Tobias Felden
5850721
Informatik B.Sc.
14
tobias@felden.com

Bachelorarbeit

Integration von PartNet in VAnnotatoR unter Gewährleistung von Objektsegmentierungen

Tobias Felden

Abgabedatum: 09.08.2021

Text Technologie Lab
Prof. Dr. Alexander Mehler

Erklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

Frankfurt am Main, 09.08.2021

Ort, Datum

Felden

Unterschrift

Zusammenfassung

Szenen automatisch aus Texten generieren zu können ist eine interessante Aufgabe der Informatik. Für diese Aufgabe wurde VAnnotatoR (Mehler und Abrami 2019, Abrami, Spiekermann und Mehler 2019, Spiekermann, Abrami und Mehler 2018) entwickelt, ein Framework, das die Beschreibung bzw. Beschriftung von VR-Szenen ermöglicht. Damit für diese Szenen die benötigten 3D-Objekte bereitgestellt werden können, sind entsprechende Datenbanken vonnöten. Diese Datenbanken müssen umfangreich annotiert sein, damit diese Aufgabe bewältigt werden kann. Deshalb wurde im Falle des VAnnotatoRs auf die ShapeNetSem Datenbank zurückgegriffen (Abrami, Henlein, Kett u. a. 2020).

Je detailreicher eine Szene dargestellt wird, desto detailreicher kann diese auch durch einen Text beschrieben werden. Aus diesem Grund wird die Datenbank um einen Teilbereich von PartNet (Mo u. a. 2019) erweitert. Dieser erlaubt die Option, Objekte zu segmentieren, und erweitert hierdurch das annotierbare Vokabular. Manche der bereits vorhandenen ShapeNetSem-Objekte verfügen über die Eigenschaft, dass sie auch PartNet-Objekte sind. Diese Arbeit befasst sich mit der Umsetzung, wie ShapeNetSem-Objekte mit hinterlegten PartNet-Objekten durch diese ersetzt werden können. Um das zu bewerkstelligen, wurde ein Panel entworfen, in welchem ein PartNet-Objekt mit samt seinen einzelnen Segmenten aufgeführt wird. Diese Segmente können nun wie ShapeNetSem-Objekte ausgewählt und in einer Szene platziert werden. Dadurch werden 1.881 Objekte mit wiederum 34.016 Unterobjekten VAnnotatoR zur Verfügung gestellt. Dieses vergrößerte Vokabular hilft *Natural Language Processing* noch effektiver und präziser voranzutreiben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AR Augmented Reality (zu Deutsch: Erweiterte Realität). Siehe Abschnitt 1.3.3, S. 10, 11

CAD Computer-aided design (zu Deutsch: Computerunterstützter Entwurf), CAD Programme, wie z.B. Autodesk, sollen dem Anwender durch Anfertigen eines 3D-Objektes die Erstellung von technischen Zeichnungen erleichtern. Siehe Abschnitt 2.2, S. 13

ID Identifikationsnummer, in dieser Arbeit ist die ID eines PartNet-Objekts immer identisch mit der des entsprechenden ShapeNet-Objekts. Siehe Kapitel 3, S. 18, 21, 23, 24, 26

NLP Natural Language Processing (zu Deutsch: Verarbeitung natürlicher Sprache). Siehe Abschnitt 1.3.1, S. 7–9

UMUX Usability Metric for User Experience (zu Deutsch: Gebrauchstauglichkeits-Metrik für das Benutzererlebnis (DIN EN ISO 9241-11)). Siehe Abschnitt 5.1, S. 33

VR Virtual Reality (zu Deutsch: Virtuelle Realität). Siehe Abschnitt 1.3.2, S. 3, 7–11, 33, 34

1 Einleitung

VAnnotatoR ist ein Programm zur Unterstützung der Annotation von Multimedia, multimodalen Netzwerken der Linguistik und objektbezogener Daten in einer virtuellen dreidimensionalen Umgebung. VAnnotatoR ermöglicht das Zerlegen eines Textes in einzelne Worte, um diese mit anderen Multimedia oder multimodalen Daten zu verbinden und so einen räumlichen Hypertext zu kreieren. Das Programm kann dazu verwendet werden, konkrete technische Anwendungen zu unterstützen, z.B. für die Rekonstruktion einer dreidimensionalen Szene aus Texten, als Vorbereitung zum Anlernen von Text2Scene Systemen (siehe Abschnitt 2.1) oder für Anwendungen im Bereich der digitalen Geisteswissenschaften, wie die räumliche Rekonstruktion und Verbindung von historischen Szenen oder Biografien mit korrespondierenden medialen Inhalten, beispielsweise Bildern oder 3D rekonstruierten historischen Gebäuden. Das gesamte Programm wird interaktiv mit einer VR-Brille in der virtuellen Realität (VR, siehe Abschnitt 1.3.2) ausgeführt und kann dementsprechend erprobt und gesteuert werden. Das *TextAnnotator* Werkzeug (Abrami, Henlein, Lücking u. a. 2021, TextTechnologyLab 2019a) ist das Rückgrat des VAnnotatoRs, weshalb VAnnotatoR auch sämtliche Funktionen für Annotation, Task Management, annotations Evaluation und für kollaborative und simultane Prozesse von Ressourcen des *TextAnnotators* benutzt. Außerdem verwendet VAnnotatoR auch das *TextImage* Werkzeug (Hemati, Uslu und Mehler 2016, TextTechnologyLab 2019c), um von dessen *Natural Language Processing* (NLP)-Routinen für automatische Textanalyse zu profitieren.

