittmenge besitzen. Dieses Verhalten kann aber ebenso erreicht werden, wenn zu einem Zeitpunkt nur auf ein einziges Ziel gesnappt werden kann. Dazu können von Hand oder automatisch aus den Schnittmengen der beteiligten Ziele (gleicher Priorität) neue Ziel-Objekte erzeugt werden, die eine höhere Priorität als die

ursprünglichen Objekte besitzen. Die Art der Schnittmenge hängt natürlich von den beteiligten Klassen der Zielprimitive ab (z. B. Schnittpunkte bei Geraden oder Liniensegmenten).

4.2 Snap-and-go

Wie bereits erläutert, hat das traditionelle Snapping mit einfacher Verschiebung des Punktes auf das Ziel innerhalb der Snap-Range zur Folge, dass dieser Bereich nicht mehr erreichbar ist. Abgesehen davon, dass die erzeugte Unstetigkeit den Grad der Immersion bei Interaktion in Virtuellen Umgebungen verringern kann, kann es bei einer Applikation unter Umständen überhaupt nicht möglich oder erwünscht sein, dass der Benutzer z. B. mit Hilfe eines Menüs die Snapping-Funktionalität deaktivieren kann, um doch einmal ein Objekt innerhalb der Snap-Range aber nicht genau auf dem Snapping-Ziel platzieren zu können. Eine Verkleinerung der Snap-Range würde die Größe des unerreichbaren Raums zwar reduzieren, erschwert auf der anderen Seite aber auch das Treffen des Ziels.

Eine Möglichkeit zur Lösung dieser Herausforderung für 2D-Interaktion mit Widgets auf dem Computer-Desktop bietet die in [Baudisch et al. 2005] vorgestellte Snap-and-go-Technik (siehe 3.1.5). In diesem Abschnitt werden Verfahren vorgestellt, die es ermöglichen, Snap-and-go für Spatial-Input-Interaktion in Virtuellen Umgebungen zu verwenden.

Snap-and-go vermeidet einen Eintrittsbereich für das Snapping. Stattdessen wird das bewegte Objekt bei Überquerung eines Snapping-Ziels solange an diesem festgehalten, bis die Distanz zwischen der Position auf dem Snapping-Ziel und der aktuellen Originalposition eine gewisse Größe überschreitet. Letzteres entspricht natürlich dem Austrittsbereich bei Standard-Snapping-Verfahren. Das ursprüngliche Snap-and-go arbeitet nur korrekt mit indirekten Eingabegeräten wie z. B. einer Maus, wo kein absoluter Bezug zwischen der tatsächlichen Bewegung des Eingabegerätes und der Cursor-Bewegung besteht. Für Spatial-Input-Interaktion in Virtuellen Umgebungen ergeben sich daher neue Herausforderungen. Insbesondere beim Verlassen eines Snapping-Ziels müssen alternative Ansätze entwickelt werden, da mit Snap-and-go dabei eigentlich ein dauerhafter Versatz (*Offset*) entsteht.

4.2.1 Eintritt

Zunächst betrachten wir Snap-and-go-Snapping auf Ebenen und konvexen Polygonen. Zur Vermeidung von unerreichbaren Bereichen wird nicht mit einer Eintritts-Snap-Range gearbeitet, sondern das bewegte Objekt erst bei Zielüberquerung an diesem festgehalten und zu snappen begonnen. Während man gesnappt ist, wird das Objekt ebenso wie bei Standard-Snapping immer auf den aktuell nächsten Punkt auf dem Ziel verschoben.

Die Überquerung eines Zieles kann festgestellt werden, indem Schnitttests zwischen dem Liniensegment, das von der Position $\mathbf{q}_1$ des bewegten Objekts im letzten Bild zu seiner aktuellen Position $\mathbf{q}_2$ verläuft, und den Snapping-Ziel-Objekten durchgeführt werden. Zusätzlich wird zu snappen begonnen, wenn der Abstand zum Ziel Null ist (bzw. kleiner ε aufgrund von

möglichen Gleitkomma-Ungenauigkeiten). So kann auch auf Oberflächen gesnappt werden, die aufgrund anderer Einschränkungen nicht durchdrungen werden dürfen (z. B. ein Boden oder eine Wand).

Die Berechnung des Schnittpunktes des Liniensegments von $\mathbf{q}_1$ nach $\mathbf{q}_2$ mit einer Ebene $e : \mathbf{n} \cdot \mathbf{x} = d$ (in Hesse-Form) kann ähnlich wie die Nächster-Punkt-Bestimmung auf Ebenen und Geraden ohne Lösen eines linearen Gleichungssystems durchgeführt werden. Zunächst betrachten wir die zum Liniensegment korrespondierende Gerade g, die derart parametrisiert wird, dass $g(0) = \mathbf{q}_1$ und $g(1) = \mathbf{q}_2$ gilt, also:

$$g(\mathbf{t}) = \mathbf{q}_1 + t(\mathbf{q}_2 - \mathbf{q}_1)$$

Voraussetzung für die Existenz eines Schnittpunktes ist, dass die Gerade nicht parallel zur Ebene verläuft, also $\mathbf{n} \cdot (\mathbf{q}_2 - \mathbf{q}_1) \neq 0$ ist. Falls diese Bedingung erfüllt ist, ergibt sich der Schnittpunkt $\mathbf{x}_0$ aus zwei Eigenschaften. Einerseits liegt er auf der Geraden, es existiert also ein Parameterwert t_0, so dass $\mathbf{x}_0 = \mathbf{q}_1 + t_0(\mathbf{q}_2 - \mathbf{q}_1)$. Desweiteren liegt er natürlich ebenso auf der Ebene, also gilt $\mathbf{n} \cdot \mathbf{x}_0 = d$. Nach Multiplikation der ersten Gleichung mit $\mathbf{n}$, Gleichsetzen und Auflösen nach t_0 ergibt sich:

$$\mathbf{x}_0 = \mathbf{q}_1 + t_0(\mathbf{q}_2 - \mathbf{q}_1), \quad t_0 = \frac{d - \mathbf{n} \cdot \mathbf{q}_1}{\mathbf{n} \cdot (\mathbf{q}_2 - \mathbf{q}_1)}$$

Der Schnittpunkt zwischen der Geraden und der Ebene liegt nur auf dem Liniensegment, falls $0 \leq t_0 \leq 1$ erfüllt ist.

Für die Schnittpunktberechnung zwischen dem Liniensegment und einem konvexen Polygon kann man zunächst die Schnittpunktberechnung mit der durch das Polygon definierten Ebene durchführen. Falls ein Schnittpunkt auf der Ebene gefunden wurde, kann z. B. wie in 4.1.1 festgestellt werden, ob der Punkt innerhalb des Polygons liegt.

4.2.2 Guiding

Bei Geraden und Punkten als Snapping-Ziele ist ähnlich wie in 2D bei punktförmigen Zielen eine Herausforderung, dass diese Ziele in zwei bzw. bei Punkten sogar in drei Dimensionen keine Ausdehnung besitzen. Die meisten Geraden haben also keinen Schnittpunkt mit diesen Elementen. Für Interaktion in Virtuellen Umgebungen bedeutet dies, dass eine Bewegung eigentlich immer an solchen Zielen vorbeiführt, so dass Snapping ohne Eintrittsbereich nicht ohne Weiteres möglich ist (vgl. dazu Abbildung 3.4 für den 2D-Fall). Analog zum im 2D eingeführten (unsichtbaren) Cross-Widget können aber ähnliche Widgets in 3D definiert werden, um auch hier Snap-and-go auf Punkte, Geraden und Liniensegmente zu ermöglichen (siehe Abbildung 4.3).

Das Widget für Snapping auf Geraden in 3D besteht aus zwei orthogonal um die Gerade angeordneten Rechtecken, auf die gesnappt wird, um die Bewegung daraufhin zur Gerade zu leiten (Guiding). Die Schnittkante der Rechtecke ist die eigentlich gewünschte Snapping-Gerade. Bei Überquerung einer der Ebenen wird das Objekt darauf festgehalten und (mit dem

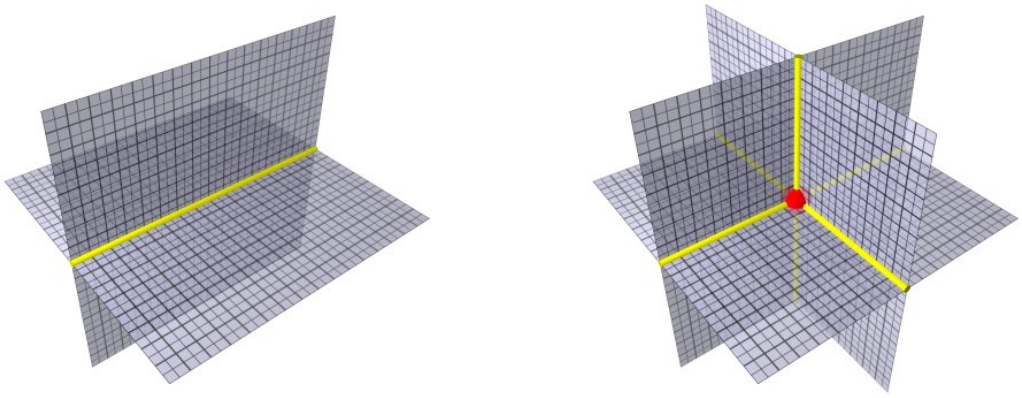

Abbildung 4.3: (Eigentlich unsichtbare) 3D-Guiding-Widgets, um Snap-and-go-Snapping auf Geraden und Liniensegmente (links) sowie Punkte (rechts) zu ermöglichen.

Snap-and-go-Verfahren) zu snappen begonnen. Solange man innerhalb ihrer Snap-Range ist, findet die weitere Bewegung findet daher in der Ebene statt, so dass, wenn gewünscht, auch die Gerade exakt überquert werden kann und man so auf sie snappen kann. Um den Übergang von der Ebene auf die Gerade zu ermöglichen, bekommt letztere eine höhere Snapping-Priorität zugewiesen.

Um Snap-and-go auf Punkte in 3D zu ermöglichen, wird das Guiding-Widget für Geraden um ein weiteres Rechteck und drei Liniensegmente erweitert. Die Rechtecke sind dabei orthogonal angeordnet und schneiden sich im gewünschten Snapping-Ziel-Punkt. Die Liniensegmente ergeben sich aus den Schnitten der Ebenen. Dabei wird diesen ähnlich wie beim Widget für Geraden eine höhere Snapping-Priorität zugewiesen. Die höchste Priorität bekommt der gewünschte Snapping-Ziel-Punkt.

Bei der Wahl der Größe und der Breite der Austritts-Snap-Range der Guiding-Objekte stehen sich einerseits die Einfachheit des Snappings auf das eigentlich Ziel-Objekt und auf der anderen Seite gegenüber, dass sonstige Bewegungen, die ein anderes Ziel haben, nicht gestört werden sollen. Die genauen Parameter müssen im Kontext der Anwendung und abhängig von der Dichte der Objekte definiert werden.

Im Übrigen kann es vorteilhaft sein, die Guiding-Elemente zu deaktivieren, sobald man auf das gewünschte Ziel gesnappt ist. Dadurch wird verhindert, dass sofort wieder auf sie gesnappt wird, wenn man beim Verlassen des Ziel-Objekts einen kleinen Abstand zu ihnen hat, wie bei Snap-and-go möglich (siehe 4.2.3). Die Reaktivierung kann erfolgen, wenn man eine gewisse (möglicherweise kleine) Distanz zum Snapping-Ziel überschritten hat.

4.2.3 Austritt

Snapping auf ein Ziel wird beim Snap-and-go-Verfahren begonnen, sobald es während der Bewegung überquert wurde oder der Abstand kleiner ε ist. Erst wenn eine bestimmte Distanz

Abbildung 4.4: Beim Beenden des Snappings auf ein Ziel (hier ein punktförmiges Ziel $\mathbf{p}$) mit der Snap-and-go-Technik entsteht bei Interaktion mit direkten Eingabegeräten ein Offset $\mathbf{o}$. $\mathbf{q}$ bezeichnet hier die ursprüngliche Objektposition und $\mathbf{q}'$ die Position nach Addition des Offsets. Die Snap-Range des Ziel-Objekts ist rot markiert.

analog zur Snap-Range bei Standard-Snapping überschritten wurde, wird das Snapping wieder aufgehoben. Allerdings wird das bewegte Objekt dann nicht einfach wieder an die durch das Eingabegerät und die Interaktionstechnik vorgegebene Position verschoben, sondern zur Vermeidung eines Sprungs ein Versatz (Offset) zur Originalposition erzeugt.

Die Länge des entstehenden Offset-Vektors $\mathbf{o}$ entspricht genau der Breite des Austrittsbereichs. Genauer ergibt er sich aus der aktuellen Originalposition $\mathbf{q}$ und dem dazu nächstliegenden Punkt auf dem Snapping-Ziel $\mathbf{p} := \nu(\mathbf{q})$ sowie der Austritts-Snap-Range r (siehe Abbildung 4.4)

$$\mathbf{o} := r\frac{\mathbf{p}-\mathbf{q}}{|\mathbf{p}-\mathbf{q}|}$$

Die Summe aus Offset-Vektor und Originalposition $\mathbf{q}' := \mathbf{q} + \mathbf{o}$ ersetzt bei allen weiteren Betrachtungen (also auch bei folgenden Snapping-Entscheidungen) die ursprüngliche Position. Dabei muss beachtet werden, dass für die neue Position damit die Distanz zum soeben verlassenen Snapping-Ziel kleiner oder gleich ε sein kann, wenn $r \leq |\mathbf{p}-\mathbf{q}| \leq r+\varepsilon$ ist. Daher muss für das aktuelle Ziel ähnlich wie beim Standard-Snapping mit unterschiedlichen Eintritts- und Austritts-Snap-Bereichen (siehe 4.1) das Snapping auf das Ziel temporär deaktiviert werden, solange man sich in dessen direkter Nähe befindet.

Durch den bei Snap-and-go entstehenden Offset, der die Bewegung effektiv an den Snapping-Zielen temporär abbremst, geht die Korrespondenz zwischen dem bewegten Objekt und dem Eingabegerät dauerhaft verloren. Bei einer strahlbasierten Interaktion führt dies z. B. dazu, dass der Strahl auch nach beendetem Snapping am Objekt vorbeizeigen kann. Durch mehrfaches Snapping und dadurch entstehendes Abbremsen in einer Richtung kann bei direkter Interaktion sogar schnell die Reichweite (Armlänge) des Benutzers erschöpft sein. Ohne weitere Eingriffe kann der Offset erst durch „Ablegen“ (z. B. durch Loslassen des entsprechenden Knopfes am Eingabegerät) und erneute Selektion des Objekts neutralisiert werden. Dabei erfordert die Neuselektion aufgrund der fehlenden Objekt-Eingabegerät-Korrespondenz auch eine Neuorientierung des Benutzers. Ablegen und Neuselektion sind allerdings nicht möglich, wenn das bewegte Objekt der 3D-Cursor selbst ist, weshalb hier in jedem Fall geeignte Maßnahmen ergriffen werden müssen.

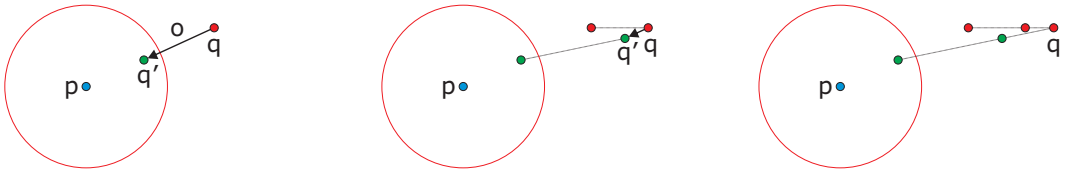

Abbildung 4.5: Korrektur des bei Snap-and-go nach Verlassen eines punktförmigen Snapping-Ziel entstehenden Offsets **o**. Die Korrektur erfolgt abhängig von der auf **o** projizierten Bewegung des Originalpunktes **q**. Zum besseren Verständnis sind die Bewegungen von **q** und des resultierenden Punktes **q**′ eingezeichnet.

Eine Möglichkeit zur Vermeidung dauerhafter Offsets bei Snap-and-go in Virtuellen Umgebungen besteht darin, einmal entstandene Offsets nachträglich wieder zu korrigieren. Eine einfache Idee ist, den Offset-Vektor über eine gewisse Zeitspanne immer kleiner werden zu lassen, bis die Korrespondenz zwischen dem Eingabegerät des Benutzers und dem bewegten Objekt schließlich wieder hergestellt ist. Dies hat allerdings den Effekt, dass sich das Objekt auch dann bewegt, wenn es vom Benutzer nicht gewünscht ist. Damit dieses Verfahren nicht einfach nur eine animierte Variante des Standard-Snappings ist, dürfte diese Korrektur im Übrigen erst nach Überschreiten einer gewissen Distanz zum Snapping-Ziel beginnen. Ansonsten würde das Objekt sofort nach Verlassen des Snapping-Ziels unweigerlich zur eigentlichen Position driften, ohne das der Benutzer dies verhindern könnte.

Besser als die Reduktion des Offsets über eine vorgegebene Zeitspanne ist, eine Korrektur abhängig von der originalen Bewegung des Objekts durchzuführen. Die einfachste Variante betrachtet den aktuellen Objekt-Bewegungsvektor **v** seit dem letzten Bild. Abhängig von dessen Länge wird nun der Offset-Vektor **o** verkleinert, um den neuen Offset $\mathbf{o}_{\text{new}}$ zu bestimmen. Dabei kann der korrigierte Anteil durch einen Faktor $\alpha \geq 0$ festgelegt werden:

$$\mathbf{o}_{\text{new}} := \begin{cases} \left(1 - \frac{\alpha|\mathbf{v}|}{|\mathbf{o}|}\right)\mathbf{o} & : \quad \alpha|\mathbf{v}| < |\mathbf{o}| \\ \mathbf{0} & : \quad \text{sonst} \end{cases}$$

Falls $\alpha = 0$ ist, findet keine Korrektur statt. Für $\alpha = 1$ wird eine Korrektur in der Größenordnung der eigentlichen Objektbewegung durchgeführt.

Dieses Verfahren zur Offset-Korrektur verursacht eine Beschleunigung der Objektbewegung (relativ zu eigentlichen Bewegung) bei Bewegung entgegen der Richtung des Offsets. Eine Bewegung in Offset-Richtung wird andererseits gebremst. Die Stärke der Korrektur kann durch Wahl des Faktors α bestimmt werden. Es ist wünschenswert, diesen Faktor abhängig von der Bewegungsrichtung relativ zum Offset-Vektor zu variieren. So wäre es beispielsweise möglich, dass sich das manipulierte Objekt bei Bewegung in Richtung des Offset-Vektors überhaupt nicht bewegt, bis der Offset verschwunden ist. Dies könnte durch eine volle Korrektur ($\alpha = 1$) in Offset-Vektor-Richtung erreicht werden.

