UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

COMPUTERVISUALISTIK

Einfaches kooperatives Modellieren von virtuellen Gegenständen

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Rodja Trappe

Erstgutachter:	Prof. Dr. -Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl. -Inform. Oliver Abert (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Januar 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☒	☐

.....

(Ort, Datum) (Unterschrift)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

COMPUTERVISUALISTIK

Aufgabenstellung für die Diplomarbeit
Rodja Trappe (Matr.-Nr. 201210811)

Thema: Einfaches kooperatives Modellieren von virtuellen Gegenständen

In der Computergrafik werden virtuelle Objekte miteinander verknüpft und dargestellt. Oft ist es nötig, solche Objekte manuell zu erstellen. Dazu wird immer ein Werkzeug benötigt, mit dem Ideen und Vorstellungen in den Computer eingegeben werden können. Anwendungen für diesen Zweck sind wegen ihrer hohen Komplexität meist nur von Experten bedienbar. Problematisch wird dies vor allem in Projekten, bei denen virtuelle Objekte nur als Beiwerk fungieren. Wenn die Qualitätsanforderungen an die Darstellung nicht sehr hoch sind, passiert es leicht, dass keine Experten greifbar sind, die ansprechende und gute Geometrie beisteuern könnten.

Werkzeuge, mit denen nach kurzer Einarbeitung ästhetische und technisch verwendbare virtuelle Gegenstände mit wenig Zeitaufwand produziert werden können, sind rar. Besonders schwierig ist in diesem Zusammenhang auch das gemeinsame Arbeiten an und mit den Geometriedaten. Organisation, Versionierung und Zugriff sind oft chaotisch, da sie dem unkundigen Benutzer überlassen werden.

Ziel dieser Diplomarbeit ist es daher, eine mögliche Lösung für die genannten Schwierigkeiten zu finden. Es soll eine Anwendung konzipiert werden, mit der relativ unerfahrene Benutzer im Team ansprechende virtuelle Gegenstände anfertigen können. Dazu muss die Mächtigkeit des Werkzeugs zu Gunsten einer einfachen Bedienung sinnvoll begrenzt werden. Es ist offensichtlich, dass so bestimmte Arten von virtueller Geometrie nicht abgedeckt werden können. Das Werkzeug wird sich dementsprechend auf die Verwaltung und Bearbeitung einer kleinen Anzahl von Einzelgegenständen geringer Komplexität ohne Texturen und Animationen beschränken.

Schwerpunkte dieser Arbeit sind:

- Einarbeiten in die Literatur und Formulieren der Anforderungen
- Entwurf der Anwendung und Darlegen der Entscheidungen
- Implementation des Werkzeugs
- Dokumentation der Ergebnisse

Koblenz, den 31. Juli 2006

– Prof. Dr. Stefan Müller –

Inhaltsverzeichnis

Abkürzungsverzeichnis

bzw. beziehungsweise

B-Rep *Boundary Representation*

ca. circa, in etwa

CNC *Computerized Numerical Control*, computerisierte numerische Steuerung

coi *center of interest*, Betrachtungsmittelpunkt

CPU *Central Processing Unit*, bezeichnet den Hauptprozessor eines Computers

CSCW *Computer-Supported Cooperative Work*

CSG *Constructive Solid Geometry*

d.h. das heißt

dt. deutsch

DVD *Digital Video Disk*, bezeichnet ein Speichermedium

EDV Elektronische Datenverarbeitung

etc. und so weiter (lat. et cetera)

engl. englisch

GPU *Graphics Processing Unit*, bezeichnet den Grafikprozessor eines Computers

GUI *Graphical User Interface*, bezeichnet die grafische Benutzungsschnittstelle

IP *Internet Protocol*

LAN *Local Area Network*

lat. lateinisch

NURBS *Non Uniform Rational B-Splines*

OpenGL *Open Graphics Library*, Spezifikation einer Programmier-Schnittstelle für Computergrafik

SBM *Sketch Based Modeling*

u.ä. und ähnliches

ugs. umgangssprachlich

USB *Universal Serial Bus*, bezeichnet eine Geräte-Anschlußtechnik für Computer

usw. und so weiter

vgl. vergleiche

WIMP *Windows, Icons, Menues, Pointer*, bezeichnet ein Software-Ergonomisches Konzept zur Gestaltung von GUIs

z.B. zum Beispiel

Abbildungsverzeichnis

Kapitel 1.

Einführung

> Our long term goal should be to design intuitive systems that require a minimum of instruction.
>
> *Christine L. Borgman*

Die rasant fortschreitende Entwicklung in der Computergrafik beeinflusst mittlerweile große Teile der Informatik und ermöglicht immer mehr Software-Projekten einen Nutzen aus der dreidimensionalen Visualisierung zu ziehen. Unternehmungen in diesem Bereich erfordern allerdings virtuelle Szenen, in denen etwas veranschaulicht, präsentiert oder vorgeführt werden soll. Solche Szenen bestehen aus virtuellen Objekten – die irgendwo her kommen müssen.

1.1. Problemstellung

Die Konstruktion virtueller Objekte muss oft manuell erfolgen. Entweder, weil diese Objekte einen speziellen Zweck zu erfüllen haben, der mit anderen Methoden schwer zu erreichen ist, oder weil keine anderen Methoden zur Verfügung stehen. Heutige Software-Produkte zu diesem Zweck richten sich fast ausschließlich an professionelle Designer, die damit nach einer Ausbildung effizient hochkomplexe Geometrien erschaffen können. Doch nicht jedes Projekt besitzt die Ressourcen, solche ausgebildeten Künstler zu beschäftigen. Es wäre daher äußerst wünschenswert, wenn 3D-Inhalte auch von weniger versierten Personen an Standard-*Desktop*-Computern nach kurzer Einarbeitung produziert werden könnten – selbst wenn der Prozess dann etwas länger dauert.

1.2. Zielvorgabe

Ziel dieser Diplomarbeit ist es, eine Anwendung zu konzipieren, mit deren Hilfe relativ unerfahrene Benutzer leicht im Team virtuelle Gegenstände[1] anfertigen können. Dazu muss die Vielfalt und Funktionalität der Werkzeuge zu Gunsten einer einfachen Bedienung sinnvoll begrenzt und durch Automatismen unterstützt werden. Da dieser Blickwinkel auf die Problematik zur Konstruktion virtueller Objekte in der Computergrafik neuartig ist, setzt die Arbeit den Fokus auf die Analyse und Beschreibung eines möglichen Lösungsweges bei gleichzeitiger prototypischer Umsetzung zur Untermauerung der aufgestellten Hypothesen.

1.3. Anwendungszenario

Im Rahmen eines größeren Projekts werden ca. ein Dutzend virtuelle Gegenstände zur Visualisierung gebraucht[2]. Eine Gruppe aus drei Personen soll sich um die Konstruktion kümmern. Niemand von ihnen hat Erfahrungen in der 3D-Modellierung und Datenorganisation im Team; der Umgang mit dem Computer ist jedoch geübt und sicher. Für die virtuellen Modelle wird kein Fotorealismus verlangt, wohl aber eine ansprechende Darstellung, einheitliches Design und technische Nützlichkeit. Das Team hat von der Einarbeitung bis zur Präsentation der Resultate nur fünf Arbeitstage Zeit. Ein gemeinsamer Arbeitsraum mit Standard-EDV-Ausstattung steht zur Verfügung.

1.4. Gang der Untersuchung

Da das Ziel dieser Diplomarbeit der Entwurf einer Anwendung ist, bietet sich ein stark software-technisches Vorgehen [Bal98] an. Daher beginnt diese Arbeit mit einer bewertenden Analyse bestehender Techniken, Prinzipien und Anwendungen. So wird ein allgemeiner Überblick geschaffen, durch den sich die möglichen Lösungswege abzeichnen. Auf dieser Basis lassen sich dann Anforderungen und Ziele begründet definieren. Sobald klar ist, welche Wünsche umzusetzen sind, wird ein Software-Design erarbeitet. Hierzu gehören vor allem die Identifizierung von Funktionalitäten, Datenflüssen, Komponenten und deren Aufbau in einer Architektur. Der letzte Schritt zum fertigen Programm besteht in der Implementation. Erst hier stellt sich die konkrete Frage nach der Programmiersprache, den Entwicklungswerkzeugen und möglichen Hilfsbibliotheken.

Benutzungstests sind unabdingbar, wenn es um die Bewertung benutzungsorientierter Software-Produkte geht. Deshalb ist diesem Thema anschließend ein

[1] Geometrie in der Computergrafik stellt fast immer einen realen Gegenstand dar, weswegen hier ausdrücklich von einer vageren Bezeichnung wie Modell, Objekt oder Geometrie abweichend der Begriff »virtueller Gegenstand« eingeführt wird.

[2] Zum Beispiel ein Schwarm Fische

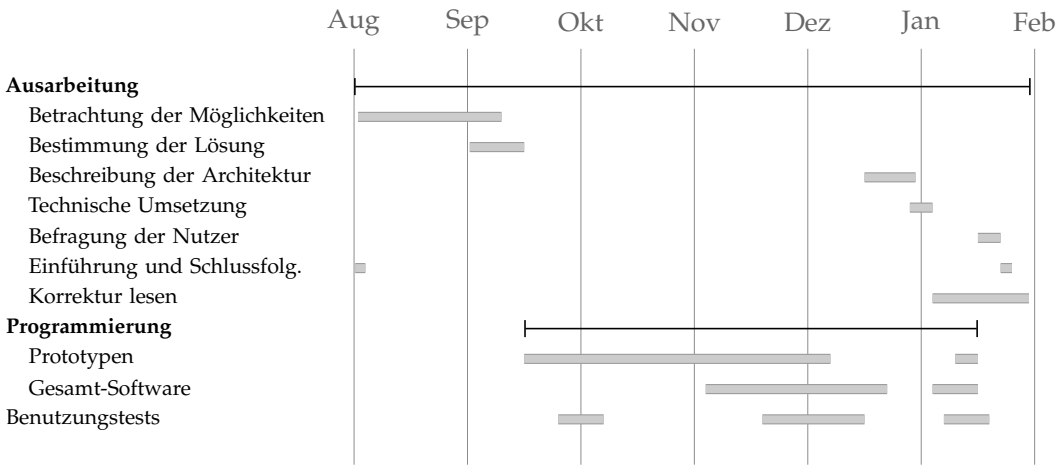


Abbildung 1.1.: Zeitlicher Ablauf der Teilaufgaben in dieser Diplomarbeit.

eigenes Kapitel gewidmet. Daraufhin endet diese Arbeit mit einem Resümee der durchgeführten Schritte und der erzielten Ergebnisse sowie einem Ausblick auf zukünftige Möglichkeiten in diesem Forschungsthema.

Es liegt auf der Hand, dass dieser Ablauf nur bedingt chronologisch fortschreiten kann. Benutzungsumfragen sollten z. B. so bald wie möglich nach der Anforderungsdefinition erfolgen, um zu verifizieren, ob das Gedachte auch in der Realität Bestand hat. Demnach werden die Schritte zwar der logischen Reihenfolge nach präsentiert, beeinflussen sich jedoch während der eigentlichen Arbeit stark. Der zeitlicher Ablauf lässt sich dem Gantt-Diagramm [Cla52] in Abbildung 1.1 entnehmen.

Kapitel 2.

Betrachtung der Möglichkeiten

Das also war des Pudels Kern

Goethes Faust I, Studierzimmer

Der Anspruch, ein einfaches kooperatives Modellierungswerkzeug für virtuelle Gegenstände zu konzipieren, ist zunächst sehr allgemein gefaßt. Darum ist es wichtig, mit einer Evaluation der Möglichkeiten auf breiter Ebene zu beginnen. In diesem Kapitel soll es daher um eine bewertende Betrachtung der Optionen im Hinblick auf das gesetzte Ziel gehen. Zu Beginn werden verschiedene Repräsentationsformen virtueller Geometrie betrachtet, denn sie sind die Grundlage für die dann folgende Untersuchung der Konstruktionstechniken. Dort finden sich die maßgeblichen Feststellungen für die Wahl der Benutzungsschnittstelle. Anschließend werden die Prinzipien der Datenorganisation betrachtet, denn erst auf dieser Basis ist eine Bewertung der Möglichkeiten zum Teamwork sinnvoll. Genau das ist Thema des vorletzten Abschnitts: Wie können mehrere Personen gemeinsam an virtuellen Gegenständen arbeiten? Beendet wird dieses Kapitel mit einem Überblick zu einigen Software-Produkten, die in diesem thematischen Zusammenhang besonders interessant sind.

2.1. Geometrie

In der Computergrafik gibt es eine Vielzahl an Datenstrukturen und Methoden, mit denen sich Geometrie beschreiben lässt (vgl. Abbildung 2.1). Viele Repräsentationen beschreiben ein Objekt durch seine Oberfläche[1] (engl. *surface*), also der zwei-mannigfaltigen[2] (engl. *two-manifold*) Grenze (engl. *boundary*) zwischen dem Inneren und Äußeren des Objektes. Diese Art der Beschreibung wird auch *Boundary Representation*, kurz *B-Rep*, genannt. Wenn eine Beschreibung der Oberfläche

[1] Die CSG Repräsentation hat weitestgehend ausgedient und bezeichnet jetzt ein Konstruktionskonzept (siehe auch Abschnitt 2.2.6 auf Seite 13).

[2] Mannigfaltigkeit (engl. *manifold*) ist ein fundamentales Konzept der algebraischen Topologie und zentraler Gegenstand der Differenzialgeometrie. Zwei-mannigfaltig bedeutet, dass ein topologischer Raum lokal einer Scheibe gleicht.

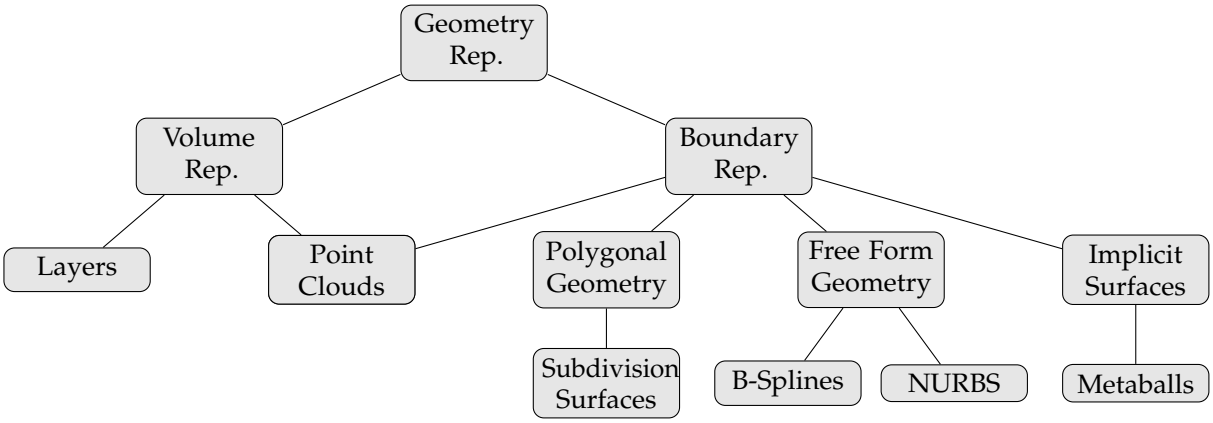


Abbildung 2.1.: Hierarchische Einordnung der Begriffe, die die verschiedenen Geometriedarstellungen im Englischen bezeichnen.

allein nicht ausreicht, kommen Volumendaten (engl. *Volume Data*) zum Einsatz. Im Folgenden werden die wichtigsten Techniken zur Repräsentation von Geometrie (siehe auch Abbildung 2.1) kurz vorgestellt und im Hinblick auf das in Kapitel 1 beschriebene Ziel bewertet.

2.1.1. Freiform-Geometrie

Freiform-Geometrie (engl. *free form geometry*) ist ein sehr verbreitetes Prinzip, um die Hülle eines Objektes zu beschreiben. Dabei wird durch mehrere *Splines* (dt. Kurven) ein *Patch* (dt. Fläche) definiert. *Splines* lassen sich am besten in ihrer Repräsentation als stückweise polynomiale Kurven verarbeiten [Rog00] (z. B. Bézier, B-Splines und NURBS). Jedes *Patch* zeichnet sich so durch eine wohl definierte, hinreichend glatte und beliebig geschwungene Oberfläche aus. Da NURBS eine Verallgemeinerung aller sonst üblichen *Splines* darstellen, wird Software für Freiform-Geometrie gern auf dieser Basis entwickelt. Dank rationalen Basisfunktionen können mit NURBS auch Kegelschnitte beschrieben werden, so dass z. B. »echte« Kreise mit dieser Technik repräsentiert werden können [Pie91]. Aus diesem Grund ist gerade im industriellen Produktentwurf[3] diese Art der Repräsentation traditionell stark vertreten[4].

Die Freiform-Geometrie – basierend auf NURBS – ist mathematisch wohlverstanden und seit Jahrzehnten im Praxiseinsatz. Durch die gute Kontrolle über weiche Formen lassen sich gekrümmte Oberflächen sehr leicht erstellen. Da sich

[3] Bei der Maschinensteuerung (CNC) zur Produktfertigung werden dann andere Datenformate verwendet [Mad96]

[4] Traditionell deshalb, weil NURBS die erste, allgemeine Beschreibungsform von Geometrie waren und sich seitdem in industriellen Standards wiederfinden [Far02].

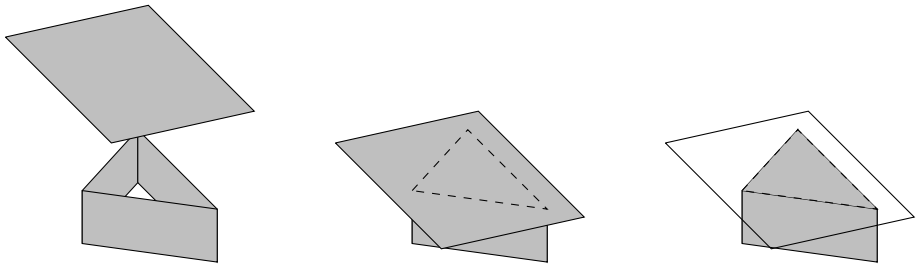


Abbildung 2.2.: Das Trimmen von Freiform-Flächen ist fast immer unvermeidbar: Da *Patches* rechteckig sind, müssen alle anderen Formen durch Schnitte im Parameterraum erstellt werden.

aus den Kontrollpunkten der *Splines* die Kurve berechnen lässt, wird relativ wenig Speicherplatz benötigt. Leider lässt sich durch ein einzelnes Patch nur Geometrie darstellen, die topologisch isomorph zu einer Scheibe, einem Zylinder oder einem Torus ist. Somit müssen schon für primitive Gegenstände immer mehrere *Patches* aneinander gefügt werden. Die daraus resultierenden Probleme werden in Abschnitt 2.2.1 aufgezeigt. Durch die topologische Unflexibilität von Freiform-Flächen ergibt sich die Notwendigkeit, diese zu *trimmen*. Das bedeutet, jedes *Patch* muss auch in seinem Parameterraum begrenzbar sein (siehe Abbildung 2.2). Der Benutzer muss also die Möglichkeit haben, weitere Kurven im Parameterraum eines einzelnen *Patches* zu definieren, um so an Hand dieser die eigentlich darzustellende Fläche festzulegen. Es liegt auf der Hand, dass ein solches Vorgehen ein grundlegendes Verständnis von *Splines* und *Patches* erfordert. Dem unversierten Benutzer erschließt sich die Problematik und Lösung nicht auf Anhieb, da es sich um eine mathematisch-technische Schwierigkeit handelt, welche keine visuelle Darstellungsform besitzt [Hol97].

2.1.2. Polygonale Netze

Die trivialste Beschreibung von Oberflächen stellen die polygonalen Netze (engl. *polygonal meshes*) dar. Wie der Name schon sagt, bestehen sie aus einem Verbund von Polygonen; im einfachsten Fall Dreiecke. Im Kontext von *Meshes* werden die Polygone *Faces* genannt, ihre aneinanderstoßenden Eckpunkte *Vertices* und die so entstehenden Kanten *Edges*[5]. Ein *Mesh* definiert einen virtuellen Gegenstand durch stückweise lineare Approximation der Oberfläche. Eine solche Approximation ist besonders gut für eine algorithmische Weiterverarbeitung (Stichwort: *Mesh Processing* [BPR+06]) geeignet.

[5] Ein *Mesh* kann daher auch als Graph angesehen werden, bei dem die *Vertices* Knoten und die *Edges* Kanten beschreiben.

Ein besonderer Vorteil von *Meshes* ist die einfache und sehr effiziente Repräsentation. Daher liegt sie auch der aktuellen Grafikhardware zugrunde[6]. *Meshes* lassen sich leicht manipulieren und mit Hilfe von Algorithmen modifizieren. Dank der primitiven Beschreibung sind beliebige topologische Gestalten darstellbar. Sollen jedoch glatte Formen entstehen, muss dies durch sehr viele und dementsprechend kleine Flächen angenähert werden. Die so genannten *Subdivision* Algorithmen beschreiben Verfahren, mit denen durch wiederholtes Zerteilen der *Faces* und Neupositionieren der *Vertices* eine hinreichend glatte Oberfläche definiert wird [ZSD+00]. Dadurch erhalten *Meshes* dieselben Eigenschaften wie B-Spline *Patches* – jedoch ohne die in Abschnitt 2.1.1 genannten topologischen Nachteile; diese Technik wird damit auch als Verbesserung von Freiform-Flächen angesehen. Die bekanntesten Algorithmen sind Loop [Loo87], Catmull-Clark [CC78] und $\sqrt{3}$ [Kob00].

Verschiedene Ansätze [DKT98, RLWH05][7] ermöglichen es auch, die *Subdivision* so anzupassen, dass dem Ursprungs-*Mesh* weitere Werte mitgegeben werden, mit denen sich dann beliebig scharfe Kanten und Ecken definieren lassen, die zwar als solche auf dem resultierenden Objekt zu erkennen sind, jedoch nichts die Glattheit[8] der Oberfläche beeinflussen. Mit einer Detail-Hierarchie (siehe Abschnitt 2.2.5) lässt sich dies noch weiter verallgemeinern. Auch das Beschreiben von Kegelschnitten [NF02, MWW01, SZSS98] und das Trimmen der Oberfläche [LLS01] ist durch Erweiterung der Standard-*Subdivision* Algorithmen möglich.

Die Positionierung und Verknüpfung der *Vertices* entscheidet über die Qualität dieser Geometriebescheibung. Zum Beispiel sollten die *Faces* möglichst uniform (d. h. die Größe ist regulär) und die Kanten immer entlang der geringsten Krümmung verlaufen. Wird bei der Erstellung des *Meshs* nicht auf solche Anforderungen geachtet, kann dies teilweise durch *Mesh Processing* nachgeholt werden.

2.1.3. Implizite Oberflächen

Implizit definierte Oberflächen (engl. *implicit surfaces*) [Blo97] sind besonders gut geeignet, um organische Objekte mit beliebiger Topologie darzustellen. Sie nutzen die Eigenschaft aus, dass ein eingeschlossenes Volumen durch eine implizite Gleichung wie z. B. $x^2 + y^2 + z^2 - 1 = 0$ mathematisch beschrieben werden kann. Natürlich sind solche Gleichungen für sinnvolle virtuelle Gegenstände um vieles komplizierter und kaum benutzungsfreundlich. Daher gibt es einige hilfreiche Vereinfachungen, wie beispielsweise die sogenannten *Metaballs* [Bli82]. Hier werden vom Anwender einzelne Punkte, so genannte *Atome*, zu komplexen Formen »ver-

[6] Grafikkarten machen sich z. B. die lokale Unabhängigkeit der Polygone zu nutze, durch die das Rendering von Geometrie stark parallelisiert werden kann.

[7] Die Pixar Animation Studios besitzen ein Patent mit der Nummer 6489960 auf den in ihrer Veröffentlichung beschriebenen Ansatz.

[8] Mit »glatt« werden in der Mathematik Abbildungen bezeichnet, die unendlich oft differenzierbar sind.

schmolzen«. Mathematisch lassen sich die Punkte und ihre Abstände zueinander als implizite Gleichungen ausdrücken. Dies entspricht genau dem Verschmelzungseffekt. Noch mehr Flexibilität kann erreicht werden, wenn neben Punkten auch andere geometrische Primitive wie etwa Linien zugelassen werden. Mit diesen lässt sich dann auf einfache Art und Weise ein Skelett konstruieren, das implizit die äußere weiche Oberfläche repräsentiert. Einen vielversprechenden Ansatz bieten die *Implicit Variational Surfaces* [TO99], bei denen – inspiriert von den *Variational Surfaces* [WW92] – die Form durch äußere *Constraints* vorgegeben wird.

Leider ist es für manche Vorhaben beim aktuellen Stand der Technik sinnvoller, die impliziten Oberflächen in eine andere Repräsentation wie z. B. Polygone [Blo88] umzuwandeln. Hierzu zählt beispielsweise das interaktive Darstellen der Geometrie [Pop04] und die Analyse von Oberflächeneigenschaften. Problematisch ist auch die Tatsache, dass sich mit intuitiven Vereinfachungen wie den *Metaballs* technisch bedingt nur sehr schwer scharfen Kanten und Ecken modellieren lassen.

2.1.4. Punktwolken

Ein virtuelles Objekt kann auch durch eine Menge von Punkten beschrieben werden. Je nach Anwendung repräsentiert die Punktwolke (engl. *Point Cloud*) allein die Hülle (engl. *Boundary*) oder das gesamte Volumen eines Gegenstandes. Letzteres ist hervorragend geeignet, um einen Körper mit all seinen innenliegenden heterogenen Details zu speichern. Beispielhaft sei hier die Visualisierung von medizinischen Daten genannt. Die Anordnung der Punkte erfolgt entweder frei im Raum oder regulär; zum Beispiel in einem *Voxel Grid*. Aufgrund ihrer diskreten Beschreibung werden Datensätze mit Punktwolken schon bei kleinen Objekten sehr groß, was die Verarbeitung erschwert. Da die einzelnen Punkte generell keine Konnektivität besitzen, also nicht mit anderen Punkten verbunden sind, müssen Nachbarschaften, Oberflächen und alles andere durch rechenintensive Algorithmen aus der räumlichen Ordnung und den Informationen, die einen einzelnen Punkt beschreiben (z. B. Farbe, Dichte, Elastizität, ...), gewonnen werden. Daraus resultiert eine hohe Funktionsvielfalt zur Verarbeitung der Daten [ZPKG02].

2.1.5. Gegenstände

Geometrische Festkörper (engl. *solids*, ugs. Gegenstände) sind keine eigene Beschreibungsform für virtuelle Objekte, sondern stellen vielmehr weitere Anforderungen an die zugrunde liegende mathematische Beschreibung [Req80]: Die Oberfläche muss eine Grenze zwischen dem innenliegenden und dem äußeren Bereich beschreiben sowie zwei-mannigfaltig ohne Rand sein. Das bedeutet, es gibt keine Teile der Oberfläche, die nicht mit dem Inneren des Objektes verbunden sind, und keine Löcher in der Oberfläche. Weiterhin ist ein Festkörper in seinen Maßen immer endlich und seine Oberfläche »regulär«, also endlich darstellbar.

Alle vier vorgestellten Geometrie-Repräsentationen können diese Kriterien erfüllen[9]. Typischerweise werden bei einer Festkörper-Repräsentation allerdings noch erweiternde Attribute benötigt, die bei realen Gegenständen vorhanden sind. Es sollte z. B. die Möglichkeit bestehen, das Volumen zu ermitteln, das Gewicht zu bestimmen, Materialinformationen zu erhalten oder den Masseschwerpunkt abzufragen.

Nahezu alle Geometrie, die in virtuellen Szenarien Verwendung findet, kann und sollte die grundlegenden Kriterien eines Festkörpers einhalten. Es sind schließlich Abbilder realer Gegenstände – oder lehnen sich daran an. Solange es nur um eine rein visuelle Darstellung von virtuellen Gegenständen geht, sind physikalische Informationen nicht unbedingt nötig. Diese erweiternden Attribute werden vor allem bei der physikalischen Simulation sowie der Konstruktionsplanung benötigt.

2.2. Konstruktionstechniken

Die Erstellung von virtuellen Gegenständen erfolgt entweder maschinell, prozedural, manuell oder in einer Mischung aus diesen drei Methoden. Eine maschinelle Fertigung besteht aus dem *Scannen* (dt. Abtasten) von realen Objekten. Als prozedural werden Techniken bezeichnet, die mit Hilfe von Algorithmen nach Eingabe einer Anzahl von Parametern selbstständig ein Resultat hervorbringen. Die manuelle Konstruktion von Geometrie wird als Modellierung bezeichnet. Um virtuelle Objekte benutzergestützt erstellen zu können, bedarf es Metaphern für die Handhabung der zugrunde liegenden Repräsentation. In diesem Abschnitt werden die wichtigsten Techniken für diesen Zweck vorgestellt, erläutert und bewertet.

2.2.1. Einzelflächen

Freiform-Geometrie (siehe Abschnitt 2.1.1 auf Seite 6) muss wegen der topologischen Beschränktheit ihrer zugrunde liegenden Mathematik (vgl. 2.1.1) durch viele Einzelflächen (engl. *Patches*) angenähert werden. Die Krümmungen jeder Einzelfläche werden durch eine Vielzahl von Kontrollpunkten im Raum bestimmt. Die Menge der Kontrollpunkte hängt von der Anzahl der verwendeten Kurven sowie dem Grad des entsprechenden Polynoms ab und unterscheidet sich somit von *Patch* zu *Patch*. Die Differenzierbarkeit einer Fläche wird durch ihren Polynomgrad (Ordnung) festgelegt und lässt sich nicht intuitiv visualisieren, kann aber oft automatisch mit sinnvollen Werten belegt werden[10]. Ähnlich verhält es sich mit den

[9] Auch Punktwolken können durch Interpolation zwischen räumlich benachbarten Punkten eine Grenze zwischen Innen und Außen definieren.