VAnnotatoR wurde in Unity3D (Unity3D 2021, Abruf: 18.05.2021) mithilfe von OpenVR programmiert und ist für verschiedene VR-Brillen ausgelegt. Erfolgreich getestet wurde mit der Oculus Rift, Oculus Rift S, Oculus Quest, HTC Vive und HTC Vive Cosmos unter Windows 10 (TextTechnologyLab 2019d, Abruf: 18.05.2021).

Die Objekte, die in dem virtuellen Raum (im folgenden Szene genannt) platziert werden können, werden aus einem Teil der ShapeNet-Datenbank, ShapeNetSem (A. X. Chang u. a. 2015, TTIC, S. University und P. University 2016, Abruf: 18.05.2021), bezogen. Um eine noch detailreichere Abbildung von Szenen gewährleisten zu können, wird nun ShapeNetSem um PartNet (Mo u. a. 2018, Mo u. a. 2019, Abruf: 18.05.2021) erweitert. PartNet-Objekte sind ShapeNet-Objekte mit Unterobjekten (siehe Abschnitt 2.3). Alle Unterobjekte eines PartNet-Objekts zusammengefasst sind demnach von Form und Größe identisch mit dem entsprechenden ShapeNet-Objekt. Die Unterobjekte eines PartNet-Objekts werden im Folgenden Segmente genannt. Diese Segmente können auch einzeln platziert und annotiert werden.

1.1 Motivation

Die menschliche Informationsverarbeitung der meisten Menschen ist stark räumlich, nicht nur was die Wahrnehmung angeht, sondern auch in der Sprache (Lakoff 1987). Um in der VR eine möglichst präzise Repräsentation der realen Welt zu gewährleisten, ist es unabdingbar

eine große Anzahl an Objekten zur Verfügung zu stellen. Ähnlich wie in einem Buch die Wörter, sind in der VR die verschiedenen Objekte dafür ausschlaggebend, wie lebendig und realistisch eine Szene wirkt. Außerdem kann mithilfe der PartNet-Objekte eine detailliertere Szene beschrieben werden als lediglich mit ShapeNet-Objekten. Auf diese Weise kann z.B. eine Szene in einem Buch getreuer wiedergegeben werden. Wenn beispielsweise ein Mörder mit einem Tischbein sein Opfer erschlug, könnte dies nun dargestellt werden, da durch die PartNet-Erweiterung das Tischbein von dem Tisch abgelöst werden kann (siehe Abbildung 4.2). Demnach erweitert PartNet auch das annotierbare Vokabular für die in VAnnotatoR eingespeisten Texte. Eine breiter aufgestellte mediale Unterstützung ist also höchst erfreulich und eine wünschenswerte Erweiterung.

Die in dem Rahmen dieser Arbeit entwickelte Erweiterung wäre in der Lage PartNet-Objekte zu platzieren, skalieren, annotieren, speichern, auseinanderzunehmen und zu entfernen. Außerdem wäre sie eine unabhängige Erweiterung. Das bedeutet, sie könnte nach Belieben benutzt oder ignoriert werden, denn sämtliche ShapeNet-Funktionen blieben unverändert und wären mit den PartNet-Funktionen kompatibel.

1.2 Problemstellung

Ziel dieser Arbeit ist es, die bereits vorhandenen Funktionen für ShapeNetSem-Objekte, die gleichzeitig als PartNet-Objekte existieren, auszuweiten. Hierzu muss sowohl die PartNet-Datenbank implementiert, als auch die vorhandenen Funktionen so erweitert werden, dass PartNet-Objekte erkannt und entsprechend verarbeitet werden können.

Hilfreich ist hier, dass die ShapeNet- und PartNet-Objekte sich immerhin stark ähneln. Allerdings lässt sich nicht erkennen, dass die ShapeNet Implementierung in Hinblick auf eine Erweiterung um PartNet gestaltet wurde. Zudem weisen PartNet-Objekte eine neue Eigenschaft auf, nämlich die hierarchische Zuordnung innerhalb des Ausgangsobjekts. PartNet-Objekte müssen sowohl als Einzelteile eines größeren Objekts als auch als individuelle Objekte betrachtet und behandelt werden.

Die finale Erweiterung soll gewährleisten, dass mit Einzelteilen eines Objekts in der Szene interagiert werden kann und dass die Einzelteile annotierbar sind.

1.3 Grundlagen

Der Verständlichkeit halber werden zunächst einige relevante Begrifflichkeiten erläutert. In den folgenden Abschnitten wird sich mit *Natural Language Processing* (NLP, siehe Abschnitt 1.3.1) befasst, da dies der Hauptantrieb für diese Arbeit ist. Auch muss die virtuelle und die erweiterte Realität (Abschnitte 1.3.2 und 1.3.3) angesprochen werden, da VAnnotatoR sich beider bedient.