Um für diese volle Korrektur in Richtung des Offsets den korrigierten Wert (bei nur

ungefährer Bewegung in Offset-Richtung) nicht zu überschätzen, darf nur die Projektion des Bewegungsvektors auf die Offset-Vektor-Richtung betrachtet werden. Auch für andere Bewegungen ist es intuitiver, wenn nur dieser Anteil betrachtet wird. Ansonsten würde sich bei Bewegung orthogonal zum Offset-Vektor das Objekt diagonal in Richtung des aktuellen Originalpunktes bewegen, was nicht der durch den Benutzer vorgegebenen Richtung entspricht. Die Projektion der Bewegung auf den Offset-Vektor ergibt sich zu:

$$\mathbf{v}_o := \frac{\mathbf{o} \cdot \mathbf{v}}{\mathbf{o}^2} \cdot \mathbf{o}$$

Für den Faktor α kann dann folgende Unterscheidung gemacht werden, um unterschiedliche Korrekturfaktoren für Bewegungen in Richtung oder entgegen der Richtung des Offset-Vektors zu benutzen:

$$\alpha := \begin{cases} \alpha_1 & : \quad \mathbf{v}_o \cdot \mathbf{o} > 0 \\ \alpha_2 & : \quad \mathbf{v}_o \cdot \mathbf{o} < 0 \\ 0 & : \quad \mathbf{v}_o \cdot \mathbf{o} = 0 \end{cases}$$

Insgesamt gilt damit bei diesem verbesserten Verfahren für den neuen Offset-Vektor $\mathbf{o}^*_{\text{new}}$ (siehe Abbildung 4.5):

$$\mathbf{o}^*_{\text{new}} := \begin{cases} \left(1 - \frac{\alpha|\mathbf{v}_o|}{|\mathbf{o}|}\right)\mathbf{o} & : \quad \alpha|\mathbf{v}_o| < |\mathbf{o}| \\ \mathbf{0} & : \quad \text{sonst} \end{cases}$$

Um schnelle Objektbewegung nicht zu stören, bietet sich insbesondere bei Snap-and-go wegen des auch nach dem Snapping zeitweise noch bestehenden Offsets die Verwendung eines *Thresholds* für Snapping an: So kann z. B. auf Snapping verzichtet werden, wenn die Geschwindigkeit der aktuellen ungesnappten Objektbewegung einen bestimmten Wert überschreitet. Die Verwendung dieser Heuristik erscheint sinnvoll, da Benutzer bei schneller Bewegung in der Regel auch kein Snapping wünschen, sondern dadurch tatsächlich eher irritiert werden.

4.2.4 Varianten

Zur Vereinfachung des Snap-and-go-Verfahrens für Interaktion in Virtuellen Umgebungen können auch nur einzelne Teile davon benutzt werden. Um z. B. Offsets nach dem Snappen vollständig zu vermeiden, kann nach Verlassen der Snap-Range wie bei Standard-Snapping wieder auf den Originalpunkt gesprungen werden. Zum Beginn des Snappings kann wie beim vollständigen Snap-and-go-Verfahren ohne ausgedehnten Eintrittsbereich und mit Guiding für Linien und Punkte gearbeitet werden.

Andererseits ist es ebenso möglich, mit einem expliziten Eintrittsbereich wie beim Standard-Snapping zu arbeiten und nur beim Verlassen des Austrittsbereichs Snap-and-go inklusive der Erzeugung eines Offsets zu benutzen. Damit dabei aber nicht sofort wieder auf das soeben verlassene Ziel gesnappt wird, wenn man sich noch mit dem hinzugerechneten Offset innerhalb

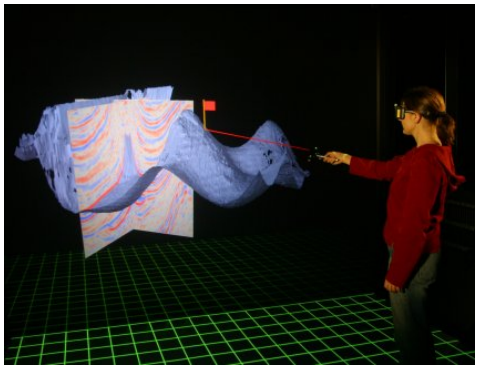

Abbildung 4.6: Eine Benutzerin platziert mit Hilfe von Snapping eine Markierung exakt auf einem triangulierten *Horizon* (vorwiegend horizontale seismische Grenzfläche).

des Eintrittsbereichs befindet, müssen hier dieselben Maßnahmen wie für unterschiedliche Snap-Ranges bei Standard-Snapping ergriffen werden (siehe 4.1).

Die Verwendung jeder dieser beiden Möglichkeiten vereinfacht zwar die konkrete Umsetzung des Snapping-Verfahrens, hat aber den Nachteil, dass nun entweder beim Beginn oder beim Beenden des Snappings auf ein Ziel ein Sprung erzeugt wird. Im Übrigen arbeiten beide Verfahren jeweils unterschiedlich bei Eintritt und Austritt, wodurch das Verhalten für den Benutzer weniger nachvollziehbar und intuitiv ist als beim vollständigen Snap-and-go.

4.3 Snapping auf Polygonmodelle

Schon ohne weitere Anpassungen kann grundsätzlich mit den bis jetzt vorgestellten Snapping-Techniken auf aus Polygonen bzw. Dreiecken bestehende Modelle gesnappt werden. Allerdings werden dabei die Polygone einzelnen betrachtet, was nur bei Verfahren, die ohne explizite Zustandsinformationen auskommen, zum gewünschten Ergebnis führt. Dies gilt bei den vorgestellten Verfahren nur für Standard-Snapping mit identischen Eintritts- und Austritts-Snap-Ranges.

Falls das benutzte Snapping-Verfahren zustandsabhängig arbeitet (wie z. B. Snap-and-go), ist es notwendig, das Modell als logisch zusammenhängendes Snapping-Ziel zu betrachten, um den Snapping-Zustand einmalig zu speichern. Für die Bestimmung des nächsten Punktes auf dem Modell kann allerdings auf die entsprechenden Berechnungen für die einzelnen Polygone zurückgegriffen werden.

Beim Standard-Snapping wird der aktuelle Snapping-Zustand nur benötigt, falls die Ranges für Eintritt und Austritt aus dem Snapping unterschiedlich groß sind. Das Snap-and-go-Verfahren ist hingegen in jedem Fall zustandsabhängig. Im ungesnappten Zustand wird dabei für jedes Polygon des Snapping-Ziels einzeln getestet, ob sich das bewegte Objekt darauf befindet oder es seit dem letzten Bild überquert hat. Während man auf das Ziel gesnappt ist, wird bei Standard-Snapping wie auch bei Snap-and-go für jedes einzelne Polygon der jeweils

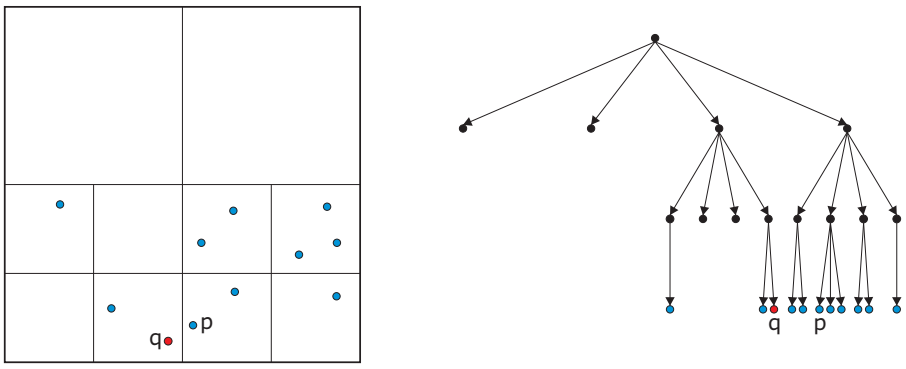

Abbildung 4.7: Nächster-Nachbar-Suche in einem Quadtree: Der nächste Nachbar **p** zum Punkt **q** befindet sich nicht zwangsläufig in derselben Zelle oder im gleichen Ast der Raumunterteilung wie **q**

nächste Punkt bestimmt und das bewegte Objekt schließlich auf den Nächsten dieser Punkte verschoben (siehe Abbildung 4.6).

4.4 Effiziente Nächster-Nachbar-Suche

Bei den vorgestellten Snapping-Techniken ist die Bestimmung des nächsten Punktes auf dem Snapping-Ziel-Objekt essentiell. Bei Standard-Snapping findet sie bei jeder Bewegung des zu snappenden Objekts statt, also unabhängig vom aktuellen Status. Techniken, die erst bei Zielüberquerung snappen, bestimmen den nächsten Punkt nur im gesnappten Zustand, dann aber ebenso bei jeder Bewegung auf der Oberfläche des Zielobjekts.

Naive Bestimmung des nächsten Punktes auf allen Primitiven beeinträchtigt schon bei aus mehreren Tausend Polygonen bestehenden 3D-Modellen die Interaktivität der Applikation. Es muss auch berücksichtigt werden, dass bei mehreren gleichzeitig bewegten Objekten wie z. B. in Mehrbenutzersystemen [Simon u. Scholz 2005; Riege et al. 2006] mehrere dieser Berechnungen pro Bild nötig sein können. Daher wird ein effizientes Verfahren benötigt, welches auch bei großen Modellen stets interaktive Bildraten ermöglicht. In diesem Abschnitt stellen wir einen Algorithmus vor, der eine Nächster-Nachbar-Suche mit Hilfe eines Octrees durchführt. Die Basis bildet der in [Hjaltason u. Samet 1999] präsentierte Nächster-Nachbar-Algorithmus.

Eine Herausforderung bei hierarchischer Nächster-Nachbar-Suche im Vergleich zu anderen räumlichen Fragestellungen ist, dass der nächste Nachbar nicht unbedingt im selben Ast der Raumunterteilung liegt und daher keine abschließenden lokalen Entscheidungen möglich sind. Abbildung 4.7 verdeutlicht dies am Beispiel eines Quadtrees. Dasselbe gilt auch für andere Datenstrukturen wie Octrees, BSP-Bäume und R-Bäume. Die zugrundeliegenden Datenstrukturen müssen weiterhin berücksichtigen, dass Datenelemente nicht nur Punkte sein können, sondern selbst eine räumliche Ausdehnung haben können.

Eine einfache Tiefensuche oder Traversierung der räumlichen Datenstruktur evtl. mit nach

Abstand sortierter Betrachtung der Kinder des aktuellen Knotens wie in [Roussopoulos et al. 1995] kann daher ineffizient sein. Nachdem man beispielsweise einen Blattknoten besucht hat, werden danach alle Geschwister und Geschwister der Elternknoten betrachtet (unter Umständen zurück bis zur Wurzel), bis man schließlich die Knoten aus einem anderen Ast besucht und dort vielleicht den nächsten Nachbarn findet.

4.4.1 Tiefensuche-Algorithmus

Für die Durchführung hierarchischer Nächster-Nachbar-Suche sind Abschätzungen über Abstände zu Begrenzungsregionen der enthaltenen Datenelemente (wie AABB, OBB, k-DOP oder Bounding-Spheres etc.) notwendig. Schon bei einfacher hierarchischer Suche wird der minimale Abstand benötigt, den ein Element aus der betrachteten Region zum Anfragepunkt haben kann. So kann frühzeitig entschieden werden, ob ein Teilbaum überhaupt betrachtet werden muss.

Es wird angenommen, dass jeder Knoten aus der hierarchischen Raumunterteilung assoziierte Datenelemente und Kindknoten besitzen kann, die vollständig in der Begrenzungsregion des Elternknotens liegen (siehe Abbildung 4.8). Weitere Annahmen über die Art der Regionen werden nicht gemacht, d. h. sie können sich z. B. auch überlappen.

Eine einfache hierarchische Nächster-Nachbar-Suche führt eine Tiefensuche angefangen beim Wurzelknoten durch (siehe Abbildung 4.9). Zunächst werden für den aktuellen Knoten die Abstände zu evtl. assoziierten Datenelementen bestimmt, bevor die Kindknoten betrachtet werden. Dabei werden nur solche Kinder besucht, die überhaupt Datenelemente enthalten können, welche näher zum Anfragepunkt liegen als bereits gefundene Elemente. Das bisher gefundene Datenelement mit dem kürzesten Abstand zur Anfrage bestimmt diese sogenannte *Pruning Distanz*, die zu Anfang auf $+\infty$ initialisiert wird. Sind schon vor Beginn der Suche einzelne Punkte $\mathbf{p}_i, i \in \mathbb{I}$ und deren Abstände zum Anfragepunkt $\mathbf{q}$ bekannt, kann die Pruning Distanz auf $\min(\delta(\mathbf{q}, \mathbf{p}_i), i \in \mathbb{I})$ initialisiert werden. Alternativ kann sie auch dazu benutzt werden, um nur nächste Nachbarn mit einem vorgegebenen maximalen Abstand zu finden.

Das der einfachen hierarchischen Nächster-Nachbar-Suche zugrundeliegende Abstandsmaß zwischen dem Anfragepunkt $\mathbf{q}$ und einer Begrenzungsregion R ist die sog. minDist, welche den minimalen Abstand angibt, den ein Punkt aus der Region zum Anfragepunkt haben kann. Befindet sich $\mathbf{q}$ innerhalb von R ist dieser Abstand natürlich 0. Allgemein ergibt sich bei Verwendung von AABBs als Begrenzungsregionen die minimale Distanz zu der durch die Punkte $\mathbf{s}$ und $\mathbf{t}$ begrenzten Region R im $\mathbb{R}^d$ zu

$$\text{minDist}(\mathbf{q}, R)^2 := \sum_{k=1}^{d} |q_k - m_k|^2 \ , \qquad m_k = \begin{cases} s_k & : \quad q_k < s_k \\ t_k & : \quad q_k > s_k \\ q_k & : \quad \text{sonst} \end{cases} ,$$

wobei o. B. d. A. $s_k \leq t_k$ gilt. Für andere Begrenzungsregionen wie z. B. Bounding Spheres lassen sich entsprechende Funktionen aufstellen.

```
class Node
{
  Bound bound;

  vector<Node> children;
  vector<Element> data;
}
```

Abbildung 4.8: Klassentyp für Knoten bei hierarchischer Nächster-Nachbar-Suche

```
Element closestElement;
float pruningDist = max<float>();

Element simpleNearestNeighborSearch(Node currentNode, Point query)
{
  // check associated data elements
  for (int i=0; i<currentNode.data.count(); ++i) {
    Element currentElement = currentNode.data[i];
    float currentDist = dist(query, currentElement);
    if (currentDist < pruningDist) {
      pruningDist = currentDist;
      closestElement = currentElement;
    }
  }

  // check children
  for (int i=0; i<currentNode.children.count(); ++i) {
    Node currentChild = currentNode.children[i];
    if (minDist(query, currentChild.bound) < pruningDist) {
      simpleNearestNeighborSearch(currentChild, query);
    }
  }

  return closestElement;
}
```

Abbildung 4.9: Hierarchische Nächster-Nachbar-Suche durch Tiefensuche

4.4.2 RKV-Algorithmus

Bei der einfachen Nächster-Nachbar-Suche durch Tiefentraversierung werden die Kinder des aktuellen Knotens in einer festen Reihenfolge betrachtet, wodurch viele Pfade unnötig verfolgt werden. Der RKV-Algorithmus [Roussopoulos, Kelley, u. Vincent 1995] führt dagegen eine lokale Sortierung der Kindknoten durch, so dass eine effizientere Traversierung möglich ist. Ansonsten entspricht er dem Tiefensuche-Algorithmus aus Abbildung 4.9.

Ziel dabei ist nun, Abschätzungen zu finden, so dass Kinder, die mit größerer Wahrscheinlichkeit den nächsten Nachbarn enthalten, zuerst besucht werden. Im optimalen Fall müssen Geschwisterknoten nicht mehr traversiert werden, wenn schon vorher ein Element gefunden wurde, dessen Abstand zum Anfragepunkt kleiner ist als der Abstand, den ein Punkt aus der jeweiligen Region überhaupt haben kann (minDist).

Eine optimistische Sortierung der Kindknoten wird durch Ordnung nach der minDist erreicht. Für eine pessimistische Sichtweise können aber auch die maximal möglichen Abstände eines Elementes aus einer Begrenzungsregion verwendet werden. Das entsprechende Abstandsmaß ist die sog. maxDist. Für eine durch die Punkte $\mathbf{s}$ und $\mathbf{t}$ begrenzte AABB ($s_k \leq t_k$) gilt (analog für andere Typen von Begrenzungsregionen):

$$\text{maxDist}(\mathbf{q}, R)^2 := \sum_{k=1}^{d} \max\left\{|q_k - s_k|, |q_k - t_k|\right\}^2$$

Eine bessere Abschätzung ist möglich für Begrenzungsregionen, deren begrenzende Hyperebenen jeweils mindestens ein Punkt eines Datenelementes enthalten. Dies gilt z. B. für AABBs. Für Bounding Spheres ist eine solche Abschätzung nicht möglich.

Im Gegensatz zur minDist garantiert die minMaxDist, innerhalb des berechneten Abstandes mindestens einen Datenpunkt einer (nichtleeren) Begrenzungsregion zu finden. Dadurch wird der Abstand zu der Region nicht wie bei der minDist meist unterschätzt. Genauer gilt für Regionen, die in jeder Dimension ausgedehnt sind (d. h. $s_k < t_k$) sogar minDist $<$ minMaxDist $<$ maxDist.

Für AABBs ergibt sich die minMaxDist, indem man in jeder Komponente $1 \leq k \leq d$ die Hyperebene H_k bestimmt, welche näher am Anfragepunkt liegt ($H_k = s_k$ oder $H_k = t_k$). Auf jeder dieser Hyperebenen bestimmt man nun den von der Anfrage am weitesten entfernten Punkt in der Region. Die minMaxDist ist der Abstand zu dem Nächstliegenden dieser Punkte.

$$\text{minMaxDist}(\mathbf{q}, R)^2 := \min_{1 \leq k \leq d} \left(|q_k - m_k|^2 + \sum_{\substack{i \neq k \\ 1 \leq i \leq d}} |q_i - M_i|^2 \right)$$

$$m_k = \begin{cases} s_k & : \quad q_k \leq \frac{s_k + t_k}{2} \\ t_k & : \quad \text{sonst} \end{cases} \qquad M_i = \begin{cases} s_i & : \quad q_i \geq \frac{s_i + t_i}{2} \\ t_i & : \quad \text{sonst} \end{cases}$$

Unabhängig von der Möglichkeit zur Sortierung der Kindknoten bietet die minMaxDist schon

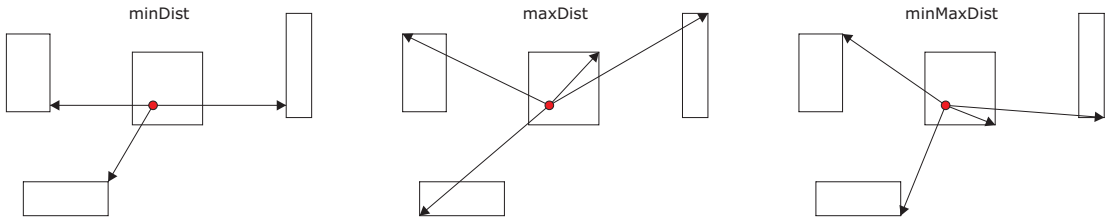

Abbildung 4.10: Verschiedene bei der Nächster-Nachbar-Suche gebräuchliche Abschätzungen für den Abstand eines Anfragepunktes zu einer AABB.

vor Betrachtung einzelner Datenelemente die Möglichkeit, die Pruning Distanz herabzusetzen, da sie garantiert, innerhalb einer bestimmten Distanz einen Punkt auf einem Datenelement zu finden. Abbildung 4.10 stellt die vorgestellten Abstandsmaße gegenüber.

4.4.3 HS-Algorithmus

Die beim RKV-Algorithmus durchgeführte lokale Sortierung der Kinder bei der Traversierung ist zwar in vielen Fällen eine gute Heuristik, aber auch sie verhindert nicht, dass ganze Teilbäume bis zu den Blattknoten traversiert werden müssen, bevor die Geschwisterknoten auf höherer Ebene betrachtet werden. Natürlich sind gute Raumunterteilungen und Abstandsabschätzungen zu den Begrenzungsregionen an dieser Stelle wesentliche leistungsbestimmende Faktoren (Roussopoulos et al. verwenden z. B. R-Bäume), eine globale Betrachtung des Problems kann aber gegenüber der starren Baumtraversierung Vorteile haben.

In [Hjaltason u. Samet 1999] wird ein Algorithmus zur hierarchischen Nächster-Nachbar-Suche vorgestellt, der mit einer globalen Priority Queue arbeitet. Anstatt bei der Traversierung nur die Kinder des aktuellen Knotens zu sortieren und in dieser Reihenfolge zu besuchen, werden sie in die globale Priority Queue eingefügt. Die Sortierung kann hier ebenso nach minDist oder minMaxDist durchgeführt werden. Statt eines rekursiven Aufrufs wird in einer Schleife immer das erste Element aus der Queue entnommen und wie bisher betrachtet (siehe Abbildung 4.11).