[10] In bestimmten Teilbereichen der Modellierung, wie z. B. bei der Konstruktionsplanung, bedarf die Ordnung jedoch einer individuellen Kontrolle und wird daher als Zahlenwert eingegeben.

bei rationalen Darstellungen[11] vorkommenden Gewichten an den Kontrollpunkten [Hon]. Sie ermöglichen beispielsweise die Beschreibung von mathematisch korrekten Kegelschnitten, können aber nicht durch simple Benutzerinteraktion zielgerichtet eingestellt werden.

Für die Beschreibung eines virtuellen Gegenstandes (siehe Abschnitt 2.1.5 auf Seite 9), müssen mehrere *Patches* »zusammengenäht« werden. Um die dabei entstehenden Unstetigkeiten an den Flächengrenzen zu verhindern, werden Algorithmen benötigt, die eine Benutzungsschnittstelle unter Umständen verkomplizieren. Dem Benutzer müssen zudem mächtige Werkzeuge für das Trimmen von *Patches* zur Verfügung gestellt werden, wie bereits in Abschnitt 2.1.1 und Abbildung 2.2 auf Seite 7 erläutert wurde.

2.2.2. Käfig

Für die Bearbeitung von polygonalen Netzen (siehe Abschnitt 2.1.2) wird in der Praxis die Metapher eines Käfigs (engl. *Cage*) verwendet. Dieser entspricht der Hülle des Objektes, wobei die Kanten der Polygone ein dreidimensionales Gitter bilden. Das Gitter kann der Benutzer mit verschiedenen Werkzeugen verändern. Beispielsweise lassen sich die Knotenpunkte verschieben, neue Kanten einfügen bzw. löschen und Teile des Gitters durch Unterteilungen verfeinern. Da bei dieser Konstruktionstechnik meist mit einer Kiste (engl. *Box*) begonnen wird, ist sie auch unter dem Namen *Box Modelling* bekannt. Prinzipiell ist die Arbeitsweise schnell verständlich, einfach zu implementieren und ermöglicht die Konstruktion von Objekten mit beliebiger Topologie. Es können auch leicht weitere Kriterien, wie z. B. die Forderung nach einer geschlossenen Oberfläche[12] oder einer uniformen *Edge*-Länge sichergestellt werden.

Die *Subdivision* Algorithmen zur automatischen Verfeinerung von Oberflächen (siehe Abschnitt 2.1.2) lassen sich hervorragend in die Modellierungs-Metapher mit aufnehmen: Entweder wird zwischen der Ansicht des groben Käfigs und dem weichen Resultat hin und her geschaltet, oder der ursprüngliche Käfig wird durch Linien direkt auf dem Ergebnis der *Subdivision* angezeigt (vgl. Abbildung 2.3).

2.2.3. Flexibel

Statt wohlbestimmte Kontrollpunkte für die Manipulation einer Oberfläche zu verwenden, kann der Benutzer beim sogenannten *Variational Modelling* [WW92] beliebige »Angriffspunkte« (*Constraints*) auf der Geometrie setzen und somit die Form beeinflussen. Dazu wird die Koppelung von Möglichkeiten zum Verformen

[11] NURBS bieten hier den allgemeinsten Standard, wenn man von Subdivision Surfaces (siehe Abschnitt 2.1.2 auf Seite 7) als Lösung absieht.

[12] Dies ist unter anderem für die Repräsentation von Gegenständen (siehe Abschnitt 2.1.5) essentiell.

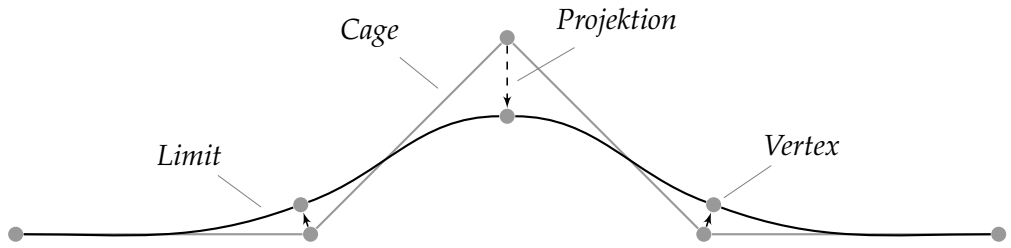


Abbildung 2.3.: Zweidimensionales Beispiel zur Verdeutlichung der Projektion des *Cage*s auf die durch *Subdivision* erzeugte *Limit*-Oberfläche.

der Objekte mit der zugrunde liegenden Repräsentation aufgelöst. Für Freiform-Flächen würde dies unter anderem bedeuten, dass der Anwender nicht mehr mit Kontrollpunkten arbeitet, sondern die Oberfläche überall »anfassen« kann. Der Benutzer arbeitet dann mit einer sogenannten »Fassade«, die größtmögliche Flexibilität bei der Gestaltung der virtuellen Objekte ermöglicht.

Es ist jedoch nach jeder Änderung eine globale Optimierung der *Constraint*-Gleichungen nötig, was je nach Menge der Angriffspunkte zu einem hohen Berechnungsaufwand führt. Hinzu kommt, dass diese enorme Flexibilität nur schwer mit der geforderten Benutzerfreundlichkeit in Einklang zu bringen ist [13]. Die vielen Freiheitsgrade machen es gerade dem unerfahrenen Anwender schwer, die richtigen Werkzeuge zu finden, um gewünschte Änderungen durchzuführen. Es muss daher ein gangbarer Weg zwischen Abstraktion und Beliebigkeit gesucht werden.

2.2.4. Physikalisch

Die physikalische Simulation[14] ermöglicht besonders intuitive Modellierungsmetaphern. Zum Beispiel lassen sich auf fast allen Arten von Geometrie sehr nützliche lokale und globale Deformationen durchführen [SP86]. Besonders interessant ist der Ansatz des *Virtual Sculpting* [GH91, Coq90]. Hier werden dem Benutzer Werkzeuge präsentiert, die auch bei der klassischen Arbeit mit Ton oder Knete Verwendung finden. Das Wirken auf die virtuelle Geometrie wird dabei möglichst akkurat dem Verhalten in der Realität nachempfunden. Mit den so genannten *Sweepers* [AWC04] ist sogar die Erhaltung des Volumens bei Deformationen in Echtzeit möglich. Die in Abschnitt 2.1.4 erwähnten Punktwolken eignen sich vorrangig für diese Art der intuitiven Modellierung [PKKG03].

Durch *Virtual Sculpting* lassen sich sehr intuitiv und schnell detailreiche Gegenstände formen. Allerdings nur, wenn auch das Eingabegerät entsprechend

[13] Dies ist besonders gut in dem in Abschnitt 2.5 auf Seite 19 beschriebene Softwareprodukt *Modo* zu beobachten.

[14] Der Begriff ist hier sehr weit gefasst. Es sind Techniken gemeint, die sich in gewisser weise »physikalisch plausibel« verhalten.

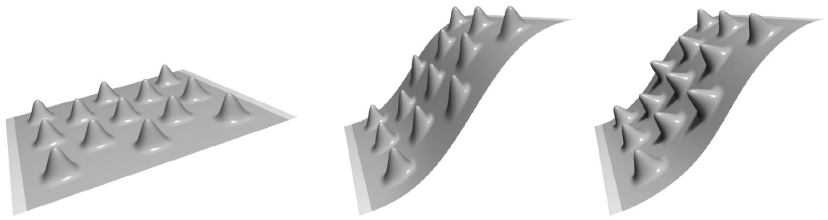


Abbildung 2.4.: Wird eine Oberfläche mit feinen Details (links) verformt, so entstehen ungewollte Verzerrungen (mitte). Durch eine Detailhierarchie lässt sich dieses Problem umgehen (rechts). Bild: [BPR+06].

realistisch bedient werden kann. Die Benutzerschnittstelle sollte daher unbedingt über ein haptisches Feedback [Bur96] verfügen. Aus diesem Grund kommt eine konsequente Umsetzung dieses Modellierungskonzeptes für diese Diplomarbeit nicht in Frage, denn in Abschnitt 1.3 wurde gefordert, dass ein Standard-PC ausreichen soll. Dieser verfügt natürlich nicht über ein solch spezielles Interaktionsgerät.

2.2.5. Hierarchisch

Um die Übersicht und Handhabung bei der Bearbeitung von Geometrie zu verbessern ist es ratsam, die Modelle in verschiedenen Detailstufen zugänglich zu machen. Dieses Vorgehen darf nicht mit *Level of Detail*-Techniken [LRC+02] verwechselt werden, die allein das Ziel haben, die Komplexität der darzustellenden Geometrie zu reduzieren. Vielmehr geht es hierbei um die Erweiterung der zugrunde liegenden Repräsentation durch eine Detail-Hierarchie [GSS99, SNBW03, PKG]. So kann die Benutzerfreundlichkeit stark verbessert werden. So ist es dann einerseits möglich, grobe Strukturen zu verändern, ohne die Details zu zerstören, und andererseits Details auszuarbeiten, ohne das Gesamtaussehen zu beeinflussen (siehe Abbildung 2.4). Besonders beim *Cage Modelling* (vgl. Abschnitt 2.2.2) ist dies eine gängige und relativ einfache Erweiterung, da die als Basis dienende Repräsentation durch Polygone in Kombination mit *Subdivision Surfaces* eine effiziente Implementierung ermöglichen.

2.2.6. Logisch

Das Prinzip der logischen Mengenoperationen zur Konstruktion von komplexen virtuellen Gegenständen ist eines der ältesten Konzepte in der Computergrafik. Zu Beginn wurden ausschließlich geometrische Primitive wie Würfel, Kugeln und Zylinder miteinander kombiniert, um die Form eines Objektes zu beschreiben. Diese Technik erhielt den Namen *Constructive Solid Geometry* (CSG). Heutzutage werden logische Operationen auf jeglicher Art von virtueller Geometrie als CSG bezeichnet.

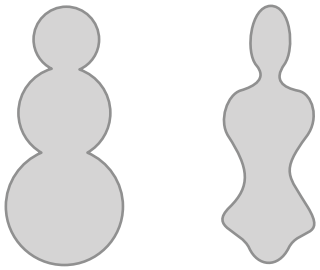


Abbildung 2.5.: Mit geometrischen Grundprimitiven lässt sich die linke Form sehr einfach durch das logische verunden von Kreisen erstellen; die rechte From hingegen ist äußerst schwer mit diesem Modellierungskonzept zu erreichen.

Wenn die Software es ermöglicht, mehrere Objekte gleichzeitig anzuzeigen und im virtuellen Raum zu positionieren, ist die Handhabung sehr einfach – natürlich nur unter der Voraussetzung, dass der Benutzer mit Mengenoperationen wie Schnitt, Vereinigung, Differenz, etc. etwas anzufangen weiß. Einige Formen, die durch solche logischen Operationen erzeugt werden können, sind äußerst schwer mit anderen Modellierungsmetaphern umzusetzen. Auf der anderen Seite lassen sich eine Vielzahl von Formen nur sehr aufwändig konstruieren (siehe Abbildung 2.5). CSG wird deshalb meist nur als Ergänzung zu anderen Modellierungstechniken verwendet.

2.2.7. Sonstiges

Sketch Based Modelling (kurz SBM) ist ein relativ junges, aber vielversprechendes Forschungsthema in der Computergrafik. Ziel ist die Produktion von 3D-Modellen mit einem Stift als Eingabegerät. Der Benutzer ist so in der Lage, mit zweidimensionalen Skizzen ein virtuelles Objekt zu beschreiben [OCJF01, IMT99, FMK+03, CHZ00]. Andere Ansätze wie z. B. *Variational Surfaces* (siehe 2.2.3 auf Seite 11) werden mit SBM kombiniert und erzielen so hervorragende Ergebnisse [KHR, NSACO05]. Da diese Arbeit jedoch das Ziel hat, Standard-Eingabegeräte zu verwenden (siehe 1.3 auf Seite 2), ist das Modellieren mit Hilfe von Skizzen nicht möglich.

In [ZPKG02] wird eine Anwendung mit dem Namen *PointShop 3D* beschrieben, mit der Punktwolken (siehe Abschnitt 2.1.4 auf Seite 9) analog zu 2D Rastergrafiken bearbeitet werden können. In dieser Software sind viele verschiedene Werkzeuge zur Modellierung wie beispielsweise Glätten, Entrauschen usw. verfügbar. Das macht eine längere Einarbeitungszeit zum Erlernen der Funktionen notwendig. Aufgrund der Forderung nach einer schnellen Erlernbarkeit der Funktionalität auch für unerfahrene Anwender, scheint ein solches Verfahren für diese Arbeit nicht geeignet.

2.3. Datenorganisation

Für ein einfaches kooperatives Modellierungswerkzeug ist eine Datenorganisation unerlässlich, mit der leicht auf die virtuellen Ggenstände zugegriffen werden kann und Veränderungen vorgenommen werden können. Das Organisieren und Strukturieren von Informationen wird seit Jahrhunderten erforscht [Tay99]; für diese Arbeit reicht es jedoch aus, die gängigsten Strukturierungsverfahren aus Sicht der Informatik zu betrachten und jeweils mit Blick auf das gesetzte Ziel (siehe 1.2) eine Bewertung vorzunehmen.

2.3.1. Liste

Die einfachste Form der Strukturierung von Objekten ist die ungeordnete Liste. Vor allem bei einer geringen Anzahl von Elementen ist dies oft ausreichend. Wie in [Tay99] beschrieben wurde, ist aber schon bei einer sehr kleinen Menge von Elementen eine Ordnung empfehlenswert. Daher sollte eine Liste von Gegenständen nach definierten und ersichtlichen Kriterien sortiert werden. Ein solches Kriterium könnte z. B. der letzte Änderungszeitpunkt eines Modells oder seine Popularität sein.

2.3.2. Baum

Bei einer steigenden Anzahl von Elementen werden Listen schnell unübersichtlich. Zur Verbesserung der Übersichtlichkeit bietet sich ein Baum als Metapher an. Beginnend mit einer Wurzel verzweigt sich die Struktur immer wieder, bis sich schließlich in den Blättern die eigentlichen Elemente finden. Die Verzweigungen fassen dadurch semantisch verwandte Elemente in Gruppen zusammen. Der Desktop-Metapher gängiger Betriebssysteme liegt eine solche Verzeichnisbaum-Struktur zugrunde. Deshalb ist das Prinzip jedem Computernutzer bekannt. In der Graphentheorie werden die Verzweigungen Knoten genannt. Knoten sind die Eltern (engl. *Parents*) ihrer nachfolgenden Elemente. Diese werden als Kinder (engl. *Childs*) bezeichnet.

Die Problematik dieses Ordnungsprinzips in Bezug auf virtuelle Gegenstände besteht in der Definition und Einordnung in Gruppen. Für diese Arbeit erscheint die Klassifikation nach Eigenschaften, wie sie in Abbildung 2.6 angedeutet wird, am sinnvollsten. Damit eine solche Ordnung verstanden und erweitert werden kann, benötigt der Benutzer ein intuitives Werkzeug zur Ansicht und Manipulation. Ein weiteres Problem ist die semantische Inflexibilität von Baumstrukturen, da sich leicht Problemfälle finden lassen, bei denen ein Element in zwei oder mehr Gruppen eingeordnet werden muss. Zum Beispiel könnte ein Auto sowohl als Fahrzeug, als auch als Spielzeug klassifiziert werden.

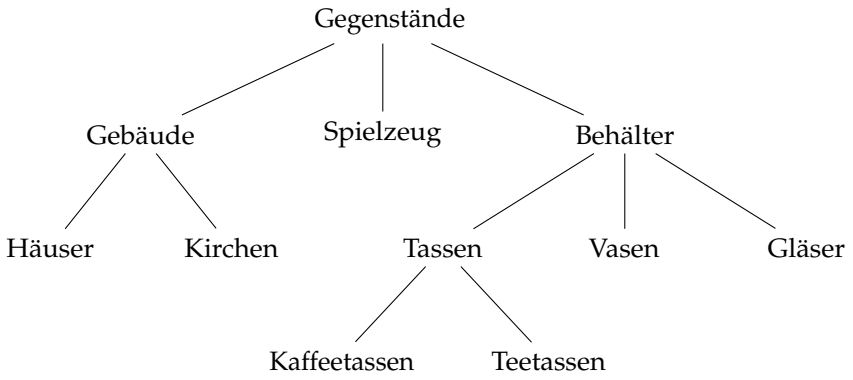


Abbildung 2.6.: Beispielhafte Ordnung in einer Baumstruktur nach Eigenschaften der virtuellen Gegenstände.

2.3.3. Katalog

In einem Katalog bekommt jedes Element eine beliebige Anzahl von Annotationen (engl. *Tags*), mit denen es assoziiert werden soll. Der Benutzer hat dann die Möglichkeit, sich alle Elemente, die ein bestimmtes *Tag* aufweisen, in einer Liste anzeigen zu lassen. Dabei sollte darauf geachtet werden, dass die Listen nicht zu lang werden. Dazu können die *Tags* in einer Baumstruktur hierarchisiert werden. Die Annotationen müssen leider meist von Hand vorgenommen werden, da sich nur dem Benutzer die volle Semantik der Elemente erschließt.

2.3.4. Suchmaschine

Bei sehr großen Mengen von Elementen ist eine Filterung durch sogenannte Suchmaschinen empfehlenswert. Das Ergebnis wird in einer Liste sortiert angezeigt. Der Unterschied zu einem Katalog besteht in der meist vollautomatischen Klassifizierung der Elemente. Demnach ist mit dem Begriff »Suchmaschine« nicht nur der Zugriff auf Teilmengen gemeint, sondern vor allem ihre maschinelle Erschließung und Ordnung. In der Computergrafik wird daher versucht, Techniken zu entwickeln, mit deren Hilfe dies für 3D-Objekte erfolgen kann [FMK+03].

2.3.5. Virtuelle Welt

Im Kontext virtueller Geometrie kann eine Strukturierung auch räumlich und zeitlich erfolgen. Dabei werden alle Gegenstände in einer virtuellen Welt arrangiert und platziert. Der Benutzer ist so in der Lage, seine kognitiven Fähigkeiten auf die dargestellte Umgebung anzuwenden. Durch die Berücksichtigung der Gestaltgesetze [Wer25] und gute Navigationsunterstützung werden Elemente leicht gefunden. Vor allem, wenn Objekte nicht nur für sich allein stehen sollen, sondern

mit anderen Dingen in einer virtuellen Welt interagieren oder sich verbinden, ist diese Art der Strukturierung sehr sinnvoll. Ein gutes Beispiel hierfür sind geographische Darstellungen, bei denen Dörfer, Häuser, Seen usw. auf einer Landkarte angeordnet werden.

2.4. Kooperationskonzepte

In diesem Abschnitt werden verschiedene Techniken vorgestellt, mit denen Menschen über das Werkzeug »Computer« miteinander kooperieren können. Unter »Kooperation« (engl. *cooperation*) wird hier der Verbund mehrerer Einzelhandlungen zweier oder mehrerer Personen zum Erreichen eines gemeinsamen Ziels verstanden. Diese Definition entspricht den Erläuterungen der Fachgruppe CSCW der Gesellschaft für Informatik und grenzt sich so von dem semantisch verwandten Begriff »Kollaboration« (engl. *collaboration*) ab, bei dem zwar gemeinsame Interessen geteilt werden, aber jeder sein eigenes Ziel verfolgt. Bei einer Kollaboration wird die Gruppe der Teilnehmer auch als *Community* bezeichnet.

2.4.1. Manueller Datenaustausch

Personen mit geringen Computerkenntnissen greifen meist auf einen manuellen Datenaustausch zurück, wenn sie mit anderen Menschen kooperieren wollen. Die entsprechenden Daten werden auf tragbaren Medien wie USB-Sticks oder DVD gespeichert oder per E-Mail an den Empfänger gesendet. Versionierung, Synchronsiation und Persistenz sind den handelnden Personen überlassen und werden daher oft vernachlässigt, vergessen oder sogar absichtlich nicht durchgeführt. Dadurch entsteht ein hoher Zeitaufwand für die Verwaltung und den Abgleich der Daten. Dieser steigt extrem schnell mit der zu administrierenden Datenmenge und der Anzahl der Benutzer.

2.4.2. Netzwerk Dateisystem

In lokalen Netzwerken (LAN) kann mit einfachen Mitteln auf einem Server ein Ordner als Dateisystem auf dem Netzwerk freigegeben werden. Dadurch erhalten alle Benutzer Zugriff auf gemeinsamen Speicherplatz, der zu Kooperationszwecken genutzt werden kann. Persistenz kann leicht durch ein automatisches Datensicherungs-System (engl. *backup*) garantiert werden. Das Team muss jedoch selbst die Regeln festlegen, wie Dateien benannt, verändert und strukturiert werden sollen. Alle Teilnehmer brauchen bei größeren Projekten viel Erfahrung und Disziplin, um nicht die Übersicht zu verlieren. Vor allem Versionierung und Synchronisation müssen wie beim manuellen Datenaustausch »von Hand« durchgeführt werden.

2.4.3. Repository

Im professionellen Umfeld wird oft ein sogenanntes *Repository* zur Verbesserung der Kooperation eingesetzt. Dazu wird auf einem Server eine spezielle Software installiert, die Daten verwalten und versionieren kann. Jeder Benutzer greift mit einem speziellen *Client*-Programm auf diese Datenbank zu und kann sich benötigte Objekte zur Bearbeitung auf einen lokalen Speicher kopieren. Nach Bearbeitung der Daten können diese wieder in das *Repository* zurückkopiert werden, wobei automatisch eine Synchronisation der Daten stattfindet. So sind auch mehrere Teilnehmer in der Lage, gemeinsam an einem Projekt zu arbeiten, ohne unnötig viel Zeit mit der Datenverwaltung zu verbringen. In der Softwareentwicklung und anderen Unternehmungen, bei denen hauptsächlich mit Textdateien gearbeitet wird, ist das Produkt *Subversion*[15] der Firma *Tigris* sehr beliebt. Im Medienbereich, wo viel mit Binärdateien wie Bildern oder 3D-Modellen gearbeitet wird, spricht man von *Asset Management* Software. Solche Programme sind in der Lage, die speziellen Medientypen sinnvoll zu verwalten und teilweise sogar zu synchronisieren.

2.4.4. Simultan

Seit einiger Zeit werden unter dem Stichwort CSCW (*Computer-Supported Cooperative Work*) unter anderem Softwareprodukte entwickelt, die das simultane Arbeiten mit ein und derselben Dateninstanz ermöglichen [RSVW94]. Die Teammitglieder besitzen dabei keine lokale Kopie mit möglicherweise inkonsistentem Zustand, sondern arbeiten jederzeit mit der persistenten Version des Servers. Durch diese Art der Zusammenarbeit kann auch jeder Teilnehmer feststellen, woran die anderen Mitglieder momentan arbeiten. Auch lassen sich so Versionskonflikte auf einer sehr feinen Granularitätsstufe beheben; sie kommen praktisch nicht mehr vor (vgl. Benutzungstest in Abschnitt 6.2 auf Seite 62); dadurch ergibt sich für alle Anwender eine deutlich angenehmere Arbeitsweise [NWWM92].

In der Computergrafik gibt es einige interessante Arbeiten, die sich mit kooperativen bzw. kollaborativen virtuellen Umgebungen auseinander setzen [BM00, Bro95]. Im Bezug auf die Erstellung virtueller Objekte ist das speziell für den kooperativen Einsatz geschaffene Netzwerk-Protokoll mit dem Namen *Verse* [Bri] besonders bemerkenswert. Auf Grundlage dieser Technik wurde im Februar 2004 das *Uni-Verse* Projekt[16] gegründet, dessen Ziel die Entwicklung einer interaktiven *Multi-User* Plattform auf IP-Basis für 3D-Grafik und Audio ist. Im Gegensatz zu der in dieser Arbeit gestellten Problematik geht es dabei jedoch um die Vernetzung bestehender Softwareprogramme beliebiger Komplexität.

[15] Januar 2007: `http://subversion.tigris.org`
[16] Januar 2007: `http://www.uni-verse.org`

2.5. Existierende Software-Produkte

Es existiert eine Vielzahl an Software-Produkten, mit denen es möglich ist, virtuelle Objekte zu konstruieren. Neben den professionellen Anwendungen hat sich mit der Zeit auch ein Markt für einfache Werkzeuge entwickelt. In den nächsten beiden Abschnitten werden daher einige repräsentative Programme aus beiden Bereichen kurz vorgestellt und deren Unterschiede zu der in dieser Arbeit behandelten Aufgabe herausgearbeitet. Screenshots dieser Anwendungen befinden sich in Anhang B.

2.5.1. Professionelle Anwendungen

Sehr bekannt und in der Film-Industrie weit verbreitet ist die Allround-Software *Autodesk Maya*[17]. Hier wird die Geometrie vorwiegend durch NURBS oder Polygon-Netze repräsentiert und mit den entsprechenden Konstruktionstechniken bearbeitet (vgl. Abschnitt 2.2). Diese äußerst komplexe Anwendung ist ein gutes Beispiel für eine große Funktionsvielfalt: *Maya* ermöglicht nicht nur die Modellierung von virtuellen Objekten und Szenen, sondern auch deren Animation, Texturierung und das Erzeugen von Spezial-Effekten. Selbst mehrere *Rendering*-Lösungen sind eingebunden. Das Besondere an *Maya* ist der modulare Aufbau. Alle Elemente der Software können beliebig miteinander verknüpft und sogar mit selbst entwickelten Komponenten erweitert, ersetzt oder verändert werden. Alle Eigenschaften sind programmierbar, konfigurierbar und über viele verschiedene Zugänge erreichbar. Diese Mächtigkeit bringt eine lange Einarbeitungszeit mit sich. Im professionellen Einsatz ist dies vertretbar, da die Software je nach Problemstellung perfekt angepasst werden kann und die Bedienung ausschließlich von geschultem Personal erfolgt.

Das Produkt *Pixologic ZBrush*[18] setzt ganz auf das Konzept des *Mesh Paintings*. Der Benutzer zeichnet dabei Details auf eine bestehende Geometrie. Die Größe und Art der Details – Ausbuchtung, Einkerbung, Muster, usw. – müssen hier vom Anwender eingestellt werden. Dies führt zu einem breiten Spektrum von Auswahlmöglichkeiten, die nur der erfahrene Künstler überblicken und gezielt einsetzen kann.

Seit 2001 wird eine Software unter dem Namen *Modo*[19] von der Firma *Luxology* entwickelt. Aufgrund ihrer verhältnismäßig kurzen Existenz finden in *Modo* viele moderne Ansätze der Computergrafik Anwendung. Die Software basiert auf *Subdivision Surfaces* (vgl. 2.1.2 auf Seite 7) und ist in ihrer Benutzung sehr flexibel (vgl. 2.2.3 auf Seite 11). Das bedeutet, es wird stark von der zugrunde liegenden Repräsentationstechnik abstrahiert, wodurch auf einem höheren Niveau gearbeitet

[17] Januar 2007: `http://www.autodesk.com/maya`
[18] Januar 2007: `http://www.pixologic.com/zbrush`
[19] Januar 2007: `http://www.modo3d.com`

werden kann. Als Beispiel sei hier das *Proxy*-Werkzeug genannt: Auf der Oberfläche kann ein konfigurierbares Kontrollgitter frei positioniert werden. Anschließend lassen sich Verformungen durch die »Angriffspunkte« des Gitters umsetzen. Alle Werkzeuge lassen sich durch die sogenannte *Tool Pipe* nahezu beliebig anpassen. Natürlich wird auch hier eine intensive Einarbeitungszeit vorausgesetzt.

2.5.2. Einfache Anwendungen

Eines der bekanntesten 3D-Modellierungsprogramme mit »einfacher« Benutzungsschnittstelle ist die zu *Google*[20] gehörende Software *SketchUP*[21]. Als Grundprimitive werden Linien eingesetzt, die in den 3D-Raum gezeichnet und dort zu Polygonen zusammengesetzt werden; demnach favorisiert die Anwendung das *Sketch Based Modelling* (siehe Abschnitt 2.2.7). Doch dieses Zeichenkonzept wird zu kompliziert umgesetzt: Der Anwender benötigt zum Modellieren mehr als ein Dutzend verschiedener Werkzeuge, die in ihrer Handhabung ebenfalls nicht auf leichte Erlernbarkeit ausgelegt sind. Keine Funktion ist in weniger als vier Arbeitsschritten abschließbar.

In eingeschränkter Weise ist es mit *SketchUP* möglich, kollaborativ tätig zu werden (vgl. Abschnitt 2.4). Über das sogenannte *3D Warehouse* lässt sich Geometrie auf Datei-Basis hochladen, suchen, gruppieren und herunterladen. Es ist auch möglich, Modelle mit Georeferenz-Daten auszustatten und dann in die eigene *Google-Earth* Visualisierung einzubauen. Jedes Objekt »gehört« aber einer Person; zum Bearbeiten muss es lokal abgespeichert werden und später wieder als neue Version ins *3D Warehouse* hochgeladen werden. Dieser Prozess ist nur von dem Besitzer durchführbar[22] und verhindert so eine unkomplizierte Kooperation mehrerer Anwender.

Bei anderen einfachen Modellierungs-Systemen wie *Milkshape 3D*[23], *Cheetah3D*[24] oder *Wings 3D*[25] ist keinerlei Unterstützung für kooperatives Arbeiten eingebaut. Auch lässt sich insgesamt feststellen, dass bei diesen Anwendungen trotz anfänglich guter Idee ein Konzept nicht konsequent verfolgt wird. Solange sich eine neue Eingabe-Techniken in der Forschung befindet, bleibt der Anwendungsbereich jedoch stark eingeschränkt (z. B. [IMT99, TWB+]). Wie bei *SketchUP* werden bei der Kommerzialisierung aber eine Vielzahl von Erweiterungen eingebaut, welches eine Einarbeitung für unerfahrene Anwender wieder stark verlängert.