1.3.1 Natural Language Processing

Mithilfe von NLP (zu Deutsch: Verarbeitung natürlicher Sprache) soll die Verarbeitung und Analyse großer Mengen an natürlicher Sprache durch Computer realisiert werden. Für Men-

schen ist es einfach aus einem Text ein lebendiges Bild im Kopf entstehen zu lassen. Ein Mensch lernt früh, Wörter mit Gegenständen zu verbinden und setzt dies im Alltag ständig um z.B., wenn er ein Buch liest oder nach dem Salz gebeten wird. Computer hingegen müssen vom Menschen trainiert werden, einen Text zu „verstehen“. Hierfür ist es wichtig, dass Wörter annotiert werden, um so einen Bezug zu anderen Wörtern herstellen zu können. Das Ziel von NLP ist es Informationen zu erforschen, wie Menschen Sprache benutzen und verstehen. Dieses Wissen wird dann benutzt, um Computer dahingehend zu trainieren, die natürliche Sprache zu verstehen und zu manipulieren, um neue Aufgabenfelder zu bewältigen. Teilprobleme der NLP entstammen der Linguistik, Informatik, Mathematik, Elektrotechnik, künstliche Intelligenz und Robotik. NLP-Programme bearbeiten verschiedene Aufgaben. Dazu zählen: Die Verarbeitung natürlicher Sprachtexte und das Zusammenfassen dieser, sowie Spracherkennung, künstliche Intelligenz, sprachübergreifendes Abrufen von Information und Weitere (Chowdhury 2003).

Eine mögliche Definition von NLP (gemäß Liddy 2001) wäre:

> Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.

Zu Deutsch:

> Die Verarbeitung natürlicher Sprache ist eine theoretisch motivierte Reihe von rechnergestützten Verfahren zur Analyse und Darstellung natürlich vorkommender Texte auf einer oder mehreren Ebenen der linguistischen Analyse, um ein menschenähnliches Sprachverständnis für eine Reihe von Aufgaben oder Programmen zu erreichen.

1.3.2 Virtuelle Realität

Virtuelle Realität (VR) ist im heutigen Sprachgebrauch ein Computerprogramm, in welchem der Nutzer mithilfe einer VR-Brille und dazugehörigen Controllern mit einer fiktiven Umgebung interagieren kann. Obgleich viele Menschen VR als eine moderne Erscheinung wahrnehmen mögen, kann ihr Ursprung in die frühen 80er-Jahre zurückgeführt werden (Erl 2020, Abruf: 18.05.2021). Tomasz Mazuryk und Michael Gervautz sehen den Ursprung noch früher: In den Jahren 1960-1962 entwickelte der Kameramann Morton Heilig einen interaktiven Film, Sensorama und könne somit als Pionier der VR gesehen werden (Mazuryk und Gervautz 1999). In Romanen wie Neuromancer von William Gibson wurde bereits 1984 VR erstmals behandelt und demonstriert somit, dass nicht nur in der Computerwelt VR thematisiert wird (Machover und Tice 1994, Gibson 1984). Nur am Rande: Das Ende 2020 erschienene Spiel *Cyberpunk 2077* von CD Projekt RED ist von dem von Neuromancer ins Leben gerufene Genre *Cyberpunk* inspiriert. Ein weitestgehend bekannter Film zu VR, Matrix von 1999 (Wachowskis 2020, Abruf: 18.05.2021), ist inzwischen über 20 Jahre alt. VR ist demnach keine Neuerscheinung, obwohl sie durchaus noch als eben dies wahrgenommen wird. Ein Grund für die geringe Verbreitung von VR, ist auch der stattliche Preis und die geringe Anwendbarkeit. Der Trend zeigt jedoch fallende Preise, durch billigere Sensoren, und ermöglicht mehr Menschen sich ein VR-Headset zu leisten. Das lockt wiederum Softwareentwickler, ein breiteres Angebot für die Verwendung dieser zu programmieren (Ballhaus 2019, Abruf:

03.08.2021). So feierte jüngst *Beat Saber* (Beat Games, 01.05.2018), ein Computerspiel, welches wie VAnnotatoR in Unity3D programmiert wurde, mit über vier Millionen verkauften Exemplaren große Erfolge für die VR (Verdu 2021).

Selbstverständlich findet die VR nicht nur in Science-Fiction oder Computerspielen Anwendung, auch die Wissenschaft macht sich diese Technik zunutze. So hat die NASA bereits 1984 ein Programm zur Untersuchung von Strömungsfeldern entwickelt (Mazuryk und Gervautz 1999). Auch das Text Technologie Lab macht sich die VR zunutze. Im Falle des VAnnotatoRs bewegt sich der Nutzer mittels der erwähnten VR-Brille und den entsprechenden Controllern durch eine zu Beginn leere, dreidimensionale Szene. Diese kann mithilfe von verschiedenen Teilprogrammen (siehe Abschnitt 2.1.1) mit Gegenständen gefüllt und im Anschluss mit den platzierten Objekten interagiert werden.

1.3.3 Erweiterte Realität

Dieser Abschnitt wird nur angeschnitten, denn Augmented Reality (AR, zu Deutsch: Erweiterte Realität) und VR ähneln sich. Außerdem wurde die PartNet Erweiterung ausschließlich für VR getestet. Das liegt daran, dass es für das Platzieren der Objekte keine Rolle spielt, ob die „Realität" in die Szene integriert ist oder nicht. Doch ist die AR Teil des VAnnotatoRs, weshalb sie hier thematisiert wird. AR ist die computergestützte Erweiterung der Realitätswahrnehmung. Diese Erweiterung geschieht in Echtzeit als Anwendung auf eine direkte oder indirekte Sicht auf eine reale Umgebung. Diese zusätzlichen Informationen können mehrere menschlichen Sinne ansprechen, jedoch wird die visuelle Darstellung am häufigsten assoziiert. Unter AR fällt beispielsweise, wenn mittels Handy und dessen Kamera das Bild auf dem Display erweitert wird, um mehr Informationen über ein vorliegendes Objekt bereitzustellen. Dieses Hilfsmittel findet u. a. im Bayerischen Nationalmuseum in München bereits Anwendung (Go 2021, Abruf: 08.06.2021). AR ist sowohl interaktiv als auch prüfend und ermöglicht das Kombinieren von realen und virtuellen Objekten (Furht 2011).