Der HS-Algorithmus ist optimal in dem Sinne, dass nur Datenelemente betrachtet werden, deren jeweilige Begrenzungsregionen sich mit der sog. *NN-Sphäre* schneiden. Dabei ist die NN-Sphäre die Kugel mit dem Anfragepunkt $\mathbf{q}$ als Mittelpunkt und Radius $\delta(\mathbf{q}, \mathbf{p})$), wobei $\mathbf{p}$ der zu $\mathbf{q}$ nächstliegende Punkt ist. Allerdings kann eine geordnete Traversierung wie beim RKV-Algorithmus abhängig von der zugrundeliegenden Datenstruktur bei nicht komplett im Hauptspeicher befindlichen Daten vorteilhaft sein.

4.4.4 Octrees für Nächster-Nachbar-Suche auf Polygonmodellen

Nächster-Nachbar-Suche wird insbesondere bei Geo-Informations-Systemen oder Systemen zum Multimedia-Information-Retrieval eingesetzt. Für die hochdimensionalen und großen

```
Element HSNearestNeighborSearch(Node root, Point query)
{
  Element closestElement;
  float pruningDist = max<float>();

  PriorityQueue queue;
  queue.push(0, root);

  while (!queue.empty()) {
    Node currentNode = queue.top();
    queue.pop();

    // already found the nearest neighbor?
    if (minDist(query, currentNode) >= pruningDist) return closestElement;

    // check associated data elements
    for (int i=0; i<currentNode.data.count(); ++i) {
      Element currentElement = currentNode.data[i];
      float currentDist = dist(query, currentElement);
      if (currentDist < pruningDist) {
        pruningDist = currentDist;
        closestElement = currentElement;
      }
    }

    // check children
    for (int i=0; i<currentNode.children.count(); ++i) {
      Node currentChild = currentNode.children[i];
      // dist is minDist or minMaxDist
      queue.push(dist(query, currentChild.bound), currentChild);
    }
  }

  return closestElement;
}
```

Abbildung 4.11: Hierarchische Nächster-Nachbar-Suche mit dem HS-Algorithmus

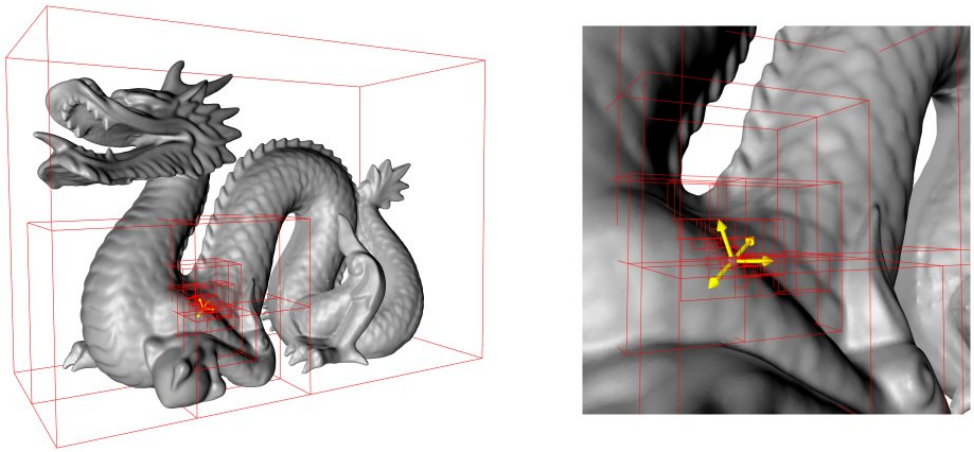

Abbildung 4.12: Nächster-Nachbar Suche mit dem auf Octrees adaptierten HS-Algorithmus. Rot eingezeichnet sind die Bounding Boxes aller Octree-Zellen, welche bei der Suche des zum Cursor nächsten Punktes auf dem Modell besucht wurden (Modell aus [GeorgiaTechLGMA]).

Datenbanken kommen wie auch beim RKV- und HS-Algorithmus häufig R-Bäume [Guttman 1984] zum Einsatz.

R-Bäume entsprechen in ihrer Struktur den eindimensionalen B*-Bäumen, d.h. sie besitzen Blattknoten, in denen die Datenelemente gespeichert werden, sind balanciert und garantieren logarithmische Suchzeit. Die inneren Knoten sind Indexknoten, welche AABBs ihrer Kinder repräsentieren, und dienen zur Navigation im Baum. Die AABBs der Indexknoten erzeugen eine nicht notwendigerweise vollständige und disjunkte Partitionierung des Raumes. Beim Einfügen eines Datenelementes kann dieses daher in mehreren Regionen oder auch in keiner bisher abgedeckten Region liegen. Im Übrigen enthalten die Knoten wie bei B*-Bäumen eine minimale und maximale Anzahl von Elementen, wodurch beim Aufbau eine Overflow-Behandlung, d.h. ein Teilen der Regionen bei Überschreitung der Maximalzahl an Kindern, notwendig ist. Bei allen nötigen Entscheidungen steht die Vermeidung von Überlappungen im Vordergrund, da diese die Suchgeschwindigkeit verringern. Die Indexstruktur erlaubt im Übrigen, dass nicht der komplette Baum im Speicher gehalten werden muss, sondern die einzelnen Seiten bei Bedarf nachgeladen werden können.

Abhängig von den benutzten Verfahren für die Auswahl der Region beim Hinzufügen eines Elementes und für das Splitting kann der Aufbau eines R-Baums sehr aufwendig sein. Insbesondere sind sie für dynamische und nicht vollständig im Hauptspeicher zu haltende Datenbestände optimiert, was zusätzlichen Verwaltungs- und Implementierungsaufwand erzeugt. Bei den vorgestellten Snapping-Verfahren ist es nötig, den nächsten Nachbarn zu einem Punkt auf aus Dreiecken oder konvexen Polygonen bestehenden 3D-Modellen zu bestimmen. Da die Modelle bei vielen VR-Anwendungen statisch sind, können Octrees eine leicht zu implementierende Alternative sein. Nach dem Laden eines 3D-Modells ermöglichen sie einen schnellen Aufbau und danach eine effiziente hierarchische Nächster-Nachbar-Suche, die interaktive Bildraten

auch bei großen Modellen ermöglicht.

Es sind lediglich Anpassungen nötig, um Polygone bzw. Dreiecke im Octree zu speichern, die über die Grenzen der Octree-Zellen hinausragen können. Wir stellen hier eine Adaption des HS-Algorithmus auf Octrees vor, welche räumliche Datenelemente unterstützt und damit eine effiziente Suche des nächsten Nachbarn auf Dreiecks- oder Polygonmodellen für die vorgestellten Translations-Snapping-Verfahren ermöglicht (siehe Abbildung 4.12). Zunächst wird der Aufbau des Octrees erläutert, bevor auf die Anpassungen bei der eigentlichen Suche eingegangen wird.

Aufbau

Beim Aufbau des Octrees ist zu beachten, dass Datenelemente mit räumlicher Ausdehnung im Gegensatz zu punktförmigen Daten die Ränder der Octree-Zellen schneiden können. Bei vielen Fragestellungen ist es üblich, solche Elemente gleichzeitig allen geschnittenen Zellen zuzuordnen (z. B. beim Raytracing [Glassner 1988] oder beim Frustum-Culling). Für die hierarchische Nächster-Nachbar-Suche haben über die Begrenzungsregionen herausragende Datenpunkte jedoch zur Folge, dass die vorgestellten Abstandsabschätzungen zu den Regionen nicht mehr korrekt sind.

Eine weitere Möglichkeit ist, die Polygone an den Grenzen der Octree-Zellen zu teilen. Allerdings sind Aufteilungen möglicherweise für mehrere Seiten einer Zelle und auf mehreren Hierarchieebenen nötig. Dadurch können viele kleine Polygone entstehen, die eine größere Anzahl von Abstandsbestimmungen auf Datenebene zur Folge haben. Dasselbe Problem besteht, wenn die die Zellen schneidende Polygone den Elternknoten zugeordnet werden. Bei der Suche müssen auch dann sehr viele Abstandsbestimmungen direkt auf den Polygonen durchgeführt werden.

Das hier vorgestellte Verfahren vermeidet die Aufteilung der Polygone, erlaubt aber weiterhin die Abschätzung der Abstände zu den Regionen mit minDist und minMaxDist. Die grundlegende Idee ist, die Bounding Box der in den Knoten enthaltenen Daten getrennt von der durch die Octree-Zelle definierten Box zu betrachten. Daher enthält der verwendetete Klassentyp für die Knoten im Octree zusätzlich ein Feld für die Box der jeweiligen Zelle (siehe Abbildung 4.13). Polygone, die mehrere Octree-Zellen schneiden, werden nur einer Zelle zugeordnet und die Bounding Box des Knoten entsprechend erweitert. Bei der Rekursion zur Aufteilung der Octree-Zellen wird die zur jeweiligen Zellen gehörende Box und nicht die Bounding Box der enthaltenen Daten verwendet, um bei jedem Schritt eine Halbierung der Seitenlänge zu gewährleisten.

Der Aufbau des Octrees (siehe Abbildung 4.14) beginnt mit der Bestimmung der Bounding Box des gesamten Modells, die die Größe des Octrees d. h. die Größe der zum Wurzelknoten gehörenden Octree-Zelle bestimmt. Danach werden sukzessiv alle Datenelemente hinzugefügt und dabei die Bounding Boxes der jeweiligen Octree-Knoten entsprechend erweitert. Die Octree-Zellen selbst dienen nur zur weiteren Unterteilung des Octrees und werden daher beim

Hinzufügen von Daten nicht verändert. Das führt dazu, dass die zugeordneten Datenelemente nicht unbedingt vollständig in den Octree-Zellen enthalten sind.

Für die Entscheidung, welcher Octree-Zelle ein Datenelement letztlich zugeordnet wird, wird seine Bounding Box (genauer eine AABB) verwendet. Sie wird zunächst benutzt, um festzustellen, ob es dem aktuellen Knoten endgültig zugeordnet wird oder es an einen Kindknoten weitergereicht wird. Dazu existieren zwei Abbruchbedingungen: Zunächst wird geprüft, ob die maximale Tiefe (`maxDepth`) des Octrees erreicht ist, damit der Verwaltungsaufwand durch die Datenstruktur statisch eingeschränkt werden kann. Desweiteren wird ein dynamisches Abbruchkriterium geprüft, welches ein Datenelement einem Knoten zuordnet, wenn das Größenverhältnis zwischen Bounding Box des Elements zur Octree-Zelle einen vorgegebenen Wert (`maxSizeRatio`) überschreitet.

Falls die Abbruchbedingungen nicht erfüllt sind, wird anhand zweier Kriterien entschieden, welches Kind für die Rekursion gewählt wird. Falls das Datenelement vollständig in eine Kind-Octree-Zelle passt, wird es dieser zugeordnet. Eine Heuristik wird verwendet, wenn das Element mehrere Octree-Zellen schneidet. In diesem Fall wird diejenige ausgewählt, die den geringsten Volumenzuwachs durch das Hinzufügen des Elementes erfahren würde. Für diese Entscheidung wird nicht die Bounding Box der bereits zugeordneten Datenelemente verwendet, sondern die sich aus der Unterteilung ergebende eigentliche Octree-Zelle, um eine gleichmäßige Verteilung der Daten zu erhalten. Andernfalls würde die Heuristik Kindknoten mit einmal gewachsenen Bounding Boxes bevorzugen und ihnen mehr Daten zuordnen, was ihre Bounding Boxes sukzessive weiter vergrößern könnte.

Suche

Für die Suche des nächstliegenden Punktes mit der Octree-Raumunterteilung kommt der HS-Algorithmus (siehe Abbildung 4.11) zum Einsatz, da er für komplett im Hauptspeicher befindliche Daten optimal arbeitet. Für die Abschätzung der Abstände zu den Octree-Zellen (mit minDist oder minMaxDist) werden die den Octree-Knoten zugeordneten Bounding Boxes der enthaltenen Datenelemente benutzt, da nur sie korrekte Abschätzungen ermöglichen. Die eigentlichen Octree-Zellen enthalten wie bereits erwähnt nicht unbedingt vollständig alle ihnen oder ihren Kindern zugeordnete Datenelemente.

Für die Sortierung der Octree-Knoten in der Priority-Queue können sowohl minDist als auch minMaxDist zum Einsatz kommen. Die für die minMaxDist notwendige Bedingung, dass jede Seite der Begrenzungsregion mindestens einen Datenpunkt enthält, ist erfüllt, da dies bereits für die AABBs der im Octree enthaltenen Dreiecke bzw. Polygone gilt und diese Eigenschaft bei Vereinigung der Bounding Boxes erhalten bleibt (siehe Abbildung 4.15). Da ein innerer Octree-Knoten immer genau acht Kinder besitzt, kann es Zellen geben, die keine Daten enthalten und daher eine leere Bounding Box besitzen. Der Abstand zu diesen Zellen wird als unendlich definiert, damit sie im weiteren Verlauf des Algorithmus nicht besucht werden. Ein Hinzufügen solcher Knoten in die Priority Queue ist daher unnötig.

```
class OctreeNode
{
  Box bound;              // aabb of contained data elements
  Box cell;               // octree cell geometry used for subdivision

  vector<Node> children;
  vector<Element> data;
}
```

Abbildung 4.13: Klassentyp für Octree-Knoten bei hierarchischer Nächster-Nachbar-Suche

```
void octreeAddElement(OctreeNode currentNode, Element element, int depth,
                      int maxDepth, int maxSizeRatio)
{
  // extend the bounding box so that it fully contains the data element
  currentNode.bound.extendBy(element.bound);

  // recursion end condition
  if (depth == maxDepth ||
      element.bound.size() / currentNode.cell.size() > maxSizeRatio) {
    currentNode.data.insert(element);
    return;
  }

  // create children nodes once
  if (currentNode.children.empty()) currentNode.createChildren();

  // recurse if element fits completely into a child's octree cell
  for (int i=0; i<currentNode.children.count(); ++i) {
    Node currentChild = currentNode.children[i];
    if (currentChild.cell.contains(element.bound)) {
      addElement(currentChild, element, depth+1, maxDepth, maxSizeRatio);
      return;
    }
  }

  // recurse to child with minimal cell box volume increase
  float minIncrease = max<float>();
  Node minIncreaseChild;
  for (int i=0; i<currentNode.children.count(); ++i) {
    Node currentChild = currentNode.children[i];
    float currentIncrease = getSizeIncrease(currentChild.cell, element.bound);
    if (currentIncrease < minIncrease) {
      minIncrease = currentIncrease;
      minIncreaseChild = currentChild;
    }
  }
  addElement(minIncreaseChild, element, depth+1, maxDepth, maxSizeRatio);
}
```

Abbildung 4.14: Aufbau des Octrees für Nächster-Nachbar-Suche

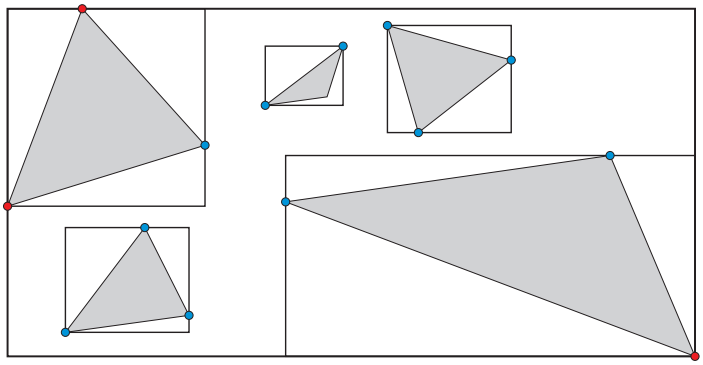

Abbildung 4.15: Bei der Vereinigung von Bounding Boxes bleibt die Eigenschaft erhalten, dass jede Hyperebene mindestens einen Datenpunkt enthält.

Bewertung

Der vorgestellte Octree-Algorithmus erlaubt eine effiziente Nächster-Nachbar-Suche auf Dreiecks- oder Polygonmodellen, welche auch bei aus mehreren Millionen Elementen bestehenden Modellen interaktive Bildraten erlaubt.

Bei der Bestimmung des nächsten Punktes auf einem Objekt beim Snapping kann je nach eingesetztem Verfahren die Pruning-Distanz auf die Breite der Snap-Range initialisiert werden, so dass keine Elemente gefunden werden, die außerhalb der Snap-Range liegen. Dies beschleunigt die Suche weiter, da weiter als die Snap-Range entfernte Regionen sofort verworfen werden können. Beim Beenden des Snappings führt dies dazu, dass kein nächster Nachbar auf dem Ziel-Objekt gefunden wird, sobald man sich außerhalb der Snap-Range befindet. Dies ist aber bei Snap-and-go nötig, um den Offset-Vektor berechnen zu können. Um trotzdem den Suchbereich für die Nächster-Nachbar-Suche einschränken zu können, kann entweder eine vom Faktor der Offset-Korrektur und der Snap-Range abhängige Toleranz zur eigentlich gewählten Pruning Distanz addiert werden oder als Alternative der zuletzt bestimmte nächste Punkt als Referenz für die Berechnung des Offsets benutzt werden, was für die bei interaktiven Bildraten auftretenden kleinen Bewegungen pro Bild eine gute Annäherung ist.

4.5 Effiziente Bereichsanfragen

Die vorgestellte Octree-Datenstruktur zur Nächster-Nachbar-Suche eignet sich ebenso zur Beantwortung von Bereichsanfragen (*Range Queries*). Genauer kann effizient beantwortet werden, welche Daten in einer gegebenen achsenorientierten Box liegen oder diese schneiden (siehe Abbildung 4.16 für den Algorithmus).

Solche Bereichsanfragen können beim Snapping genutzt werden, um im nicht gesnappten Zustand bei Verfahren, die ohne Eingangs-Snap-Range arbeiten (z. B. Snap-and-go), die Durchführung von Schnitttests zur Erkennung der Zielüberquerung des Cursors oder allgemein

```
vector<Element> result;

void octreeRangeQuery(OctreeNode currentNode, Box query)
{
  // check associated data elements
  for (int i=0; i<currentNode.data.count(); ++i) {
    Element currentElement = currentNode.data[i];
    if (query.intersects(currentElement.bound) {
      result.insert(currentElement);
    }
  }

  // check children
  for (int i=0; i<currentNode.children.count(); ++i) {
    Node currentChild = currentNode.children[i];
    if (query.intersects(currentChild.bound) {
      octreeRangeQuery(currentChild, query);
    }
  }
}
```

Abbildung 4.16: Bereichsanfragen mit der Octree-Datenstruktur aus der Nächster-Nachbar-Suche

des bewegten Objekts zu beschleunigen. Dabei wird von der Objektposition im vorigen Bild zur Position im aktuellen Bild ein Liniensegment erzeugt, für das Schnitttests mit den Polygonen des Objekts durchgeführt werden. Die Anzahl dieser Tests kann nun reduziert werden, indem nur die Polygone betrachtet werden, deren Bounding Box sich innerhalb der Bounding Box des Liniensegments befindet. Da die Bewegung des Cursors von Bild zu Bild bei Interaktion in Virtuellen Umgebungen in der Regel sehr klein ist, ist die Verwendung von achsenorientierten Bounding Boxes ausreichend effizient.

4.6 Zusammenfassung

In diesem Kapitel wurden unterschiedliche Verfahren für das Snapping von Punkten bzw. des Translationsanteils von Transformationen präsentiert. Die vorgestellten Snapping-Techniken für Spatial-Input-Interaktion in Virtuellen Umgebungen erlauben als primitive Ziele Punkte, Geraden und Liniensegmente, Ebenen und konvexe Polygone. Darüberhinaus wurde ein Verfahren vorgestellt, das Snapping auf Polygonmodelle erlaubt. Insbesondere dafür wurde ein Octree-Algorithmus vorgestellt, der eine effiziente Nächster-Nachbar-Bestimmung auch auf großen Modellen ermöglicht.