Ganz anders verhält es sich bei Computerspielen. Seit einigen Jahren werden immer mehr Spielideen vorgestellt, bei denen die Spieler selbst digitale Inhalte

[20] Januar 2007: `http://www.google.de`
[21] Januar 2007: `http://sketchup.google.com`
[22] Die Google-Hilfe schreibt: »*You can only edit and replace your own models; you can't replace someone else's model.*«
[23] Januar 2007: `http://www.milkshape3d.com`
[24] Januar 2007: `http://http://www.cheetah3d.com/`
[25] Januar 2007: `http://http://www.wings3d.com/`

erstellt müssen[26]. Die dazu notwendigen Modellierungstechniken müssen einfach zu erlernen sein und die Motivation aufrecht erhalten. Sehr lehrreich ist in diesem Zusammenhang das 1993 erschienene Computerspiel *The Incredible Machine*. Hier muss der Spieler aus einem gegebenen Set von Gegenständen eine Maschine bauen, die einen vorgegebenen Zweck erfüllt. Heutzutage kommt das Weitergeben der selbstproduzierten Elemente an andere Spieler hinzu. Zum Beispiel werden laut Herstellerangaben in dem noch nicht veröffentlichten Spiel *Spore*[27] alle von Spielern erstellten Dinge automatisch an alle anderen Spieler weiterverteilt. Angeblich basiert die gesamte Modellierung auf einer intuitiven Steuerung von Parametern, aus denen dann prozedural Geometrie und Bewegungen erstellt werden. Wie mächtig diese Technik ist, lässt sich vermutlich erst nach der Veröffentlichung des Spiels bewerten.

In dem stark an Popularität gewinnenden Internet-Spiel *Second Life*[28] hat die Konstruktion von virtuellen Gegenständen eine ganz besondere Bedeutung: Alle Dinge in der Spielwelt sind von Spielern produziert worden und müssen mit einer virtuellen Währung erworben werden. Das Modellierungs-Konzept basiert auf Polygonen und macht starken Gebrauch von der CSG-Metapher (vgl. 2.2.6). Mehrere Spieler können gemeinsam Gegenstände zusammenbauen und so Geld verdienen. Dies ist aber auch gleichzeitig das Problem im Hinblick auf die »einfache« Modellierung. Bei der Programmierung der Anwendung wurde darauf geachtet, dass es nicht leicht ist, etwas Außergewöhnliches zu produzieren; wenn jeder Spieler mit wenig Aufwand seine eigenen Wünsche umsetzen könnte, würde der Reiz am Handel verloren gehen und das virtuelle Wirtschaftssystem zusammenbrechen.

[26] Es ist zu vermuten, dass damit auch die Produktionskosten gedrückt werden sollen. Bei aktuellen Spieleproduktionen ist die Anzahl professioneller 3D-Künster um vieles höher als die der Software-Entwickler.

[27] Januar 2007: `http://www.spore.com`

[28] Januar 2007: `http://www.second-life.com`

Kapitel 3.

Bestimmung der Lösung

> ...when you have excluded the impossible, whatever remains, however improbable, must be the truth.
>
> *Sherlock Holmes*

Ausgehend von dem Anwendungsszenario in Abschnitt 1.3 auf Seite 2 und den im vorherigen Kapitel betrachteten Möglichkeiten, wird im folgenden die zu entwickelnde Software im Hinblick auf ihre qualitativen und quantitativen Eigenschaften definiert. Dazu sind zunächst in einer Analyse die betrachteten Möglichkeiten kurz zusammengefasst; mit den daraus resultierenden Überlegungen lässt sich auf die zu erarbeitende Lösung schließen.

Diese Folgerungen sind anschließend ausführlich im Abschnitt 3.2 über die Anforderungen an das Produkt zusammengefaßt. Sie beschreiben die Lösung aus Anwendersicht mit Blick auf ihren Zweck und die damit verbundene Nützlichkeit. Dadurch wird das zu entwickelnde System beschrieben, ohne die technische Umsetzung vorweg zu nehmen. Das Kapitel schließt mit einer Abgrenzung zu verwandten Themen und zeigt weiterführende Problemstellungen auf, die im Umfang dieser Arbeit nicht behandelt werden können.

3.1. Analyse und Auswahl

3.1.1. Geometrie

Wie bereits in den Abschnitten auf Seiten 6–9 festgestellt wurde, bieten vor allem polygonale Netze eine sinnvolle Basis für ein einfaches Modellierungswerkzeug. Freiform-Geometrie ist prinzipbedingt sehr schwer zu bedienen und topologisch höchst unflexibel. Implizite Oberflächen sind zwar topologisch flexibel, jedoch ist es mit ihnen sehr schwierig, technische Formen mit harten Kanten nachzubilden. Daher kommen auch sie als grundlegendes Repräsentationsschema für die prototypische Umsetzung innerhalb dieser Arbeit nicht in Frage. Die Modellierung mit

Punktwolken bietet vor allem bei physikalischen Modellen verschiedene Vorteile (siehe Abschnitt 2.2.4 auf Seite 12), bringt jedoch einen hohen Berechnungsaufwand mit sich (vgl. 2.1.4 auf Seite 9). Für diese Arbeit ist es daher am besten, polygonale Netze für die Repräsentation der Geometrie zu verwenden.

3.1.2. Konstruktionstechnik

Die einfachste und intuitivste Metapher zur Konstruktion von polygonalen Netzen ist das *Cage-* oder *Box Modelling*, welches in Abschnitt 2.2.2 auf Seite 11 beschrieben wurde. Zur Umsetzung eines Modellierungs-Werkzeuges, bei dem die Einfachheit der Bedienung im Vordergrund stehen soll, bietet sich diese Technik geradezu an. Ein hoch flexibler Ansatz wie das *Variational Modelling* aus Abschnitt 2.2.3 würde den unerfahrenen Anwender durch eine Vielzahl von Freiheitsgraden überfordern und kann daher nur in Maßen eingesetzt werden. Auch eine pure physikalische Modellierungsmetapher ist nicht umsetzbar, da hierzu ein haptisches Feedback (siehe Abschnitt 2.2.4) nötig sein würde. Dennoch könnte ein geringes Maß an physikalischer Plausibilität bei der Mausinteraktion die Modellierung von Gegenständen erleichtern. Die Möglichkeit zur Bearbeitung der Geometrie auf verschiedenen Detailstufen, wie in Abschnitt 2.2.5 auf Seite 13 erläutert, ist eine zwingend erforderliche Eigenschaft. Nur so lassen sich grobe Formen unabhängig von feinen Strukturen manipulieren.

Logische Mengenoperationen auf virtueller Geometrie (CSG) benötigen eine grafische Benutzungsoberfläche, mit der die Gegenstände arrangiert und verknüpft werden können (siehe Abschnitt 2.2.6). Aus Gründen der Funktionsminimierung und Übersichtlichkeit ist es daher angemessen, diese Art der Modellierung von der des *Cage Modellings* zu trennen. Aufgrund ihres beschränkten Anwendungsfeldes ist CSG außerdem für die Problemstellung dieser Arbeit nicht zwingend erforderlich. Wie schon in Abschnitt 2.2.7 auf Seite 14 erläutert, wird für das *Sketch Based Modelling* ein digitaler Stift als Eingabegerät benötigt, welcher jedoch nicht zur Standardausstattung heutiger Computer gehört. Damit ist *SBM* für die hier gesuchte Lösung ungeeignet.

3.1.3. Datenorganisation

Für die Problemstellung dieser Arbeit ist eine intuitive und einfache Lösung für den strukturierten Zugriff auf die virtuellen Gegenstände sehr wichtig. Da es ausschließlich um die Modellierung von unabhängigen Einzelgegenständen geht, scheint die Anordnung in einer virtuellen Welt, wie in Abschnitt 2.3.5 auf Seite 16 beschrieben, nicht angemessen. Besser ist die Präsentation der Gegenstände in einem Katalog, wie er in Abschnitt 2.3.3 skizziert wurde, da so auch größere Mengen von Objekten leicht zu durchsuchen sind. Zusätzliche Annotationen können leicht vom Benutzer selbst hinzugefügt werden. Deshalb ist diese Lösung

einem automatischen Klassifizierungssystem (vgl. Abschnitt 2.3.4) mit ungenauen Ergebnissen vorzuziehen.

3.1.4. Kooperation

Die Kooperation zwischen mehreren Benutzern sollte transparent, vollautomatisch und ohne technisches Vorwissen funktionieren, denn eine zeitintensive Einarbeitung schreckt viele potenzielle Anwender ab und die Zeit fehlt später während der kreativen Arbeit. Daher ist es unbedingt erforderlich, nicht auf das Konzept von Dateien zu setzen, wie es bei Repositories (Abschnitt 2.4.3) und primitiver Kooperation (Abschnitte 2.4.1 und 2.4.2) der Fall ist. Dateien, die Geometrie beschreiben, sind nicht leicht zu versionieren und äußerst schwer zu synchronisieren. Wie in Abschnitt 2.4.4 auf Seite 18 erläutert wurde, ist das direkte und simultane Arbeiten auf einer für alle Anwender identischen Instanz ein gangbarer Lösungsansatz, der jedoch von existierenden Softwareprodukten (Abschnitt 2.5 auf Seite 19) standardmäßig nicht unterstützt wird. Die Benutzer sollten in der hier angestrebten Lösung nicht mehr mit Dateien »hantieren«, sondern ganz unproblematisch mit den virtuellen Objekten auf dem Server arbeiten, ohne sich um die Datenhaltung kümmern zu müssen. Synchronsiation und Versionierung erfolgen dann implizit bei der gleichzeitigen kooperativen Arbeit an den Gegenständen. Dazu ist es notwendig, während der Arbeit eine Netzwerkverbindung aufrecht zu erhalten. Da nicht *offline* gearbeitet werden kann, ist vor allem ein Serverausfall höchst problematisch. Alle Tätigkeiten kommen in diesem Fall zum Erliegen. Dies muss durch Backup-Systeme vermieden werden.

3.2. Anforderungen

Eine Anforderung ist nach Balzert [Bal98] eine »Aussage über eine zu erfüllende qualitative und/oder quantitative Eigenschaft eines Produkts ... um ein System für den Entwickler zu definieren«. Im folgenden Abschnitt werden daher alle nötigen Eigenschaften des zu entwickelnden Systems aufgelistet und ihre Notwendigkeit begründet. Es wird wegen der besseren Übersicht eine Gruppierung der Anforderungen nach direkter Benutzerinteraktion und automatischem Verhalten vorgenommen. Letzteres ist entweder aus technischer Sicht notwendig oder vereinfacht die Interaktion mit der Software.

3.2.1. Benutzerinteraktion

Minimale Komplexität

Der Benutzer sollte einerseits eine möglichst geringe Anzahl von Handlungsschritten durchführen müssen, um ein gesetztes Ziel zu erreichen. Andererseits sollte er aber auch möglichst wenige unterschiedliche Handlungsweisen kennen

müssen, um dies zu erreichen. Im Zweifel muss eher zu Gunsten von wenigen Handlungsweisen entschieden und eine längere Arbeitszeit in Kauf genommen werden. Nur so wird eine schnelle Einarbeitung in die Software gewährleistet (siehe Abschnitt 6.1 auf Seite 59). Dies ist für das gestellte Problem essentiell.

Interaktive Bedienbarkeit

Die Software muss sich interaktiv bedienen lassen, denn nur so ist ein angenehmes Arbeiten möglich. Dadurch ist der Detailgrad und die Größe der virtuellen Gegenstände beschränkt. Vor allem die weiter unten formulierte Anforderung an eine verständliche Darstellung ist sehr rechenaufwändig, wie sich in der Umsetzung gezeigt hat. Ziel muss es daher sein, die in dem Szenario auf Seite 2 geforderte Qualität bei flüssiger Interaktion sicher zu stellen.

Direkte Maniputlation

Die Benutzungsschnittstelle sollte sich soviel wie möglich durch »direkte Manipulation« steuern lassen, da diese Art der Interaktion eine besonders schnelle Einarbeitung verspricht und besser von der zugrunde liegenden Technologie abstrahiert [SP05].

Freie Verformung

Das Verformen eines virtuellen Gegenstandes sollte viele Freiheitsgrade besitzen, damit der Anwender möglichst intuitiv modellieren kann. Trotzdem muss die Bedienbarkeit so einfach wie möglich bleiben und es darf keinesfalls zu unbehebbaren Fehlern in der Geometrie kommen. Deshalb ist an geeigneten Stellen die Freiheit sinnvoll zu begrenzen.

Technische Abstraktion

Der Anwender braucht keinerlei technisches Wissen über Computergrafik und kein Spezialwissen aus den Fachrichtungen Mathematik, Informatik oder den Ingenieurwissenschaften. Nur so kann die Einarbeitungszeit kurz und die Motivation bei themenfremden Anwendern hoch gehalten werden.

Verständliche Darstellung

Der Benutzer muss erkennen können, welche Form ein Gegenstand besitzt. Daher ist es notwendig, darauf zu achten, dass evtl. nötige Markierungen und Benutzungshilfen möglichst dezent eingebaut werden. Eine Simulation der Selbstverschattung (engl. *Ambient Occlusion*) trägt enorm zum Verständnis von Einbuchtungen und Wölbungen bei (siehe Abschnitt 4.3.3) und sollte daher unbedingt verwendet werden.

Kausale Automatik

Automatismen müssen sich dem Benutzer ankündigen und nachvollziebar bleiben. Das bedeutet nicht, dass die Gründe für die Automatik verstanden werden sollen, sondern lediglich dass der Anwender einen kausalen Zusammenhang zwischen seinem Handeln und dem resultierenden Ergebnis erkennen und vorhersagen kann – unabhängig von zwischengeschalteten Automatismen.

Kontextsensitive Funktionalität

Dem Anwender werden immer nur die in einem Kontext sinnvollen Handlungsfreiräume aufgezeigt. Durch diese klare Trennung der verschiedenen Anwendungsmöglichkeiten wird eine Funktionsüberflutung des Benutzers bei der Durchführung eines Arbeitsvorganges vermieden. Der Übergang von einem Kontext zum anderen sollte dabei möglichst angenehm, aber dennoch für den Benutzer wahrnehmbar erfolgen.

Mehrbenutzer-Kenntnis

Es muss leicht erkennbar sein, welche anderen Personen zur gleichen Zeit an einem Gegenstand arbeiten. Dadurch wird die Kommunikation im Team einfacher und der »Spaßfaktor« bei der gemeinsamen Arbeit erhöht.

Beliebige Topologie

Es muss möglich sein, Gegenstände mit beliebiger, zwei-mannigfaltiger Topologie erstellen zu können. Daraus folgt, dass der Anwender den *Genus*[1] (dt. Geschlecht) des Gegenstandes ändern kann, denn viele Dinge in der Realität weisen Löcher und Henkel auf.

Gegenstand finden

Die Liste der Gegenstände muss durchsuchbar (siehe 2.3.3 auf Seite 16) sein, damit die Teammitglieder schnell einzelne Gegenstände in der Sammlung auffinden können.

[1] Das topologische Geschlecht bezeichnet die Anzahl der Schnitte durch die Oberfläche, die gemacht werden können, ohne das die Oberfläche auseinader fällt[BSG+03]. Bildlich gesprochen ist das Geschlecht die Anzahl der »Henkel«.

3.2.2. Interaktionsunterstützende Automatismen

Kamerapositionierung

Die Ansicht auf den virtuellen Gegenstand erfolgt durch eine virtuelle Kamera, die immer optimal für die aktuelle Handlungsfolge des Benutzers ausgerichtet sein sollte. Wie in Kapitel 6 beschrieben, ist die freie Navigation im virtuellen Raum mit herkömmlichen Eingabegeräten nicht ohne weiteres verständlich und muss daher so weit wie möglich vom System übernommen werden.

Persistenz

Die Gegenstände sind jederzeit in einem persistenten Zustand. Sowohl an ihrem physischen Ort, als auch bei den Benutzern, die damit arbeiten. Dadurch wird sichergestellt, dass sich zu jedem Zeitpunkt nur eine einzige Instanz des Gegenstandes im gesamten Team befindet. So werden Synchronisationsprobleme vermieden und den handelnden Personen die Kommunikation erleichtert.

Neuer Gegenstand

Benutzer können zu jeder Zeit neue Gegenstände erstellen. Diese haben zunächst eine vordefinierte Ursprungsform, die bereits die Eigenschaften eines Festkörpers u. ä. erfüllt. So kann jeder Nutzer direkt mit der Bearbeitung beginnen.

3.2.3. Technisch notwendige Automatismen

Minimalste Krümmung

Während der Translation von *Vertices* unterstützt die Software durch leichte Positionsanpassungen in Richtung der minimalsten Krümmung der Oberfläche ein gutes polygonales Netz, wie in Abschnitt 2.1.2 auf Seite 7 erläutert. Dies ist für unerfahrene Benutzer sehr wichtig, da sie kein Wissen über eine vorteilhafte *Edge*-Richtung besitzen.

Reguläres polygonales Netz

Das Programm blockt Manipulationen, die zu starke Abweichungen der Kantenlänge vom Durchschnitt zur Folge hätten. Denn auf jeder Detailstufe sollte das Polygonale Netz eine möglichst reguläre Struktur besitzen, um eine spätere Weiterarbeitung zu erleichtern und technisch professionell zu bleiben.

Bearbeitungsverbot

Wie in Abschnitt 2.4.4 auf Seite 18 vorgestellt wurde, werden Bearbeitungskonflikte auf möglichst kleiner Granularitätsstufe behoben, um ein flüssiges Arbeiten

sicherzustellen. Es hat sich in den Benutzungstests (6.2.2) gezeigt, dass durch ein kurzzeitiges Bearbeitungsverbot (*locking*) einzelner *Vertices* der Arbeitsablauf nur sehr gering behindert wird.

Backup

Alle Daten und Serverfunktionen sollen für die Anwender nutzbar bleiben, wenn die Hardware eines Servers ausfällt. Daraus folgt, dass die Benutzerschnittstelle im laufenden Betrieb den Server wechseln können sollte und sich dabei für den Benutzer keine Unterbrechung der Arbeit ergibt.

Festkörper

Alle Gegenstände, die mit der Software erstellt werden, entsprechen den Anforderungen eines geometrischen Festkörpers, wie in Abschnitt 2.1.5 auf Seite 9 beschrieben. Damit ist sichergestellt, dass die virtuellen Objekte dem allgemeinen Verständnis von Gegenständen entsprechen. Die physikalischen Eigenschaften bleiben in dieser Arbeit jedoch unberücksichtigt, da dies zu weit über die in Abschnitt 1.1 auf Seite 1 formulierte Problemstellung hinaus gehen würde.

3.3. Abgrenzung

In dieser Arbeit soll kein universelles und frei konfigurierbares, sondern in erster Linie ein einfach zu bedienendes Modellierungswerkzeug entwickelt werden. Dabei wird der Fokus auf die Konstruktion von virtuellen Gegenständen gelegt, wie sie ein Künstler aus Ton herstellen könnte. Pflanzen und Kleidungsstücke gehören beispielsweise nicht unbedingt in diese Kategorie.

Des Weiteren werden weder verschiedene Geometrie-Repräsentationen unterstützt noch mehrere Manipulations-Metaphern. Das hier entworfene System dient auch nicht der Konvertierung bestehender Datensätze, verzichtet vollkommen auf den Import bestehender Daten und unterstützt keinen Export. Auch wenn es auf den ersten Blick nötig erscheint die Gegenstände als Datei zugreifbar zu machen, wurde bewusst darauf verzichtet. Um das Konzept zu Wahren, ist nur ein stetiger, persistent synchroner Zugriff durch einen Server möglich. Asynchrone, unversionierte Kopien wären Kontraproduktiv. Es ginge über die Aufgabenstellung dieser Arbeit hinaus, eine Texturierung der Geometrie vorzunehmen oder sie zu animieren. Ein 2D-Rendering der virtuellen Gegenstände wird allein zu Darstellungszwecken innerhalb des Modellierungsprogramms vorgenommen. Es ist weiterhin nicht beabsichtigt, Daten zu produzieren, die physikalische Simulationen oder eine maschinelle Fertigung ermöglichen.

Kapitel 4.

Beschreibung der Software-Architektur

> Everything should be as simple as it is, but not simpler.
>
> *Albert Einstein*

In diesem Kapitel wird die Software-Architektur des Systems beschrieben, mit dessen Hilfe ein einfaches kooperatives Bearbeiten von virtuellen Gegenständen möglich ist. Zu Beginn ist es dabei hilfreich, die gesamte benötigte Funktionalität zu analysieren, welche sich aus den Anforderungen (siehe Abschnitt 3.2) ergibt. Darauf aufbauend kann ein Datenfluss-Modell veranschaulichen, wie *Client* und *Server* miteinander agieren und sich die vorher identifizierten Funktionalitäten aufteilen. Abschließend werden in diesem Kapitel die einzelnen Komponenten der Software-Architektur spezifiziert und deren Beziehungen untereinander erläutert.

4.1. Funktionalität

Im Folgenden werden die einzelnen Funktionen der Software definiert und geordnet. Jede Funktion beschreibt eine klar umrissene Aufgabe innerhalb der Anwendung. Werden sie alle kombiniert, lässt sich die in dem Szenario auf Seite 2 vorgestellte Aufgabe vollständig lösen.

Bei der benötigten Funktionalität lässt sich zwischen der Verwaltung von Gegenständen und der Modellierung unterscheiden. Letztere teilt sich wiederum in Darstellung und Interaktion auf (vgl. Abbildung 4.1 auf der nächsten Seite). Wie schon in Abschnitt 3.2 erläutert wurde, befindet sich die meiste Funktionalität »versteckt« hinter einer sehr geringen Anzahl von Interaktionsmöglichkeiten. Die Komplexität der Anwendung schlägt sich daher nicht in einer Vielzahl von Werkzeugen nieder, sondern verbirgt sich hinter intelligenten Automatismen, die kontextbezogen den Benutzer bei seiner Arbeit unterstützen. Funktionen, die

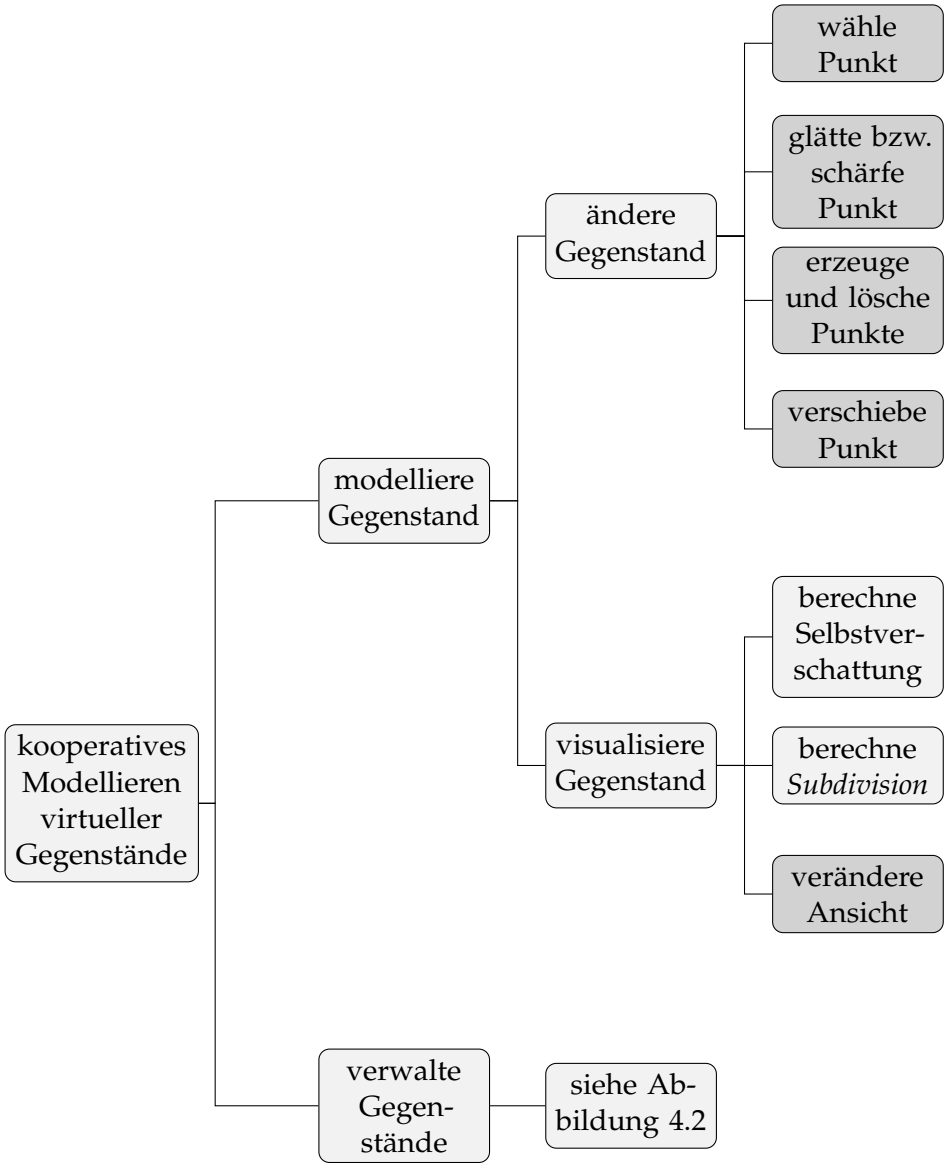


Abbildung 4.1.: Hierarchische Ordnung der benötigten Funktionalität für das kooperative Modellieren virtueller Gegenstände. Dunkler abgesetzte Kästen markieren Funktionen, die zum Teil Benutzerinteraktion erfordern.

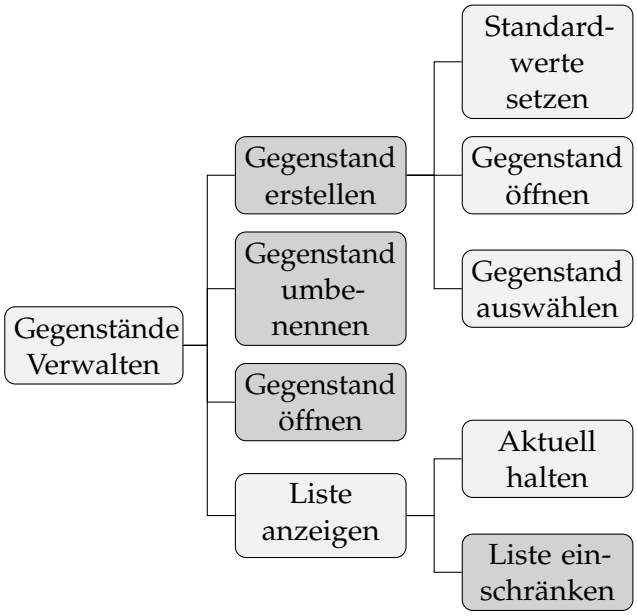


Abbildung 4.2.: Hierarchische Ordnung der benötigten Funktionalität zur kooperativen Verwaltung virtueller Gegenstände. Dunkler abgesetzte Kästen markieren Funktionen, die zum Teil Benutzerinteraktion verlangen.

eine direkte Manipulation erfordern, werden in den Grafiken mit einem Rahmen gekennzeichnet und auf den Seiten 33 bis 36 inklusive den dazugehörenden Automatismen im Hinblick auf ihre Eigenschaften erläutert.

4.1.1. Gegenstände verwalten

Wie Abbildung 4.2 zeigt, ist die Verwaltung von Gegenständen funktionell sehr überschaubar. Dem Anwender wird eine Liste mit Gegenständen angezeigt, in der er suchen und einzelne Elemente umbenennen kann. Wird der selektierte Gegenstand geöffnet, besteht die Möglichkeit, diesen zu bearbeiten (siehe Abschnitt 4.1.2). Des Weiteren kann jederzeit ein neuer Gegenstand erstellt werden.

Umbenennen

In der angezeigten Namensliste können Gegenstände umbenannt werden. Dabei werden Änderungen zuerst an den *Server* gesendet, welcher dann alle *Clients* über den aktualisierten Namen informiert.

Öffnen

Wird ein Gegenstand zur Bearbeitung geöffnet, präsentiert sich dem Anwender ein neues Fenster, in dem der Gegenstand dargestellt (vgl. Abschnitt 4.1.3) und durch Interaktion (vgl. 4.1.2) modifiziert werden kann.

Liste anzeigen

Die Gegenstände werden in einer Liste mit ihren Namen angezeigt. Diese wird ständig mit dem *Server* abgeglichen und zeigt so immer den aktuellen Namen an. Der Benutzer kann die Liste einschränken, indem er eine Zeichenkette angibt, die in dem Namen des Gegenstandes vorkommen muss. Unbenannte Objekte sind davon nicht betroffen und werden immer angezeigt.

Erstellen

Soll ein neuer Gegenstand erstellt werden, legt der *Server* die Standardeinstellungen fest und sendet diese Informationen an alle *Clients*. Die neue Geometrie beschreibt einen einfachen Gegenstand mit dem Namen »Unbenannt«, der ab sofort von allen Anwendern bearbeitet werden kann. In der Liste des Erstellers wird das neue Objekt ausgewählt und der Gegenstand wird automatisch zum Bearbeiten geöffnet.

4.1.2. Gegenstand ändern

Das Ändern der Form eines Gegenstands erfolgt ausschließlich über die *Vertices* des umschließenden Käfigs (siehe Abschnitt 2.2.2 auf Seite 11). Sie werden im Folgenden »Punkte« genannt[1]. Da die Positionen dieser Punkte die Form des Gegenstandes bei den gemachten Annahmen[2] vollständig beschreiben, ist so eine beliebige Verformung möglich. Die Punkte können vom Benutzer angewählt werden, indem er sie mit der Maus angeklickt. Erst danach kann durch sie die Form beeinflusst werden. Der Anwender hat dazu zwei Funktionen zur Auswahl: Er kann sie im Raum verschieben, ohne dabei die Anzahl der Punkte zu verändern, oder aber dabei automatisch Punkte hinzufügen bzw. löschen. Diese beiden Aktionen sind aber nur möglich, wenn nicht ein anderer Anwender eine Sekunde zuvor den selben Punkt bereits bearbeitet hat (vgl. Abschnitt 2.4.4).

[1] Die Bezeichnung »Punkt« abstrahiert von der zugrunde liegenden Repräsentationsform und greift damit die Idee der *Variational Surfaces* (vgl. 2.2.3 auf Seite 11) auf.

[2] Die Annahmen wurden in Kapitel 3 beschrieben. Dort ist definiert, dass der virtuelle Gegenstand durch ein polygonales Netz repräsentiert und über seinen *Cage* verformt wird. Durch *Subdivision Surfaces* ist Oberfläche glatt und die Eigenschaften eines Festkörper werden eingehalten.