1.3.4 LitJSON

Für dieses Projekt wurde die Konvertierung von den gegebenen JSON-Datein zu Strings benötigt, um die Daten von der Webseite ins Projekt zu laden (siehe Abschnitt 4.5). Die Konvertierung von JSON-Dateien zu Strings ist weder Teil von Unity3D noch von C#. *LitJSON* ist eine Open-Source Bibliothek, welche die nötigen Anforderungen (Konvertierung JSON zu String) erfüllt (Glick 2021, Abruf: 13.06.2021). Die umgekehrte Konvertierung (String zu JSON) bietet *LitJSON* zwar auch, diese wird allerdings nicht benötigt.

2 Stand der Technik

Im folgenden Kapitel wird auf die Ausgangslage eingegangen, von welcher diese Arbeit aus startet. Hierzu wird das VAnnotatoR Framework näher beleuchtet und auf die beiden Datenbanken der Objekte eingegangen.

2.1 VAnnotatoR

VAnnotatoR ist ein Framework zur Generierung und Visualisierung von multimodalen Hypertexten. Durch die Entstehung der Netzwerke von animierten, ikonischen oder symbolischen Einheiten, die in der VR oder AR repräsentiert werden, kann ein Benutzer verschiedene Wege der Informationsbeschaffung zum Lesen oder Erstellen von Hypertexten beschreiten. Beispielsweise kann der Benutzer diese Texte durch sehen, hören, bewegen oder berühren dieser Einheiten bearbeiten. Wenn das der Fall ist, dann sprechen wir von multimodalen Hypertexten (Mehler, Abrami, Spiekermann u. a. 2018). Das Framework beinhaltet Text2Scene (Abrami, Henlein, Kett u. a. 2020), Stolperwege, eine App, die sich dem Gedenken der Holocaust Opfer widmet (Mehler, Abrami, Bruendel u. a. 2017), *TextAnnotator*, ShapeNetSem (siehe Abschnitt 2.2) und künftig auch PartNet (siehe Abschnitt 2.3). Der *TextAnnotator* ist browserbasiert und kann somit unter dem Link `http://www.textannotator.texttechnologylab.org/` benutzt werden. Er kann diverse Daten und Annotationsebenen verarbeiten (Abrami und Mehler 2018, TextTechnologyLab 2019a, TextTechnologyLab 2019b Abruf: 22.05.2021).

Die Aufgabe Text aus einer Szene oder umgekehrt zu erstellen, kann nur erfüllt werden, indem eine umfangreiche und gut annotierte Datenbank zur Verfügung steht. In unserem Fall bedeutet umfangreich, dass die Objekte über Informationen wie Ausrichtung und Skalierung verfügen. Hiermit beschäftigt sich Text2Scene in der VR und erlaubt seinen Benutzern räumliche Hypertexte zu generieren (Abrami, Henlein, Kett u. a. 2020).

VAnnotatoR bedient sich der VR sowie der AR und ermöglicht die Annotation und Zuweisung semiotischer Aggregate, wie Texte, Bilder und deren Segmente mit begehbaren Animationen von Plätzen und Gebäuden. Mithilfe des VAnnotatoRs kann multimodaler Hypertext generiert und auch gelesen werden (Mehler und Abrami 2019). VAnnotatoR ist ein Werkzeug, das dazu dient, nicht nur einzelne Objekte zu annotieren, sondern gleich ganze Szenen.

Neben VAnnotatoR existieren noch weitere Frameworks, die sich mit dem Problem, aus Texten eine Szene zu erstellen, befassen. Beispielsweise *Language-Driven Synthesis of 3D Scenes from Scene Databases* (Ma u. a. 2018) oder *Text to 3D Scene Generation* (A. Chang u. a. 2015). Anders als VAnnotatoR ist der Ansatz des *Language-Driven Synthesis of 3D Scenes from Scene Databases* Frameworks, aus bereits erstellten und annotierten Szenen zu lernen. Hierfür benutzt das Framework die *SUNCG-Datenbank* (Song u. a. 2017) für die Szenen und, wie

VAnnotatoR, ShapeNetSem für die einzelnen Objekte (Ma u. a. 2018). Beide Alternativen verfügen derzeit nicht über PartNet-Strukturen.

2.1.1 DataBrowser

Der *DataBrowser*, Abbildung 2.1, ist ein Teil des VAnnotatoR-Interfaces. Er ermöglicht, dass die Datenbank nach den vorhandenen Daten und Objekten durchsucht werden kann. Er besteht aus den Fenstern *DataSpaceControl*, *DataPanel*, *DataFilterPanel*, *DataSearchPanel* und künftig auch aus dem *DataHierarchyPanel*. Wird der *DataBrowser* geöffnet so erscheint zuerst nur das *DataSpaceControl*, das *DataPanel* und das *DataSearchPanel*. Wird nun im *DataSpaceControl* ShapeNet ausgewählt und anschließend im *DataPanel* PartNet Models oder ShapeNet Models, so öffnet sich das *DataFilterPanel*. Innerhalb des *DataPanels* kann zwischen einzelnen Komponenten des Frameworks navigiert werden. Im Folgenden wird davon ausgegangen, dass der Benutzer sich bei der Wahl immer entweder für ShapeNet Models oder PartNet Models entscheidet.