5 Rotations-Snapping in Virtuellen Umgebungen

Bei den bislang vorgestellten Snapping-Techniken wurden nur die Bewegung von Punkten bzw. des Translationsanteils von Objekttransformationen berücksichtigt. In diesem Kapitel werden verschiedene Techniken für das Snapping von Rotationen vorgestellt. Zwar kann Rotations-Snapping mit geeigneten Widgets (wie z. B. in 3D-Modellierungsprogrammen) auch auf das Snapping von Translationen zurückgeführt werden. Jedoch erfordert dies eine Indirektion, bei der die Translationen auf geeignete Rotationswerte abgebildet werden müssen. Der Fokus dieses Kapitels liegt stattdessen auf Rotations-Snapping-Techniken, bei denen Rotationen direkt betrachtet und geeignet gesnappt werden.

Noch stärker als bei Translations-Snapping-Verfahren sind beim Snapping von Rotationen für die Interaktion in Virtuellen Umgebungen die verwendeten Interaktionstechniken von Bedeutung. Die hier präsentierten Verfahren sind primär ausgelegt für solche Techniken, bei denen eine relativ direkte Übertragung der Rotation eines 3D-Eingabegerätes auf das bewegte Objekt stattfindet. Dies ist beispielsweise beim Scaled Grab oder der Virtual Hand der Fall. Bei anderen Interaktionsparadigmen kann es sinnvoll sein, stattdessen den Rotationsanteil der Eingabegerät-Transformation direkt zu snappen. Insbesondere gilt das beim Simple Dragging, bei dem das manipulierte Objekt in fester Entfernung am Pick-Ray hängt und das Rotationszentrum in der Hand des Benutzers liegt.

5.1 Vorbereitungen

Vor der Betrachtung oder Modifikation einer Transformation muss festgestellt werden, ob die entsprechende 4×4-Matrix überhaupt eine Rotationsmatrix darstellt bzw. ob eine Rotation extrahiert werden kann. Ungültig sind in diesem Zusammenhang z. B. Transformationen, die einen projektiven Anteil, eine nicht-orthogonale linke obere 3×3-Submatrix oder eine Spiegelung enthalten.

Bei einer Matrix M, die eine homogene Skalierung enthält (d.h. $M_{44} \neq 1$), kann die linke obere 3×3-Submatrix extrahiert werden und getrennt von der Skalierung betrachtet werden. Auch Matrizen mit einer in den 3×3-Anteil integrierten uniformen Skalierung können in einen Skalierungs- und Rotationsteil aufgeteilt werden, wobei der Skalierungsfaktor der Länge der Bilder der Basisvektoren entspricht.

5.2 Snapping auf Orientierungen

Ein Variante des Rotations-Snappings ist das Snapping auf vorgegebene Orientierungen. Sobald die aktuelle Orientierung eines Objekts zu einer der angegebenen Snapping-Zielorientierungen ähnlich genug ist, wird sie darauf gesnappt. Dazu kann die ursprüngliche Orientierung einfach durch die Zielorientierung ersetzt werden.

Eine einfache und intuitive Möglichkeit zur Angabe der Zielorientierungen besteht in der Verwendung von Eulerwinkeln. Snapping könnte dann stattfinden, wenn die jeweiligen Differenzen von *Head-* (h), *Pitch-* (p) und *Roll-* (r) Winkeln zwischen Original- und Zielorientierung alle innerhalb eines kleinen Bereichs liegen. Dies wird allerdings dadurch erschwert, dass bei Eulerwinkeln ein *Gimbal Lock* auftreten kann, da die einzelnen Rotationen im jeweils lokalen Koordinatensystem durchgeführt werden, d. h. die aktuelle Drehachse von den vorherigen Drehungen mitbestimmt wird. Beim Gimbal Lock fallen die Rotationen um die erste und dritte Achse zusammen, wodurch die Eulerrepräsentation nicht mehr eindeutig ist. Zwar kann ein Gimbal Lock leicht erkannt werden ($|p - \frac{\pi}{2}| < \varepsilon$), eine weitere Herausforderung ist dabei aber, dass die meist in Form einer Rotationsmatrix angegebene Originalorientierung zunächst in Eulerrepräsentation umgewandelt werden muss, um das Snapping auf die *hpr*-Winkel durchzuführen. Dies ist nicht nur unnötig aufwendig, sondern in der Nähe eines Gimbal Locks außerdem numerisch instabil. Es ist daher wünschenswert, die Darstellung der Zielorientierungen unabhängig von einer bestimmten Repräsantationsform zu machen, um Mehrdeutigkeiten bei Gimbal Locks und die Umrechnung der Originaltransformation in eine bestimmte Repräsentationsform während des Snappings zu vermeiden.

Es gibt unterschiedlichste Möglichkeiten zur Darstellung von Rotationen wie z. B. Eulerwinkel, Quaternionen und Exponential Maps. Sie alle können aber relativ leicht in Transformationsmatrizen umgewandelt werden. Die zu snappende Objektrotation ist zudem auch meist als Matrix gegeben, weswegen sich die Verwendung dieser Repräsentationsform anbietet. Ein Maß μ für die Ähnlichkeit zwischen der Originalorientierung und den Snapping-Zielorientierungen kann definiert werden, in dem die jeweiligen Skalarprodukte zwischen den durch die Matrizen transformierten Hauptachsen bestimmt werden. Diese lassen sich dabei leicht aus den einzelnen Spalten der Matrizen ablesen. Seien O die originale Rotationsmatrix, T eine Matrix, die eine Snapping-Zielorientierung repräsentiert, sowie die (normierten) Basisvektoren $\mathbf{e}_i$ gegeben. Dann ergibt sich das Maß für die Ähnlichkeit bei Orientierungs-Snapping zu:

$$\mu_o(O, T) := \min \{(O\mathbf{e}_i) \cdot (T\mathbf{e}_i),\ i \in [1:3]\}$$

Diejenige transformierte Achse $O\mathbf{e}_i$, die am weitesten von der zugehörigen Zielorientierung entfernt ist (und damit das niedrigste Skalarprodukt erzeugt), gibt hierbei schließlich den Ausschlag. Insbesondere gilt natürlich $\mu_o(T, T) = 1$.

Der Benutzer kann für das Rotations-Snapping ebenso wie beim Snapping von Translationswerten Eintritts- und Austritts-Snap-Ranges angeben. Für Rotationen ist dies intuitiv durch die

Angabe von Winkeln als Abstand zur Zielorientierung möglich. Mit Hilfe des Ähnlichkeitsmaßes kann nun der minimale Winkelabstand bzgl. der drei Hauptachsen berechnet und so bestimmt werden, ob die ursprüngliche Orientierung innerhalb der Snap-Range eines Ziels liegt:

$$\delta_o(O,T) := |\arccos(\mu_o(O,T))|$$

Zusammen mit einer Snap-Range $r \in \mathbb{R}$ ergibt sich die Orientierungs-Snapping-Funktion σ_o damit insgesamt zu:

$$\sigma_o(O,T) := \begin{cases} T & : \quad \delta_o(O,T) \leq r \\ O & : \quad \text{sonst} \end{cases}$$

Ebenso wie beim Translations-Snapping kann beim Rotations-Snapping gleichzeitig nur ein Snapping-Ziel aktiv sein. Stehen mehrere Snapping-Ziele zur Auswahl, wird auf dasjenige gesnappt, welches die größte Ähnlichkeit (d. h. das maximale μ) zur ursprünglichen Orientierung besitzt. Analog zum Translations-Snapping können auch hierbei unterschiedliche Snap-Ranges für Eintritt- und Austritt sowie unterschiedliche Snapping-Ziel-Prioritäten verwendet werden.

5.3 Snapping auf Rotationsachsen

Eine weitere Variante des Rotations-Snappings ist das Snapping auf vorgegebene Achsen. Dadurch wird dem Benutzer beispielsweise ermöglicht, ein Objekt ohne Verwackeln präzise um seine Hauptachsen zu drehen.

Die Parametrisierung von Rotationsachsen-Snapping-Zielen erfolgt durch Angabe einer Quellachse $\mathbf{s}$ und einer Zielachse $\mathbf{t}$. Wie beim zum Orientierungs-Snapping definieren wir auch hier ein Maß für die Ähnlichkeit der ursprünglichen Orientierung zum Snapping-Ziel. Dabei wird mit Hilfe des Skalarprodukts die durch die Originalmatrix O transformierte Quellachse mit der Zielachse verglichen:

$$\mu_a(O,\mathbf{s},\mathbf{t}) := (O\mathbf{s}) \cdot \mathbf{t}$$

$\mathbf{s}$ und $\mathbf{t}$ müssen hierbei normiert sein. Die Bestimmung des Winkelabstandes zwischen Original- und Zielachse erfolgt wie beim Orientierungs-Snapping, so dass hier die Snap-Ranges identisch angegeben werden können:

$$\delta_a(O,\mathbf{s},\mathbf{t}) := |\arccos(\mu_a(O,\mathbf{s},\mathbf{t}))|$$

Der Winkelabstand wird hier zusätzlich benutzt, um die Rotation R zu bestimmen, die die durch die Originalmatrix transformierte Quellachse auf die Zielachse dreht. Mit dieser Rotation wird dann die Originalmatrix transformiert, um sie auf die Zielachse zu snappen. Der Rotationsanteil kann hier nicht wie beim Orientierungs-Snapping einfach ersetzt werden, da beim Snapping auf Rotationsachsen nicht alle Freiheitsgerade durch das Snapping-Ziel bestimmt sind. Sei $r \in \mathbb{R}$ wiederum die Snap-Range, dann gilt für die Rotationsachsen-Snapping-Funktion σ_a:

$$\sigma_a(O,\mathbf{s},\mathbf{t}) := \begin{cases} RO & : \quad 0 < \delta_o(O,\mathbf{s},\mathbf{t}) \leq r \\ O & : \quad \text{sonst} \end{cases},$$

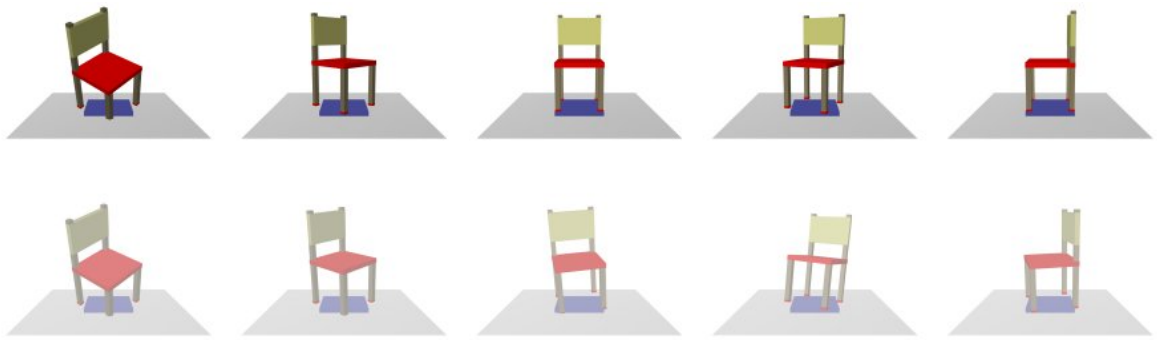

Abbildung 5.1: Snapping des Rotationsanteils einer Objekttransformation. Oben ist die sichtbare Bewegung bei aktiviertem Snapping zu sehen, unten die ursprüngliche Rotation. Als Snapping-Ziele werden hier eine Achse ($\mathbf{s} = \mathbf{t} = (0, 1, 0)$) sowie zwei Orientierungs-Ziele (in Eulerdarstellung $hpr = (0, 0, 0)$ sowie $hpr = (-\frac{\pi}{2}, 0, 0)$) mit höherer Priorität verwendet.

wobei R wie die Rotationsmatrix ist, die $O\mathbf{s}$ um die Achse $\mathbf{u} := O\mathbf{s} \times \mathbf{t}$ mit dem Winkel $\delta_o(O, \mathbf{s}, \mathbf{t})$ auf $\mathbf{t}$ rotiert ($O\mathbf{s}, \mathbf{t}, \mathbf{u}$ bilden ein Rechtssystem, falls $\delta_o(O, \mathbf{s}, \mathbf{t}) \neq 0$ ist).

Die Definitionen von μ bzw. δ bei Orientierungs- und Rotationsachsen-Snapping sind miteinander verträglich. Das heißt insbesondere, dass unterschiedliche Snapping-Ziele parallel verwendet werden können und es eine einheitliche Bestimmung der Winkelabstände gibt.

Ein denkbares Szenario für die Kombination unterschiedlicher Rotations-Snapping-Ziele ist z. B. das Snapping der Hauptachsen eines Objektes auf die Welt-Koordinatenachsen. Gleichzeitig können höher priorisierte Orientierungs-Snapping-Ziele verwendet werden, so dass das bewegte Objekt während der (gesnappten) Rotation um eine der Hauptachsen bevorzugt auf eine vorgegebene Orientierung snappt, sobald der Winkelabstand dazu kleiner als ihre Snap-Range ist (siehe Abbildung 5.1).

5.4 Rotations-Snapping und Translationen

Bei vielen Interaktiontechniken in Virtuellen Umgebungen werden Objekte nicht um ihren Ursprung im lokalen Koordinatensystem gedreht. Stattdessen findet z. B. beim Simple Dragging eine Rotation um die Hand des Benutzers statt. Bei der Scaled-Grab-Technik erfolgt die Rotation um den zuletzt ausgewählten sog. *Pivot*-Punkt.

Beim Rotations-Snapping muss dies beachtet werden, da Rotationen um einen beliebigen Punkt im lokalen Koordinatensystem des bewegten Objektes einen Translationsanteil in der Transformationsmatrix erzeugen: Zur Durchführung einer Rotation um einen Punkt wird dieser zunächst in den Ursprung verschoben. Die gewünschte Rotation kann nun um den neuen Ursprung erfolgen. Nachdem die Translation rückgängig gemacht wurde, liegt der Rotationspunkt wieder an seiner alten Position. Falls die Rotationsachse nicht genau der Verbindungsvektor zwischen dem Rotationspunkt und dem alten Ursprung ist, liegt letzterer

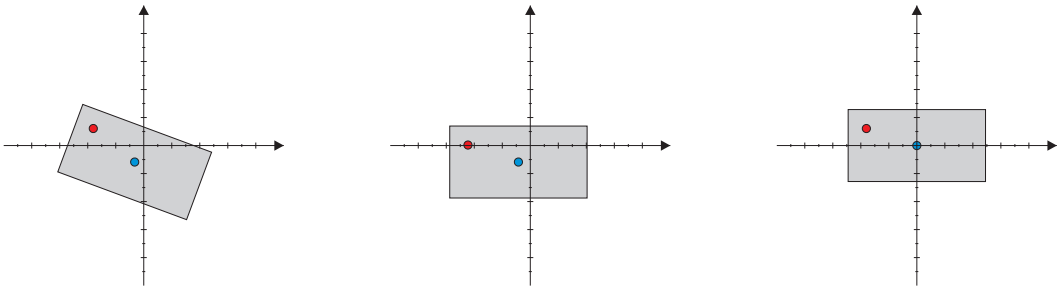

Abbildung 5.2: Rotations-Snapping und beliebige Rotationspunkte (links) Augangsrotation um einen vom Ursprung des lokalen Koordinatensystems verschiedenen Rotationspunkt (mitte) Rotations-Snapping ohne Beachtung des Rotationspunktes verschiebt diesen (rechts) Rotations-Snapping mit Beachtung des Rotationspunktes erhält dessen Position, verschiebt aber den Ursprung des lokalen Koordinatensystems.

nun nicht mehr an seiner alten Position, wodurch die Transformation einen Translationsanteil enthält.

Wird in einem solchen Fall beim Snapping die Orientierung des bewegten Objekts geändert und nicht beachtet, dass die Rotation um einen vom Ursprung verschiedenen Punkt stattfindet, ändert sich durch Ersetzen oder Anpassen des Rotationsanteils die Position des Rotationspunktes, was u. U. unerwünscht ist. Ist der Rotationspunkt bekannt bzw. kann er ermittelt werden (möglich, wenn die Transformation keinen Translationsanteil außer dem bei der Rotation entstehenden enthält), dann kann dies auch beim Rotations-Snapping berücksichtigt werden: Dazu muss der Rotationspunkt vor Durchführung des Snappings in den Ursprung und danach wieder zurück an sein ursprüngliche Position verschoben werden (siehe Abbildung 5.2).

Beachtet werden muss dabei allerdings, dass dann beim Rotations-Snapping auch der Translationsanteil der Transformation geändert wird, was im Zusammenspiel mit manchen Interaktionstechniken aber auch beim Translations-Snapping unerwünschte Nebeneffekte haben kann. Damit der beim Translations-Snapping gesnappte Punkt nicht nachträglich verschoben wird, muss in jedem Fall das Rotations-Snapping zuerst ausgeführt werden.

5.5 Zusammenfassung

In diesem Kapitel wurden zwei Ansätze für das Snapping von Rotationen vorgestellt, die die direkte Betrachtung des Rotationsanteils von Objekttransformationen erlauben. Die Definition der Snapping-Ziele kann aufgrund der internen Verwendung von Rotationsmatrizen in beliebiger Form erfolgen, also z. B. durch Eulerwinkel oder Quaternionen.

6 Snapping-Feedback

Wie bereits in den vorangehenden Kapiteln erwähnt, geht durch Snapping in Virtuellen Umgebungen bei Verwendung von Spatial-Input-Interaktionstechniken die Korrespondenz zwischen der Hand- bzw. Armbewegung des Benutzers und der Bewegung des Cursors bzw. manipulierten Objekts verloren.

Weitere Herausforderungen entstehen durch Einsatz spezieller Interaktionstechniken, die dem Benutzer Manipulationen außerhalb seiner Reichweite erlauben. Der dazu häufig verwendete Pick-Ray ist meist auch während der Bewegung sichtbar, um die Verbindung zum Objekt anzuzeigen und dem Benutzer die Arbeitsweise der verwendeten Interaktionstechnik transparent zu machen. Wird nun wie z. B. beim Translations-Snapping das Objekt unabhängig von der durch den Benutzer direkt verursachten Bewegung verschoben, geht natürlich auch die Korrespondenz zwischen Pick-Ray und Objekt verloren.

In diesem Kapitel werden Methoden vorgestellt, die dem Benutzer visuelles Feedback über den aktuellen Snapping-Status geben und so eine koordinierte Bewegung ermöglichen. Dabei besteht ein enger Bezug zur verwendeten Interaktionstechnik. Es wird in diesem Zusammenhang insbesondere auf vollkommen direkte Manipulation mit der Virtual-Hand-Technik sowie auf die strahlbasierten Techniken Simple-Dragging und vor allem den Scaled Grab [Simon et al. 2005] eingegangen. Für letzteren wird zunächst ein Verfahren vorgestellt, welches auch während der Objektmanipulation die Darstellung des Pick-Rays erlaubt.

6.1 Connected Scaled Grab

Bei der Scaled-Grab-Technik wird ein Pick-Ray zur Objektselektion verwendet. Während der Bewegung wird der Strahl jedoch deaktiviert, da die Rotation des Eingabegerätes auf eine Rotation des Objektes um den selektierten Punkt (Pivot-Punkt) abgebildet wird. Wäre der immer gerade aus dem Eingabegerät zeigende Strahl weiterhin aktiv, würde er vor dem Benutzer ins Leere zeigen und ihn irritieren, da es keinen erkennbaren direkten Bezug zur Objektbewegung gibt.