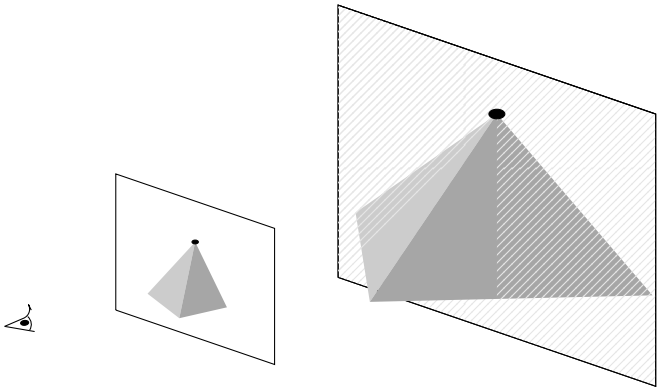


Abbildung 4.3.: Der Betrachter kann von seiner Position im Raum den schwarzen Punkt nur auf einer Ebene verschieben. Dazu wählt er auf dem Bildschirm den Punkt aus und verschiebt ihn. Der Punkt bleibt dann immer unter dem Mauszeiger und zugleich parallel zur Bildebene; dadurch ergibt sich im dreidimensionalen Raum eine Ebene mit potentiellen Positionen. Um den Punkt an eine nicht auf dieser Ebene liegende Stelle zu bewegen muss die Ansicht gedreht werden.

Punkt verschieben

Möchte der Anwender einen Punkt verschieben, so wird dieser mit der Maus durch Drücken der linken Maustaste »festgehalten« und über den Bildschirm bewegt. Der Punkt bleibt dabei immer unter dem Mauszeiger und bewegt sich auf der Ebene, die im Abstand der alten Punktposition parallel zur Bildebene verläuft (siehe Abbildung 4.3). Die Praxistauglichkeit dieser Interaktionstechnik wurde in Benutzungstests (siehe Abschnitt 6.1.2) nachgewiesen und hat sich als sehr intuitiv herausgestellt. Während des Verschiebens müssen einige technische Randbedingungen für das zugrunde liegende *Mesh* erhalten bleiben. Dazu zählt die möglichst reguläre Verteilung der *Vertices* und die Eigenschaften eines Festkörpers (siehe Abschnitt 2.1.5). Um dies zu erreichen dürfen auch benachbarte Punkte mitverschoben werden, falls dies den Arbeitsablauf unterstützt. Eine Übersicht der notwendigen Funktionalität lässt sich Abbildung 4.4 entnehmen.

Punkte erzeugen und löschen

Wählt der Anwender die Funktion des Erzeugens und Löschens von Punkten, indem er die rechte Maustaste verwendet, so kann er einen Punkt wie beim oben erläuterten Verschieben »festhalten« und im Raum bewegen. Anstatt jedoch andere

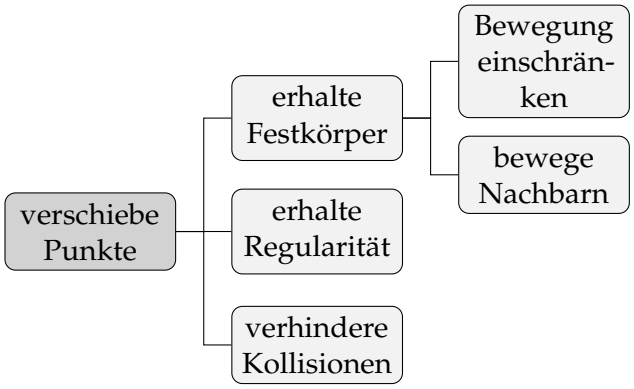


Abbildung 4.4.: Hierarchische Ordnung der benötigten Funktionalität für das Verschieben eines Punktes. Dunkler abgesetzte Kästen markieren Funktionen, die zum Teil Benutzerinteraktion verlangen.

Punkte aus dem Weg zu räumen bzw. nachzuziehen, wird – wie Abbildung 4.5 zu entnehmen ist – die Topologie und Konnektivität (engl. *connectivity*) verändert. Mit Topologie ist hier das Geschlecht (engl. *Genus*) des Gegenstandes gemeint. Die Verknüpfung von *Vertices* durch *Edges* wird Konnektivität genannt. Eine zu lange *Edge* wird so zerteilt, dass in der Mitte ein neuer Punkt entsteht. Ist eine *Edge* zu kurz, muss der zu nah gekommene Punkt absorbiert werden. Selbstkollisionen der Oberfläche resultieren automatisch in einer Neukonfiguration des *Meshes*, so dass Henkel und Löcher intuitiv vom Anwender produziert werden können.

4.1.3. Gegenstand visualisieren

Der virtuelle Gegenstand wird so dargestellt, dass der Benutzer seine Form erkennen und mit dieser arbeiten kann. Dazu muss die *Subdivision* (siehe auch Abschnitt 2.1.2 auf Seite 7) und Selbstverschattung nach jeder Änderung neu berechnet werden. Für das Anpassen der Ansicht ist beides nicht nötig, da keine blickwinkelabhängigen Effekte verwendet werden. Auswählbare Punkte sind auf der Oberfläche als blaue Kreise dargestellt. Als zusätzliche Hilfestellung wird in einer Farbkodierung von grün nach rot angezeigt, wie stark von einer regulären Aufteilung der Punkte abgewichen wird. Dadurch wird das Eingreifen der Automatismen (vgl. Abschnitt 4.1.2) schon frühzeitig durch eine Rotfärbung angekündigt und eine Gleichverteilung durch die Farbe grün signalisiert. Dieser kausale Zusammenhang wurde vielen Anwendern erst nach 15 bis 30 Minuten klar (vgl. Abschnitt 6.1). Alternativ könnten deshalb einander nahe kommende Punkte, farblich »zusammen fließen« und bald entstehende Punkte, langsam »einblenden«.

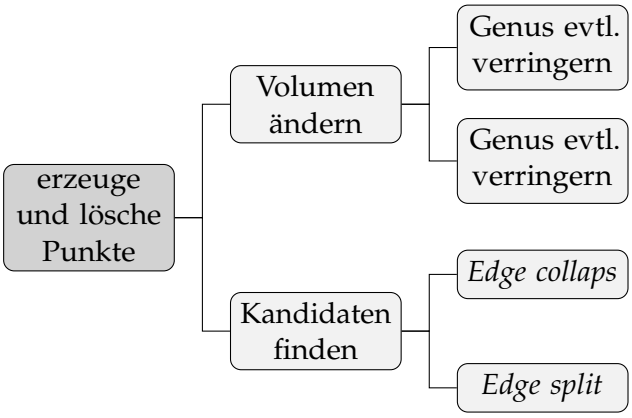


Abbildung 4.5.: Hierarchische Ordnung der benötigten Funktionalität zur Veränderung der *Mesh*-Konnektivität und Objekt-Topologie. Dunkler abgesetzte Kästen markieren Funktionen, die zum Teil Benutzerinteraktion verlangen.

Subdivision

Der virtuelle Gegenstand ist durch ein geschlossenes Netz von Dreiecken (siehe allgemein Abschnitt 2.1.2 auf Seite 7 und konkret Abschnitt 4.3.1) beschrieben, welche durch mehrmalige *Subdivision* (vgl. Abschnitt 4.3.2) in eine hinreichend glatte Oberfläche überführt werden. Auf dem so entstandenen feinen Gitter bleiben alle Kanten und *Vertices* des Ursprungs-*Meshes* erhalten.

Selbstverschattung

Um den Oberflächenverlauf eines Gegenstandes besser begreifen zu können, wird eine unterstützende Schattierung vorgenommen [LB]. Dazu wird *Ambient Occlusion* (dt. Selbstverschattung) simuliert. Als Beleuchtung wird eine gleichförmige Helligkeit der gesamten Umgebung angenommen, da so eine Verschattung in jeder Orientierung des Raum gleich aussieht. Dadurch ergibt sich eine direkte Visualisierung der lokalen Gestalt an jedem Punkt auf der Oberfläche (siehe Beispiel in Abbildung 4.6). Die Darstellung der Verschattung ergibt sich dann stark vereinfacht durch das Abdunkeln weiter innen liegender Regionen. Der verwendete Ansatz zur Simulation von *Ambient Occlusion* kann in Abschnitt 4.3.3 auf Seite 43 nachgelesen werden.

Ansicht verändern

Durch die Benutzungstests (siehe Kapitel 6) wurde klar, dass die Steuerung einer Kamera mit Rotation, Zoom und Translation vielen Nutzern schwer fällt. Nach einigen Experimenten wurde klar, das Zoomen und Verschieben vollständig von Au-

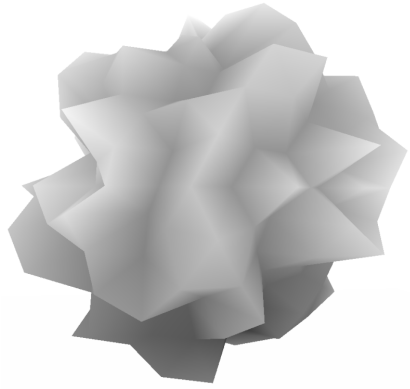


Abbildung 4.6.: Bei einer gleichmäßigen Beleuchtung der gesamten Hemisphäre sind – lokal betrachtet – weiter innen liegende Bereiche der Oberfläche immer etwas abgedunkelt. Diese Information trägt stark zum Verständnis des Oberflächenverlaufs bei.

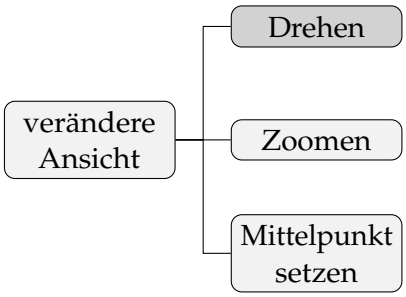


Abbildung 4.7.: Hierarchische Ordnung der benötigten Funktionalität zum Wechseln der Kameraposition. Dunkler abgesetzte Kästen markieren Funktionen, die zum Teil Benutzerinteraktion verlangen.

tomatismen übernommen werden können (siehe Kamerasteuerungs-Komponente in Abschnitt 4.3.8). Durch Klicken und Ziehen auf dem Hintergrund oder auf Regionen der Oberfläche, die nicht in unmittelbarer Nähe eines Punktes liegen, dreht der Anwender sich um den zuletzt angewählten Punkt. Der Abstand zu dem Objekt wird dabei automatisch angepasst (vgl. Abbildung 4.7). Ein einfacher Klick auf den Hintergrund verändert die Entfernung so, dass der gesamte Gegenstand im Bild zu sehen ist.

4.2. Datenfluss

Um die Struktur der Software zu veranschaulichen, bietet sich ein Datenflussdiagramm nach DeMarco [DeM79] an. Abbildung 4.8 können die Wege der Daten zwischen Anwender, *Server* und *Client* entnommen werden. Die Schnittstelle nach außen bildet die grafische Benutzungsschnittstelle (GUI). Der eigentliche Datenspeicher für Geometrie und Meta-Informationen befindet sich auf dem *Server*. Von ihm bekommt der *Client* Informationen über alle Gegenstände (z. B. Namen, eindeutige Nummer, ...) und *Mesh*-Daten, für die er ein Abonnement besitzt. Eine lokale Kopie der Daten beim *Client* wird nur für die Darstellung und Auswahl benötigt. Alle Änderungen werden immer erst an den *Server* gesendet, der diese verarbeitet und in seinen Datenspeicher einpflegt. Erst dann werden die Änderungen vom *Server* an alle Abonnenten geschickt. Auf diesem Weg bekommt jeder *Client* seine Informationen – auch derjenige, der Änderungen durchführen möchte.

Für eine visuelle Rückkoppelung beim Modellieren müssen die Manipulationen daher in »Echtzeit« über den *Server* geleitet werden. Bei den heutigen Computer- und Netzwerk-Leistungen ist diese Daten-Schleife performant genug, um einer großen Anzahl von Teilnehmern das kooperative Arbeiten an virtuellen Gegenständen zu ermöglichen.

4.3. Systemkomponenten

Ein Software-Produkt lässt sich hierarchisch in Systemkomponenten aufteilen. Diejenigen Teile der Anwendung, die nicht weiter zerlegbar sind oder zerlegt werden sollen, tragen die Sonderbezeichnung Systemelement [Bal98]. Eine solche Ordnung kann Abbildung 4.9 entnommen werden. In diesem Abschnitt werden die Systemelemente der entworfenen Architektur mit ihren Abhängigkeiten detailliert dargestellt, so dass eine Implementation möglich wird.

4.3.1. Mesh

Wie in Abschnitt 2.1.2 auf Seite 7 vorgestellt wurde, besteht ein *Mesh* aus Polygonen. Im einfachsten Fall hat jedes Polygon drei Ecken. Mit einem Dreiecksgitter lassen sich alle Arten von Flächen beschreiben; dies prädestiniert sie daher als Grundlage zur Repräsentation der Geometrie. Da sich die *Mesh*-Topologie, d. h. die Verknüpfung der *Vertices* und *Edges*, laufend ändert, wird eine Datenstruktur benötigt, die effizient aktualisiert und abgefragt werden kann. Die Komponenten *Subdivision*, Selbstverschattung und die Gültigkeitsprüfung stellen besondere Anforderungen an das *Mesh* (vgl. Abschnitte 4.3.2, 4.3.3 und 4.3.6). Aus diesen Gründen bietet sich eine Verwendung der *Halfedge*-Datenstruktur an [CKS]. Diese bietet sich dank der hohen Performanz und hervorragender Flexibilität für viele Anwendungsfälle besonders gut an (siehe Abbildung 4.10).

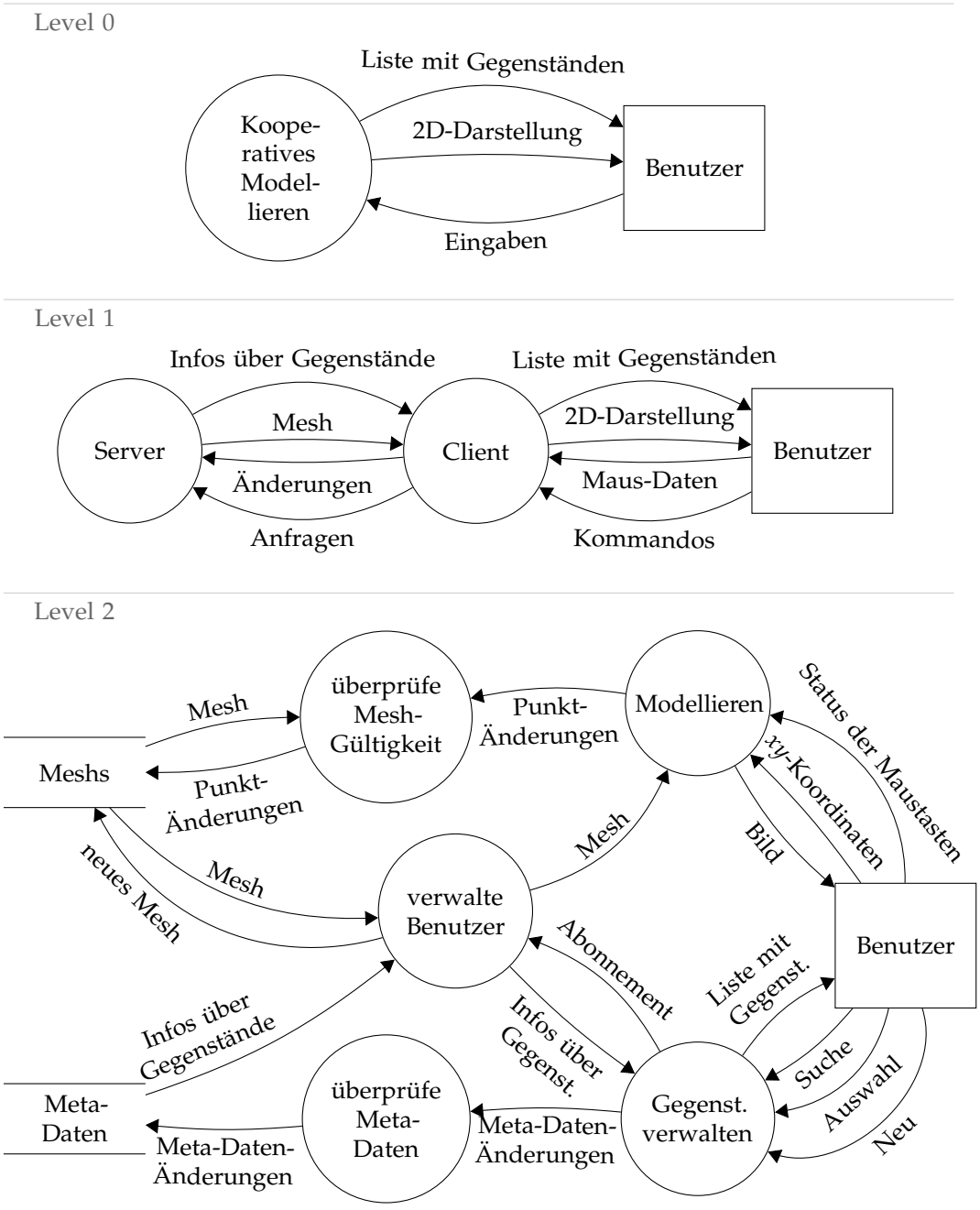


Abbildung 4.8.: Schematische Darstellung der Datenflüsse in drei Verfeinerungsstufen. Prozesse sind durch Kreise und Datenspeicher mit zwei horizontalen Linien markiert. Die Schnittstelle ist durch einen rechteckigen Rahmen gekennzeichnet.

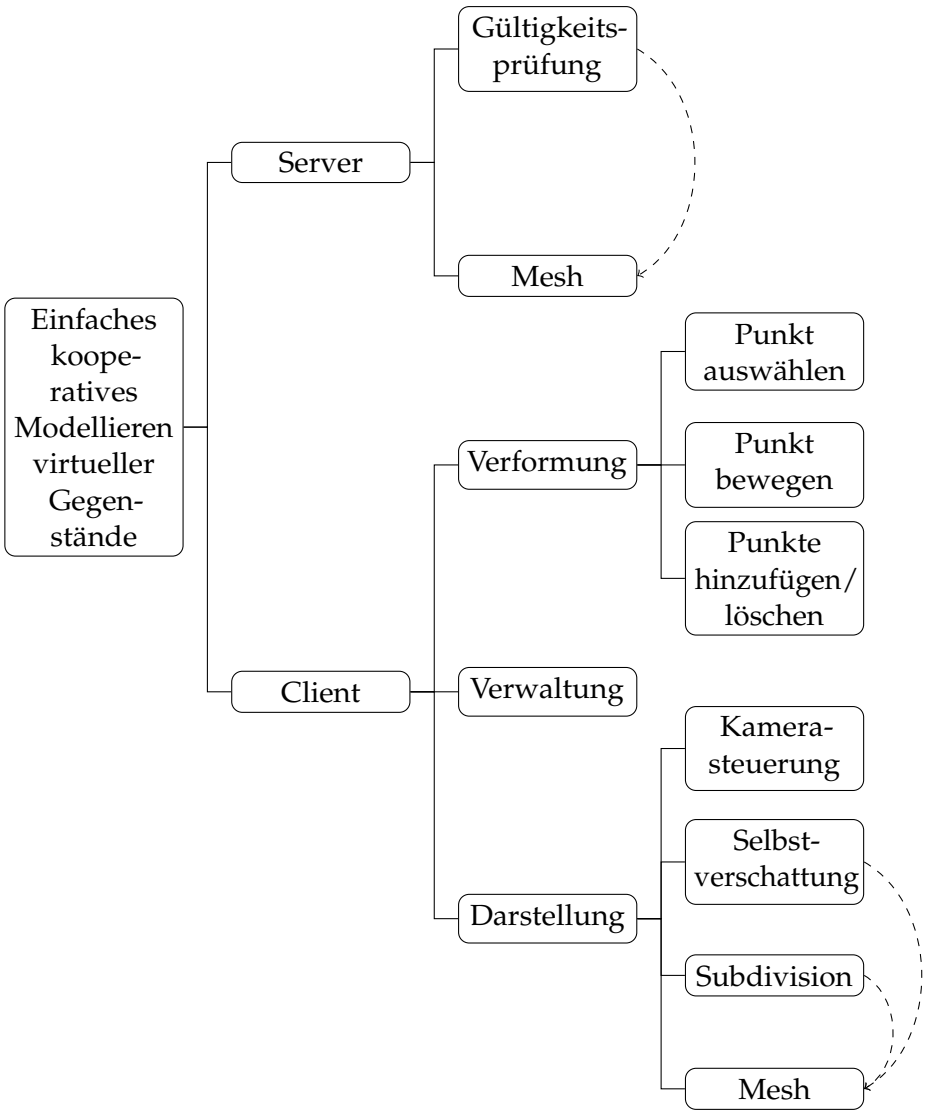


Abbildung 4.9.: Hierarchische Ordnung der Systemkomponenten und der nicht weiter zu zerlegenden Systemelemente. Gestrichelte Linien kennzeichnen besondere Anforderungen an andere Komponenten.

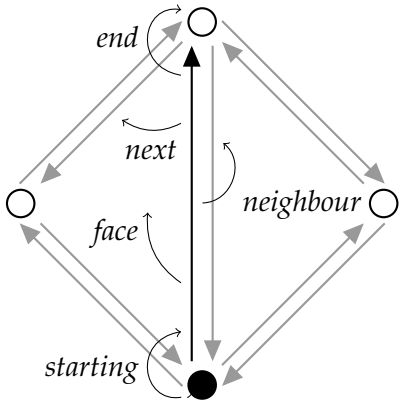


Abbildung 4.10.: Bei der *Halfedge*-Datenstruktur ist jedem *Vertex* eine einzige ausgehende *Halfedge* zugeordnet (*starting*). Jede *Halfedge* kennt 1. den *Vertex*, auf den sie zeigt (*end*), 2. das *Face*, zu dem es gehört (*face*), 3. die gegen den Uhrzeigersinn nächste *Halfedge* des *Faces* (*next*) und 4. die entgegenkommende *Halfedge* (*neighbour*). Jedes *Face* hat außerdem Zugriff auf eine seiner *Halfedges*.

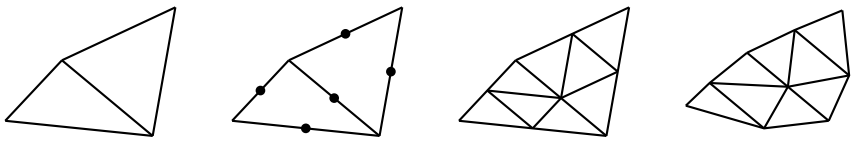


Abbildung 4.11.: Bei der *Loop-Subdivision* werden alle Kanten geteilt und die so entstandenen *Vertices* zu jeweils vier neuen Dreiecken verbunden. Anschließend werden alle Punkte neu positioniert, so dass sich nach mehreren Iterationen eine weiche Oberfläche bildet.

4.3.2. Subdivision

Zur Verfeinerung des Basis-*Meshs* werden mehrere Iterationen der *Loop-Subdivison* [Loo87] durchgeführt. Dieses Schema basiert auf Dreiecken, ist relativ einfach zu implementieren und liefert eine ausreichend glatte Oberfläche. In jedem Schritt wird jedes Dreieck in vier neue aufgeteilt (vgl. Abbildung 4.11), in dem jede Kante in der Mitte aufgeteilt wird und so einen neuen *Vertex* bekommt. Die daraus resultierenden *Edges* müssen daraufhin in die Datenstruktur eingepflegt werden. In einem zweiten Schritt werden dann die Positionen der Punkte angepasst[3], damit die Oberfläche einen glatten Verlauf bekommt. Nach mehreren Iterationen dieses

[3] Die genauen Masken für die Gewichtung können [Loo87] oder dem SIGGRAPH Kurs *Subdivision for Modeling and Animation* [ZSD+00] entnommen werden.

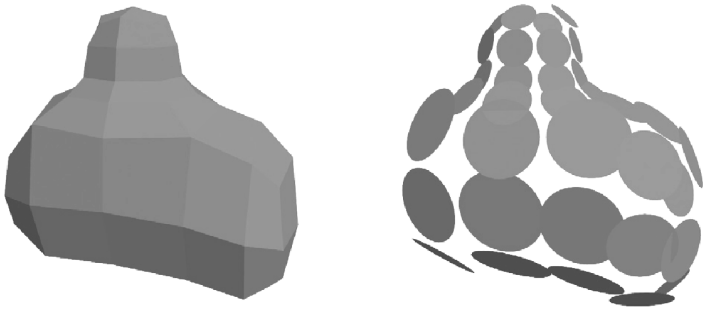


Abbildung 4.12.: Annähern der Oberfläche durch Scheiben zur Simulation von *Ambient Occlusion*. Bild: [Bun05]

Vorgangs konvergiert das *Mesh* gegen die so genannte *Limit*-Fläche, was bedeutet, dass zu jeder Startkonfiguration eines *Meshs* eine eindeutige, hinreichend glatte Endkonfiguration der Oberfläche existiert.

Die *Subdivision* wird der einfacheren Implementation wegen bei jeder Änderung an dem Objekt aktualisiert. Bei der Implementation ist deshalb auf eine besonders effiziente Umsetzung zu achten.

4.3.3. Selbstverschattung

Die interaktive Simulation von *Ambient Occlusion* stellt eine besondere Herausforderung dar und kann deshalb nur approximiert werden. Wie in [Bun05] beschrieben, wird dazu die Oberfläche des darzustellenden *Meshes* durch eine Vielzahl von Scheiben (engl. *disks*) angenähert (siehe Abbildung 4.12). Es kann dann in einem Algorithmus der Komplexität $O\left(n^2\right)$ für jede Scheibe R der nicht durch andere Scheiben $E_{1..n}$ verdeckte Teil der Hemisphäre av berechnet werden:

$$\mathbf{re}_i = \frac{(E_i).\texttt{position} - (R).\texttt{position}}{(E_i, R).\texttt{distance}} \tag{4.1}$$

$$s_i = 1 - \left(\frac{1}{\sqrt{\frac{(E).\texttt{area}}{(E_i,R).\texttt{distance}^2}} + 1} \right) \tag{4.2}$$

$$av = \sum_{i=1}^{n} \left((E_i).\texttt{normal} \circ \mathbf{re}_i \right) \cdot \left((R).\texttt{normal} \circ \mathbf{re}_i \right) \cdot s_i \tag{4.3}$$

Dieser Wert liegt im Bereich $[0 \ldots 1]$ und wird zur Abdunkelung der betroffenen Umgebung verwendet. Je höher av, desto heller ist die Oberfläche an dieser Stelle, da die anderen Scheiben in dieser Hemisphähre wenig Schatten werfen. Um eine Verdunkelung durch den Boden zu approximieren, kann zusätzlich av von einer

unendlich großen Scheibe mit Höhe und Ausrichtung des Bodens beeinflusst werden.

Dieser Algorithmus ist stark parallelisierbar und sollte hoch effizient implementiert werden, da ansonsten eine Darstellung von größeren *Meshes* nicht mehr möglich wäre. Eine Umsetzung auf der GPU, wie in [Bun05] beschrieben, steigert die Performanz enorm und rechtfertigt so den erhöhten Entwicklungsaufwand.

4.3.4. Punkt auswählen

Damit ein Punkt vom Benutzer mit der Maus ausgewählt werden kann, müssen Regionen auf dem *Mesh* eindeutig auf der zweidimensionalen Bildfläche identifizierbar gemacht werden (engl. *picking*). Eine elegante Lösung mit *OpenGL*[4] wird in [W+99] als *Object Selection Using the Back Buffer* bezeichnet und ist hervorragend zur Lösung des Problems geeignet. Jede selektierbare Region auf dem *Mesh* wird beim Klicken der Maus mit einer eindeutigen Farbe (engl. *color index*) in den *Back Buffer* gezeichnet. Ebenso können Hintergrund und nicht selektierbare Teile des *Meshes* kodiert werden. Anschließend lässt sich mit `glReadPixels` an der Position des Mauszeigers die Farbe abrufen und damit feststellen, was der Anwender auf dem Bildschirm auswählen möchte. Der gesamte Prozess bleibt für den Benutzer unsichtbar, da in den *Back Buffer* vor dessen Darstellung (engl. *swap*) wieder die richtige Szene gerendert wird.

4.3.5. Punkt bewegen

Wird ein Punkt mit der Maus an die Bildschirmkoordinaten (x, y) verschoben, so muss er sich, wie schon auf Seite 35 erläutert, parallel zur Bildebene bewegen und unter dem Mauszeiger bleiben. Die neue Position $\mathbf{p}_{new}$ lässt sich mit linearer Algebra berechnen. Dazu werden die Koordinaten der Maus mit `gluUnproject` in eine 3D-Position $\mathbf{m}$ umgewandelt. Mit Hilfe der Betrachterposition $\mathbf{a}$ kann daraus der Richtungsvektor $\mathbf{l}$ der Linie bestimmt werden, auf der sich der Punkt bewegen kann, um unter dem Mauszeiger zu bleiben. Der genaue Ort wird durch die Projektion der alten Position des Punktes $\mathbf{p}$ in Betrachterkoordinaten auf diesen Richtungsvektor bestimmt.

$$\mathbf{l} = \frac{\mathbf{a} - \mathbf{m}}{|\mathbf{a} - \mathbf{m}|} \tag{4.4}$$

$$\mathbf{p}_{new} = \mathbf{a} + ((\mathbf{p} - \mathbf{a}) \circ \mathbf{l}) \cdot \mathbf{l} \tag{4.5}$$

Um einen guten Verlauf der *Edges* zu unterstützen sollte die Krümmung der Oberfläche an dem zu verschiebenden Punkt analysiert werden. Eine Mausbewe-

[4] Januar 2007: `http://www.opengl.org`

gung entgegen der geringsten Krümmung wäre dann tendenziell abzuschwächen; so erhält der Benutzer beim Verschieben eine automatische Führung.