Das *DataFilterPanel* dient zum Filtern der zahlreichen Objekte. Durch dieses kann zwischen den einzelnen Hauptkategorien und anschließend zwischen den dazugehörigen Unterkategorien gewählt werden. Wurde eine Kategorie ausgewählt, so werden sofort die auf dem *DataPanel* angezeigten Objekte aktualisiert. Im *DataPanel* kann nun eine *Virtuelle-Ressource* erzeugt werden. Diese wird an den *ObjectTab* übergeben und ein Objekt kann platziert werden. Abbildung 2.1 zeigt den neuen *DataBrowser* mit sämtlichen Komponenten.

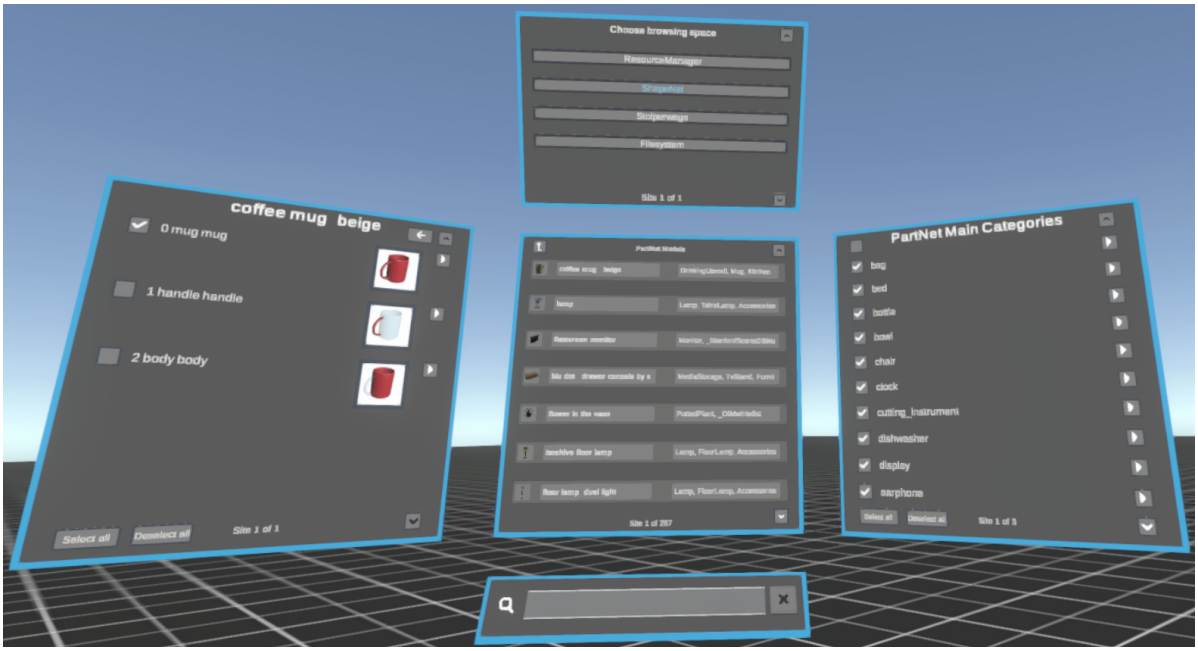


Abbildung 2.1: Zeigt den *DataBrowser* mit seinen einzelnen Komponenten am Beispiel einer Kaffeetasse. Links befindet sich das neue *DataHierarchyPanel*, oben in der Mitte der *DataSpaceControl*, zentral in der Mitte das *DataPanel*, unten in der Mitte das *DataSearchPanel* und rechts das *DataFilterPanel*

2.1.2 ObjectTab

Der *ObjectTab* befindet sich im selben Fenster wie der *AnmeldeTab* und der *AnnotationsTab*, wobei im *AnnotationsTab* der Text repräsentiert wird, der annotiert werden soll. Im *ObjectTab* können einfache Formen wie Würfel oder Ebenen erstellt werden. Außerdem dient er der Erzeugung der PartNet- und ShapeNet-Objekte.

Wurde sich für ein PartNet- oder ShapeNet-Objekt entschieden so wird eine *Virtuelle-Ressource* erstellt. Diese wird nun zum *ObjectTab* getragen und in ein kleines Fenster gelegt. Daraufhin kann über die linke bzw. rechte Hand das entsprechende *GhostObject* platziert werden. Ein *GhostObject* ist ein zu dem eigentlichen Objekt identisches Objekt, welches beim Platzieren der Objekte helfen soll. Um das *GhostObject* herum ist ein rechtwinkliges Gitter, welches auf Kollisionen mit bereits existierenden Objekten hinweist. Nachdem das Objekt platziert wurde, wird das *GhostObject* deaktiviert.