Es ist wünschenswert, auch während der Manipulation eines Objekts eine sichtbare Verbindung zu diesem zu haben. Dafür gibt es mehrere Gründe: In Mehrbenutzersystemen sollen die anderen Benutzer zuordnen können, welcher Benutzer für die Objektmanipulation verantwortlich ist. Dies gilt sogar schon bei nur zwei aktiven Benutzern, damit erkennbar ist, wer bei einer konkurrierenden Objektselektion erfolgreich war. Zwar tritt der Strahl bei der Verwendung

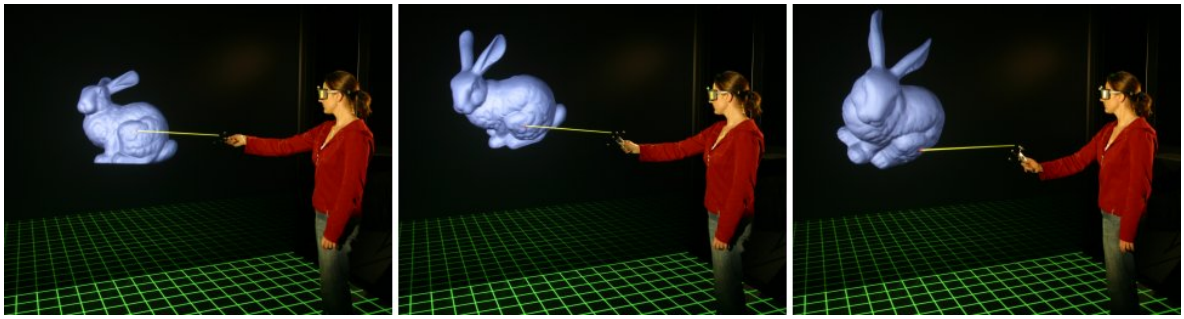

Abbildung 6.1: Bei der Connected-Scaled-Grab-Technik wird der Pick-Ray auch während der Objektmanipulation dargestellt. Er zeigt dabei immer auf den bei der Selektion festgelegten Pivot-Punkt. Durch die skalierte Translation und die Rotation um den Pivot-Punkt tritt der Pick-Ray nicht mehr unbedingt gerade aus dem Eingabegerät aus (Modell aus [GeorgiaTechLGMA]).

von Multi-Viewpoint Images [Simon u. Scholz 2005] aus Sicht der anderen Benutzer nicht exakt aus dem Gerät des aktiven Benutzers aus. Bei ähnlicher Perspektive ist der Fehler aber gering, so dass die Zuordnung, welcher Strahl zu welchem Benutzer gehört, trotzdem möglich ist.

Vorteilhaft ist ein sichtbarer Strahl auch, falls das Objekt während der Bewegung aus dem auf dem Screen sichtbaren Bereich geschoben wurde. Wenn eine Verbindung zum Objekt besteht, ist es für den Benutzer einfacher, das Objekt zurück in den sichtbaren Bereich zu holen. Durch die skalierte Bewegung sind dabei auch große Entfernungen zu bewältigen.

Für die dauerhafte Sichtbarkeit des Pick-Rays spricht schließlich, dass Benutzer es als irritierend und nicht intuitiv empfunden haben, wenn der Pick-Ray bei Objektselektion plötzlich deaktiviert wurde. Diese Einschätzungen wurde von mehreren Benutzern nach Sitzungen zur Evaluation der Scaled-Grab-Technik geäußert.

Eine einfache aber effektive Idee zur dauerhaften Darstellung des Pick-Rays beim Scaled Grab besteht darin, den Strahl stets vom Eingabegerät zum Pivot-Punkt des selektierten Objekts zeigen zu lassen, was als *Connected Scaled Grab* bezeichnet werden kann. Dies hat u. a. die Konsequenz, dass der Strahl durch die skalierte Translation und die durch die vom Eingabegerät auf den Pivot-Punkt abgebildete Rotation meist nicht mehr gerade aus dem Gerät austritt (siehe Abbildung 6.1). Von einigen Benutzern wurde dies zunächst als ungewohnt empfunden, was aber schon nach kurzer Eingewöhnungsphase relativiert wurde.

Beim Beenden der Interaktion mit einem Objekt ist eine sprunghafte Veränderung der Pick-Ray-Richtung nicht zu vermeiden, da er dann wieder gerade aus dem Eingabegerät austritt. So ist eine folgende Neuselektion des letzten Objekts u. U. erst nach einer Neuausrichtung des Eingabegerätes möglich. Dies kann verbessert werden, indem bei Selektion ins Leere oder in den für den Benutzer auf dem Screen nicht sichtbaren Bereich das zuletzt gewählte Objekt erneut aktiviert wird. Der letzte Pivot-Punkt kann dabei beibehalten werden, sollte dann aber auch dauerhaft angezeigt werden. Ein Vorteil bei dieser Art der Selektion ist, dass

der Benutzer die Ausgangshaltung des Eingabegerätes relativ frei wählen kann, da er nicht zwangsweise auf das Objekt zeigen muss. Dies ist insbesondere relevant, da die Veränderung gegenüber der Ausgangsorientierung und nicht die absolute Rotation des Eingabegerätes auf die Objektrotation um den Pivot-Punkt abgebildet wird.

Die permanente Pick-Ray-Darstellung erlaubt im Folgenden die Verwendung strahlbasierter Snapping-Feedback-Techniken auch für die bevorzugte Scaled Grab Interaktionstechnik.

6.2 Translations-Snapping-Feedback

In diesem Abschnitt stellen wir Verfahren vor, die dem Benutzer Feedback über den aktuellen Status bei Translations-Snapping geben können. Dabei soll nicht nur visualisiert werden, ob das bewegte Objekt momentan auf ein Ziel gesnappt ist. Während des Snappings kann der Benutzer seine Bewegungen besser koordinieren, wenn ihm bekannt gemacht wird, wo sich die eigentliche Objektposition innerhalb der Snap-Range befindet. Darüberhinaus ist es wünschenswert, dem Benutzer noch vor Verlassen der Snap-Range einen Hinweis zu geben, dass er sich an ihrem Rand befindet, damit er die Objektbewegung anpassen kann, wenn ein Verlassen des aktuell gesnappten Ziels nicht gewünscht ist.

6.2.1 Highlight

Die einfachste Form des Snapping-Feedbacks ist die Verwendung eines Highlights. Für die strahlbasierten Interaktionstechniken kann z. B. der Pick-Ray aufgehellt oder in einer anderen Farbe dargestellt werden, wenn das bewegte Objekt gerade gesnappt ist. Alternativ kann auch ein Highlight auf das bewegte Objekt oder falls sichtbar das Snapping-Ziel gelegt werden. Die letzten beiden Möglichkeiten sind auch bei der ohne Pick-Ray arbeitenden Virtual-Hand-Interaktion anwendbar.

Um dem Benutzer das baldige Verlassen der Snap-Range anzuzeigen, kann ein graduelles Abschalten des Highlights erfolgen. Überschreitet die Entfernung der eigentlichen Objektposition vom Snapping-Ziel einen festgelegten Grenzwert (z. B. 75% der Snap-Range), wird das Highlight linear in Abhängigkeit von der Entfernung bis zum Verlassen des Ziels ausgeblendet. So wird dem Benutzer Gelegenheit gegeben, seine Bewegung u. U. zu korrigieren, falls kein Verlassen des aktuellen Snapping-Ziels gewünscht ist.

Werden als Snapping-Feedback ausschließlich Highlights verwendet, bleibt bei strahlbasierten Interaktionstechniken die Frage der Korrespondenz zwischen Pick-Ray und Objekt ungelöst. Während des Snappings zeigt der Pick-Ray also u. U. nicht mehr auf das Objekt. Um dies zu verhindern, könnte der Pick-Ray ähnlich wie beim Connected Scaled Grab auf den durch Snapping transformierten Punkt auf dem Objekt zeigen. Damit würde der Pick-Ray auch hier das Eingabegerät nicht mehr gerade verlassen. Daher hätte der Benutzer abesehen vom Highlight keine Information über die aktuelle Entfernung vom Snapping-Ziel. Die eigentliche Objektposition könnte er nur über die Veränderung des Highlights bei Bewegung oder durch die

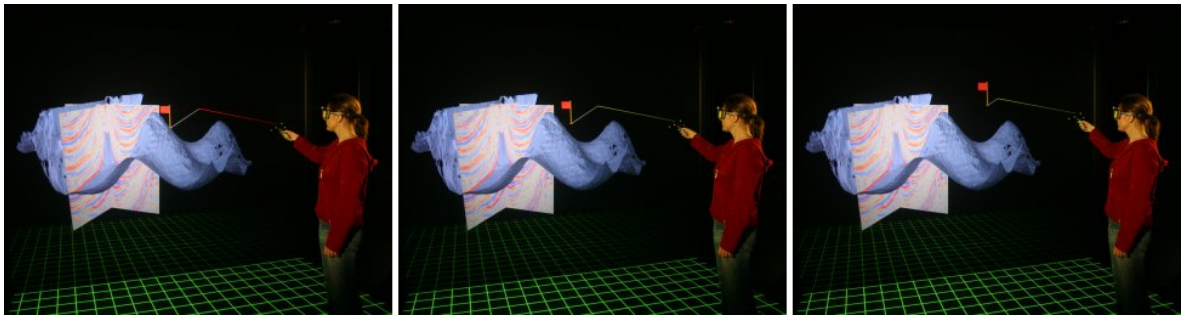

Abbildung 6.2: Gummiband Snapping-Feedback

Ausrichtung und Position seines Eingabegerätes erahnen. Beim Scaled Grab können zusätzliche Irritationen dadurch entstehen, dass nicht mehr transparent ist, was der aktuelle Grund für den nicht gerade aus dem Eingabegerät austretenden Strahl ist, d. h. hier wäre die Metapher doppelt besetzt.

6.2.2 Gummiband

Zur Erhaltung der ursprünglichen Orientierung des Pick-Rays kann die Gummiband-Metapher [Newman u. Sproull 1979; Foley et al. 1995] eingesetzt werden. Der Pick-Ray zeigt hier immer zu der ursprünglichen ohne Snapping bestehenden Position. Dabei wird er an dieser Stelle abgeschnitten und ausgehend von seiner Spitze ein Gummiband zur durch Snapping transformierten Position dargestellt.

Das Gummiband-Feedback ist kombinierbar mit der Verwendung von Highlights. Als weitere Alternative kann ähnlich wie beim graduellen Highlight des Pick-Rays der Durchmesser, die Helligkeit oder die Transparenz des Gummibandes abhängig von der Position innerhalb der Snap-Range variiert werden.

Bei Verwendung von Snap-and-go Translations-Snapping besteht auch nach Beenden des Snappings ein Offset zur ursprünglichen Position, der erst im Laufe weiterer Bewegungen neutralisiert wird. Im Gegensatz zur ausschließlichen Verwendung von Highlights bietet das Gummiband auch dann noch Feedback darüber, wie groß und in welcher Richtung dieser Offset besteht (siehe Abbildung 6.2).

In Kombination mit der Connected-Scaled-Grab-Technik führt das Gummiband zu einem etwas indirekten Eindruck bei der Kontrolle des Objekts. Dies liegt daran, dass die Verbindung zwischen dem Eingabegerät und dem Objekt über zwei Ecken läuft: Einerseits gibt es einen Knick zwischen Eingabegerät und Pick-Ray und dann einen weiteren zwischen Pick-Ray und Gummiband. Im nächsten Abschnitt wird eine für Scaled Grab geeignetere Feedback-Technik vorgestellt, die dies vermeidet.

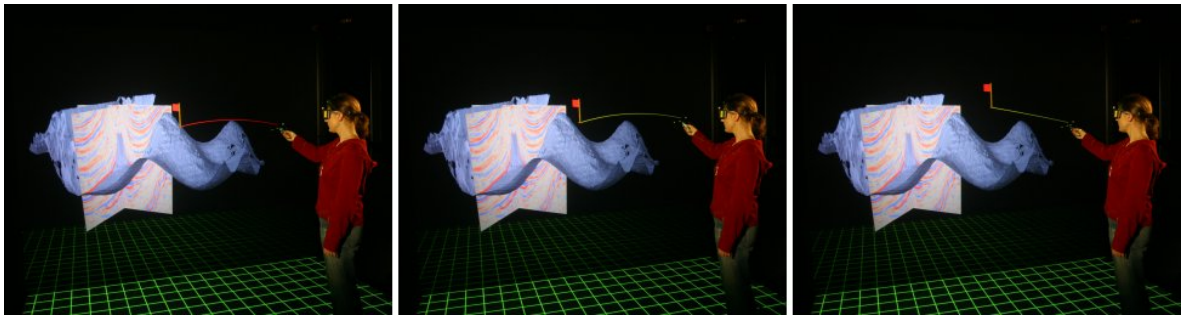

Abbildung 6.3: Bent Pick-Ray Snapping-Feedback

6.2.3 Bent Pick-Ray

Der ursprünglich für die Kollaboration mehrerer Benutzer entwickelte *Bent Pick-Ray* [Riege et al. 2006] kann auch als Feedback für Translations-Snapping eingesetzt werden. Bei der Kollaboration visualisiert er Krafteinwirkung der Benutzer bei einer gemeinsamen Objektmanipulation. Eine ähnliche Metapher kann man beim Snapping benutzen, wo ein pseudo-haptischer Eindruck z. B. beim Durchdringen oder Festhalten an einer Oberfläche entsteht [Lecuyer et al. 2000; Lindeman et al. 2001].

Am Startpunkt des Strahls am Eingebegerät liegt der Bent Pick-Ray tangential am eigentlichen ungekrümmten Strahl. Beim Connected Scaled Grab zeigt er beispielsweise in die Richtung des originalen Pivot-Punktes. Der Pick-Ray-Endpunkt liegt in dem durch das Snapping verschobenen Punkt. Die Strahlkrümmung wird gemäß des Verfahrens aus [Riege et al. 2006] durch einen Kreisbogen erreicht, der in der von Start-, End- und untransformierten Originalpunkt aufgespannten Ebene gezeichnet wird (siehe Abbildung 6.3). Der Radius und Mittelpunkt des zugehörigen Kreises ergibt sich aus der Entfernung zwischen dem untransformierten und dem gesnappten Punkt sowie der Entfernung vom Benutzer.

Für den Scaled Grab ergibt sich aufgrund der unmittelbaren Verbindung zum bewegten Objekt eine sich wesentlich direkter anfühlende Interaktion als beim Gummiband-Feedback. Wie auch beim Gummiband kann das beste Feedback durch zusätzliche Verwendung gradueller Highlights erreicht werden. Auf eine zusätzliche Visualisierung des ungekrümmten Strahls wie in [Lindeman et al. 2001] zur Unterstützung von Selektionen (siehe 3.3.4) kann verzichtet werden, da sie dem Benutzer während der Manipulation keine zusätzliche Information bietet.

6.3 Rotations-Snapping-Feedback

Auch für Rotations-Snapping ist eine Darstellung des aktuellen Snapping-Status wünschenswert. Eine Möglichkeit während des Snappings ist, sowohl gesnappte als auch originalen Achsen bspw. durch Pfeile zu visualisieren. Um eine Verdeckung der Pfeile durch das gesnappte Objekt zu vermeiden, können sie z. B. relativ zur Bounding Box des Objekts skaliert werden.

Analog zum Pick-Ray-Highlighting beim Translations-Snapping kann die ursprüngliche Achse in Abhängigkeit von der Entfernung zur Zielachse eingefärbt werden, damit der Benutzer Feedback darüber bekommt, ab wann das Snapping gelöst wird.

Um auch im ungesnappten Zustand zu visualisieren, in welchen Orientierungen oder auf welche Achsen Rotations-Snapping durchgeführt wird, können ab einer bestimmten Entfernung zum jeweiligen Ziel die Zielachsen eingeblendet und ebenso abhängig vom Abstand zum Snapping-Eintrittsbereich eingefärbt werden.

7 Implementierung

Wie bereits in Kapitel 2 angedeutet, kombiniert ein Virtual-Reality-Software-Framework die in einem VR-System benötigten Technologien, so dass eine effiziente Applikationsentwicklung ermöglicht wird. Neben der Integration eines auf einer Low-Level-Graphik-API aufbauenden Szenengraphen sind weitere benötigte Komponenten beispielsweise die Anbindung der Eingabegeräte, eine Abstraktion der Ausgabegeräte sowie Möglichkeiten des Rapid-Application-Developments.

Für eine effektive Verwendung in VR-Applikationen müssen sich auch die in dieser Arbeit vorgestellten Snapping-Techniken in das verwendete Framework integrieren. In diesem Kapitel werden daher nach einer kurzen Einführung in das unserer Implementierung zugrundeliegende VR-Framework Avango die Konzepte der entwickelten Komponenten präsentiert, die eine einfache Benutzung von Snapping in bereits existierenden und neu entwickelten VR-Applikationen ermöglicht.

7.1 Avango

Avango [Avango; Tramberend 1999, 2003] ist ein objektorientiertes Framework zur Entwicklung verteilter und interaktiver VR-Applikationen. Hauptkomponenten sind die auf OpenGL aufbauende Szenengraph-API Performer, ein an Inventor angelehnter Field-Mechanismus, sowie ein auf der funktionalen Programmiersprache Scheme aufbauendes Scripting-Interface. Weitere Komponenten, auf die hier nicht näher eingegangen wird, sind u. a. ein Netzwerk-Layer zur Realisierung verteilter Applikationen, eine Display-Device-Abstraktion und ein Device-Daemon zur Anbindung von Eingabegeräten und Tracking-Systemen (siehe Abbildung 7.1). Ein Scripting-Layer zur Anbindung weiterer Sprachen und die Migration von Performer auf OpenSceneGraph [OpenSceneGraph] ist in Arbeit.

7.1.1 Fields

Die Szenengraph-API Performer [SGI] benutzt zum Zugriff auf die Attribute der Objekte Member-Funktionen. Dabei kann die Änderung eines Attributes eines Objekts den Wert eines anderen beeinflussen. Weil aber keine Möglichkeiten der Introspektion vorhanden sind, kann die Applikation dies weder einfach erkennen, noch weitere Metainformationen über die Typen oder die Anzahl der Attribute eines Objekts abfragen.

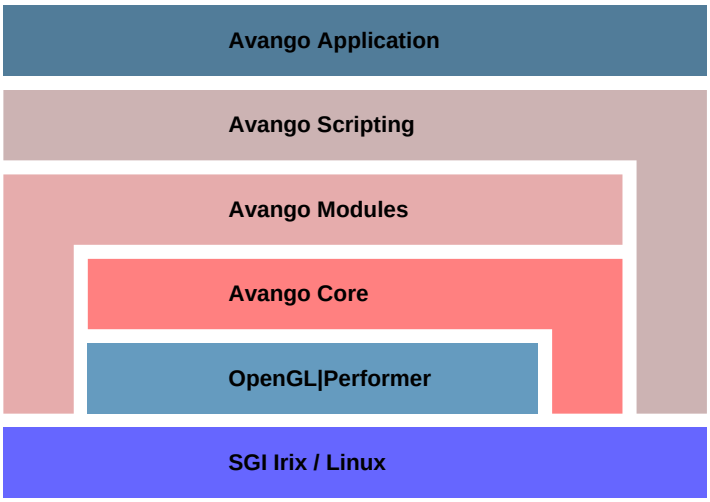

Abbildung 7.1: Schematischer Aufbau des Avango VR-Frameworks

Avango nutzt das *Field*-Konzept aus Inventor [SGI], um die Attribute von Objekten Performer-abgeleiteter und eigener Typen zu kapseln. Die sog. *Field-Container* benutzen die Fields als ihre Schnittstelle zur Außenwelt. Metainformationen sind für die Applikation verfügbar, wodurch die Field-Container als Komponenten benutzt werden können. Darüberhinaus unterstützen Fields und Field-Container ein Streaming-Interface, mit dem eine Netzwerk-Verteilung der Avango-Objekte realisiert wird. Diese kann zur Kollaboration oder auch zur Darstellung der Applikation auf mehreren Render-Clients verwendet werden, wodurch große Display-Systeme wie die i-Cone (siehe 2.1) auch mit normalen PCs angesteuert werden können.

Avango-Fields repräsentieren einzelne Werte (*Single-Fields*) oder eine variable Anzahl von Werten (*Multi-Fields*). Der Zugriff erfolgt dabei mit `getValue()` und `setValue()` Funktionen sowie bei Multi-Fields zusätzlich mit `add1Value()` und `remove1Value()` Funktionen, die eine effiziente Manipulation der zugrundeliegenden Datenstrukturen erlauben. Zusätzlich zur expliziten Speicherung von Werten in gewöhnlichen Fields erlauben *Adaptor-Fields* die Abbildung des Zustandes externer Objekte auf die Werte eines Fields. Der Zugriff auf die Werte erfolgt dabei mit Hilfe von Callback-Funktionen. Dies wird z. B. intensiv beim Wrappen der Performer-Typen benutzt.