4.3.6. Gültigkeitsprüfung

Der Server prüft, ob ein Änderungswunsch gültig ist und daher auf den virtuellen Gegenstand angewendet werden darf oder nicht. Ist die Änderung legal, werden alle Clients darüber in Kenntnis gesetzt und die Änderung an der Geometrie vorgenommen. Die zu erfüllenden Kriterien für eine Änderung sind im Einzelnen:

- Der zu ändernde Punkt darf nicht vor sehr kurzer Zeit von einer anderen Person bearbeitet worden sein. Dazu werden Listen über Punkte geführt, die von einem Benutzer in der letzten Sekunde bearbeitet wurden und zu gültigen Änderungen führten.

- Das *Mesh* darf keine zu scharfen Kanten bekommen; es ist daher zu Überprüfen, dass kein Diederwinkel[5] an einer *Edge* größer als 170 Grad wird.

- die Dreiecke bleiben alle in etwa gleich groß. Wenn die geodätische Linie[6] zwischen zwei Punkten nicht mehr zwischen einem Minimal- und Maximalwert liegt, darf die Änderung nicht durchgeführt werden.

- Das *Mesh* muss weiterhin einen Festkörper beschreiben. Dazu ist eine Überprüfung auf Selbstdurchdringung erforderlich.

Kommen sich zwei Punkte zu nah, wird ein Mindestabstand durch das Verschieben von Punkten oder – wenn nötig – auch dem Blockieren (engl. *locking*) der Bewegung sichergestellt. Entfernen sich zwei Punkte, die durch eine *Edge* verbunden sind, zu weit voneinander, wird ebenso verfahren. Zwischen beiden Fällen besteht ein Freiraum, in dem sich die Punkte ungehindert bewegen dürfen. So kann eine sehr reguläre Verteilung eingehalten werden, ohne dass der Anwender Kenntnis über die technischen Gründe besitzen muss. Die selben beiden Mechanismen kommen zum Einsatz, um die Eigenschaften eines Festkörpers (siehe 2.1.5 auf Seite 9) beizubehalten. Dazu muss bei jeder Bewegung überprüft werden, ob die Oberfläche des Gegenstandes nicht mit sich selbst kollidiert.

[5] Der Diederwinkel (engl. *dihedral angle*) ϕ_{AB} zwischen zwei Flächen A, B ist definiert als das Skalarprodukt ihrer Normalen: $\cos\phi_{AB} = \mathbf{n}_A \circ \mathbf{n}_B$ und beschreibt den Winkel zwischen den beidenen Ebenen an ihrer Schnittkante.

[6] Eine geodätische Linie ist immer die kürzeste Verbindung zwischen zwei Punkten. Im Euklischen Raum ist dies immer eine Line, innerhalb einer Mannigfaltikeit jedoch auch davon verschieden.

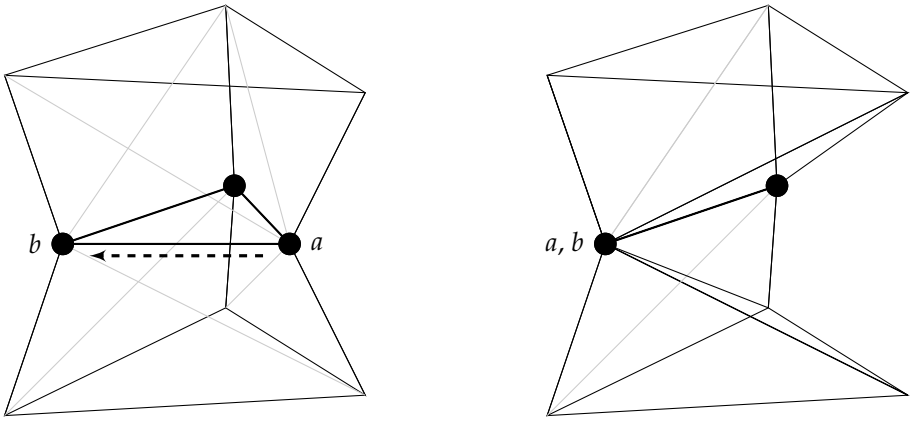


Abbildung 4.13.: Bei einem *Edge split* kann es passieren, dass die Eigenschaft der Zwei-Mannigfaltigkeit verloren geht. Wenn *Vertex a* auf *Vertex b* bewegt wird, ist die entstehende Konfiguration nicht mehr zwei-mannigfaltig.

4.3.7. Punkte hinzufügen und löschen

Wird das Gültigkeitskriterium zur regulären Punktverteilung verletzt, kann auf Wunsch des Clients der Server einen neuen Punkt hinzufügen bzw. einen Bestehenden löschen, um die Gültigkeit wieder herzustellen.

Ist eine *Edge* zu lang, wird dort ein *Edge split* durchgeführt. Das bedeutet, die Kante wird in zwei geteilt. Dies geschieht, indem in ihrer Mitte ein neuer Punkt erzeugt und dieser mit den umliegenden Punkten durch *Edges* verbunden wird. Ist eine *Edge* zu kurz, wird sie mit einem *Edge collapse* entfernt. Dabei wird der Punkt am Ende der *Edge* gelöscht und alle mit ihm verbundenen *Edges* an den übrig bleibenden Punkt gehängt. Dabei kann es passieren, das die Zwei-Mannigfaltigkeit verletzt wird. In einem solchen Fall muss der *Edge collapse* unterdrückt werden. Dazu wird vor der Operation sichergestellt, dass die beiden 1-Ringe[7] der sich zu nah gekommenden *Vertices* höchstens zwei *Vertices* teilen (siehe Abbildung 4.13).

Etwas aufwändiger ist die Unterstützung zur Änderung des *Genus*: Wenn das Gültigkeitskriterium der Selbstdurchdringung verletzt wird, muss an dieser Stelle ein Loch entstehen um die Gültigkeit wieder herzustellen. Dazu werden die beiden sich zu nahe gekommenen *Faces* gelöscht und die sich jeweils nächsten *Vertices* der gegenüberliegenden Seiten zusammengefügt. Wenn ein *Edge collapse* an einem Loch durchgeführt wird, welches nur durch drei *Edges* definiert ist, so muss dies wieder geschlossen werden. Hierfür muss die weiter oben beschriebene Überprüfung auf Zwei-Mannigfaltikeit um eine Berechnung der Euler-Charakteristik[8]

[7] Die direkten Nachbar-*Vertices* eines *Vertex* »*V*« werden »1-Ring von *V*« genannt. Diese Nachbarn sind *Vertices*, mit denen er durch eine *Edge* verbunden ist.

[8] Die Euler-Charakteristik χ ist eine Kennzahl für geschloßene Flächen, mit der unter

ergänzt werden. Ändert sich der Wert der Euler-Charakteristik nach oben, handelt es sich um eine Abtrennung von Geometrie und wird verhindert, da die Anwendung nur Einzelgegenstände verwaltet. Ändert sich der Wert jedoch nach unten, so müssen anstatt eines *Edge collapse* zwei *Faces* in dem Loch erzeugt werden, die dann ein Stück auseinander geschoben werden, um wieder die Kriterien eines Festkörpers sicher zu stellen.

4.3.8. Kamerasteuerung

Am einfachsten ist das automatische Verschieben des Betrachtungsmittelpunktes (engl. *center of interest*, *coi*) inklusive eines angenehmen *ease out*-Effekts umzusetzen. Wird ein neuer Punkt $\mathbf{p}$ angewählt, lässt sich der aktuelle Bildmittelpunkt $coi_{current}$ für jeden Frame bestimmen durch

$$coi_{current} \quad = \quad s \cdot (\mathbf{p} - coi_{old}) + coi_{old} \tag{4.6}$$

wobei s ein skalarer Wert im Bereich $[0..1]$ ist, der je nach Bildwiederholrate (engl. *Framerate*) angepasst werden muss. Dadurch fährt die Kamera zu beginn mit hoher Geschwindigkeit auf den neuen Punkt zu. Je näher dieser der Bildmitte kommt, desto langsamer wird auch die Kamerafahrt.

Für das automatische Zoomen wird der Abstand zum Betrachtungsmittelpunkt so gewählt, dass eine gegebene Menge von 3D-Koordinaten innerhalb des Bildausschnittes liegen und diesen gut ausfüllen (siehe Pseudocode 4.14). Ist kein Punkt selektiert, da auf den Hintergrund geklickt wurde, sind die 3D-Koordinaten durch die sechs Ecken der *Bounding Box*[9] des virtuellen Gegenstandes gegeben. Während mit einem Punkt gearbeitet wird kann etwas näher heran gefahren werden; die Menge der 3D-Koordinaten entspricht dann den Nachbarn des ausgewählten Punktes und deren Nachbarn; dem sogenannten 2-Ring. Um ein gezielteres Arbeiten zu ermöglichen, werden Kamerabewegungen pausiert, solange ein Punkt in Bewegung ist.

Damit sich der Anwender besser zurecht findet, ist es äußerst hilfreich, einen Untergrund visuell darzustellen. Dieser muss sich aber immer unterhalb des Gegenstands befinden, da er nur eine Orientierungshilfe darstellt und keinenfalls den Modellierungsprozess behindern darf.

---

anderem das Geschlecht einer Mannigfaltikeit bestimmt werden kann. Formel: $\chi = \#vertices - \#edges + \#faces$

[9] Die *Bounding Box* wird durch die minimale und maximale Ausdehnung des virtuellen Objektes definiert

```
zoomToBoundingPoints(boundingPoints){

   // first check for points outside of the screen
   maxOutlier = 0
   foreach 3D-point in boundingPoints {
      2D-point = projectToNearPlane(3D-point)
      if (isOutsideScreen(2D-point))
          maxOutlier = maximumOf(maxOutlier,
             getDistanceFromScreen(2D-point))
   }
   if (maxOutlier != 0){
      zoomOut(maxOutlier);
      return;
   }

   // if we reach this line, everything is "in screen"
   minBorder = 1000;
   foreach 3D-point in boundingPoints {
      2D-point = projectToNearPlane(3D-point)
      minBorder = minimumOf(minBorder,
            getDistanceFromScreen(2D-point))
   }
   zoomIn(minBorder);
}
```

Abbildung 4.14.: Pseudocode zur Berechnung des Kameraabstands.

Kapitel 5.

Technische Umsetzung

> Der Einfall ist ein Schritt
> mit Siebenmeilenstiefeln,
> die Ausführung der Weg
> zurück zu Fuß.
>
> *Peter Tille*

In diesem Kapitel wird die Realisierung der Anwendung mit Bezug auf die Software-Architektur (Kapitel 4) und dem Ziel eines Benutzungstests (Kapitel 6) beschrieben. Der erste Abschnitt erläutert kurz die Wahl der Techniken und Hilfskonstrukte, die bei der Umsetzung eine wesentliche Rolle spielten. Daraufhin werden einzelne Prototypen zur Realisierung wichtiger Teilkomponenten vorgestellt und die daraus gewonnenen Erkenntnisse zusammengefasst. Diese trugen stetig zur Erweiterung und Verbesserung der in dieser Arbeit vorgestellten Lösung bei. Das Kapitel schließt mit einer Beschreibung der konzipierten Gesamt-Software. Diese implementation ermöglichte es, Benutzungstests durchzuführen sowie das Zusammenspiel der einzelnen Komponenten zu ergründen.

5.1. Wahl der Werkzeuge und Bibliotheken

Wenn es darum geht, Software zu implementieren, stellt sich sehr bald die Frage nach den zu verwendenden Hilfsmitteln. Das betrifft einerseits Entwicklungsumgebungen, Programmiersprachen und ähnliche Werkzeuge, andererseits aber auch bereits bestehende Bibliotheken und Komponenten. Letztere sind natürlich schon in einer bestimmten Porgrammiersprache verfasst und schränken damit unter Umständen die Wahl der Werkzeuge ein.

5.1.1. Bibliotheken

Bibliotheken stellen wiederverwendbare Funktionalität in einer eigenen Architektur zur Verfügung, so dass konkrete Anwendungen davon profitieren können. Existieren für Teile der eigenen entworfenen Architektur bereits Bibliotheken,

kann unter Umständen der Implementierungsaufwand drastisch gesenkt werden. Einfache und oft verwendete Datenstrukturen (Container, Matrizen, ...) und Operationen (sortieren, suchen, ...) werden allen heute gängigen Programmiersprachen bereits beigelegt oder können leicht selbst implementiert werden. Interessanter sind deshalb spezielle Bibliotheken, die ein bestimmtes Problemfeld adressieren. Hier schienen zu Beginn der Entwicklung die Bibliotheken *OpenMesh*[1] und *Verse*[2] einen wertvollen Beitrag leisten zu können.

OpenMesh ist eine C++-Implementierung der effizienten *Halfedge*-Datenstruktur zur Beschreibung beliebiger polygonaler Netze (siehe 2.1.2). Es sind nahezu alle wichtigen Operationen zur Manipulation und Analyse vorhanden, und die Bibliothek ist seit langem im Praxiseinsatz. Da die *Mesh*-Repräsentation eine sehr kritische Komponente der in Abschnitt 4.3 beschriebenen Architektur darstellt, schien eine Verwendung vorteilhaft. Während der Entwicklung wurde jedoch deutlich, dass die Bausteinarchitektur von *OpenMesh* eine zu starke Abstraktion vornimmt und an vielen Stellen die Implementation einer auf den Anwendungsfall zugeschnittenen Lösung vorteilhafter gewesen wäre.

Ähnlich verhält es sich mit dem Netzwerk-Protokoll *Verse*. Dies wird im Rahmen eines Projekts entwickelt, welches sich zum Ziel gesetzt hat, viele verschiedene Anwendungen so miteinander zu vernetzen, dass sie gemeinsam 3D-Daten nutzen und bearbeiten können. Eine Analyse führte zu dem Schluß, dass die dafür entwickelte Bibliothek auch in dieser Arbeit für die Kommunikation zwischen *Server* und *Client* geeignet sei. Jedoch wurden zu einem fortgeschrittenen Zeitpunkt der Arbeit inhärente Schwierigkeiten mit dieser Bibliothek deutlich:

- Den Beispielserver zu erweitern hätte eine sehr lange Einarbeitungszeit erfordert.
- *Verse* zielt darauf ab, beliebige Geometrie zu verarbeiten und hat daher keinerlei Kontrollmechanismen für Mannigfaltigkeit, Volumen, usw.
- Komplexere *Mesh*-Operationen wie *Edge split* und *Edge merge* erfordern eine Zerlegung in Basisbefehle (*add vertex*, *add face*, ...). Da sich bei *Verse* aber zu jeder Zeit das gesamte *Mesh* ändern kann, ist die Ausführung aufeinander aufbauender Operationen extrem kompliziert.
- Eine Detailhierarchie, wie sie in Abschnitt 2.2.5 beschrieben wurde, wäre nur sehr aufwändig zu realisieren und würde die Kompatibilität mit anderen *Verse*-Applikationen zerstören.
- Das Protokoll sieht keinerlei Konstruktions-Historie vor.

[1] Januar 2007: `http://www.openmesh.org`
[2] Januar 2007: `http://verse.blender.org`

- Die Protokoll-Implementation in der Programmiersprache C erfordert einen selbst zu implementierenden objektorientierten *Wrapper*[3].

Andere Bibliotheken wie *OpenGL*[4],*SigC++*[5] und *GTK* (siehe Abschnitt 5.1.2) boten hingegen willkommene Arbeitserleichterungen. So konnten trotz der erwähnten Schwierigkeiten mit den oben genannten Bibliotheken eine Gesamtanwendung entstehen, in der alle Komponenten zusammenwirken und Benutzungstests durchgeführt werden konnten.

5.1.2. Zielplattform

Als Zielplattform wurde *GNU/Linux*[6] unter Verwendung des *GNOME Desktops*[7] gewählt. Diese Konfiguration ist frei erhältlich[8] und zeichnet sich durch jahrelange Bemühungen um eine benutzungsfreundliche Umgebung aus. Da zu den Grundprinzipien der Software-Ergonomie die Konsistenz gehört, sollte daher die *Gnome*-Standard-Bibliothek *GTK*[9] zur Gestaltung der WIMP-GUI zum Einsatz kommen.

5.1.3. Programmiersprache

Zur Implementation einer benutzergesteuerten grafischen Anwendung mit 3D-Ansicht gibt es drei infrage kommende Programmiersprachen: Java[10], C#[11] oder C++[12]. Da C# viele Vorteile von Java und C++ vereint, wurde ganz zu Beginn der Arbeit auf diese Sprache gesetzt. Als sich jedoch abzeichnete, dass eine Verwendung von *OpenMesh* (siehe Abschnitt 5.1.1) hilfreich wäre, musste die Entwicklung auf C++ umgestellt werden. Die in *OpenMesh* sehr exzessiv betriebene Verwendung von C++-*Templates* machte eine Anbindung durch einen *Wrapper*[13] unmöglich.

Trotz weitreichender Erfahrung mit der Sprache C++ gab es aufgrund ihrer Systemnähe immer wieder Schwierigkeiten mit der Implementation. Viele Merkmale von C++ sind nur nützlich, wenn es um hochperformante, hardwarenahe

[3] *Wrapper* kapseln Funktionalität in sich und stellen eine neue Schnittstelle für diese zu Verfügung.
[4] Januar 2007: `http://www.opengl.org`
[5] Januar 2007: `http://libsigc.sourceforge.net`
[6] Januar 2007: `http://de.wikipedia.org/wiki/Linux`
[7] Januar 2007: `http://www.gnome.org`
[8] Die Bezeichnung »frei« ist hier sowohl im Sinne von »Freiheit« als auch im Sinne von »kostenlos« gemeint.
[9] Januar 2007: `http://www.gtk.org`
[10] Januar 2007: `http://de.wikipedia.org/wiki/Java_(Programmiersprache)`
[11] Januar 2007: `http://de.wikipedia.org/wiki/C-Sharp`
[12] Januar 2007: `http://de.wikipedia.org/wiki/C++`
[13] Um C++ in C# zu *wrappen* wurde in dieser Arbeit das Programm *Swig* (`http://www.swig.org`) verwendet.

Programmierung geht (z. B. `const`, `mutable` und *Pointer*). In den meisten Bereichen einer Anwendung ist diese geringe Abstraktion aber sehr hinderlich und kann nicht »ausgeblendet« werden. Andere Merkmale sind wiederum höchst umstritten; Mehrfachvererbung führt z. B. nicht nur zu einem komplizierteren Objektmodell, sondern erfordert auch vom Entwickler größte Vorsicht und die Verwendung einiger spezieller *Modifier* (`using`, `virtual public`, ...). Auch die Datei-Aufteilung des Quelltextes in Deklaration (`.hpp`) und Definition (`.cpp`) wird spätestens bei der Verwendung von `Templates` unsinnig und behindert vor allem einen flüssigen Arbeitsablauf beim Hinzufügen neuer Methoden und Attribute. Besonders störend ist während der Programmierung mit dieser Sprache auch, dass es zwar spezielle *Low Level*-Konstrukte wie `inline` und `sizeof` gibt, jedoch z. B. für Methoden-*Callbacks*, Typ-Informationen, String-Operationen und das Einschränken von `Template`-Typen unschöne Hilfskonstruktionen verwendet werden müssen.

5.2. Prototypen

Zur Prüfung bestimmter Eigenschaften wurden viele der benötigten Funktionalitäten – soweit es möglich war – in einem eigenständigen Prototypen entwickelt (vgl. D). In den nächsten Unterabschnitten werden diese kurz vorgestellt und ihre Integration in das Gesamtsystem erläutert.

5.2.1. GTK mit OpenGL

Der erste Prototyp in dieser Arbeit befasste sich mit der Kombination von *GTK* und *OpenGL*. Ersteres sollte, wie in Abschnitt 5.1.2 beschrieben, die Integration in den *Gnome Desktop* sicherstellen, zweiteres war unabdingbar um eine interaktive Darstellung der virtuellen Gegenstände zu ermöglichen. Da *GTK* in seiner Klassenbibliothek kein eigenes *Widget* zum Einbetten von *OpenGL* besitzt, wurde die freie Erweiterung *gtkglext*[14] verwendet. Mit diesem Zusatz ließen sich problemlos *Buttons* und virtuelle Szenen gemeinsam in einem Fenster darstellen. Alle weiteren Prototypen verwenden diese Technologie für die GUI.

5.2.2. OpenMesh

Um sich in die gewöhnungsbedürftige Handhabung von *OpenMesh* (siehe Abschnitt 5.1.1) einzuarbeiten und die Möglichkeiten dieser Bibliothek zu evaluieren, wurde ein einfaches Dreiecks-*Mesh* erzeugt. Die Darstellung mit *OpenGL* und der mitgelieferte *Loop-Subdivision* Algorithmus (vgl. 4.3.2 auf Seite 42) funktionierten auf Anhieb. Daher wurde in der Gesamtanwendung die komplette Datenhaltung der Geometrie darauf aufgebaut.

[14] Januar 2007: `http://gtkglext.sourceforge.net/`

5.2.3. Interaktion mit dem Mesh

Für Selektion und Bewegung von *Vertices* entstand ein weiterer Prototyp. Hier wurde auf dem *OpenMesh*-Prototypen aufbauend die Selektion per Farbkodierung (siehe Abschnitt 4.3.4 auf Seite 44) umgesetzt. Für das Verschieben des ausgewählten Punktes musste nach kurzen Tests ein Umfragebogen (vgl. Abschnitt 6.1.2) entworfen werden. Dessen Ergebnis deutete darauf hin, dass die in Abschnitt 4.3.5 vorgestellte Interaktionstechnik eine besonders einfach zu erlernende Handhabung ermöglichen würde. Ein abschließendes Urteil konnte jedoch bei diesem Prototypen noch nicht erfolgen, da sich die Ansicht auf das *Mesh* nicht verändern ließ und so ein wesentlicher Teil der Metapher ausblieb.

5.2.4. Kamera-Steuerung

Um sich frei im Raum bewegen zu können, musste eine intuitive Navigation mit der Maus getestet werden. Die in [CMS88] vorgestellte und untersuchte »*virtual trackball*«-Interaktion versprach genau dies zu ermöglichen: Der Anwender dreht eine virtuelle Kugel, die das Objekt umschließt, mit festgehaltener Maus um eine durch die Drehung automatisch gegebene Achse. Dazu muss die in den Bereich $[-r \dots r]$ normalisierte aktuelle Position der Maus $\mathbf{m}_{current}$ und die zeitlich vorhergehende Position $\mathbf{m}_{old}$ auf eine Kugel projiziert werden. Während x und y gleich bleiben, wird die z-Koordinate durch die Formel

$$z = \sqrt{r^2 - \left|\begin{pmatrix} x \\ y \end{pmatrix}\right|} \tag{5.1}$$

ermittelt. So ergibt sich für Mauskoordinaten am Rand der virtuellen Kugel ein z-Wert nahe Null und in deren Mitte nahe r. Die Rotationsaxe lässt sich anschließend durch das Kreuzprodukt zwischen aktueller und alter projizierter Position und der Rotationswinkel α durch

$$\alpha = \arccos\left(1 - |\mathbf{m}_{current} - \mathbf{m}_{old}|\right) \tag{5.2}$$

bestimmen. Mit diesen Werten kann dann eine Rotation um den Kugelmittelpunkt durchgeführt werden. Dieses Konzept musste noch etwas erweitert werden, da vorerst nur Mausbewegungen innerhalb der virtuellen Kugel mit dem Radius r zu sinnvollen Ergebnissen führten. Um eine Navigation über den gesamten Bildschirm zu gestatten, wurden daher Mauskoordinaten außerhalb von r mit der Formel

$$x' = r - x + r \tag{5.3}$$

$$y' = r - y + r \tag{5.4}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ -1 \cdot \sqrt{r^2 - \left| \begin{pmatrix} x' \\ y' \end{pmatrix} \right|} \end{pmatrix} \tag{5.5}$$

auf die Rückseite der virtuellen Kugel projiziert.

5.2.5. Client-Server Modell

Um das Netzwerkprotokoll *Verse* (siehe Seite 50) auf seine Funktionstauglichkeit hin zu überprüfen, wurde der Interaktionsprototyp (siehe Abschnitt 5.2.3) um eine Netzwerk-Klasse erweitert. Diese kapselte alle puren C-Funktionen des Protokolls und stellte so eine saubere Schnittstelle zur Kommunikation mit dem *Verse*-Beispielserver bereit. Damit konnte auf diesem *Server* sehr einfach ein Polygon angelegt und durch Maus-Interaktion verschoben werden; auch gleichzeitig mit mehreren Benutzern und inklusive automatischer Sicherung der Daten (engl. *Backup*).

5.2.6. Selbstverschattung

Während der Arbeit an der Gesamtanwendung wurde klar, dass der Verlauf der Oberfläche nicht immer einfach zu erkennen war. Eine Simulation der Selbstverschattung erschien deshalb als Erweiterung angebracht. Der zu diesem Zweck entwickelte Prototyp implementierte den in Abschnitt 4.3.3 auf Seite 43 vorgestellten Algorithmus für die CPU, um die Funktionsweise der in [Bun05] beschriebenen Technik zu analysieren. Die visuellen Resultate waren sehr zufriedenstellend und selbst die Performanz reichte für die im Anwendungszenario auf Seite 2 beschriebenen Gegenstände, so dass der Quelltext direkt in das Endergebnis mit einfloss.

5.2.7. Topologisches Geschlecht

Um herauszufinden, wie sich das topologische Geschlecht eines Gegenstandes intuitiv verändern lässt, wurde zu einem sehr späten Zeitpunkt der Arbeit ein weiterer Prototyp entwickelt. Die Erkenntnisse wurden bereits im Abschnitt 4.3.7 beschrieben, konnten jedoch leider nicht mehr in die Endanwendung eingebaut werden.

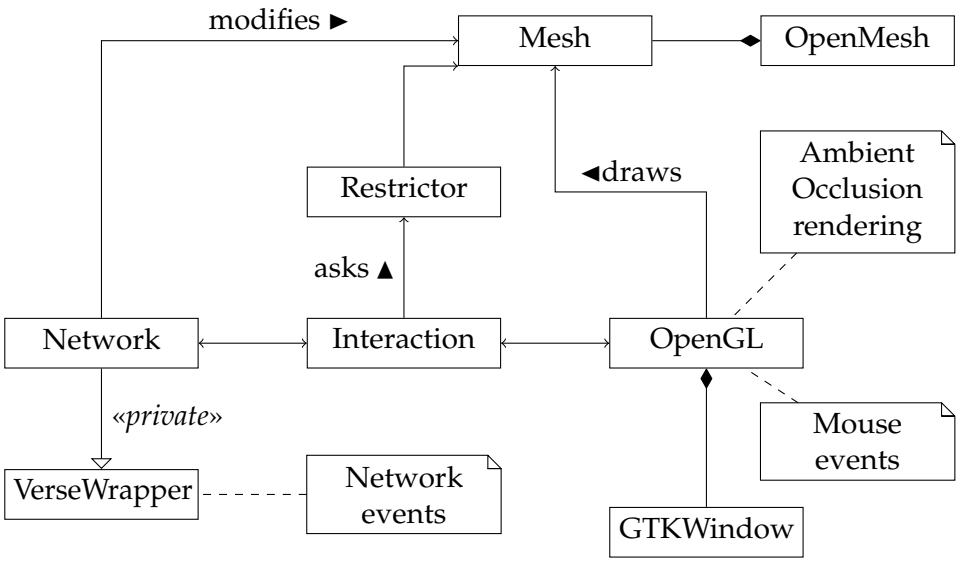


Abbildung 5.1.: Dieses UML-Klassendiagramm veranschaulicht den Aufbau des implementierten Systems. Natürlich sind hier nur die wichtigsten Klassen aufgeführt, so dass die Struktur ersichtlich werden kann.

5.3. Ergebnis

Das Endergebnis der Implementierung ist eine Gesamt-Anwendung, die das einfache Arbeiten mit virtuellen Gegenständen in kleinen Teams ermöglicht (siehe Kapitel 6 und das Video in Anhang D). Dazu wurden die vorher entworfenen Prototypen kombiniert und erweitert, sowie die noch fehlenden Komponenten hinzugefügt. In einigen Details unterscheidet sich die Implementierung jedoch von den in Kapitel 3 gestellten Anforderungen sowie der in Kapitel 4 entworfenen Architektur, da die technische Umsetzung einen wesentlichen Beitrag zu den Erkenntnissen in dieser Arbeit leistete und so spätere Design- und Anforderungsänderungen nicht mehr berücksichtigt werden konnten. Diese Unterschiede werden in den folgenden Unterabschnitten herausgearbeitet.

5.3.1. Server und Client

Da bei der Implementierung vollkommen auf die Verwendung des *Verse*-Protokolls und dessen mitgelieferten *Servers* gesetzt wurde (siehe Abschnitt 5.1.1), mussten alle Überprüfungen der Gültigkeit (vgl. Abschnitt 4.3.6) auf *Client*-Seite implementiert werden. Bei einer eigenen Implementation des *Servers* könnte der Quelltext dank des objektorientierten Aufbaus (vgl. Abbildung 5.1) aber wiederverwendet werden.

5.3.2. Detailhierarchie

Es ist unter Verwendung von *Verse* und *OpenMesh* nahezu unmöglich eine praktikable Lösung für die Umsetzung einer Detailhierarchie zu finden. Hierzu muss die zugrundeliegende Repräsentation und der Informationsaustausch stark erweitert werden. Mit einer Kombination aus Detailhierarchie und Mehrfachauswahl wäre es sogar möglich, benutzungsfreundlich beliebige Abweichungen von der Standard-Positionierung des *Loop*-Schemas einzustellen (siehe auch Seite 7.2).

Aus der Notwendigkeit heraus, wenigstens pro Punkt auf der Oberfläche schärfere Kanten hervortreten zu lassen, als sie bei der Standard-*Loop-Subdivision* bei einer regulären *Vertex*-Verteilung entstehen, wurde die *Subdivision*-Implementation von *OpenMesh* um einen »Spitzigkeits«-Faktor erweitert.

Dieser Faktor c liegt im Wertebereich $[0 \ldots 1]$ und kann mit der mittleren Maustaste direkt mainipuliert werden. Die endgültige Position $\mathbf{p}$ jedes *Vertices* bestimmt sich durch lineare Interpolation zwischen der Ursprungsposition $\mathbf{p}_{old}$ und der durch das *Loop*-Schema ermittelten neuen Position $\mathbf{p}_{loop}$:

$$\mathbf{p} \quad = \quad c \cdot \mathbf{p}_{old} + (1 - c) \cdot \mathbf{p}_{loop} \tag{5.6}$$

Der Punkt bleibt beim Drücken der mittleren Maustaste durch Anpassung von c genau unter dem Mauszeiger. So wird die Umgebung des Punktes glatter bzw. spitzer, je nachdem in welche Richtung der Anwender die Bewegung ausführt.