2.2 ShapeNet

Die Wahl der Datenbank für *Text2Scene* fiel auf ShapeNetSem eine Teildatenbank von ShapeNet, denn ShapeNetSem ist eine umfangreiche und ausreichend annotierte Datenbank, die frei zugänglich ist (Abrami, Henlein, Kett u. a. 2020). Auch ShapeNet ist ein gut annotiertes Repository von Formen, die als 3D CAD-Modelle repräsentiert werden. ShapeNet-Daten werden aus verschiedenen Datenbanken gesammelt (z.B. aus Trimble 3D Warehouse (Trimble 2021), Yobi3D (Malavida 2021) oder anderen bereits existierenden wissenschaftlichen Datensätzen). Mit rund drei Millionen Formen aus online 3D-Modell Repositories, von denen wiederum rund 300.000 in der WordNet-Taxonomie (Miller 1995) kategorisiert sind, ist ShapeNet eine bedeutende Datenbank. Die hier verwendeten Zahlen berufen sich auf den Artikel vom 09.12.2015 von A. X. Chang u. a. 2015. Da die Zahlen des VAnnotatoRs für diese Arbeit wesentlich wichtiger sind, sind nur diese aktuell gehalten. Umfangreich annotiert ist jedoch nur ein Teil dieser großen Menge. Für dieses Projekt werden die Modelle aus ShapeNetSem (Abrami, Henlein, Kett u. a. 2020, A. X. Chang u. a. 2015, TTIC, S. University und P. University 2016) verwendet, da im VAnnotatoR ganze Szenen annotiert werden sollen und es daher essenziell ist, dass die Objekte bereits über Informationen wie Ausrichtung und Skalierung verfügen.

Im VAnnotatoR gemäß `http://shapenet.texttechnologylab.org/loadedobjects` existieren Stand 27.05.2021 9.783 ShapeNet-Objekte wobei davon 1.881 PartNet-Objekte sind. Diese Objekte verfügen im Gegensatz zu reinen ShapeNet-Objekte stets über die Eigenschaft *has_parts* (siehe Abschnitt 4.1) und sind deshalb PartNet-Objekte.

Die Ziele von ShapeNet sind:

- 3D-Modell Datensätze zu sammeln und zu zentralisieren, um so der wissenschaftlichen Gemeinschaft den organisatorischen Aufwand zu ersparen.
- Unterstützung von datenbasierten Methoden, die 3D-Modelle benötigen.
- Evaluationen und Vergleiche von Algorithmen für fundamentale Aufgaben der Geometrie zu ermöglichen.
- Als Wissensbasis für reale Objekte und deren Semantik zu dienen.

Die einzelnen, annotierten Objekte besitzen Attribute, die Auskunft über die Orientierung, physische Eigenschaften wie Gewicht oder Reibungskoeffizient und die wesentlichen Punkte, Formsymmetrie und Größe in der Realität geben (A. X. Chang u. a. 2015).

2.3 PartNet

Objekte können in verschiedenen Feinheitsstufen wahrgenommen und repräsentiert werden. Genauer, grobe Objekte vermitteln ein globales, feine Objekte hingegen ein detaillierteres Verständnis (Mo u. a. 2018).

PartNet ist ähnlich wie ShapeNetSem eine Erweiterung von ShapeNet. PartNet erweitert die einzelnen 3D-Modelle um deren Einzelteile. Dies bedeutet, es gibt eine Auswahl an ShapeNet-Objekten, die nun auch in einzelne Unterobjekte bzw. Segmente, wie in Abbildung 2.2 zu sehen ist, zerteilt werden können. In Abbildung 2.2 lässt sich außerdem erkennen, dass die Unterobjekte aus weiteren Unterobjekten zusammengesetzt sind. Je feiner die Darstellung eines Objektes ist, desto mehr Segmente besitzen wiederum eigene Unterobjekte. So verfügt z.b. der Schreibtisch erst über nur zwei Segmente; die Tischplatte und der Rest. Anschließend wird der Rest aufgeteilt in drei Schubladen, Rück- und Seitenwände.

Wie auch bei ShapeNet existiert eine beachtliche Anzahl an Objekten, von denen wir jedoch aus denselben Gründen wie zuvor bei ShapeNet nicht alle verwenden können (siehe Abschnitt 2.2). Verwendet werden demnach ausschließlich die PartNet-Objekte, welche in ShapeNetSem repräsentiert sind. Dies bedeutet zugleich, dass jedes PartNet-Objekt besitzt eine ID, welche auch in ShapeNetSem existiert. Dieses ursprüngliche Objekt ist gleich, wenn das PartNet-Objekt mit der *ori_id* 0 ausgewählt ist.

Der Teil von PartNet, der in VAnnotatoR implementiert wird, beläuft sich auf 1.881 solcher Objekte. Jedes dieser Objekte besitzt weitere Segmente. Jedes Segment besteht aus weiteren atomaren Objekten. Sind diese selbst keine Segmente, so sind sie lediglich eine Stütze zum Kreieren der Segmente und bieten keinen Mehrwert zum Annotieren. Insgesamt besitzen alle PartNet-Objekte des VAnnotatoRs zusammen über 34.016 Segmente, welche aus 57.960 atomaren Objekten bestehen.

PartNet selbst besitzt 26.671 Objekte, die in 24 Kategorien unterteilt sind. Diese Objekte setzen sich aus insgesamt 573.585 Segmenten zusammen. Demnach werden im VAnnotatoR nicht einmal zehn Prozent der von PartNet zur Verfügung gestellten Objekte verwendet.