Orthogonal zum Szenengraph erlaubt Avango sog. *Field-Connections*. Fields gleichen Typs können verbunden werden, so dass der Wert eines *Source-Fields* bei Änderung an alle verbundenen Fields weitergegeben wird. Durch die Field-Connections können zusätzliche logische Verknüpfungen zwischen Szenengraph-Knoten (*Nodes*) hergestellt werden, die sich nicht alleine durch einen Szenengraph ausdrücken lassen.

Die Field-Connections dienen zur Anbindung der Applikation an die reale Welt: Die einmalig pro Bild stattfindende Evaluation des durch die Verbindungen erzeugten Datenfluss-Graphen beginnt bei den sog. Sensor-Knoten (z. B. `avDeviceSensor`, `avTimeSensor`), die ihre Werte

Performer/C++

```
// object      op        value
parentNodePtr->addChild(childNodePtr);
```

Avango/C++

```
// object      field    op         value
parentNodePtr->Children.add1Value(childNodePtr);
```

Avango/Scheme (object-oriented)

```
;;      object      field       op          value
(-> (-> parent-node 'Children) 'add-1value child-node)
```

Avango/Scheme (functional)

```
;; op           object       field    value
(av-add-1value parent-node 'Children child-node)
```

Abbildung 7.2: Performer vs. Avango-Field-Interface

über Fields der Applikation zur Verfügung stellen. Durch Field-Connections werden Fields in den sog. Effektor-Knoten (`avObject`, `avNode`) angestoßen, die die eigentlichen Applikationsknoten repräsentieren, auf die Field-Änderungen reagieren können und Ergebnisse selbst wieder über entsprechende Fields nach außen geben. Am Ende der Evaluationskette stehen die sog. Aktuatoren, die selbst keine eigenen Field-Werte berechnen und nur entsprechend gesetzter Fields agieren (z. B. `avViewActuator`).

7.1.2 Scripting

Applikationsentwicklung in Avango beginnt in der Regel mit der Erzeugung benutzerdefinierter Komponenten in C++, die von Avango-Typen abgeleitet sind und dynamisch zur Laufzeit geladen werden. Das eigentliche Design der Applikationslogik inklusive der Instanziierung der Objekte und der Herstellung von Field-Connections geschieht mit Hilfe eines auf der funktionalen Programmiersprache Scheme aufbauenden Scripting-Interfaces [Springer et al. 2000] (siehe Abbildung 7.2). Die Applikation reagiert dabei typischerweise in Script-Callbacks auf Field-Änderungen in Sensor- und Effektor-Knoten anstatt selbst durchgängig zu agieren.

7.2 Matrixfilter

Die in dieser Arbeit vorgestellten Snapping-Techniken arbeiten auf den Matrizen von Transformationsknoten, um z. B. beim Translations-Snapping die Verschiebung eines Objekts auf den zur ursprünglichen Position nächsten Punkt der aktiven Snapping-Ziele zu realisieren oder

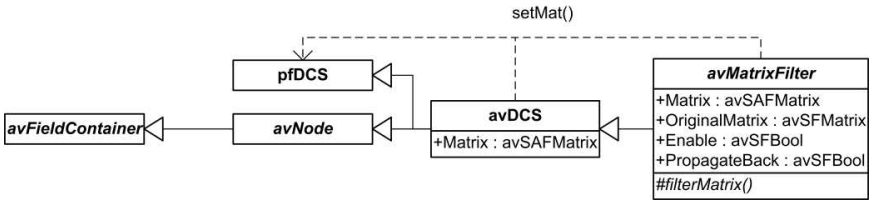

Abbildung 7.3: Die Basis-Ableitungshierarchie der Matrixfilter-Abstraktion.

beim Rotations-Snapping das Objekt auf eine Zielachse zu drehen. Um diese Manipulationen einer Matrix auf eine gemeinsame Basis zu stellen und so die Verwendung in Applikationen zu vereinheitlichen, haben wir eine sog. Matrixfilter-Abstraktion entwickelt, die gemeinsame Funktionalität für derartige Operationen zur Verfügung stellt. Die Realisierung erfolgt mit Hilfe eines Avango-Typs, der ein Matrix-Field besitzt. Beim Setzen des Field-Wertes wird die Matrix entsprechend der Aufgabe des Filters manipuliert. Zusätzlich werden über das gemeinsame Field-Interface des abstrakten Basistyps `avMatrixFilter` häufig benötigte Informationen und Einstelloptionen bereitgestellt. Dazu zählt u. a. neben einer Möglichkeit, die Manipulations-Funktionalität temporär zu deaktivieren (`Enable`) auch ein Field, welches die zuletzt gesetzte unveränderte Transformation bereitstellt (`OriginalMatrix`).

In Avango gibt mehrere grundsätzliche Möglichkeiten zur Realisierung von Komponenten, die Matrizen manipulieren. So können sie beispielsweise selbst als Transformationsknoten (`avDCS`[1]) im Szenengraph verwendet werden (siehe Abbildung 7.3). Jeder `avDCS` besitzt ein Matrix-Adaptor-Field, das in einer Member-Funktion des umgebenden Fieldcontainers die Transformation auf dem Performer-Basis-Typ setzt (mit `pfDCS::setMat()`). Durch Überschreiben dieser Funktion in einem `avDCS`-abgeleiteten Typ ist eine Manipulation der Transformationsmatrix möglich. Statt des Adaptor-Field-Callbacks aus `avDCS` wird somit bei Field-Änderungen die Callback-Funktion aus `avMatrixFilter` benutzt. In dieser wird die (abstrakte) Funktion `filterMatrix()` aufgerufen, welche die eigentliche Matrix-Manipulation in einer konkreten Subklasse durchführt. Anschließend wird auf dem `pfDCS` die möglicherweise veränderte Transformationsmatrix gesetzt. Durch die Ableitung von `avDCS` kann der Matrixfilter so direkt als die Transformationsknoten mit der veränderten Matrix im Szenengraph eingesetzt werden.

Anstatt einen Matrixfilter-Knoten in den Szenengraphen einzufügen, kann alternativ auch der Field-Connection-Mechanismus von Avango und ein zusätzlicher Transformationsknoten im Szenengraph verwendet werden, um eine Manipulation der Objekttransformation zu erreichen. Dazu wird eine Verbindungskette bspw. ausgehend vom Matrix-Field eines sog. *Dragger*-Knotens [Tramberend et al. 1999], das die sich aus der Benutzerinteraktion ergebende Transformation enthält, über den Matrixfilter-Knoten hin zum eigentlichen Transformationsknoten erzeugt (siehe Abbildung 7.4). Durch Angabe eines bei der Matrixmanipulation berücksichtigten Szenengraph-Referenz-Knotens kann auch der Nachteil wettgemacht werden,

[1] DCS = Dynamic Coordinate System

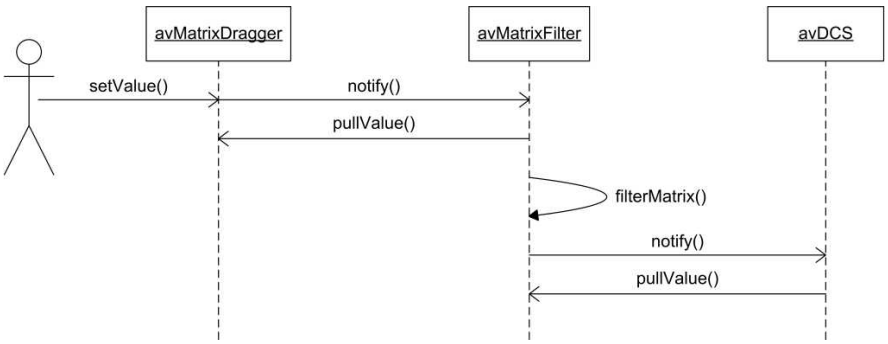

Abbildung 7.4: Manipulation von Matrizen mit Hilfe von Avango Field-Connections. Die Matrix-Fields der beteiligten Objekte sind (von links nach rechts) miteinander verbunden. Die Werte verbundener Fields werden bei Änderung weiterpropagiert. Im Matrixfilter erfolgt zuvor allerdings mit der in einer konkreten Subklasse implementierten `filterMatrix()`-Funktion eine Manipulation der Matrix.

dass die Manipulation der Transformation im Matrixfilter bei dieser Methode eigentlich nur in lokalen Koordinaten durchgeführt werden kann.

Eine im Rahmen der Diplomarbeit entstandene Erweiterung dieses Konzeptes erlaubt als dritte Möglichkeit, einen Matrixfilter explizit an ein anderes Field anzuhängen. Sobald auf dem Field ein neuer Wert gesetzt wird, wird dieser über eine Field-Connection an den Matrixfilter weitergereicht. Nach der wie bisher ablaufenden Veränderung der Transformation wird der neue Wert an das ursprüngliche Field zurückpropagiert, wenn dies im Matrixfilter über das entsprechende Field (`PropagateBack`) aktiviert wurde. Alle weiteren Fields, die durch eine Connection mit diesem verbunden und in der Benachrichtigungskette hinter dem Matrixfilter stehen, bekommen nun ebenfalls den veränderten Wert. Diese Vorgehensweise eignet sich insbesondere für bereits existierende Anwendungen, in denen z. B. Snapping-Funktionalität integriert werden soll. Ohne Änderung der bestehenden Verbindungen im Field-Connection-Datenfluss-Graphen und der Typen der existierenden Knoten können somit konkrete Matrixfilter einfach an in der Applikation definierte Objekte angehängt werden (siehe Abbildung 7.5).

Zusätzlich zu den im nächsten Abschnitt vorgestellten Snapping-Modulen sind während der Diplomarbeit mehrere konkrete Matrixfilter-Komponenten entstanden, die die vielseitige Einsetzbarkeit des Konzepts demonstrieren. U. a. können Translationen von bestimmten Knoten auf einen gewissen Bereich bzgl. lokaler oder Weltkoordinaten eingeschränkt werden. Eine andere Komponente erlaubt es, bestimmte Teile einer gegebenen Orientierung herauszufiltern (z. B. Pitch- und Roll-Rotationen). Weiteres Beispiel ist eine Komponente, die die gegebenen Matrix-Werte bzgl. der eigenen Position im Szenengraphen von lokalen in Weltkoordinaten umrechnet. Der universellste Matrixfilter ruft einen frei definierbaren Script-Callback auf, der die transformierte Matrix berechnet. Dadurch, dass die Matrixfilter mit schon vorhandenen Mechanismen in Avango arbeiten, können sie einzeln oder auch verbunden durch Field-

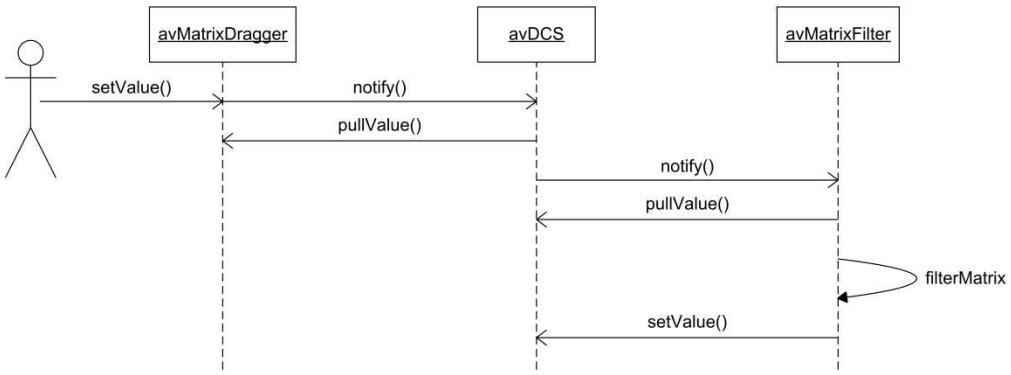

Abbildung 7.5: Durch Setzen der entsprechenden Option auf einem Matrixfilter-Modul wird die Rückpropagation der veränderten Matrix an das Source-Field einer Field-Connection aktiviert. Dies eignet sich insbesondere zur Erweiterung bestehender Applikationen, da die bisherigen Typen und die Struktur des Datenflusses beibehalten werden kann. Auch hier sind die Matrix-Fields der beteiligten Objekte über Field-Connections (von links nach rechts) miteinander verbunden.

Connections zusammen verwendet werden, wobei die jeweilige Semantik natürlich beachtet werden muss.

7.3 Translations-Snapping

Zur Durchführung von Translations-Snapping in einer Avango-Applikation benutzen wir für die manipulierten und an bestimmten Zielen zu snappenden Objekte jeweils ein eigenes Matrixfilter-Modul, welches die Translation über eine der oben erklärten Verfahrensweisen während des Snappings anpasst. Die Benutzung einer eigenen Komponente für jedes Objekt ist vorteilhaft, wenn die Snapping-Module als Transformationsknoten eingesetzt werden oder mehre Benutzer hintereinander mit denselben Objekten interagieren möchten. So können objektbezogene Status-Informationen verwaltet werden. Zum Beispiel ist bei Beginn einer neuen Manipulations-Interaktion ein evtl. noch bestehender Offset vom zuletzt durchgeführten Snap-and-go-Snapping zu löschen. Darüberhinaus können pro bewegtem Objekt dadurch weitere Einstellungen wie beispielsweise das Ausmaß der Snap-and-go-Offset-Korrektur oder Geschwindigkeits-Grenzwerte für die Bewegung eingestellt werden, ab der kein Snapping mehr stattfinden soll (siehe 4.2.3).

Um auf der anderen Seite den Aufwand für die Snapping-Ziele so gering wie möglich zu halten, ist eine einmalige Definition zu bevorzugen, anstatt die Ziele für jedes manipulierte Objekt neu parametrisieren zu müssen. Konkret bedeutet dies, dass nach einmaliger Erzeugung z. B. eines Punkt-Snapping-Ziels dieses in mehreren Snapping-Modulen eingesetzt werden kann. Die Aufgabe, anfallende Statusinformationen pro Snapping-Ziel zu verwalten, obliegt dem Snapping-Modul. Zur Erfüllung dieser Anforderung existiert in unserer Implementierung ein abstrakter Basistyp für die Snapping-Ziele. Konkrete Ziel-Objekte können zur Laufzeit beliebig

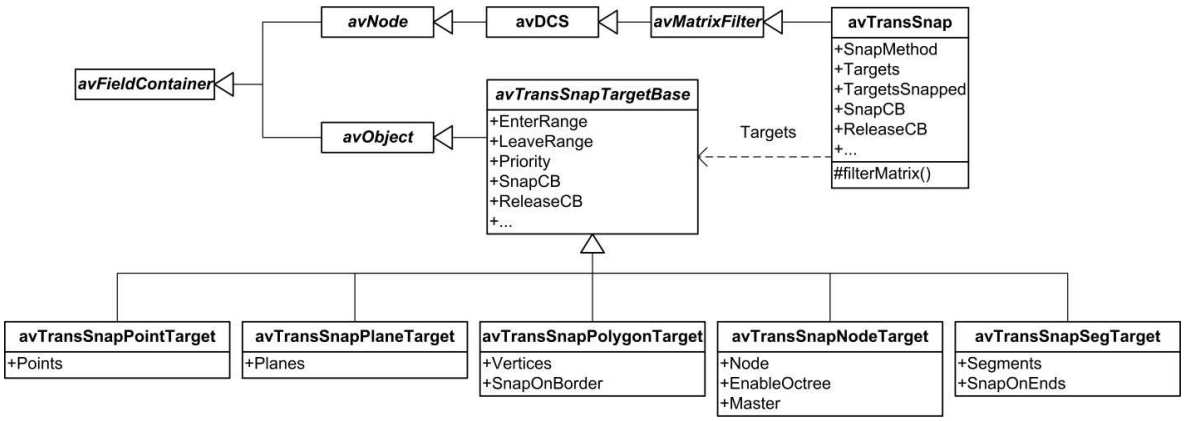

Abbildung 7.6: Klassendiagramm der beim Translations-Snapping beteiligten Typen.

in das `Targets`-Multi-Field verschiedener Snapping-Filter eingesetzt und auch wieder entfernt werden (siehe Abbildung 7.6). Bei Änderung des Matrix-Fields des Snapping-Modules werden nun z. B. beim Standard-Snapping (siehe 4.1) alle für dieses Modul aktiven Snapping-Ziele nach dem zur Ursprungsposition nächstliegenden Punkt auf dem jeweiligen Ziel gefragt. Aus den Antworten wird danach der insgesamt nächste Punkt ermittelt.

Wie bereits erwähnt, liegt die Verantwortung für die Verwaltung von Zustandsinformationen nicht bei den Snapping-Zielen, sondern bei den die Ziele verwendenden eigentlichen Snapping-Modulen. Allerdings bauen auch erstere bestimmte Datenstrukturen auf. Genauer legt z. B. das Node-Snapping-Ziel (`avTransSnapNodeTarget`), welches die Angabe eine Szenengraph-Knoten erlaubt, auf dessen darunterliegende Geometrie gesnappt wird, nach der Änderung des entsprechenden `Node`-Fields einen Octree zur effizienten Nächster-Nachbar-Suche an. Hierbei wird die Geometrie unterhalb des angegebenen Nodes zwar als statisch erachtet, im Szenengraph darüber liegende Transformationen können jedoch ohne Neuberechnung des Octrees variiert werden, so dass auch statische Snapping-Ziel-Geometrien in begrenztem Maße transformiert werden können. Darüberhinaus kann man über ein `Master`-Field ein anderes Node-Snapping-Ziel angeben, mit dem die Datenstrukturen geteilt werden, was die Speichereffizienz erhöht und auch Vorberechnungen für mehrfach in der Szene vorkommende Geometrien erlaubt.

Zur besseren Integration des Snapping-Status in die Logik der Applikation erlauben die Snapping-Filter wie auch die Snapping-Ziele die Ausführung von Script-Callbacks beim Beginn und Beenden des Snappings. Die Callbacks werden dabei jeweils mit beiden Objekten parametrisiert aufgerufen. Abbildung 7.7 zeigt beispielhaft eine minimales Snapping-Setup.

7.4 Rotations-Snapping

Die Implementierung des Rotations-Snappings ist analog zum Design des Snappings für Translationen strukturiert. Auch hier können die Snapping-Ziel-Objekte unabhängig von den

```
;; define callbacks
(define snap-cb (lambda args (display "snapped!\n")))
(define release-cb (lambda args (display "released!\n")))

;; add some geometry to the scene
(define sphere (make-instance-by-name "avFile"))
(av-set-value sphere 'Filename "sphere.iv")
(av-add sphere)

;; add dragger to geometry (for interaction)
(define matrix-dragger (make-instance-by-name "avMatrixDragger"))
(av-add-1value sphere 'Dragger matrix-dragger)

;; define snap filter and build field connection chain
(define snap-filter (make-instance-by-name "avTransSnap"))
(av-connect-from snap-filter 'Matrix matrix-dragger 'Matrix)
(av-connect-from sphere 'Matrix snap-filter 'Matrix)

;; define regular grid
(define grid (make-instance-by-name "avGrid"))
(av-set-value grid 'Dimensions (make-vec3 5 5 5))

;; add point snapping target
(define snap-target (make-instance-by-name "avTransSnapPointTarget"))
(av-set-value snap-target 'SnapCB snap-cb)
(av-set-value snap-target 'ReleaseCB release-cb)
(av-connect-from snap-target 'Points grid 'Points)
(av-add-1value snap-filter 'Targets snap-target)
```

Abbildung 7.7: Minimales Grid-Snapping-Setup mit Script-Callbacks für Snapping-Events

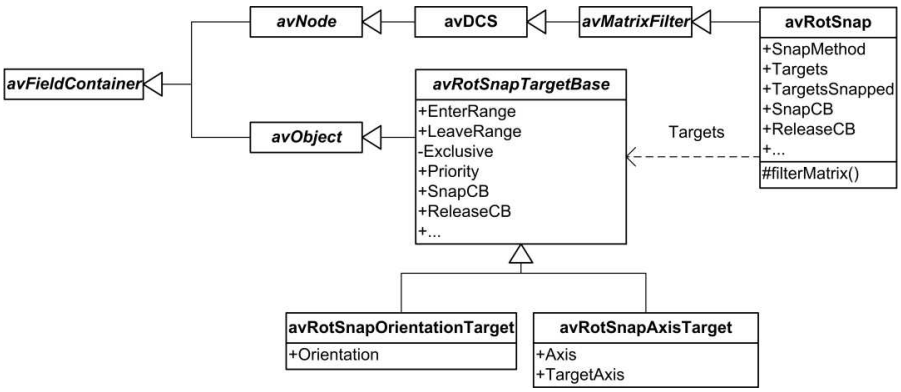

Abbildung 7.8: Klassendiagramm der beim Rotations-Snapping beteiligten Typen.

die Transformation manipulierenden Rotations-Snapping-Modulen verwendet werden und auch in mehreren Modulen eingesetzt werden. Auf diese Weise kann mit einmaliger Definition der Rotations-Snapping-Ziele ohne zusätzlichen Aufwand gleichartiges Rotations-Snapping für viele Objekte in der Szene realisiert werden. Abbildung 7.8 zeigt die Klassenstruktur der beim Rotations-Snapping beteiligten Typen.