5.3.3. Rotation um einen Punkt

Die im Prototypen in Abschnitt 5.2.4 entwickelte Methode zur Rotation der Kamera wurde aufgrund früher Benutzungstests stark verändert. Denn bei einer Kamera-Navigation ist es sehr viel besser, den Horizont immer waagerecht zu halten. Anstatt die Mauskoordinaten auf eine Kugel zu projizieren und damit eine Drehung um eine beliebige Achse zu berechnen, ist es zielführender, die Szene erst um die x-Achse und dann die y-Achse des Weltkoordinatensystems zu drehen. Der Winkel für die x-Achse entspricht der Differenz zwischen den x Werten von $\mathbf{m}_{current}$ und $\mathbf{m}_{old}$. Der Winkel der y-Achse berechnet sich analog.

5.3.4. Benutzungsschnittstelle

Über eine *GTK*-GUI (siehe Abbildung 5.2) können neue Gegenstände erstellt und alte weiter bearbeitet werden. Über ein Suchfeld ist es möglich, die Menge der in einer Liste aufgeführten Gegenstände einzuschränken. Für die Aufgabenstellung war eine Auflistung und Suche nach Namen hinreichend, die verwendete Methode lässt sich aber ohne weiteres auch auf Kategorien und Meta-Daten jeglicher Art erweitern.

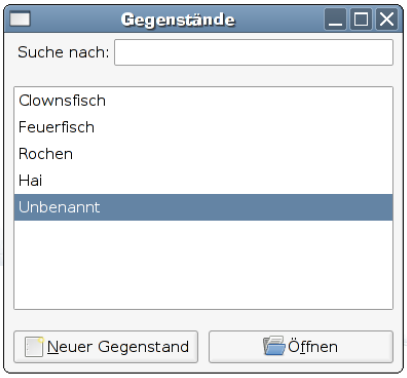


Abbildung 5.2.: Zur Verwaltung der virtuellen Gegenstände wird dem Anwender eine einfache Oberfläche mit einem Suchfeld und einer Liste von Namen angezeigt. Es besteht sowohl die Möglichkeit neue Gegenstände zu erstellen, als auch nach Auswahl eines Namens in der Liste den zugehörigen Gegenstand zu öffnen. Durch einen Klick auf den Namen kann dieser direkt verändert werden.

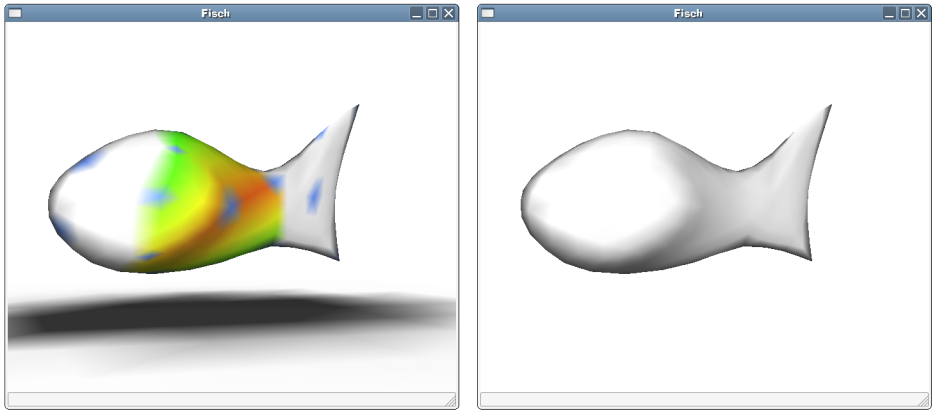


Abbildung 5.3.: Jeder Gegenstand wird in einem eigenen Fenster dargestellt und kann über die blauen Punkte verformt werden. Die Farben visualisieren sehr große sowie kleine Abstände zwischen Punkten. Wird die Maus eine Zeit lang nicht bewegt, verschwinden die Interaktionshilfen.

Die Darstellung des aktuell geöffneten virtuellen Gegenstands erfolgt in einem zweiten Fenster (siehe Abbildung 5.3). Da alle Interaktionen direkt mit der Maus durchgeführt werden, besitzt dieses Fenster keinerlei Buttons und Menus. Der Inhalt wird mithilfe von *OpenGL* visualisiert und lässt sich interaktiv verformen. Bewegt der Anwender für zwei Sekunden nicht die Maus, blendet die Anwendung langsam visuelle Hilfestellungen – wie die Punkte auf der Oberfläche und den Untergrund – aus.

Kapitel 6.

Befragung der Nutzer

> Simplicity is the ultimate sophistication.
>
> *Leonardo Da Vinci*

Essentiell für den Erfolg einer Software-Entwicklung ist es, die der Endanwender miteinzubeziehen. Nur so können Fehler im Konzept identifiziert und neue Lösungswege gefunden werden. Während der gesamten Arbeit wurden daher Tests mit Anwendern durchgeführt. In diesem Kapitel sind einige besonders bemerkenswerte Einflüsse auf die Entstehung der hier präsentierten Lösung zusammengefasst.

Der Großteil der folgenden Abschnitte behandelt jedoch die erzielten Resultate mit der Endanwendung in ihrer vollständig umgesetzten Form (vgl. Kapitel 5). Insgesamt nahmen zwölf Einzelpersonen und zwei Teams à drei Mitgliedern teil. Die Teilnehmer hatten äußerst verschiedene Vorkenntnisse und bekamen alle eine ca. einminütige Erläuterung zum Thema und dem grundlegenden Bedienungskonzept. Während des Umgangs mit der Software wurden sie gebeten, ihre Gedanken laut auszusprechen – gleichzeitig wurden alle Interaktionen von der Software mitprotokolliert. Im Anschluss wurden die Probanden noch zu besonderen Merkmalen der Anwendung befragt.

6.1. Einfache Handhabung

Bei der Beobachtung der Anwender ist aufgefallen, dass das Bedienungskonzept zu Beginn sehr ungewohnt ist. Nutzer, die zuvor schon 3D-Modellierung betrieben hatten, sahen sich mit einer minimalistischen Anzahl von Werkzeugen konfrontiert, die ihre »Freiheit« einschränkte; vollkommen unerfahrene Anwender mussten sich erst einmal mit der Vorstellung des Modellierens im Allgemeinen anfreunden. Wie Abbildung 6.1 veranschaulicht, ist die Einarbeitungszeit trotzdem überaus gering. Die Benutzer lassen sich schnell auf die Metaphern der direkten Manipulation ein und nehmen die Automatismen als inhärente Eigenschaften wahr. Nach dem Erlernen der Anwendungsfunktionalität, dauerte es noch bis zu

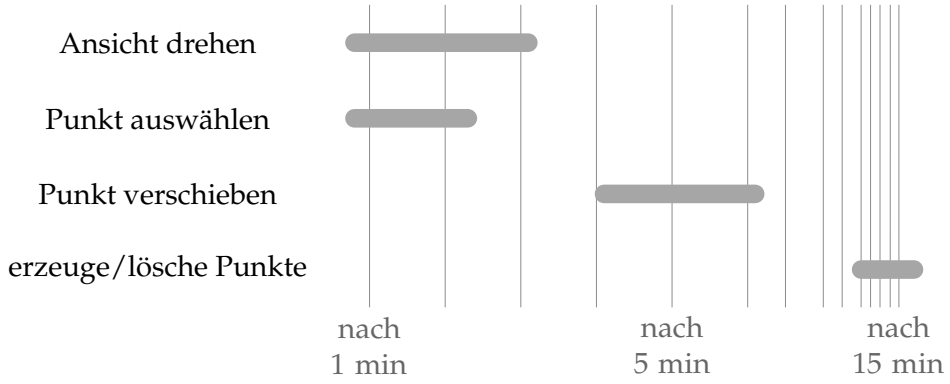


Abbildung 6.1.: Ungefähre Zeitspannen, in denen eine Interaktionsmöglichkeit von den Anwendern gezielt eingesetzt wird. Die sinnvolle und zielgerichtete Kombination der Techniken dauert einige Minuten länger. Diese Werte entstanden durch intensives Beobachten der Probanden während der Arbeit.

einer halben Stunde, bis auch zielgerichtet das Gesamtaussehen des Gegenstands in eine gewünschte Form gebracht werden konnte.

6.1.1. Navigation

Während der ersten Benutzungstests war nahezu bei jeder Testperson die erste aufkommende Frage »Wie bewege ich mich?« Daraufhin wurden die Automatismen für Zoom und Bildmittelpunkt (siehe Abschnitt 4.3.8) entwickelt. Der Anwender brauchte nur noch zwischen »Betrachten« und »Bearbeiten« per GUI zu wechseln, um die Ansicht zu drehen bzw. den virtuellen Gegenstand zu verformen. Doch viele Probanden taten sich mit diesem manuellen Moduswechsel immer noch schwer. Als Konsequenz wurden beide Modi vereint: Wird ein Punkt angeklickt und »festgehalten«, lässt er sich bewegen. Wird auf den Hintergrund oder eine nicht durch einen Punkt besetzte Region auf dem Gegenstand geklickt, kann die Szene gedreht werden. Diese Interaktionsmetapher wurde sehr schnell verstanden.

Alle Testpersonen konnten gezielt nach ein bis drei Minuten die Ansicht rotieren. Die automatischen Kamerafahrten riefen jedoch oft bei deren erstmaliger Beobachtung Erstaunen hervor. Diese Ungewohntheit verflog allerdings nach ein paar Minuten. Während Nutzer mit Erfahrung im Modellierungs- und Grafikbereich gern etwas mehr Freiheit in der Kameraführung hätten, konnten unerfahrene Anwender keinen besonderen Kontrollverlust benennen. Dies liegt vermutlich daran, dass sie noch nie eine Kamera frei gesteuert hatten. Trotz dieser Einschränkung waren aber alle Probanden in der Lage, nach vier bis sechs Minuten gezielt Punkte anzusteuern und die Ansicht in die gewünschte Lage zu rotieren.

Zur Orientierungshilfe wurde ein Untergrund visualisiert. Dadurch wird dem

Anwender klar, wo oben und unten ist. In manchen Situationen erwies sich die Darstellung jedoch als hinderlich. Schob ein Proband einen Punkt in den Untergrund hinein, befand sich dieser Teil des Gegenstands im Boden. Von Oben gesehen war er dadurch wie abgeschnitten; erst beim Rotieren der Ansicht in den Untergrund wurde der untere Teil wieder sichtbar. Deshalb wäre es besser, den Boden immer unterhalb des Gegenstandes zu positionieren.

Bei den temperamentvolleren Teilnehmern kam es manchmal zu unerwünschten Punktverschiebungen, wenn ohne Bedacht die Maus geklickt und gezogen wurde, um schnell die Ansicht zu ändern. Um solche Fehlannahmen über die Intention des Benutzers auszumerzen bedarf es leider einer weitaus aufwändigeren Verhaltensanalyse während der Interaktion. Bis dahin könnte das Problem durch eine *Undo*-Funktion gemildert werden.

6.1.2. Verformungen

Um die Oberfläche des virtuellen Gegenstandes zielgerichtet verformen zu können, musste eine intuitiv verständliche und mit einer normalen Computermaus durchführbare Interaktionsmetapher gefunden werden. Damit schon im frühen Stadium der Arbeit erste Erkenntnisse hierzu gesammelt werden konnten, bot sich eine Papier-Umfrage an. In sieben Iterationen entstand ein A4-Bogen (siehe Anhang A.1), auf dem – neben einigen Angaben zum persönlichen Kenntnisstand – vier mal der Schatten eines verschobenen Punktes eingezeichnet werden sollte. Die Ergebnis-Tabelle der 24 vollständig abgegebenen Bögen kann in Anhang A.2 nachgeschlagen werden.

Aus der Umfrage ließ als einzige signifikant hervortretende Methode zur Punktverschiebung der konstanten Abstand des Punktes zur Bildebene folgern. Dabei streuten sich die Antworten ansonsten sehr stark, so dass diese Aussage nur als Anhaltspunkt dienen konnte. Eine Umsetzung der Technik (vgl. Abschnitt 4.1.2) zeigte anschließend jedoch deutlich, dass diese Art der Manipulation sehr schnell erlernt wird. Bis ein Anwender die Punkte auf der Oberfläche gezielt an eine neue Position verschieben kann, vergehen nur zwischen vier und sieben Minuten (vgl. Abbildung 6.1). Es kommt aber immer wieder vor, dass ein Punkt nicht genau an die gewünschten 3D-Position verschoben werden kann und durch einige Ansichtswechsel nachjustiert werden muss.

Ganz anders sieht es jedoch bei dem Erzeugen und Löschen von Punkten aus. Technisch gesehen wird ein *Edge split/Edge merge* immer dann durchgeführt, wenn eine Kante zu lang/kurz wird und der Anwender die rechte Maustaste verwendet. Dass neue Punkte hinzukommen bzw. verschwinden, ist auch vielen Anwendern nach sehr kurzer Zeit klar, doch durch welches Verhalten sie dies ganz genau steuern können, bleibt lange unverstanden. Erst nach einer für diese Anwendung langen Zeit – zwischen zehn und dreißig Minuten – entwickelt sich ein Gefühl für das kausale Verhalten der Punkte. Durch eine Farbkodierung von zu großen und kleinen Abständen wie in Abbildung 5.3 auf Seite 57 ließ sich zwar die

in Abbildung 6.1 veranschaulichte Einarbeitungszeit erreichen, jedoch hat sich gezeigt, dass eine weitergehende Verbesserung der »kausalen Automatik« (vgl. Seite 27) dringend erforderlich ist. Der Anwender bekommt zwar ein Gespür für das Verhalten der Software, aber noch keine intuitiv verständliche Rückkoppelung.

Die Performanz der Anwendung reichte für kleine Gegenstände mit wenigen Punkten vollkommen aus. Versuchten sich Probanden jedoch an Gegenständen, für die mehr als etwa dreißig Punkte benötigt wurden, war ein flüssiges Arbeiten nicht mehr möglich (vgl. Abschnitt 5.2.6). Bekam der Anwender keine direkte Information über die Auswirkungen seiner Mausbewegung, konnte nicht mehr nachjustiert werden und es wurde sehr schwer, die Verformungen akkurat anzuwenden. Eine Verbesserung der Performanz ist daher unbedingt notwendig.[1]

6.2. Kooperatives Arbeiten

Eines der großen Ziele dieser Arbeit ist es, einen einfach zu handhabenden Ansatz für kooperatives Modellieren zu finden. Der Prüfstein für die in Kapitel 4 vorgestellte Architektur besteht natürlich in einem Gruppentest. Daher wurde zwei Teams mit jeweils drei Teilnehmern aufgetragen, kooperativ einige Gegenstände für eine Unterwasserwelt zu modellieren. Jeder saß dazu an einem eigenen Rechner in einem gemeinsamen Raum. Die Gruppe *A* bestand aus drei Personen, denen jegliches Wissen über Computergrafik fehlte. Gruppe *B* hatte sowohl Informatik-Wissen in diesem Bereich, als auch Erfahrung mit professioneller 3D-Software. Die erste Gruppe konnte lediglich fünfzehn Minuten lang beobachtet werden; die zweite Gruppe hat hingegen eine volle Stunde gearbeitet. Alle Gruppenteilnehmer hatten zuvor einen Einzeltest (siehe Abschnitt 6.3) durchgeführt und besaßen so dass nötige Wissen für eine zielgerichtete Verformung der Gegenstände.

6.2.1. Koordination und Kommunikation

Technik kann zwar die Kommunikation in der Gruppe erleichtern, aber niemals vollständig automatisch ablaufen lassen. Für eine Kooperation ist daher immer auch eine Koordination zwischen den einzelnen Mitgliedern notwendig. Diese war in beiden Gruppen zu Beginn sehr unterschiedlich. Während die unerfahrenen Anwender (Gruppe *A*) direkt anfingen, gemeinsam einen Fisch nach dem anderen zu erstellen und sich per Zuruf verständigten, beratschlagten sich die Mitglieder aus Gruppe *B* zunächst über das generelle Vorgehen. Nachdem geklärt war, welche Arten von Fischen und Gegenständen modelliert werden sollten, begaben sie sich ganz ähnlich der Gruppe *A* an die Arbeit.

[1] Für die Simulation der Selbstverschattung (siehe 4.3.3) ist bei weitem der größte Rechenaufwand nötig. Eine Implementation auf der GPU könnte das Problem höchstwahrscheinlich vollkommen lösen.

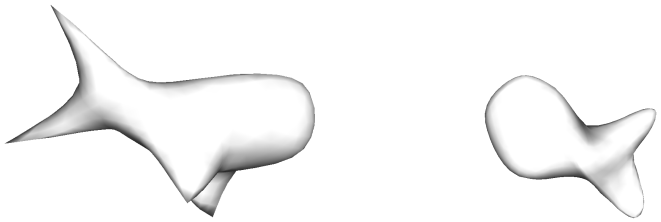


Abbildung 6.2.: Der linke Fisch entstand in ca. sieben Minuten kooperativ in Team *B*. Der Rechte wurde von Team *A* in ca. sechs Minuten modelliert.

Beide Teams begannen mit dem gleichzeitigen Modellieren an einem einzigen Fisch (siehe Abbildung 6.2). Die Kommunikation erfolgte ausschließlich über Zuruf: »Ich kümmere mich mal um die Flossen«. In Team *A* kam es dabei anfangs zu Verwirrung, da Aussagen wie »Rechts wird der Kopf« natürlich der falschen Voraussetzung unterliegen, dass alle die selbe Sicht auf den Gegenstand besitzen. Um herauszufinden wo jemand anderes arbeitet, beobachteten die Probanden die Oberfläche und forderten die anderen Teammitglieder auf, eine Verformung vorzunehmen, so dass sie eine visuelle Rückkopplung bekamen. Diese Methodik funktionierte für die benötigten Absprachen hinreichend gut, könnte aber durch eine bessere Visualisierung von Benutzer-Aktivitäten noch stark verbessert werden.

6.2.2. Kollisionen

Natürlich kam es trotz der Absprachen bei Zeiten zu Kollisionen zwischen den Verformungen, die verschiedene Gruppenmitglieder durchführen wollten. In diesem Fall lässt die Software denjenigen, der mit dem Verschieben begonnen hat, weiter arbeiten; für alle anderen wird der betroffene Bereich der Oberfläche gesperrt (siehe auch Abschnitt 4.3.6). Dies wurde von allen Probanden als wenig hinderlich angesehen und beeinflusste das kontinuierliche Modellieren nur marginal. Es kam jedoch der Wunsch auf, dass eine Sperrung eines Bereiches visuell besser mitgeteilt werden sollte. Dank der feinen Granularität auf der Kollisionen behandelt werden, ist eine Blockierung der Arbeit aber eher die Ausnahme, und wird erst lässtig, wenn auf jeden Benutzer nur etwa fünf manipulierbare Punkte kommen.

6.2.3. Motivation

Eine sich voll und ganz bestätigende Hoffnung dieser Arbeit bestand in der erhöhten Motivation beim Modellierungs-Prozess. In beiden Gruppen war eine starke Euphorie beim ersten gemeinsamen Verformen des ersten Gegenstands zu erkennen: »Alle drei zusammen . . . sieht schon gut aus!«, »Das ist total lustig!«

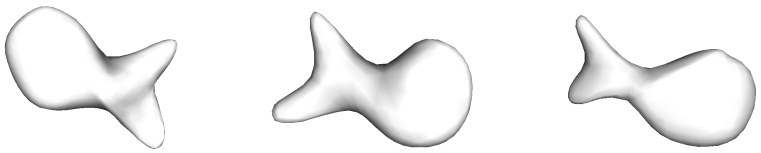


Abbildung 6.3.: Diese drei Fische modellierte das Team *A* gemeinsam und aufeinanderfolgend in insgesamt fünfzehn Minuten.

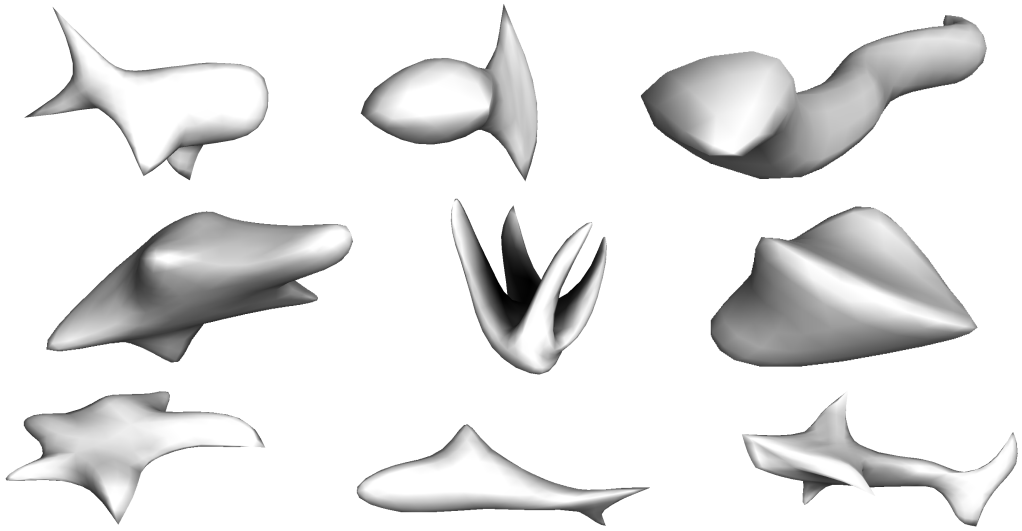


Abbildung 6.4.: Diese Gegenstände modellierte das Team *B* innerhalb einer Stunde in kooperativer Zusammenarbeit. In Leserichtung: Fisch1, Fisch2, Aal, Schildkröte, Seetang, Stein, Seestern, Delfin und Haifisch.

Etwas später schien die Kooperation bereits selbstverständlich, verstärkte aber weiterhin den Ergeiz und ließ die Arbeit leichter erscheinen: »Es geht einfacher, wenn ihr mitmacht.« »Ja, jeder sieht eine andere Beule!« Diese und ähnliche Aussagen lassen vermuten, dass sich die Lernphase in der Gruppe möglicherweise noch schneller und effektiver durchführen lässt. Ein diesbezüglicher Benutzungstest wäre jedoch noch durchzuführen.

6.2.4. Kreativität

Während Team *A* nur fünfzehn Minuten Zeit für die Modellierung von drei Fischen zur Verfügung hatte (vgl. Abbildung 6.3), konnte Team *B* der Kreativität freien Lauf lassen. Innerhalb von einer Stunde produzierten sie eine Vielzahl von aufwendigen Gegenständen (vgl. Abbildung 6.4), deren Komplexität die Performanz der implementierten Anwendung stark einbrechen ließ. Dies wurde von

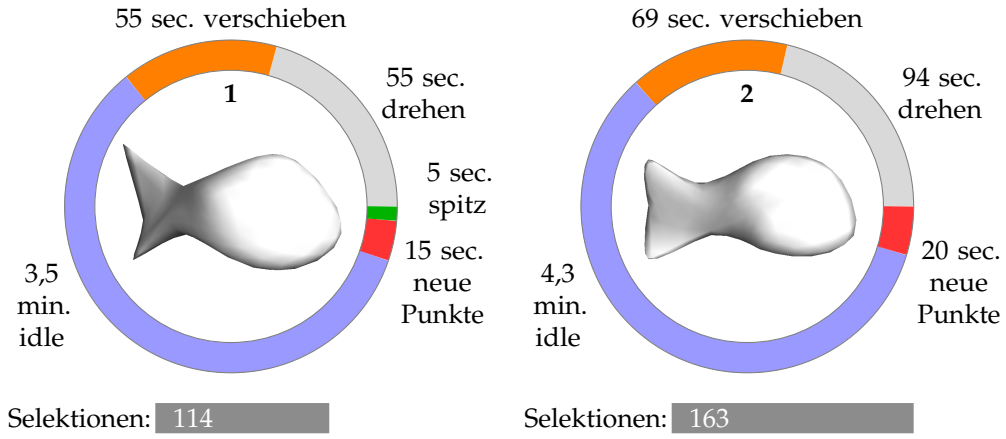


Abbildung 6.5.: Beim ersten Kontakt mit der Anwendung wurden diese beiden Fische von einem Probanden in Einzelarbeit direkt aufeinanderfolgend modelliert. Der Benutzer hatte vorher schon etwas Erfahrung mit professioneller 3D-Software.

den Probanden als der am stärksten kreativitätshemmende Faktor identifiziert.

Team *B* nutzte den Vorteil der Gruppenarbeit auch zur Aufteilung von kreativer Leistung. Während zwei Mitglieder modellierten, widmete sich die dritte Person eine Zeit lang dem Erdenken neuer Gegenstände und legte diese mit Namen an, so dass später direkt mit der Modellierung begonnen werden konnte.

6.3. Zielgerichtete Modellierung

Um herauszufinden, ob relativ unerfahrene Anwender nach kurzer Einarbeitungszeit zielgerichtet Formen und Gestalten modellieren können, wurden ein Dutzend Einzeltests durchgeführt. Dabei protokollierte die Software die Aktionen der Benutzer mit. Im folgenden werden die Ergebnisse dieser Tests zusammengefasst.

In den Diagrammen 6.5 bis 6.7 und im Anhang C sind die jeweiligen Aktionen visualisiert, mit denen die in der Mitte abgebildete Form erstellt wurde. Jeder Gegenstand ist von Grund auf neu entstanden. Der besseren Vergleichbarkeit halber wurden die Probanden darum gebeten, immer wieder eine einfache Fischform zu erstellen.

Die Benutzertests zeigen, dass fast jeder Proband nach spätestens einer Stunde in der Lage ist, zielgerichtet eine gewünschte Form zu erstellen. Zwei Probanden waren jedoch nach eigenen Angaben besonders »lustlos« und gaben nach einigen Minuten auf. Sie wiesen darauf hin, dass sie auch sonst nicht viel Motivation aufbingen können, wenn es um das kreative Erstellen von Formen geht.

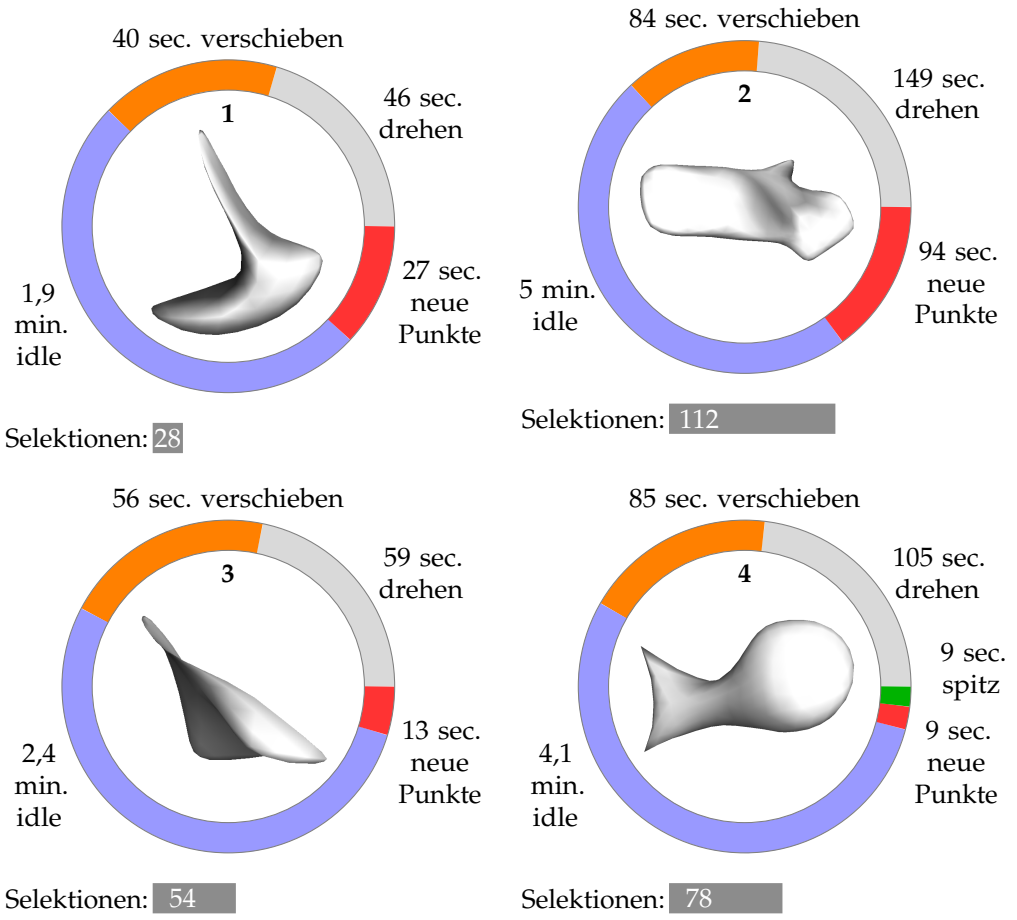


Abbildung 6.6.: Beim ersten Kontakt mit der Anwendung wurden diese vier Gegenstände von einer Probandin in Einzelarbeit direkt aufeinanderfolgend modelliert. Jedes Mal war ein Fisch das Ziel. Es bestand vorher etwas Erfahrung mit professioneller 3D-Software.

Unter den sechs Testpersonen mit Vorkenntnissen in Computergrafik und Grafik-Software waren teilweise herausragend schnelle Erfolge zu erkennen (z. B. Abbildung 6.5). Es ist jedoch keinesfalls so, das diese Teilgruppe signifikante Vorteile besitzt (vgl. Abbildung 6.6, 6.7 und Anhang C).