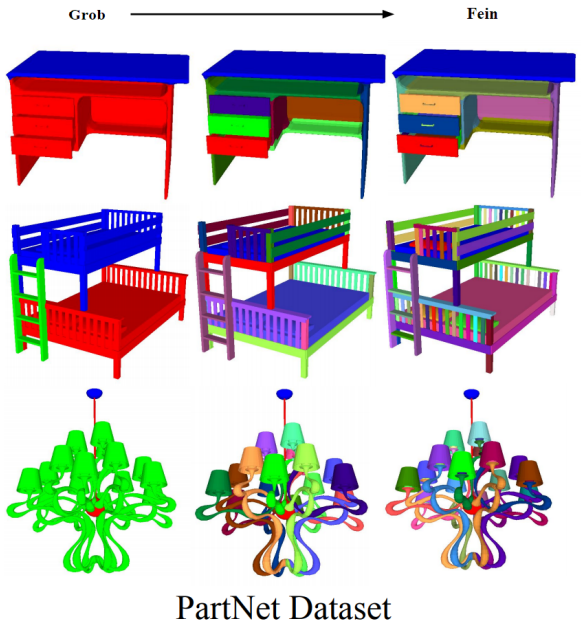


Abbildung 2.2: Feiner werdende Zerlegung von Objekten in Teilobjekte (Mo u. a. 2018)

Um diese dennoch beachtliche Menge an Objekten präsentieren zu können, wurde ein Template entwickelt, welches für jedes Objekt anwendbar ist. Laut Mo u. a. 2018 sind keine allgemein anerkannten Regeln vorhanden, wie für ein solches Problem ein Template zu erstellen ist. Deshalb würde sich die Aufgabe ein für alle PartNet-Objekte gültiges Template zu kreieren als schwierig gestalten.

Letzten Endes wurde sich für einen hierarchischen *And-Or-Graph* (zu Deutsch: Und-Oder-Graphen) entschieden (Mo u. a. 2018). Demnach wird sich zu Beginn für eine Objektkategorie entschieden. Diese Objektkategorie besitzt nun mehrere Oder-Verbindungen, die auf ein Objekt oder eine Unterkategorie verweisen. Das bedeutet, nur eine der Verbindungen muss gewählt werden. Wurde sich für die Unterkategorie entschieden, stehen die gleichen Wahlmöglichkeiten wie zuvor zur Verfügung. Wird sich jedoch für das Objekt entschieden werden die Und-Verbindungen erreicht. Diese teilen die Objekte in weitere Segmente auf. Die Segmente können wiederum aus Und- oder Oder-Verbindungen bestehen, bis so die unterste Instanz erreicht wurde. Abbildung 2.3 zeigt am Beispiel Lampe, wie diese Umsetzung aussieht.

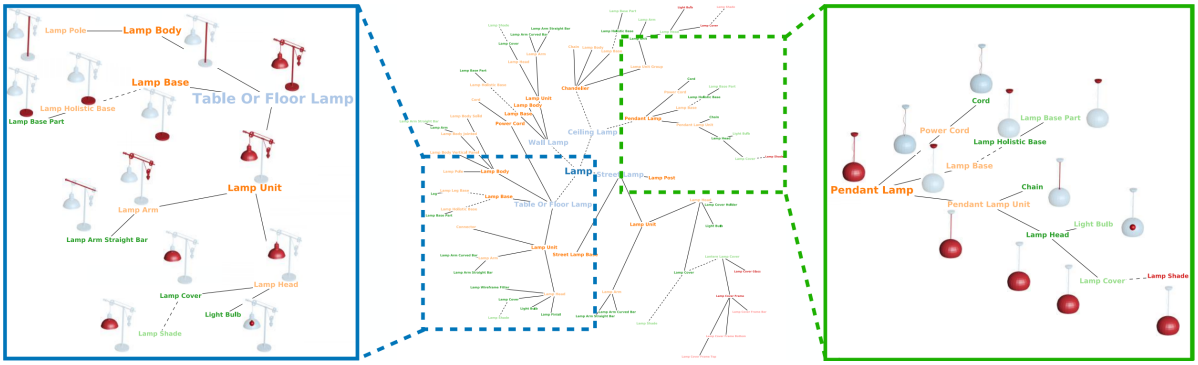


Abbildung 2.3: Hierarchie-Template einer Lampe. Die gestrichelten Linien symbolisieren die Oder-Verknüpfung während die durchgezogenen Linien die Und-Verknüpfungen kennzeichnen (Mo u. a. 2018). Lamp ist die Kategorie, sie besitzt vier Unterkategorien, von denen Ceiling Lamp zwei weitere Unterkategorien besitzt. Das Beispiel Pendant Lamp zeigt, welche Segmente fest vorgegeben sind (Und-Verbindung) und bei welchen Segmenten eine Wahlmöglichkeit besteht (Oder-Verbindung).

3 Technische Konzeption

In diesem Kapitel wird sich kritisch mit verschiedenen Lösungsansätzen des gegebenen Problems auseinandergesetzt. So kann nachempfunden werden, weshalb sich letztlich für die endgültige technische Umsetzung (siehe Kapitel 4) entschieden wurde.

Ziel ist die möglichst gute Erweiterbarkeit der resultierenden Anwendung. Eine Erweiterung sollte einen Benutzer nicht unnötig überfordern. Aus diesem Grund wird die Implementation so nahe wie möglich an den bereits existierenden Funktionen für ShapeNet-Objekten gehalten. Dies soll Unübersichtlichkeit vermeiden und Benutzerfreundlichkeit gewährleisten. Außerdem gilt dies auch als Leitfaden für den Programmcode, der nach Möglichkeit ebenso einheitlich gehalten werden soll. Dadurch kann sich schnell zurechtgefunden werden und ähnliche Module weichen nicht zu stark voneinander ab.