8 Evaluation

Zur Evaluation der im Rahmen der Diplomarbeit entwickelten Snapping-Techniken wurden Teile davon im Rahmen des VRGeo-Projekts [VRGeo] zu Demonstrationszwecken in eine Applikation integriert.

Der VRGeo-Demonstrator ist eine prototypische immersive VR-Applikation für die Exploration seismischer Volumendaten im Kontext der Öl- und Gasindustrie. Neue Technologien werden integriert und im Halbjahresrhythmus Experten aus verschiedenen Fachgebieten (u. a. Virtuelle Realität, Geologie und Geophysik) präsentiert und anschließend diskutiert. Aktuell unterstützt der Demonstrator Multi-User-Interaktion mit Hilfe von OmniStereo und Multi-Viewpoint-Images in der i-Cone oder mit Head-Tracking im TwoView (siehe 2.1), sowie einen semi-immersiven Aufbau mit Interaktion an autostereoskopischen Displays am Desktop. Dabei wird die Notation der sog. *Workspaces* benutzt, welche jeweils einen volumetrischen Datensatz und zugehörige benutzerdefinierte Elemente enthalten. Benutzer können einzeln oder gemeinsam mit mehreren Workspaces arbeiten.

Die Benutzer interagieren im VRGeo-Demonstrator mit getrackten 6DOF-Eingabegeräten (Wireless Styli bzw. PDAs, siehe 2.3) sowie optional Tablet-PCs mit einer 2D-GUI zur Systemkontrolle. Bei vergangenen Evaluationen hat sich herausgestellt, dass die Benutzer bei der Exploration der Daten Objekte häufig unter bestimmten Randbedingungen (z. B. präzise Ausrichtung von Objekten aneinander) manipulieren wollen. Hierbei können Snapping-Techniken unterstützen.

Bei der im Rahmen der Diplomarbeit entstandenen Integration der Snapping- und Snapping-Feedback-Techniken in den VRGeo-Demonstrator wurde versucht, für verschiedene Anwendungsfälle die jeweils beste Metapher zu verwenden, so dass eine möglichst präzise und natürliche Objektkontrolle erreicht wird. In den folgenden Abschnitten werden diese Ideen vorgestellt und anschließend das Feedback der Benutzer präsentiert.

8.1 Snapping für Workspaces

Die jeweils einen volumetrischen Datensatz repräsentierenden Workspaces werden durch Quader umschlossen, deren Oberflächen-Normalen umgedreht sind, so dass nur die Innenseiten sichtbar sind, während die Außenseiten durch Backface-Culling entfernt werden. Dies erlaubt den Benutzern, unabhängig von der aktuellen Orientierung in die Workspaces hineinzublicken. An der rückwärtigen Begrenzung können die Workspaces angefasst und somit bewegt werden.

Die Manipulation erfolgt mit der auf Pick-Ray-Selektion beruhenden Scaled-Grab-Technik (siehe 2.4.3). Der am Workspace-Quader gegriffene Punkt ist der Pivot-Punkt der Rotation. Die Translation wird bei Ferninteraktion skaliert, so dass eine effiziente Kontrolle im Nahbereich wie auch die Bewegung über weitere Strecken möglich ist (wie z. B. bei Panorama-Displays nötig). Die Benutzer können festlegen, ob ihr Pick-Ray während der Workspace-Manipulation ausgeblendet wird oder sichtbar bleibt (Connected Scaled Grab, siehe 6.1). Zusätzlich wird eine Manipulation durch Selektion in den leeren Raum erlaubt, die die Parameter der zuletzt durchgeführten Aktion beibehält und somit eine für den Benutzer möglicherweise bequemere Handausrichtung während der Objektmanipulation erlaubt. Weiterhin ermöglicht diese Art der Selektion bei Anzeige des Pick-Rays während der Manipulation die Lokalisierung und Verschiebung eines versehentlich aus dem sichtbaren Bereich geschobenen Workspaces.

Um eine gerade Ausrichtung der Workspaces zu ermöglichen, wird Rotations-Snapping durchgeführt. Als Snapping-Ziele werden hierbei die Welt-Hauptachsen benutzt, auf die die Hauptachsen der Workspace-Koordinatensysteme snappen können. Dies wird derart eingerichtet, dass auch nach evtl. Verkippen eines Workspaces exaktes Rotieren um diese Achsen möglich ist (wie z. B. eine Rotation der x-Achse des Workspaces um die z-Achse des Weltkoordinatensystems). Zusätzlich wird Orientierungs-Snapping der Workspace-Hauptachsen auf die Hauptachsen des Weltkoordinatensystems durchgeführt, so dass nicht nur die Rotation um eine Achse erlaubt wird, sondern auch bevorzugte Orientierungen in natürlicher Weise exakt erreicht werden können.

Zu beachten ist beim Zusammenspiel von Rotations-Snapping und Pivot-Punkt-Rotation beim Scaled Grab, dass die Position des Pivot-Punkts beim Snapping erhalten bleibt. Um dies zu erreichen, wird nach dem Snapping des Rotations-Anteils der Workspace-Transformation eine Translation durchgeführt, die den dadurch verschobenen Pivot-Punkt wieder an seine ursprüngliche Position bringt.

Um dem Benutzer möglichst viel Freiheit bei der Positionierung der Workspaces zu lassen, werden für das Rotations-Snapping kleine Eintritts-Snapping-Bereiche benutzt. Die Austritts-Bereiche sind wesentlich größer gewählt, damit eine einmal eingerastete Achse oder Orientierung nicht zu schnell ungewollt wieder verlassen wird.

8.2 Snapping für Volume Lenses

Benutzer können innerhalb eines Workspaces mehrere sog. *Volume Lenses* (auch als *Volumenproben* bezeichnet) erzeugen, die in ihrer Größe veränderbar sind und eine direkte Volumendarstellung der sich innerhalb ihrer Ausdehnung befindlichen seismischen Daten beinhalten. Die Volume Lenses können innerhalb des jeweiligen Workspaces frei orientiert und bewegt werden.

Die Manipulation erfolgt mit Hilfe des Scaled Grab. Allerdings ist der Pivot-Punkt hier nicht der selektierte Punkt auf dem Rand der würfelförmigen Volume Lens, sondern immer ihr

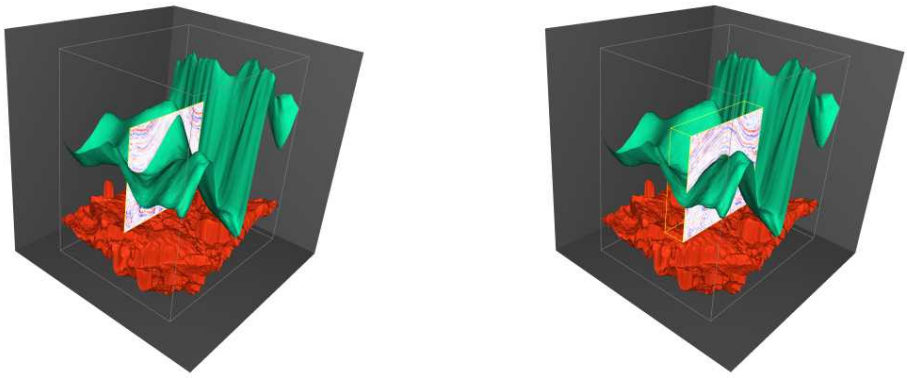

Abbildung 8.1: Volume-Slice-Snapping im VRGeo-Demonstrator

Mittelpunkt, da die in ihr dargestellten Daten möglicherweise transparent sind und daher der Rand nicht sichtbar ist. Gerade bei kleineren und entfernten Objekten wirkt eine symmetrische Rotation um den Mittelpunkt wesentlich natürlicher als die Rotation um einen Pivot-Punkt auf dem u. U. unsichtbaren Rand.

Da sich die Volume Lenses stets innerhalb des Workspaces befinden, ist ein hoher Skalierungsfaktor bei der Translation nicht nur unnötig, sondern kann bei großer Entfernung zum Benutzer dazu führen, dass die aktuell bewegte Volume Lens schon bei kleinster Bewegung des Eingabegerätes den Rand des Workspaces erreicht. Stattdessen haben wir einen maximalen Skalierungsfaktor festgelegt, der derart gewählt ist, dass sie innerhalb der Armreichweite maximal in der Größenordnung des Workspaces bewegt werden kann. Wie bei der Workspace-Interaktion ist auch hier optional die Anzeige des Pick-Rays während der Manipulation möglich.

Bei den Volume Lenses wird Rotations-Snapping benutzt, um eine Ausrichtung an den Hauptachsen des Workspaces zu ermöglichen. Die Parametrisierung der Snapping-Ziele geschieht hier analog zum Workspace-Rotations-Snapping, außer dass die Zielachsen nicht in Welt- sondern in Workspace-Koordinaten referenziert werden. Zusammen mit der Einschränkung der Position auf die Ausdehnung des Workspaces erlaubt diese Snapping-Konfiguration eine präzise Orientierung der Volume Lenses auf die Workspace-Hauptachsen und -Ränder.

8.3 Snapping für Volume Slices

Ähnlich wie die Volume Lenses erlauben sog. *Volume Slices* einen Blick in die volumetrischen seismischen Daten, wobei allerdings nur jeweils eine Schicht dargestellt wird. Die Volume Slices können ebenso mit Hilfe von Scaled Grab durch den Workspace bewegt werden, wobei auch hier wie bei den Volume Lenses der Pivot-Punkt der Mittelpunkt der Slice ist, so dass die Rotation unabhängig vom durch den Benutzer selektierten Punkt auf der Slice stattfindet.

Rotations-Snapping für Volume Slices arbeitet genauso wie das Snapping für die Volume Lenses, d. h. es findet eine Ausrichtung an den Hauptachsen des sie enthaltenden Workspaces

statt. Zusätzlich wird bei einer Volume-Slice-Manipulation auf die letzte Orientierung gesnappt, so dass die Slice in beliebiger Richtung mit festgehaltener Orientierung durch den Workspace bewegt werden kann.

Unterstützt wird diese Art der Bewegung durch zusätzlich stattfindendes Translations-Snapping mit dem Snap-and-go-Verfahren. Ein Punkt-Snapping-Ziel (siehe 4.2.2) erlaubt einerseits die exakte Rückkehr zur letzten Position und andererseits die präzise Bewegung entlang einer ebenfalls durch die letzte Position vorgegebenen orthogonal zur Slice liegenden Achse. Mit Hilfe eines von der letzten Position ausgehenden Rahmens wird den Benutzern die Arbeitsweise des Slice-Snappings verständlich gemacht (siehe Abbildung 8.1). Bei Entfernung vom Snapping-Ziel wird zusätzlich durch graduelles Ausblenden des Rahmens Feedback über die aktuelle Entfernung vom Snapping-Target präsentiert, was insbesondere bei während der Manipulation deaktiviertem Pick-Ray die nötige Unterstützung für eine präzise Interaktion bietet.

8.4 Snapping für Data Picker

Für die direkte Darstellung eines Datenpunktes aus dem volumetrischen Datensatz enthält der VRGeo-Demonstrator einen sog. Data Picker, der durch ein Flaggen-Modell repräsentiert wird. Wird dieses durch einen Benutzer bewegt oder darauf mit dem Pick-Ray gezeigt, wird ein Label eingeblendet, welches die den Dichtewert des volumetrischen Datensatzes an der Stelle des des Flaggenfußes anzeigt. Das Label wird dabei immer zum interagierenden Benutzer hin ausgerichtet.

Mit Hilfe des Data Pickers ist es darüberhinaus möglich, relevante Punkte in den seismischen Daten zu markieren. Vor allem dafür wird Translations-Snapping eingesetzt, welches präzise Platzierung des Pickers auf Volume Slices, den Rändern von Volume Lenses und auch auf Polygonmodellen (z. B. aus den seismischen Daten extrahierte Grenzschichten, sog. *Horizons*) erlaubt (siehe Abbildung 8.2).

Im Gegensatz zur Manipulation von Workspaces, Slices und Lenses bleibt der Pick-Ray bei der Data-Picker-Manipulation stets eingeschaltet, um auch bei diesem relativ kleinen Objekt kontinuierlich Feedback über den Snapping-Status zu geben. Bei Beginn der Manipulation ändert der Pick-Ray zunächst seine Richtung, so dass er zur Verdeutlichung des für das Snapping relevanten Referenzpunkts stets auf den Fuß der Flagge zeigt. Darüberhinaus wird der ansonsten in der Farbe des jeweiligen Benutzers gezeichnete Pick-Ray grau eingefärbt, damit einheitliches Snapping-Feedback möglich ist. Wird nun mit dem Snap-and-go-Verfahren auf eine Oberfläche gesnappt, wird ein von der aktuellen Entfernung und der Snap-Range abhängiges rotes Highlight auf den Pick-Ray gelegt (siehe 6.2.1). Zusätzlich hat der Benutzer die Möglichkeit zwischen Bent-Pick-Ray- und Gummiband-Feedback zu wählen, welches den Snapping-Status zusätzlich zum Pick-Ray-Highlight visualisiert.

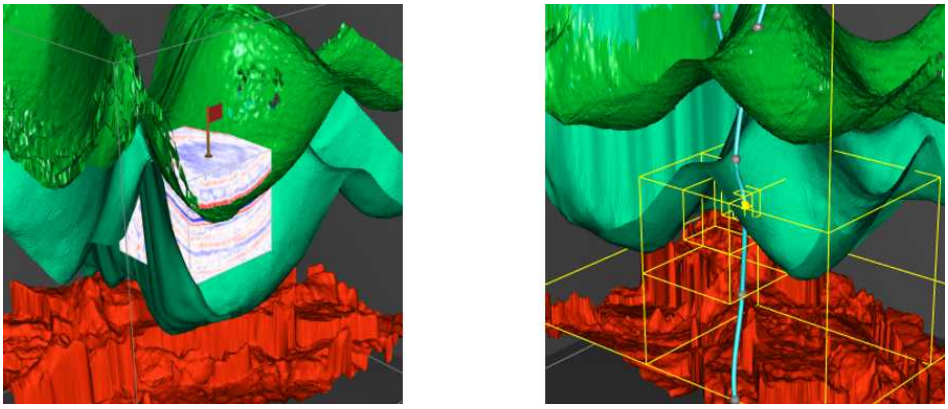

Abbildung 8.2: Data-Picker- und Drilling-Well-Snapping im VRGeo-Demonstrator (rechts ist zusätzlich der Octree zur Bestimmung des nächsten Nachbarn auf dem Horizon eingeblendet).

8.5 Snapping für Well-Kontrollpunkte

Der VRGeo-Demonstrator unterstützt prototypisch die Planung von Bohrpfaden (*Wells*). Mit denselben Methoden wie beim Data-Picker wird auch für die Kontrollpunkte der Wells Translations-Snapping und damit präzise Ausrichtung an Oberflächen im Workspace unterstützt (siehe ebenso Abbildung 8.2).

8.6 Feedback

Nach Erklärung der in den VRGeo-Demonstrator integrierten Snapping-Verfahren hatten ca. 20 Benutzer aus dem Umfeld der Öl- und Gasindustrie mit Expertise in Geophysik und Virtual Reality in einer dreistündigen Session Gelegenheit, die Techniken auszuprobieren und Feedback zu geben. Dabei fiel zunächst auf, dass alle Benutzer nach einer kurzen Einführung in die verwendeten Eingabegeräte sofort intuitiv mit den Objekten in der Applikation interagierten und eine gute Kontrolle über sie bewiesen. Einige Benutzer bemerkten sogar bei der Manipulation der Workspaces zunächst nicht, dass dort Rotations-Snapping eingesetzt wurde, obwohl sie eine gerade Ausrichtung bevorzugten und sie dadurch auch leicht erreichen konnten. Erst auf explizite Nachfrage hin wurde dies positiv bemerkt.

Ähnliche Erfahrungen gelten für die Interaktion mit den Volume Lenses und Volume Slices. Insbesondere bei den Slices gab es durchweg positives Feedback für die Möglichkeit, die Orientierung beliebig im Raum zu fixieren und sich dann entlang dieser Achse bewegen zu können. Nach Aussage einiger Experten ermöglicht diese Interaktion eine sehr detaillierte Betrachtung von meist nicht an den Hauptachsen der Volumendaten ausgerichteten Strukturen.

Die Benutzer interagierten insgesamt in natürlicher Weise auch mit den Objekten innerhalb eines Workspaces, ohne von der maximal erreichbaren Skalierung beim Scaled Grab in der Bewegung eingeschränkt zu werden. Erst auf Nachfrage fiel auf, dass der Pick-Ray beim

Connected Scaled Grab nicht gerade aus dem Eingabegerät heraustritt. Auch nachdem die Benutzer darauf aufmerksam gemacht wurden, hatten alle das Gefühl einer intuitiven und direkten Objektkontrolle. Ebenso wenig fühlten sich die Benutzer durch den beim Verlassen einer mit der Snap-and-go-Technik gesnappten Oberfläche entstehenden temporären Offset beeinträchtigt.

Das Pick-Ray-Highlight bei der Interaktion mit Data-Picker und Well-Kontrollpunkten wurde nach kurzer Erklärung sehr positiv aufgenommen. Die damit kombinierte Bent-Pick-Ray-Technik kam nicht nur visuell gut an, sondern erlaubte gutes Feedback nicht nur für die Distanz sondern auch den tatsächlichen Vektor zur aktuell gesnappten Oberfläche. Auch wenn der Bent-Pick-Ray bei Bewegungen genau in Strahlrichtung nicht so deutliches Feedback wie das Gummiband gibt, wurde diese Technik aufgrund der direkter wirkenden Objektkontrolle bevorzugt.

9 Zusammenfassung und Ausblick

In dieser Arbeit wurden Snapping-Techniken für präzise Objektmanipulation in Virtuellen Umgebungen vorgestellt. Dabei wurde insbesondere der Bezug zu den verwendeten Interaktionstechniken hergestellt und geeignete Snapping-Feedback-Verfahren entworfen und diskutiert. In diesem Zusammenhang wurde die Scaled-Grab-Technik erweitert, um während der Manipulation die Darstellung eines Pick-Rays zu ermöglichen, der die Basis für das Gummiband- und das pseudo-haptische Bent-Pick-Ray-Feedback ist.

Zur Vermeidung von Sprüngen beim Snapping von Translationen wurde das Snap-and-go-Verfahren für direkte 3D-Interaktion in Virtuellen Umgebungen adaptiert. Effizientes Snapping nicht nur auf einfache Primitive sondern auch auf komplexe aus Polygonen bestehende 3D-Modelle ermöglicht ein in dieser Arbeit präsentierter Octree-Algorithmus zur Nächster-Nachbar-Bestimmung.