Aus den Diagrammen lässt sich ersehen, dass ein Großteil der Zeit untätig (engl. *idle*) verbracht wird. Als untätig ist hier die gesamte Zeitspanne definiert, während der keine Maustaste gedrückt wurde. Eine Aufteilung in Mausstillstand und freie Mausbewegung wäre in einem Folgetest zu erwägen; aus der Beobachtung der Probanden kann aber vermutet werden, dass die Maus die meiste Zeit in Bewegung ist und kaum richtige Denkpausen eingelegt werden müssen. Desweiteren ist das Verhältnis zwischen der Verwendung von linker und rechter Maustaste sehr interessant. Ersteres resultiert in einer Punkt-Verschiebung (siehe Abschnitt 4.1.2 auf Seite 35); zweiteres in dem Verschieben eines Punktes mit gleichzeitig automatischem Löschen und Erzeugen von weiteren Punkten (siehe Abschnitt 4.1.2 auf Seite 35). Zwei Testpersonen (siehe Anhang C) verwendeten zu Beginn fast ausschließlich die zweite Funktionalität. Alle anderen nutzten erfolgreich beide Techniken im Wechsel. Es sollte daher darüber nachgedacht werden, ob sich nicht beide Aktionen in einer einzigen Maustaste vereinen lassen[2]. Dazu wären weitere umfangreiche Benutzungstests unabdingbar.

6.4. Virtuelle Gegenstände

6.4.1. Technischer Anspruch

Ein besonderer Vorteil der Lösung, welche in dieser Arbeit präsentiert wird, besteht in den nahezu beliebig erweiterbaren Automatismen, mit denen der Anwender nur indirekt in Berührung kommt. So lässt sich ohne bewusste Handlungen eine technische Qualität (siehe Abbildung 6.8) der zugrundeliegenden Repräsentation (siehe Abschnitt 2.1.2) erreichen, die mit anderen Software-Produkten (vgl. Abschnitt 2.5) nur nach sehr langer Einarbeitungszeit und hohem Kenntnisstand in der Computergrafik möglich ist.

Gegenstände, bei denen sich schon in der Visualisierung der Oberfläche eine Dissonanz wiederfindet, haben jedoch auch in technischer Hinsicht eine niedrigere Qualität – wie es beispielhaft in Abbildung 6.9 veranschaulicht wird. Während der Benutzungstests wurde aber klar, dass alle Anwender glatte und angenehme Formen anstrebten, bei denen dann automatisch auch die technische Verwendbarkeit sichergestellt ist. Eine verbesserte Unterstützung für harmonische Formen könnte die Benutzung jedoch weiter erleichtern.

[2] Evtl. ließen sich langsame Mausbewegungen als normale Punktverschiebung interpretieren und erst ab einer bestimmten Geschwindigkeit entstehen neue Punkte

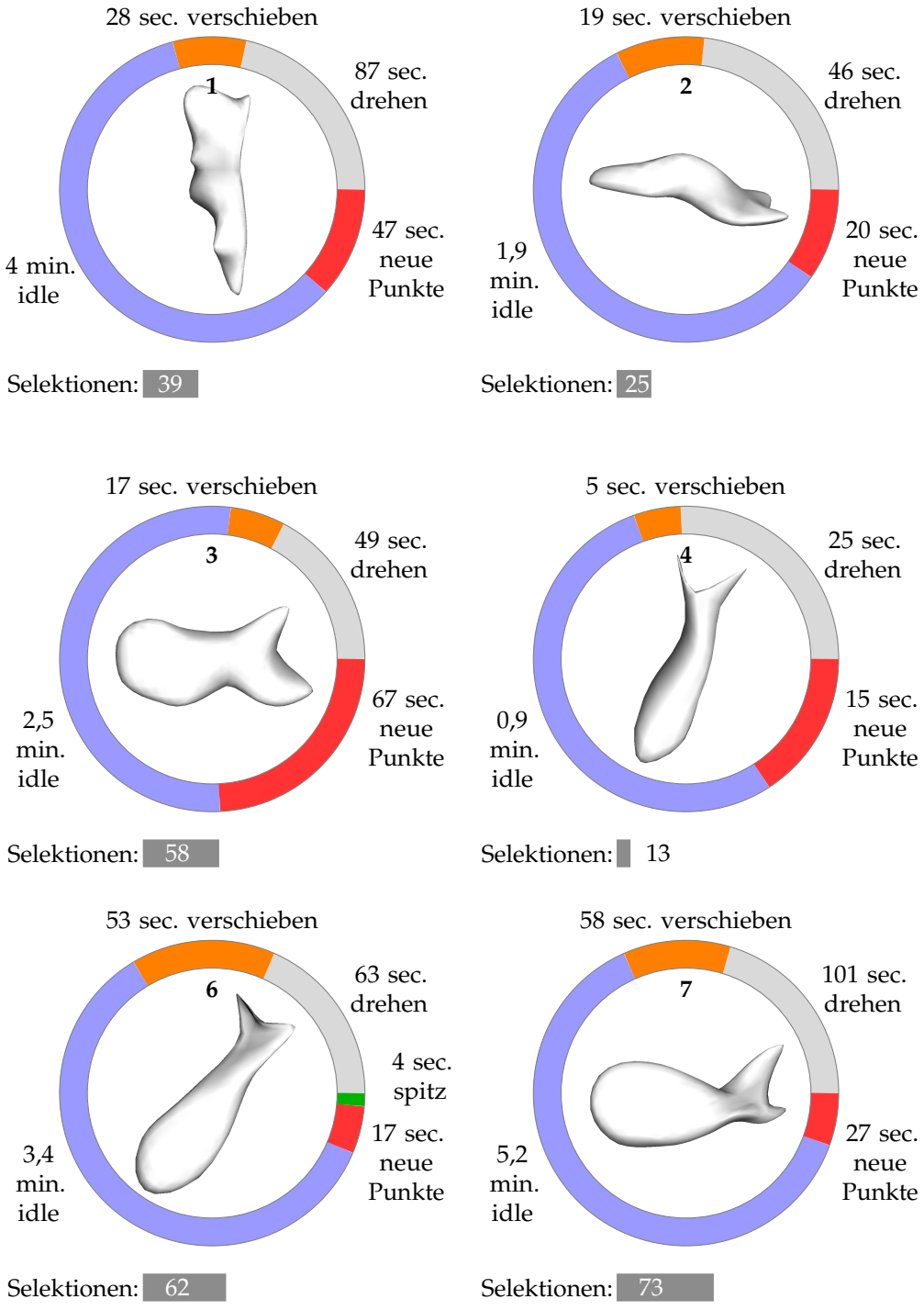


Abbildung 6.7.: Ohne jegliche Vorerfahrung mit 3D-Software und Computergrafik wurden sieben Fische von einer Probandin in Einzelarbeit direkt aufeinanderfolgend modelliert. Der fünfte Fisch in der Reihe fehlt leider aufgrund eines Software-Fehlers.

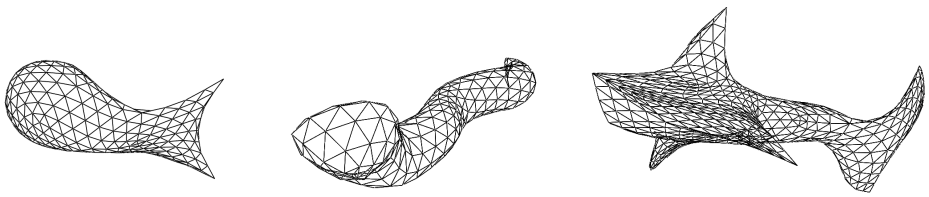


Abbildung 6.8.: Diese Drahtgitter-Darstellungen veranschaulichen die technisch nützliche Struktur des *Meshes*. Während der Konstruktion mußten die Probanden auf keinerlei Kriterien achten, sondern nur den kausalen Zusammenhängen der Anwendung folgen.

Abbildung 6.9.: Ein etwas verzerrter Gegenstand hat auch eine technisch mindere Qualität.

6.4.2. Ästhetischer Anspruch

Im Alltag werden laufend Aussagen über die Schönheit von Dingen getroffen. Da ästhetische Urteile nur subjektive Allgemeingültigkeit besitzen, kann hier keine strikt objektive Aussage getroffen werden [Kan13]. In Abbildung 6.10 werden deshalb Beispiele für besonders schöne Gestalten wiedergegeben. Ihre ästhetische Qualität ist ein reflexives Geschmacksurteil des Autors und es wird bloß erwartet, dass der Leser hiermit übereinstimmt.

Es ist noch hinzuzufügen, dass die entwickelte Software allein als Werkzeug für den künstlerischen Menschen gedacht ist und nicht auf Knopfdruck schöne Gegenstände produziert. Sie ist deshalb nur Mittel einer handelnden Person und ersetzt keinesfalls künstlerisches Talent.

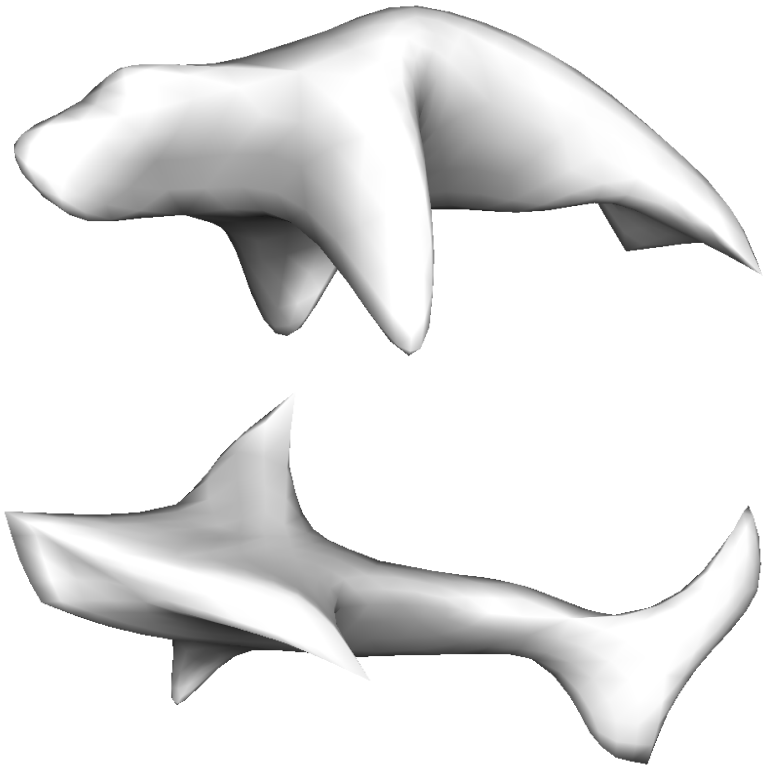


Abbildung 6.10.: Auf die beiden Gegenstände im Diagramm 6.5 folgend, entstand dieser Seehund in Einzelarbeit in etwa zehn Minuten. Der Haifisch ist ein Teilprodukt des Gruppentests mit Team *B* (vgl. Abbildung 6.4); die Herstellungszeit betrag hier ca. 15 Minuten. Beides sind hervorragende Beispiele für das ästhetische Potential der entwickelten Software.

Kapitel 7.

Schlussfolgerungen

> Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest.
>
> *Isaac Asimov*

In dieser Arbeit wurde eine Software entworfen, implementiert und getestet, mit der unerfahrene Anwender in kurzer Zeit gemeinsam virtuelle Gegenstände modellieren können. Nach umfangreicher Analyse stand fest, dass die beste Lösung in einer *Client-Server*-Architektur liegt. Die Wahl der Geometrie-Repräsentation fiel auf ein geschlossenes Dreiecks-*Mesh*, welches durch eine *Loop-Subdivision* in eine glatte Oberfläche überführt wird. Einzelne virtuelle Gegenstände können von Anwendern erstellt, ausgewählt, umbenannt und gemeinsam verformt werden. Letzteres geschieht durch eine direkte Manipulation von Kontrollpunkten auf der Oberfläche. Die Änderungen sind in Echtzeit für alle Teilnehmer sichtbar. Für die interaktive Darstellung wird eine Selbstverschattung simuliert, wodurch die Gestalt des visualisierten Gegenstandes besser wahrnehmbar wird.

7.1. Bewertung

Das gesetzte Ziel dieser Diplomarbeit ist vollständig erreicht worden. Es entstand nicht nur ein umfangreiches Konzept, mit dem die gestellte Problematik aufgelöst werden kann; vielmehr konnten die definierten Anforderungen und Architektur-Entwürfe auch in einer stabilen Software-Implementation umgesetzt werden. So wurde es in dieser Arbeit auch möglich, aussagekräftige Benutzungstests zu präsentieren. Diese untermauern die Eignung des entwickelten Systems und sind sehr hilfreich für weitere Forschung in diesem Bereich.

Mit der momentanen Implementation lassen sich jedoch nur eine kleine Anzahl von Einzelgegenständen geringer Komplexität verwalten und bearbeiten. Diese

Einschränkungen sind jedoch mit den in Abschnitt 7.2 beschriebenen Erweiterungen zu überwinden. Dabei entsteht in keinerlei Hinsicht ein Wiederspruch zu dem in dieser Arbeit entworfenen Konzept. Deshalb sind die Grenzen dieser Modellierungstechnik vor allem bei Gegenständen zu suchen, die auch ein professioneller Künstler nicht aus Ton oder ähnlich formbarem Material herstellen könnte. Für die Teamgröße scheint es – abgesehen von Rechenleistung – keine erkennbare Obergrenze zu geben, solange ausreichend große oder viele Gegenstände zur Verfügung stehen, um ein angenehm flüssiges Arbeiten sicher zu stellen (vgl. Abschnitt 6.2.2).

Im Gegensatz zu kommerziellen Software-Produkten (siehe Abschnitt 2.5 auf Seite 19) stand in dieser Arbeit das einfache kooperative Modellieren eindeutig an erster Stelle. Für eine erfolgreiche kooperative Arbeit mit virtuellen Objekten ist eine persistente, synchrone und eindeutige Datenhaltung auf einem *Server* unabdingbar. Es ist wichtig, dass diese zusammen mit einer Semantik durchsuch- und abfragbar ist. Die Benutzung selbst muss möglichst einfach in der Handhabung sein, damit auch Nicht-Spezialisten ihre kreative Leistung einbringen können: »*good training is not a substitute for good system design*« [Bor96]. Diese Arbeit zeigt, dass auch im Bereich komplexer digitaler Medien durch das sorgfältige Umsetzen direkter Manipulations-Metaphern in Kombination mit starkem Kontext-Bezug eine überaus kurze Einarbeitungszeit erzielt werden kann. Es hat sich herausgestellt, dass diese einfache Interaktion mit nahezu beliebig komplizierten Automatismen verbunden werden kann, um z. B. eine Mehrbenutzerplattform anzubieten oder technische Anforderungen einzuhalten. Der Schlüssel hierzu ist die kausale Verknüpfung von Interaktion und der im Hintergrund laufenden Automatik.

Diese Arbeit eröffnet somit durch eine Kombination von Methoden aus der Informatik und besonders der Computergrafik sehr interessante Perspektiven, die im Ausblick auf der nächsten Seite weiter ausgeführt werden.

7.2. Erweiterungen

Natürlich ist das vorgestellte Konzept noch sinnvoll erweiter- und verbesserbar. Wie schon in [Bri] vorgeschlagen wurde, sollten Änderungswünsche der Punktposition nicht durch feste Werte, sondern durch Kräfte und Beschleunigung abgebildet werden. Dadurch könnten sich Interaktionen mehrerer Benutzer akkumulieren und so eine Kooperation ohne Blockierungen ermöglichen (vgl. Abschnitt 6.2.2).

Die automatische Kameraführung erfüllt in der getesteten Implementation zwar ihren Zweck, muss jedoch für einen universellen Einsatz auf jeden Fall genauer untersucht, verbessert und erweitert werden. In Anbetracht der bisher erreichten Nutzbarkeit (siehe Abschnitt 6.1.1 auf Seite 60) scheint eine Weiterführung dieses Ansatzes Erfolgversprechend.

Viele Anwender wünschten sich während der Benutzungstests, dass sie an beliebiger Stelle auf der Oberfläche eine Verformung durchführen könnten. Dies wäre in

Kombination mit einer Detailhierarchie sicherlich realisierbar und würde den Detailgrad der Gegenstände enorm verbessern. Jedoch ist es bei dieser Erweiterung besonders wichtig, auf eine einfache Handhabung zu achten. Es müssen intuitive Metaphern für die direkte Manipulation der Oberfläche auf verschiedenen Detailstufen geschaffen werden. Dafür ist es unter anderem erforderlich, die Auswahl von beliebigen Regionen auf der Oberfläche sehr flexibel und unkompliziert zu gestalten.

Die prototypische Implementation zur Veränderung des topologischen Geschlechts (siehe Abschnitt 5.2.7) sollte unbedingt in die Software mit eingebaut werden. Dadurch wäre es möglich, viele weitere Arten von Gegenständen zu modellieren, ohne dem Benutzer weitere Werkzeuge oder Handlungsweisen aufzubürden.

Um wirklich sämtliche Vorteile einer persistenten, synchronen und eindeutigen Repräsentation der Daten auf einem *Server* zu erhalten, muss die gesamte Entstehungsgeschichte der Objekte jederzeit von Anwendern genutzt werden können. Sie müssen in den Änderungen navigieren können und an beliebiger Stelle ihre Arbeit fortsetzen dürfen. Es wäre auch zu überlegen, wie sich nur Teile eines Gegenstandes in einen früheren Zustand zurückversetzen ließen.

7.3. Ausblick

Das hier präsentierte Konzept zur Entwicklung eines einfachen, kooperativen Modellierungswerkzeugs bietet einen hervorragenden Startpunkt für andere digitale Inhalte. Trotz der oben genannten Erweiterungen lässt sich mit dieser Modellierungstechnik nur eine bestimmte Klasse von virtuellen Gegenständen effektiv konstruieren. Beispielsweise gibt es sicherlich bessere Methoden um Kleidungsstücke oder Pflanzen zu erstellen. Desweiteren wäre eine Ausweitung auf Materialeigenschaften, Animationen und das Arrangement der Objekte innerhalb eines virtuellen Szenarios erstrebenswert. Eine umfangreiche semantische Suche und Verwaltung der Inhalte ist dabei ebenso notwenig wie eine praktikable Benutzer- und Gruppenverwaltung. Für ein wirklich weltweit kollaborativ und kooperativ einsetzbares Werkzeug müsste außerdem die Kommunikation zwischen Personen an unterschiedlichen Orten und zu unterschiedlichen Zeiten hervorragend unterstützt werden.

Bei den genannten Ergänzungen muss aber tunlichst auf eine konsequente Umsetzung der hier beschriebenen Schlussfolgerungen geachtet werden. Ansonsten könnte – wie bei vielen anderen Anwendungen – trotz anfänglich guter Idee die Ausweitung des Konzepts in einer langen Einarbeitungszeit und komplizierter Datenorganisation enden. Metaphorisch lässt sich die Problematik folgendermaßen ausdrücken: Der Kugelschreiber ist ein wundervoll einfaches Werkzeug. Es gibt nur einige, direkt einleuchtende Funktionen, die jeder im Handumdrehen erlernen kann. Auch die gemeinsame Arbeit auf einem Blatt Papier gestaltet sich

intuitiv und ist schnell verstanden. Im Gegensatz dazu ist die Aquarellmalerei eine hochkomplexe Angelegenheit. Es gibt verschiedene Farben, Pinsel, Schwämmchen und Papiere. Das Wasser verläuft ineinander und trocknet schnell aus. An ein gemeinsames Aquarell-Kunstwerk ist kaum zu denken. Die Aufgabe weiterer Forschung wäre daher, einen Kugelschreiber zu erfinden, mit dem Aquarelle erstellt werden können.

Was in der Realität aussichtslos erscheint, ist in der virtuellen Realität durchaus erreichbar.

Literaturverzeichnis

[AWC04] A. Angelidis, G. Wyvill, and M.P. Cani.
Sweepers: swept user-defined tools for modeling by deformation.
Shape Modeling Applications, 2004. Proceedings, pages 63–73, 2004.

[Bal98] Helmut Balzert.
Lehrbuch der Software-Technik.
Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998.

[Bli82] J.F. Blinn.
A Generalization of Algebraic Surface Drawing.
ACM Transactions on Graphics (TOG), 1(3):235–256, 1982.

[Blo88] J. Bloomenthal.
Polygonization of implicit surfaces.
Computer Aided Geometric Design, 5(4):341–355, 1988.

[Blo97] J. Bloomenthal.
Introduction to Implicit Surfaces.
Morgan Kaufmann, 1997.

[BM00] A. Bouazza and P. Molli.
Unifying coupled and uncoupled collaborative work in virtual teams.
ACM CSCW2000 workshop on collaborative editing systems, Philadelphia, Pennsylvania, USA, December, 2000.

[Bor96] C.L. Borgman.
Why are online catalogs still hard to use?
Journal of the American Society for Information Science, 47(7):493–503, 1996.

[BPR+06] M. Botsch, M. Pauly, C. Rössl, S. Bischoff, and L. Kobbelt.
Geometric modeling based on triangle meshes.
International Conference on Computer Graphics and Interactive Techniques, 2006.

[Bri] E. Brink.
The Verse Networked 3D Graphics Platform.

[Bro95] W. Broll.
Interacting in distributed collaborative virtual environments.
Virtual Reality Annual International Symposium, 1995. Proceedings., pages 148–155, 1995.

[BSG+03] IN Bronstein, KA Semendjajew, G. Grosche, V. Ziegler, D. Ziegler, and E. Zeidler.
Teubner-taschenbuch der Mathematik.
BG Teubner, 2003.

[Bun05] M. Bunnell.
Dynamic ambient occlusion and indirect lighting.
GPU Gems, 2:223–233, 2005.

[Bur96] G.C. Burdea.
Force and touch feedback for virtual reality.
John Wiley & Sons, Inc. New York, NY, USA, 1996.

[CC78] Ed Catmull and Jim Clark.
Recursively generated B-spline surfaces on arbitrary topological meshes.
Computer Aided Design, 10(6):350–5, 1978.

[CHZ00] Jonathan M. Cohen, John F. Hughes, and Robert C. Zeleznik.
Harold: a world made of drawings.
In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 83–90, New York, NY, USA, 2000. ACM Press.

[CKS] S. Campagna, L. Kobbelt, and H.P. Seidel.
Directed edges - A scalable representation for triangle meshes.

[Cla52] W. Clark.
The Gantt Chart.
Pitman and Sons, London,, 1952.

[CMS88] M. Chen, S.J. Mountford, and A. Sellen.
A study in interactive 3-D rotation using 2-D control devices.
Proceedings of the 15th annual conference on Computer graphics and interactive techniques, pages 121–129, 1988.

[Coq90] S. Coquillart.
Extended free-form deformation: a sculpturing tool for 3D geometric modeling.
ACM Press New York, NY, USA, 1990.

[DeM79] T. DeMarco.
Structured analysis and system specification.
Prentice-Hall Englewood Cliffs, NJ, 1979.

[DKT98] T. DeRose, M. Kass, and T. Truong.
Subdivision surfaces in character animation.
Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pages 85–94, 1998.

[Far02] G. Farin.

History of Curves and Surfaces in CAGD.
Handbook of CAGD, G. Farin, MS Kim, J. Hoschek (eds), Elsevier, 2002.

[FMK+03] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs.
A search engine for 3d models.
ACM Trans. Graph., 22(1):83–105, 2003.

[GH91] Tinsley A. Galyean and John F. Hughes.
Sculpting: an interactive volumetric modeling technique.
SIGGRAPH Comput. Graph., 25(4):267–274, 1991.

[GSS99] I. Guskov, W. Sweldens, and P. Schroder.
Multiresolution signal processing for meshes.
Proceedings of SIGGRAPH, 99:325–334, 1999.

[Hol97] S.M. Hollister.
The Dirty Little Secrets of Hull Design by Computer.
New Wave Systems Inc, 1997.

[Hon] Dan Hong.
Effects of Weights in NURBS.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka.
Teddy: a sketching interface for 3d freeform design.
In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Kan13] I. Kant.
Kritik der Urteilskraft.
Meiner Hamburg, 1913.

[KHR] Olga Karpenko, John F. Hughes, and Ramesh Raskar.

[Kob00] L. Kobbelt.
sqrt (3) subdivision.
Proceedings of SIGGRAPH 2000, pages 103–112, 2000.

[LB] M.S. Langer and H.H. Bülthoff.
Perception of shape from shading on a cloudy day.
Technical report, Technical Report Technical Report.

[LLS01] N. Litke, A. Levin, and P. Schröder.
Trimming for subdivision surfaces.
Computer Aided Geometric Design, 18(5):463–481, 2001.

[Loo87] C.T. Loop.
Smooth subdivision surfaces based on triangles.
Master's thesis, Dept. of Mathematics, University of Utah, 1987.

[LRC+02] D. Luebke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson, and R. Huebner.

Level of Detail for 3 D Graphics.
Morgan Kaufmann, 2002.

[Mad96] J. Madison.
Cnc Machining Handbook: Basic Theory, Production Data, and Machining Procedures.
Industrial Press Inc., 1996.

[MWW01] G. Morin, J. Warren, and H. Weimer.
A subdivision scheme for surfaces of revolution.
Computer Aided Geometric Design, 18(5):483–502, 2001.

[NF02] A. Nasri and G. Farin.
A subdivision algorithm for generating rational curves.
Journal of Graphics Tools, 6(1):35–47, 2002.

[NSACO05] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or.
A sketch-based interface for detail-preserving mesh editing.
ACM Trans. Graph., 24(3):1142–1147, 2005.

[NWWM92] RE Newman-Wolfe, ML Webb, and M. Montes.
Implicit locking in the ensemble concurrent object-oriented graphics editor.
Proceedings of the 1992 ACM conference on Computer-supported cooperative work, pages 265–272, 1992.

[OCJF01] Manuel Oliveira, Vladimiro Colaco, Joaquim Jorge, and Manuel Fonseca.
Modeling solids and surfaces with sketches: an empirical evaluation.
In V. Skala, editor, *WSCG 2001 Conference Proceedings*, 2001.

[Pie91] L. Piegl.
On NURBS: a survey.
Computer Graphics and Applications, IEEE, 11(1):55–71, 1991.

[PKG] M. Pauly, L. Kobbelt, and M. Gross.
Multiresolution modeling of point-sampled geometry.
CS Techinical Report, 378.

[PKKG03] M. Pauly, R. Keiser, L.P. Kobbelt, and M. Gross.
Shape modeling with point-sampled geometry.
ACM Transactions on Graphics (TOG), 22(3):641–650, 2003.

[Pop04] T. Popa.
Survey of Triangulation Algorithms of Implicit Functions Represented on Regular Grids.
Computer Graphics, 20:1–9, 2004.

[Req80] A.G. Requicha.
Representations for Rigid Solids: Theory, Methods, and Systems.
ACM Computing Surveys (CSUR), 12(4):437–464, 1980.

[RLWH05] B.Y. REN, X.D. LI, Z.Q. WU, and I. HAGIWARA.
Local Sharp Feature Generation and Shape Control of Recursive Subdivision Surface.
JSME International Journal Series C, 48(2):170–175, 2005.

[Rog00] D.F. Rogers.
An Introduction to NURBS: With Historical Perspective.
Morgan Kaufmann, 2000.

[RSVW94] W. Reinhard, J. Schweitzer, G. Volksen, and M. Weber.
CSCW tools: concepts and architectures.
Computer, 27(5):28–36, 1994.

[SNBW03] F.F. Samavati, M.A. Nur, R. Bartels, and B. Wyvill.
Progressive curve representation based on reverse subdivision.
the 2003 International Conference on Computational Science and Its Applications, pages 67–78, 2003.

[SP86] Thomas W. Sederberg and Scott R. Parry.
Free-form deformation of solid geometric models.
In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160, New York, NY, USA, 1986. ACM Press.

[SP05] B. Shneiderman and C. Plaisant.
Designing the User Interface: Strategies for Effective Human-Computer Interaction.
Pearson–Addison Wesley,, 2005.

[SZSS98] T.W. Sederberg, J. Zheng, D. Sewell, and M. Sabin.
Non-uniform recursive subdivision surfaces.
Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pages 387–394, 1998.

[Tay99] A.G. Taylor.
The organization of information.
Libraries Unlimited Englewood, Colo, 1999.

[TO99] G. Turk and J.F. O'Brien.
Variational Implicit Surfaces.
1999.

[TWB+] E. Turquin, J. Wither, L. Boissieux, M.P. Cani, and J.F. Hughes.
A sketch-based interface for clothing virtual characters.

[W+99] M. Woo et al.
OpenGL programming guide.
Addison-Wesley Reading, MA, 1999.

[Wer25] Max Wertheimer.

Über Gestalttheorie : Vortrag gehalten in d. Kant-Ges. Berlin, am 17. Dez. 1924.
Verl. d. Philos. Akad., Erlangen, 1925.

[WW92] W. Welch and A. Witkin.
Variational surface modeling.
Proceedings of the 19th annual conference on Computer graphics and interactive techniques, pages 157–166, 1992.

[ZPKG02] M. Zwicker, M. Pauly, O. Knoll, and M. Gross.
Pointshop 3D: an interactive system for point-based surface editing.
Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pages 322–329, 2002.

[ZSD+00] Dennis Zorin, Peter Schröder, Tony DeRose, Leif Kobbelt, Adi Levin, and Wim Sweldens.
SIGGRAPH 2000 Course Notes: Subdivision for Modeling and Animation.
ACM Siggraph, 2000.

Anhang A.

Punktverschiebung

A.1. Umfragebogen

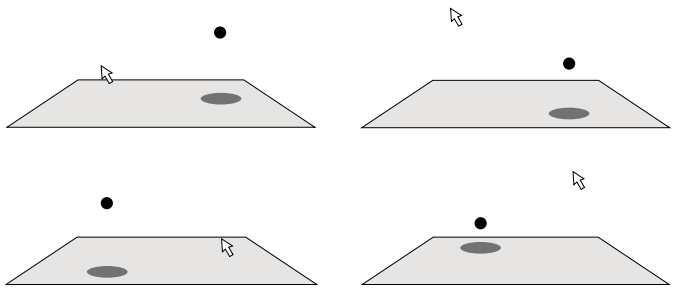
Umfrage zur Punktverschiebung im dreidimensionalen Raum

Teil der Diplomarbeit von Rodja Trappe

COMPUTERVISUALISTIK

Allgemeine Informationen:	☐ männlich Alter: ☐ ☐ weiblich
Kennst Du dich mit dem Computer soweit aus, dass Du im Internet surfen, Briefe schreiben und E-Mails verschicken kannst?	☐ Ja ☐ Nein
Hast Du schon einmal einen virtuellen Gegenstand hergestellt?	☐ Ja ☐ Nein
Ist Dir der Begriff des Kartesischen Koordinatensystems geläufig?	☐ Ja ☐ Nein
Kennst Du dich ein wenig mit Computergrafik aus?	☐ Ja ☐ Nein

Mit dieser Umfrage soll ermittelt werden, was bei der Arbeit mit virtuellen Gegenständen von dem Computer erwartet wird. Es gibt daher kein Richtig oder Falsch, sondern nur die Aussage »So sollte das sein!«. In den folgenden Grafiken ist jeweils eine Kugeln abgebildet die einen geraden Schatten von oben auf die Bodenplatte wirft, so wie es bei Mittagssonne zu beobachten wäre. Der Mauszeiger deutet an, wohin die Kugel verschoben wird. Zeichne die neue Position der Kugel und ihren Schatten, so wie Du es erwarten würdest.

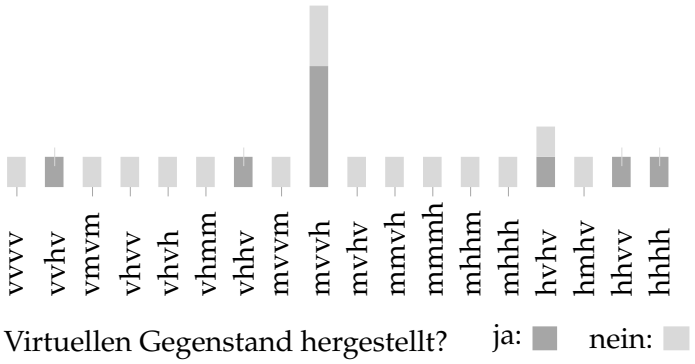


A.2. Ergebnisse

Der Schatten wurde immer entweder vorne, mittig oder hinten auf dem Untergrund genau unterhalb des Mauszeigers gezeichnet. Daraus ergibt sich folgende Tabelle:

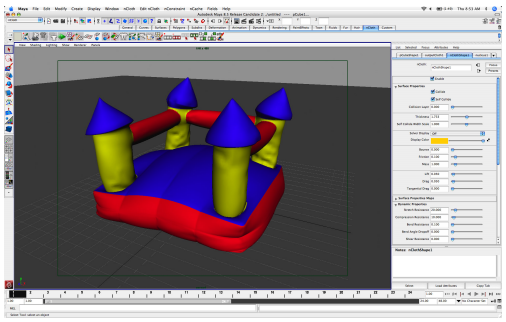
Ges.	Alter	Comp.	Gegen.	Koor.	CG	Abb. 1	Abb. 2	Abb. 3	Abb. 4
m	21	ja	ja	ja	ja	hinten	hinten	hinten	hinten
m	20	ja	ja	nein	ja	hinten	hinten	vorne	vorne
w	19	ja	nein	nein	nein	hinten	mitte	hinten	vorne
m	20	ja	ja	nein	ja	hinten	vorne	hinten	vorne
w	20	ja	nein	nein	nein	hinten	vorne	hinten	vorne
w	20	ja	nein	ja	nein	mitte	hinten	hinten	hinten
w	22	ja	nein	nein	nein	mitte	hinten	hinten	mitte
w	22	ja	nein	nein	ja	mitte	mitte	mitte	hinten
w	20	ja	nein	nein	nein	mitte	mitte	vorne	hinten
m	24	ja	nein	nein	nein	mitte	vorne	hinten	vorne
w	21	ja	nein	ja	nein	mitte	vorne	vorne	mitte
w	23	ja	nein	nein	nein	mitte	vorne	vorne	hinten
w	20	ja	ja	nein	ja	mitte	vorne	vorne	hinten
m	34	ja	ja	ja	ja	mitte	vorne	vorne	hinten
m	28	ja	ja	ja	ja	mitte	vorne	vorne	hinten
m	41	ja	ja	ja	ja	mitte	vorne	vorne	hinten
w	20	ja	nein	ja	nein	mitte	vorne	vorne	hinten
m	27	ja	ja	ja	ja	vorne	hinten	hinten	vorne
w	19	ja	nein	nein	nein	vorne	hinten	mitte	mitte
w	21	ja	nein	nein	nein	vorne	hinten	vorne	hinten
w	21	ja	nein	nein	nein	vorne	hinten	vorne	vorne
w	19	ja	nein	nein	nein	vorne	mitte	vorne	mitte
w	24	ja	ja	ja	ja	vorne	vorne	hinten	vorne
w	22	ja	nein	nein	nein	vorne	vorne	vorne	vorne

Signifikant tritt vor allem die Kombination [mitte, vorne, vorne, hinten] (mvvh) hervor. Andere Möglichkeiten der Schattenpositionierung waren bis auf eine Ausnahme Einzelmeinungen.

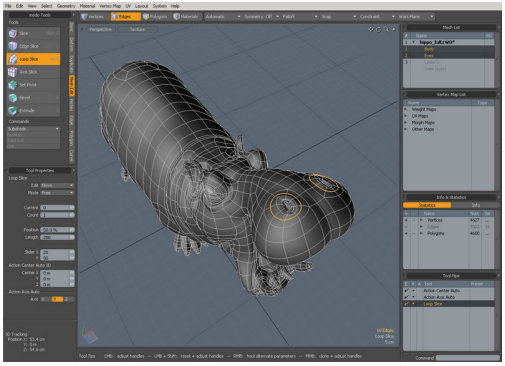


Anhang B.

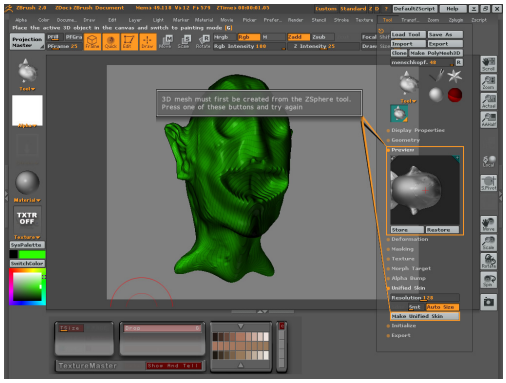
Screenshots bestehender Anwendungen



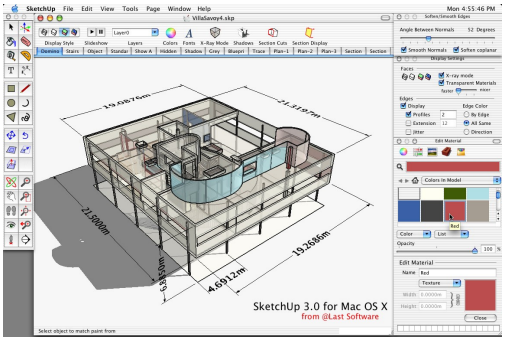
Screenshot von *Autodesk Maya*. Quelle: `http://www.highend3d.com`



Screenshot von *Luxology Modo*. Quelle: `http://images.digitalmedianet.com`



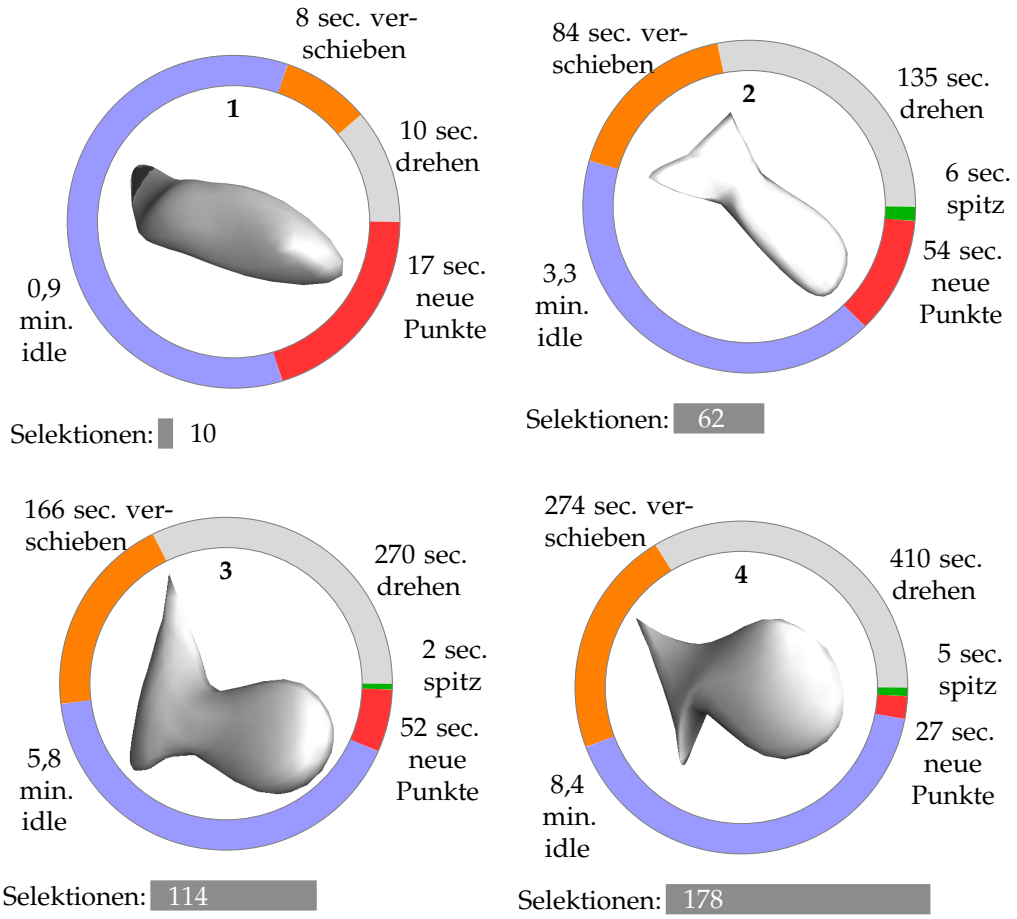
Screenshot von *Pixologic ZBrush*. Quelle: `http://www.zbrush.de`



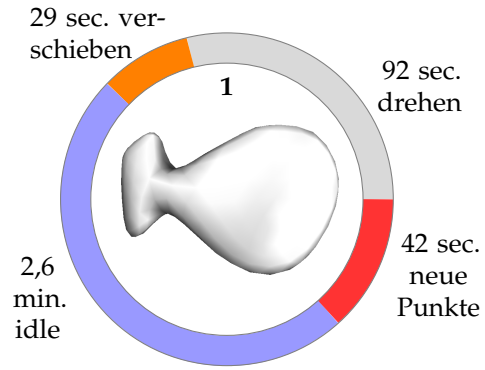
Screenshot von *Google Sketchup*. Quelle: `http://www.sketch3d.de`

Anhang C.

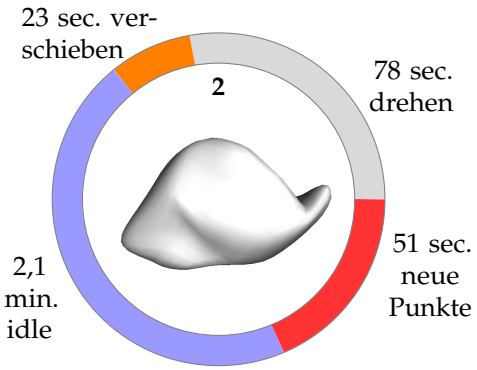
Weitere Diagramme zu den Benutzungstests



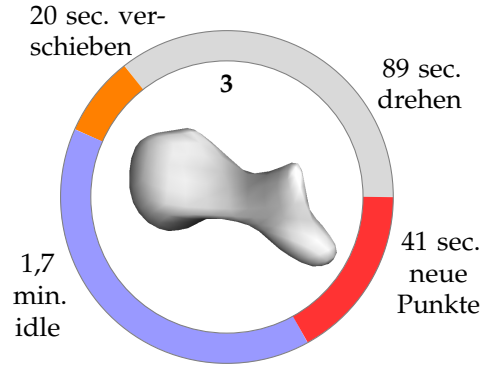
Diese vier Fische modellierte eine Probandin mit Vorerfahrungen in der 3D-Modellierung sowie Computergrafik-Kenntnissen.



Selektionen: 48

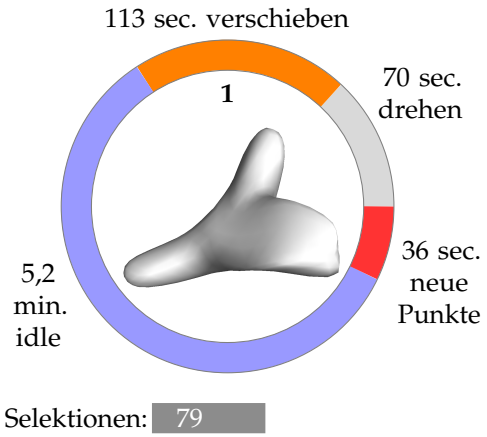


Selektionen: 47

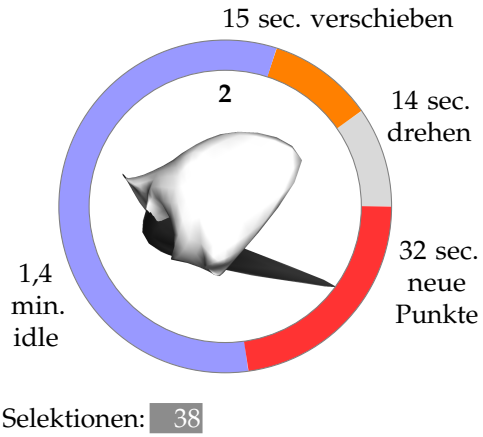


Selektionen: 35

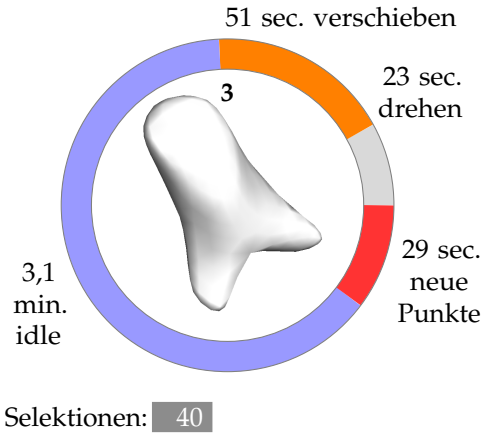
Diese drei Fische modellierte ein Proband ohne Computergrafik Wissen direkt aufeinanderfolgend.



Selektionen: 79

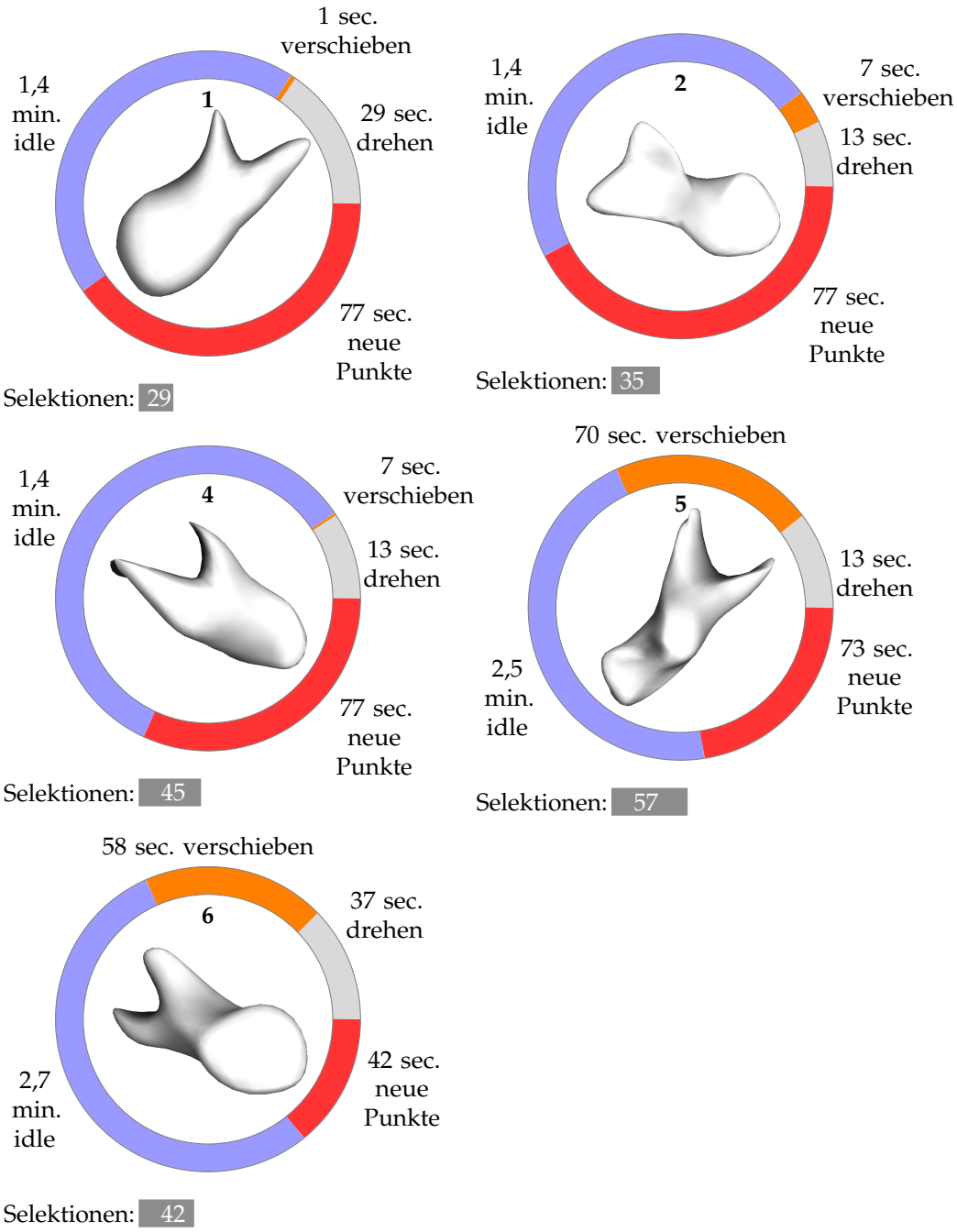


Selektionen: 38

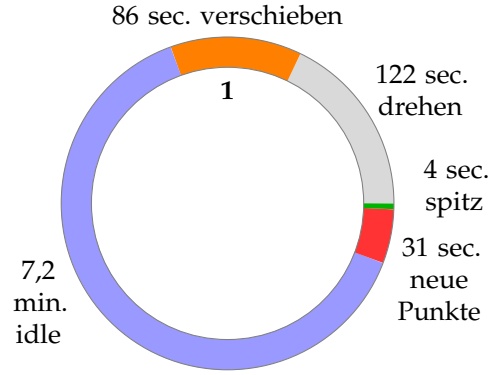


Selektionen: 40

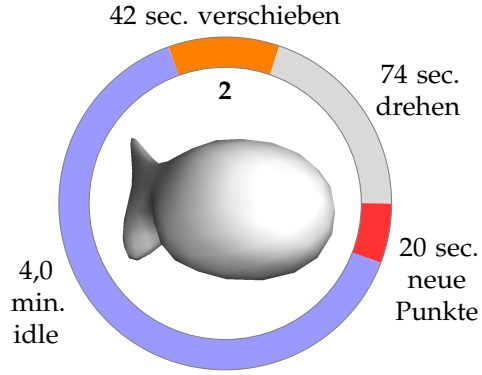
Diese drei Fische wurden von einem in Computergrafik Themen und Grafik-Software vollkommen unerfahrenen Anwender erstellt. Der Grafik im zweiten Diagramm kann man entnehmen, dass der Gegenstand kein Volumen mehr beschreibt und somit der Benutzer keine Chance der weiterführenden Modellierung hatte.



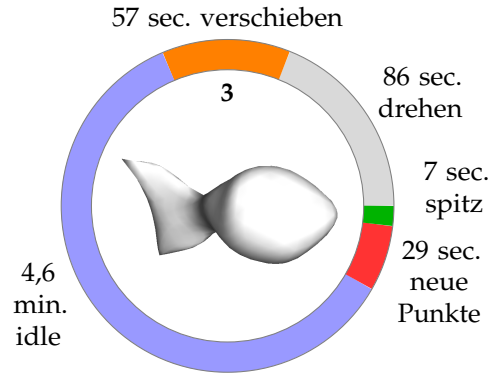
Diese Fische wurde von einer Probandin ohne Computergrafik-Wissen erstellt. Zuvor hatte sie sich 20 Minuten lang eingearbeitet. Der dritte Fisch fehlt leider. Besonders interessant ist die Entwicklung des Verhältnisses zwischen dem einfachen Verschieben von Punkten und dem neu Erzeugen. Erst nach einigen Versuchen wird der Versuch unternommen, beide Arten der Verformung gezielt einzusetzen.



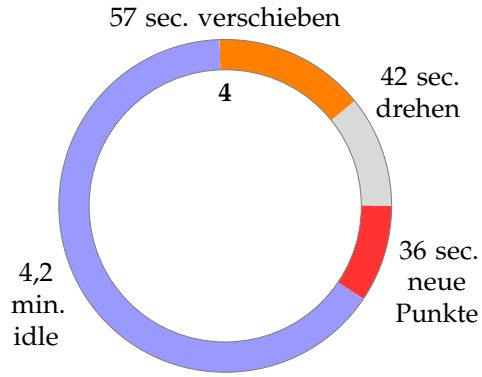
Selektionen: 148



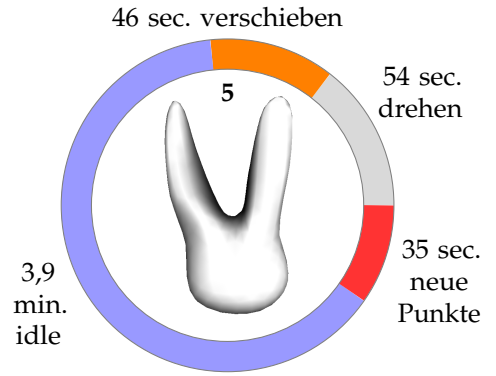
Selektionen: 109



Selektionen: 113

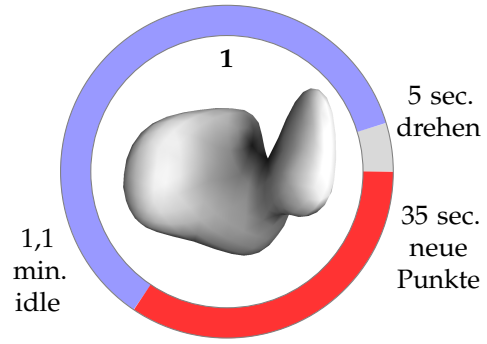


Selektionen: 102

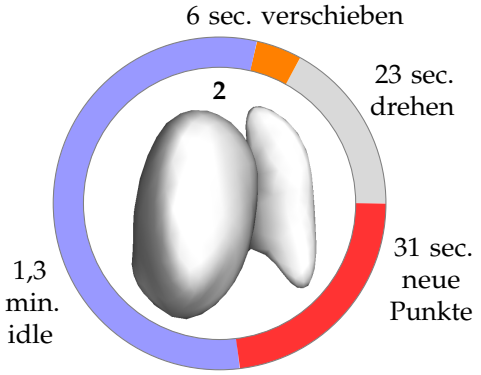


Selektionen: 110

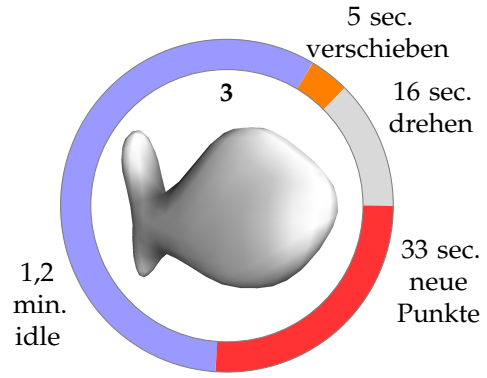
Diese Probandin hatte weitreichende Erfahrungen mit 2D Grafik-Anwendungen und Kenntnisse über Computergrafik. Von Fisch eins und vier fehlen aus technischen Gründen leider die Abbildungen. Das fünfte Diagramm dokumentiert den Versuch, einen Hasenkopf zu modellieren.



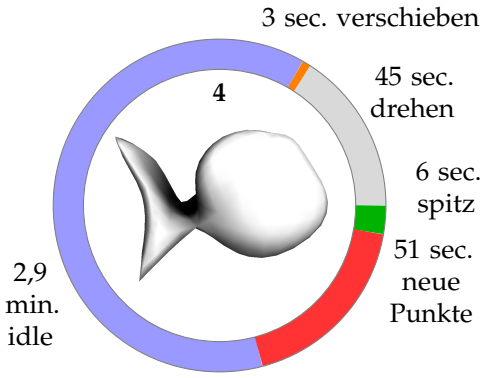
Selektionen: 21



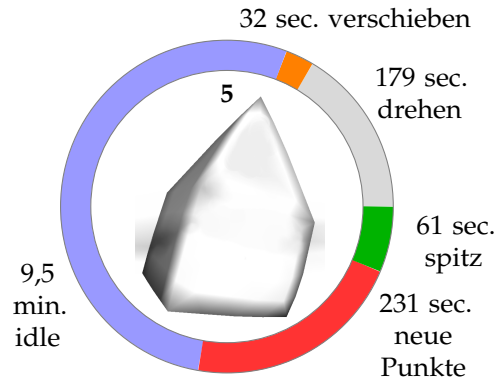
Selektionen: 37



Selektionen: 42

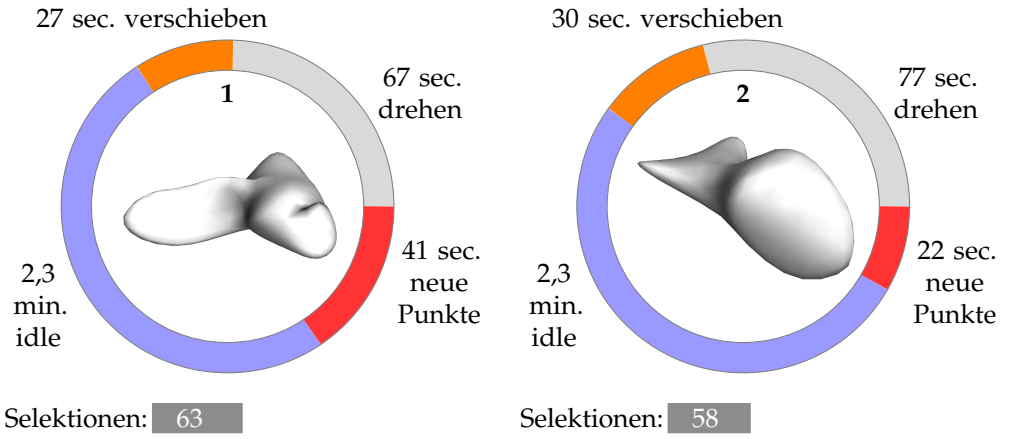


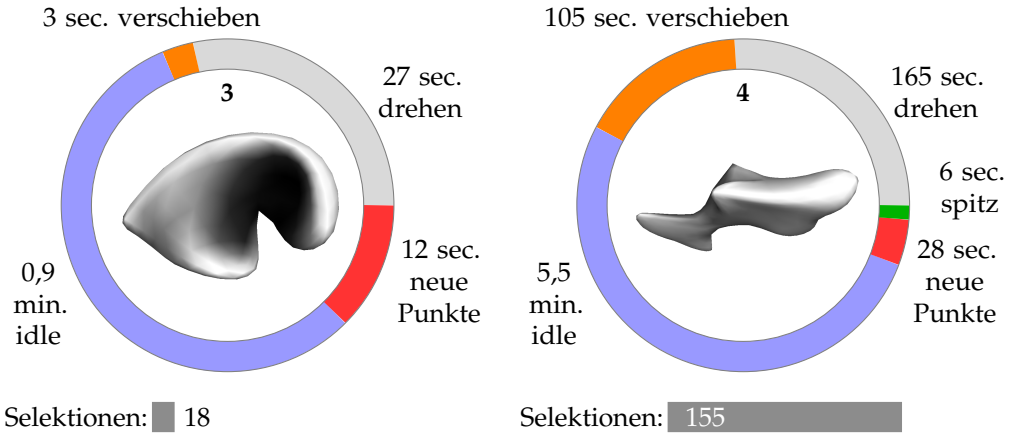
Selektionen: 104

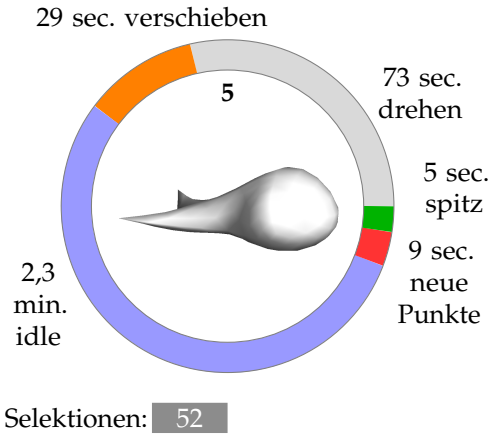


Selektionen: 297

Diese Probandin hatte weitreichende Kenntnisse von 2D und 3D Grafik-Software und Kenntnisse über Computergrafik. Nach vier Fischen bestand die Herausvorderung in der Modellierung eines Hauses. Dafür hat sie vier mal neu angefangen.







Der Proband hatte Vorerfahrungen in der 3D-Modellierung sowie Computergrafik-Kenntnisse. Diese fünf Fische entstanden aufeinander folgend nach dem er sich einige Tage zuvor schon ca. zehn Minuten mit der Software auseinander gesetzt hatte.

Anhang D.

CD-ROM mit Quelltext und Video

Auf der innen am Buchrücken befestigten CD-ROM befindet sich:

- Die Videoaufnahme eines Bildschirms, während mit drei anderen Personen zwei Fische modelliert werden (Dateiformat: Ogg Theora).
- Der Quelltext der entwickelten Protoypen.
- Die fertige Anwendung im Quelltext inklusive einer funktionsfähigen Version von *OpenMesh* und *Verse*.
- Dieses Dokument in elektronischer Form (Format: PDF).