Zwei unterschiedliche Rotations-Snapping-Ziel-Typen erlauben effektives Rotations-Snapping insbesondere zusammen mit der Scaled-Grab-Interaktionstechnik. Zur Visualisierung des Snapping-Status wurden auch hierfür Feedback-Techniken entwickelt.

Zur Evaluation der vorgestellten Snapping-Verfahren wurden Teile in eine prototypische immersive VR-Anwendung zur interaktiven Exploration seismischer Daten im Kontext der Öl- und Gasindustrie integriert. Diese wurde in einer mehrstündigen Session von Benutzern mit Expertise in Virtual Reality und Geophysik benutzt und bewertet.

In Zukunft planen wir, beim Translations-Snapping nicht nur einzelne Punkte bzw. einfach den Translations-Anteils einer Objekttransformation zu betrachten, sondern auch in Kombination mit Rotations-Snapping effiziente Verfahren zu entwickeln, die auch mehrere Einrastpunkte für Objekte erlauben. Darüberhinaus ist es wünschenswert, ähnlich wie beim Snap-and-go-Verfahren für Translations-Snapping auch beim Snapping von Rotationen Sprünge zu vermeiden, also eine graduelle Interpolation zwischen gesnappter und aktueller Originalorientierung zu erlauben.

Interessant wäre über das Snapping auf Polygonmodelle hinausgehend gerade im Kontext von Seismiken auch automatisches Snapping auf Merkmale in volumetrischen Daten, also eine Technik, die das in Kapitel 3 vorgestellte Image Snapping auf dreidimensionale Daten erweitert.

Literaturverzeichnis

3Dconnexion

3Dconnexion. http://www.3dconnexion.com. 13, 21, 33

Adobe

Adobe Systems Incorporated. http://www.adobe.de. 25

Andujar et al. 2006

Andujar, C ; Fairen, M. ; Argelaguet, F.: A Cost-Effective Approach for Developing Application-Control GUIs For Virtual Environments. In: *3DUI '06: Proceedings of the 3D User Interfaces*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 1–4244–0225–5, S. 45–52. http://dx.doi.org/10.1109/VR.2006.6. 36

Avango

Avango VR-Framework. http://sourceforge.net/projects/avango. 23, 73

Batagelo u. Wu 2005

Batagelo, Harlen C. ; Wu, Shin-Ting: What You See Is What You Snap: Snapping to Geometry Deformed on the GPU. In: *SI3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. New York, NY, USA : ACM Press, 2005. – ISBN 1–59593–013–2, S. 181–86. http://dx.doi.org/10.1145/1053427.1053440, http://www.dca.fee.unicamp.br/projects/mtk/batagelo. 32

Baudisch 1996

Baudisch, Patrick: The Cage: Efficient Construction in 3D Using a Cubic Adaptive Grid. In: *UIST '96: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA : ACM Press, 1996. – ISBN 0–89791–798–7, S. 171–172. http://dx.doi.org/10.1145/237091.237117, http://www.patrickbaudisch.com. 33

Baudisch et al. 2005

Baudisch, Patrick ; Cutrell, Edward ; Hinckley, Ken ; Eversole, Adam: Snap-and-go: Helping Users Align Objects Without the Modality of Traditional Snapping. In: *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press, 2005. – ISBN 1–58113–998–5, S. 301–310. http://dx.doi.org/10.1145/1054972.1055014, http://www.patrickbaudisch.com. 29, 30, 42

Bier 1988

BIER, Eric A.: Snap-Dragging: Interactive Geometric Design in Two and Three Dimensions / EECS Department, University of California at Berkeley. Berkeley, CA, USA, 1988 (UCB/CSD-88-416). – Also available as Xerox PARC report EDL-89-2 and his Ph.D. Thesis, 1989. 26

Bier 1990

BIER, Eric A.: Snap-Dragging in Three Dimensions. In: *SI3D '90: Proceedings of the 1990 Symposium on Interactive 3D Graphics*. New York, NY, USA : ACM Press, 1990. – ISBN 0–89791–351–5, S. 193–204. http://dx.doi.org/10.1145/91385.91446. 30, 35

Bier u. Stone 1986

BIER, Eric A. ; STONE, Maureen C.: Snap-Dragging. In: *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM Press, 1986. – ISBN 0–89791–196–2, S. 233–240. http://dx.doi.org/10.1145/15922.15912. 26

Bowman et al. 2005

BOWMAN, Doug A. ; KRUIJFF, Ernst ; LAVIOLA, Joseph J. ; POUPYREV, Ivan: *3D User Interfaces – Theory and Practice*. Reading, MA : Addison-Wesley, 2005. 17, 22

Butterworth et al. 1992

BUTTERWORTH, Jeff ; DAVIDSON, Andrew ; HENCH, Stephen ; OLANO, Marc. T.: 3DM: A Three Dimensional Modeler Using a Head-Mounted Display. In: *SI3D '92: Proceedings of the 1992 Symposium on Interactive 3D Graphics*. New York, NY, USA : ACM Press, 1992. – ISBN 0–89791–467–8, S. 135–138. http://dx.doi.org/10.1145/147156.147182. 33

Conner et al. 1992

CONNER, Brookshire D. ; SNIBBE, Scott S. ; HERNDON, Kenneth P. ; ROBBINS, Daniel C. ; ZELEZNIK, Robert C. ; DAM, Andries van: Three-Dimensional Widgets. In: *SI3D '92: Proceedings of the 1992 Symposium on Interactive 3D Graphics*. New York, NY, USA : ACM Press, 1992. – ISBN 0–89791–467–8, S. 183–188. http://dx.doi.org/10.1145/147156.147199. 31

Corel

Corel Corporation. http://www.corel.de. 25

Cruz-Neira et al. 1993

CRUZ-NEIRA, Carolina ; SANDIN, Daniel J. ; DEFANTI, Thomas A.: Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In: *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM Press, 1993. – ISBN 0–89791–601–8, S. 135–142. http://dx.doi.org/10.1145/166117.166134. 17

Dumas et al. 1998

DUMAS, C. ; DEGRANDE, S. ; SAUGIS, G. ; CHAILLOU, C. ; VIAUD, M.-L.: A 3-D Interface for Cooperative Work. In: *Proceedings of CVE'98 (Collaborative Virtual Environments 1998)*, 1998. 34

Fakespace

Fakespace Systems. http://www.fakespace.com. 17

Fitts 1954

FITTS, P. M.: The Information Capacity of the Human Motor System in Controlling Amplitude of Movement. In: *Journal of Experimental Psychology* 47 (1954), Nr. 6, S. 381–391. – Reprinted in Journal of Experimental Psychology: General, 121(3):262–269, 1992. 25, 30

Foley et al. 1995

FOLEY, James D. ; DAM, Andries van ; FEINER, Steven K. ; HUGHES, John F.: *Computer Graphics: Principles and Practice in C (2nd ed.)*. Reading, MA : Addison-Wesley, 1995. 27, 70

Fröhlich et al. 2005

FRÖHLICH, Bernd ; BLACH, Roland ; STEFANI, Oliver ; HOCHSTRATE, Jan ; HOFFMANN, Jörg ; KLÜGER, Karsten ; BUES, Matthias: Implementing Multi-Viewer Stereo Displays. In: *WSCG (Full Papers)*, 2005. – ISBN 80–903100–7–9, S. 139–146. 18

GeorgiaTechLGMA

Large Geometric Models Archive at Georgia Institute of Technology. http://www-static.cc.gatech.edu/projects/large_models. 55, 68

Gimp

GIMP - GNU Image Manipulation Program. http://www.gimp.org. 25

Glassner 1988

GLASSNER, A. S.: Space Subdivision for Fast Ray Tracing. (1988), S. 160–167. ISBN 0–8186–8854–4. 56

Gleicher 1995

GLEICHER, Michael: Image Snapping. In: *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM Press, 1995. – ISBN 0–89791–701–4, S. 183–190. http://dx.doi.org/10.1145/218380.218441. 28, 31

Gleicher u. Witkin 1994

GLEICHER, Michael ; WITKIN, Andrew: Drawing with Constraints. In: *The Visual*

Computer: International Journal of Computer Graphics 11 (1994), Nr. 1, S. 39–51. – ISSN 0178–2789. http://dx.doi.org/10.1007/BF01900698. 27

Gnome

Gnome Desktop. http://www.gnome.org. 25

Guttman 1984

GUTTMAN, Antonin: R-trees: A Dynamic Index Structure for Spatial Searching. In: *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM Press, 1984. – ISBN 0–89791–128–8, S. 47–57. http://dx.doi.org/10.1145/602259.602266. 55

de Haan et al. 2005

HAAN, Gerwin de ; KOUTEK, Michal ; POST, Frits H.: IntenSelect: Using Dynamic Object Rating for Assisting 3D Object Selection. In: *Eurographics Symposium on Virtual Environments*, The Eurographics Association, 2005, S. 201–209. http://dx.doi.org/10.2312/EGVE/IPT_EGVE2005/201-209. 35, 36

Hjaltason u. Samet 1999

HJALTASON, Gisli R. ; SAMET, Hanan: Distance Browsing in Spatial Databases. In: *ACM Transactions on Database Systems* 24 (1999), Nr. 2, S. 265–318. – ISSN 0362–5915. http://dx.doi.org/10.1145/320248.320255. 49, 53

Hsu et al. 1997

HSU, C. ; ALT, G. ; HUANG, Z. ; BEIER, E. ; BRÜDERLIN, B.: A Constraint-based Manipulator Toolset for Editing 3D Objects. In: *SMA '97: Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications*. New York, NY, USA : ACM Press, 1997. – ISBN 0–89791–946–7, S. 168–180. http://dx.doi.org/10.1145/267734.267779. 31

Hudson 1990

HUDSON, Scott E.: Adaptive Semantic Snapping - A Technique for Semantic Feedback at the Lexical Level. In: *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press, 1990. – ISBN 0–201–50932–6, S. 65–70. http://dx.doi.org/10.1145/97243.97253. 25, 28

Igarashi u. Hughes 2001

IGARASHI, Takeo ; HUGHES, John F.: A Suggestive Interface for 3D Drawing. In: *UIST '01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA : ACM Press, 2001. – ISBN 1–58113–438–X, S. 173–181. http://dx.doi.org/10.1145/502348.502379. 31

Infitec

INFITEC GmBH. http://www.infitec.net. 18

Inkscape

Inkscape. http://www.inkscape.org. 25

InterSense

InterSense Inc. http://www.intersense.com. 21

KDE

The K Desktop Environment. http://www.kde.org. 25

KeyPress

Key Curriculum Press. http://www.keypress.com. 25

Kim 2005

KIM, Gerard J.: *Designing Virtual Reality Systems*. London, UK : Springer-Verlag, 2005. – ISBN 1–85233–958–6. 38

Lecuyer et al. 2000

LECUYER, Anatole ; COQUILLART, Sabine ; KHEDDAR, Abderrahmane ; RICHARD, Paul ; COIFFET, Philippe: Pseudo-Haptic Feedback: Can Isometric Input Devices Simulate Force Feedback? In: *VR '00: Proceedings of the IEEE Virtual Reality 2000 Conference*. Washington, DC, USA : IEEE Computer Society, 2000. – ISBN 0–7695–0478–7, S. 83. http://dx.doi.org/10.1109/VR.2000.840369. 29, 71

Li et al. 2004

LI, Yin ; SUN, Jian ; TANG, Chi-Keung ; SHUM, Heung-Yeung: Lazy Snapping. In: *ACM Transactions on Graphics* 23 (2004), Nr. 3, S. 303–308. – ISSN 0730–0301. http://dx.doi.org/10.1145/1015706.1015719. 28

Lindeman et al. 2001

LINDEMAN, Robert W. ; SIBERT, John L. ; TEMPLEMAN, James N.: The Effect of 3D Widget Representation and Simulated Surface Constraints on Interaction in Virtual Environments. In: *VR '01: Proceedings of the Virtual Reality 2001 Conference*. Washington, DC, USA : IEEE Computer Society, 2001. – ISBN 0–7695–0948–7, S. 141. http://dx.doi.org/10.1109/VR.2001.913780. 71

Masui 2001

MASUI, Toshiyuki: HyperSnapping. In: *HCC '01: Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments*. Washington, DC, USA : IEEE Computer Society, 2001. – ISBN 0–7695–0474–4, S. 188. http://dx.doi.org/10.1109/HCC.2001.995258. 27

Microsoft

Microsoft Deutschland GmbH. http://www.microsoft.de. 25

Mine 1995

MINE, Mark R.: Virtual Environment Interaction Techniques / University of North Carolina at Chapel Hill. Chapel Hill, NC, USA, 1995 (TR95-018), http://www.cs.unc.edu/~mine/. 21

Nelson 1985

NELSON, Greg: Juno, A Constraint-Based Graphics System. In: *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM Press, 1985. – ISBN 0–89791–166–0, S. 235–243. http://dx.doi.org/10.1145/325334.325241. 27

Newman u. Sproull 1979

NEWMAN, William M. ; SPROULL, Robert F.: *Principles of Interactive Computer Graphics (2nd ed.)*. New York, NY, USA : McGraw-Hill, Inc., 1979. – ISBN 0–07–046338–7. 27, 70

Oh u. Stuerzlinger 2005

OH, Ji-Young ; STUERZLINGER, Wolfgang: Moving Objects with 2D Input Devices in CAD Systems and Desktop Virtual Environments. In: *GI '05: Proceedings of the 2005 Conference on Graphics Interface*. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada : Canadian Human-Computer Communications Society, 2005. – ISBN 1–56881–265–5, S. 195–202. 31

OpenSceneGraph

OpenSceneGraph. http://www.openscenegraph.org. 73

Poupyrev et al. 1996

POUPYREV, Ivan ; BILLINGHURST, Mark ; WEGHORST, Suzanne ; ICHIKAWA, Tadao: The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR. In: *UIST '96: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA : ACM Press, 1996. – ISBN 0–89791–798–7, S. 79–80. http://dx.doi.org/10.1145/237091.237102. 22

Riege et al. 2006

RIEGE, Kai ; HOLTKÄMPER, Thorsten ; WESCHE, Gerold ; FRÖHLICH, Bernd: The Bent Pick Ray: An Extended Pointing Technique for Multi-User Interaction. In: *3DUI '06: Proceedings of the 3D User Interfaces*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 1–4244–0225–5, S. 62–65. http://dx.doi.org/10.1109/VR.2006.127. 18, 49, 71

Roussopoulos et al. 1995

ROUSSOPOULOS, Nick ; KELLEY, Stephen ; VINCENT, Frederic: Nearest Neighbor Queries. In: *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on*

Management of Data. New York, NY, USA : ACM Press, 1995. – ISBN 0–89791–731–6, S. 71–79. http://dx.doi.org/10.1145/223784.223794. 50, 52, 53

SGI

SGI. http://www.sgi.com. 73, 74

Sherman u. Craig 2003

SHERMAN, William R. ; CRAIG, Alan B.: *Understanding Virtual Reality*. San Francisco, CA, USA : Elsevier Science, 2003. – ISBN 1–55860–353–0. 17, 38

Simon u. Doulis 2004

SIMON, Andreas ; DOULIS, Mario: NOYO: 6DOF Elastic Rate Control for Virtual Environments. In: *VRST '04: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA : ACM Press, 2004. – ISBN 1–58113–907–1, S. 178–181. http://dx.doi.org/10.1145/1077534.1077571. 20

Simon et al. 2005

SIMON, Andreas ; DRESSLER, Armin ; KRÜGER, Hans-Peter ; SCHOLZ, Sascha ; WIND, Jürgen: Interaction and Co-located Collaboration in Large Projection-Based Virtual Environments. In: *INTERACT*, 2005, S. 364–376. http://dx.doi.org/10.1007/11555261_31. 19, 22, 67

Simon u. Göbel 2002

SIMON, Andreas ; GÖBEL, Martin: The i-Cone – A Panoramic Display System for Virtual Environments. In: *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1784–6, S. 3. http://dx.doi.org/10.1109/PCCGA.2002.1167834. 18

Simon u. Scholz 2005

SIMON, Andreas ; SCHOLZ, Sascha: Multi-Viewpoint Images for Multi-User Interaction. In: *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*. Washington DC, USA : IEEE Computer Society, 2005. – ISBN 0–7803–8929–8, S. 107–113. http://dx.doi.org/10.1109/VR.2005.55. 19, 49, 68

Simon et al. 2004

SIMON, Andreas ; SMITH, Randall C. ; PAWLICKI, Richard R.: OmniStereo for Panoramic Virtual Environment Display Systems. In: *VR '04: Proceedings of the IEEE Virtual Reality 2004*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7803–8415–6, S. 67. http://dx.doi.org/10.1109/VR.2004.56. 19

Springer et al. 2000

SPRINGER, J. ; TRAMBEREND, H. ; FRÖHLICH, B.: On Scripting in Distributed Virtual Environments. In: *4. Immersive Projection Technology Workshop*, 2000. 75

Stork 2000

STORK, André: An Algorithm for Fast 3D Picking and Snapping Using a 3D Input Device and 3D Cursor. In: *CAD Tools and Algorithms for Product Design (Dagstuhl Seminar, November 1998)*. London, UK : Springer-Verlag, 2000. – ISBN 3–540–66204–9, S. 113–127, http://www.igd.fhg.de/igd-a2/publications/Dagstuhl98.pdf. 34

Stork u. Maidhof 1997

STORK, André ; MAIDHOF, Martin: Efficient and Precise Solid Modelling Using a 3D Input Device. In: *SMA '97: Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications*. New York, NY, USA : ACM Press, 1997. – ISBN 0–89791–946–7, S. 181–194. http://dx.doi.org/10.1145/267734.267782. 34

Sutherland 1963

SUTHERLAND, Ivan E.: Sketchpad: A Man-Machine Graphical Communication System / MIT, Lincoln Laboratory. 1963 (296). 26, 27, 38

Sutherland 1964

SUTHERLAND, Ivan E.: Sketchpad: A Man-Machine Graphical Communication System. In: *DAC '64: Proceedings of the SHARE Design Automation Workshop*. New York, NY, USA : ACM Press, 1964, S. 6.329–6.346. http://dx.doi.org/10.1145/800265.810742. 26

Tramberend et al. 1999

TRAMBEREND, H. ; HASENBRINK, F. ; FRÖHLICH, B.: Tools, Mediators, and Interaction Operators. In: *Proceedings of the 3. Immersive Projection Technology Workshop*, 1999, S. 77–79. 76

Tramberend 1999

TRAMBEREND, Henrik: Avocado: A Distributed Virtual Reality Framework. In: *IEEE Virtual Reality 1999 Conference (VR'99)*, 1999, S. 14–21. 23, 73

Tramberend 2003

TRAMBEREND, Henrik: *Avocado: A Distributed Virtual Environment Framework*, Universitat Bielefeld, Diss., 2003. http://bieson.ub.uni-bielefeld.de/volltexte/2003/299/. 23, 73

Venolia 1993

VENOLIA, Dan: Facile 3D Direct Manipulation. In: *CHI '93: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press, 1993. – ISBN 0–89791–575–5, S. 31–36. http://dx.doi.org/10.1145/169059.169065. 33, 34

VRGeo

VRGeo Konsortium. http://www.vrgeo.org. 14, 83

Wu et al. 2003

WU, Shin-Ting ; ABRANTES, Marcel ; TOST, Daniel ; BATAGELO, Harlen C.: Picking and Snapping for 3D Input Devices. In: *XVI Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'03)*. Los Alamitos, CA, USA : IEEE Computer Society, 2003. – ISSN 1530–1834, S. 140. http://dx.doi.org/10.1109/SIBGRA.2003.1241002. 32, 34, 35

Yoo u. Ha 2004

YOO, Kwan-Hee ; HA, Jong S.: Geometric Snapping for 3D Meshes. In: *Computational Science - ICCS 2004: 4th International Conference*, Springer, 2004. – ISBN 978–3–540–22129–6, S. 90–97. http://dx.doi.org/10.1007/b98005. 31