Zur Bewältigung der gestellten Aufgabe bestehen mehrere verschiedene Lösungswege. Dieser Abschnitt geht auf mögliche Lösungswege ein und erläutert, wieso letzten Endes der eingeschlagene Weg gewählt wurde. Um die Erweiterung zu implementieren, sind drei wesentliche Schritte vonnöten;

- erstens die Integration der PartNet-Datenbank,
- zweitens die Repräsentation der PartNet-Datenbank im VAnnotatoR und
- drittens die Platzierung der PartNet-Objekte in der Szene.

Der erste Schritt ist der simpelste, da die Daten genau wie bei den ShapeNet-Daten entweder als ZIP-Datei für die 3D-Objekte und Thumbnails oder als JSON-Datei für die Hierarchie und Eigenschaften der PartNet-Objekte hinterlegt sind. Alle Daten können von Webseiten heruntergeladen werden. Hierbei existiert nicht viel Spielraum. Unity selbst unterstützt die Funktionen *unitywebrequest* (Unity3D 2019b) und *www*, letztere wurde durch *unitywebrequest* abgelöst (Unity3D 2019c). Auch die ShapeNet-Datenbank wurde mithilfe des *unitywebrequests* implementiert. Um das Schreiben einer komplett neuen Bibliothek zum Herunterladen von Dateien zu vermeiden, muss demnach zwischen diesen beiden Optionen entschieden werden. Deshalb fiel die Wahl auf *unitywebrequest*. *Unitywebrequest* erlaubt auch den erwünschten Zugriff auf lokal gespeicherte Dateien. Dies ist hilfreich, um auf den Cache zuzugreifen. Der Cache ist ein lokaler Ordner, der von VAnnotatoR angelegt wird. In diesem werden Dateien wie Thumbnails, 3D-Objekte, Textdokumente und Bilder zu PartNet- und ShapeNet-Objekten gespeichert. So können bereits auf der Festplatte gespeicherte Modelle unabhängig der Internetverbindung geladen werden.

Die Strings der JSON-Dateien (siehe Abschnitt 1.3.4) sind in Arrays gespeichert. Dennoch wird sich letzten Endes dafür entschieden diese in HashSets zu speichern, weil ShapeNet

mithilfe von HashSets realisiert wurde und so Kompatibilität gewährleistet wird. Deshalb wurde entschieden, weiterhin mit HashSets zu arbeiten. Des Weiteren wird für jede JSON-Datei eine entsprechende Datenstruktur angelegt, sodass alle Einträge abgedeckt sind. Diese Datenstrukturen werden anschließend in einem Dictionary unter der passenden ID gespeichert. Da Kategorien keine ID besitzen, ist der Key für das Dictionary ihre Hauptkategorie. Nun sind die benötigten Daten von den entsprechenden Webseiten in das Programm geladen und diese dort in verschiedenen Dictionaries gespeichert.

Der zweite Schritt erlaubt mehr Freiheiten, denn die Repräsentation der PartNet-Daten in der Benutzeroberfläche ist hauptsächlich Geschmackssache. Um die Funktionalität und das Design zu testen, wird eine Evaluation ausgeführt (Nielsen 1994). Diese findet sich im Abschnitt 5.1. In Abbildung 2.1 wurde bereits das Kernstück der Benutzeroberfläche vorgestellt. Um den Benutzer nicht mit unnötigen neuen Optionen zu überwältigen, wurde entschieden PartNet in den vorhandenen *Databrowser* einzuflechten.

Man kann ShapeNet-Objekte generell als PartNet-Objekte ausgeben. Das heißt beim Erstellen eines ShapeNet-Objekts wird dieses durch das PartNet-Objekt gleicher ID ersetzt, sofern dieses über die *has_parts* Eigenschaft verfügt. Indem alle ShapeNet- durch PartNet-Objekte ersetzt werden, könnte dies realisiert werden. Diese PartNet-Objekte würden dann ohne Weiteres als ganzes erstellt werden. Allerdings war dies nicht gewünscht, da die Option offen gehalten werden sollte, weiterhin ohne PartNet arbeiten zu können. Andererseits, kann natürlich auch jedes nicht segmentierte PartNet-Objekt als ShapeNet-Objekt erstellt werden.

Eine weitere Methode wäre einen Schalter, z.b. im *FilterPanel* anzubringen, der einen Wechsel zwischen ShapeNet- und PartNet-Objekten bieten würde. Auch wäre eine weitere Auswahlmöglichkeit im *DataPanel*, nachdem sich im *DataSpaceControl* für ShapeNet entschieden wurde, denkbar. Diese Möglichkeit wurde auch getestet, erwies sich jedoch als unnötig, da das Navigieren zwischen den Bereichen bereits schnell und einfach möglich ist.

Da ShapeNet- und PartNet-Kategorien für Objekte unterschiedlich sind, wurde sich letztlich für die Lösung einer neuen Auswahlmöglichkeit entschieden.

Bei ShapeNet wurde bereits eine Trennung zwischen Thumbnails und den Objekten erstellt. Deshalb schien es angemessen auch ShapeNet und PartNet getrennt zu halten. Durch diese zusätzliche Auswahlmöglichkeit wird zudem sichergestellt, dass die PartNet-Erweiterung eine unabhängige Option ist, die wie gewünscht beim Arbeiten auch ausgelassen werden kann.

Um PartNet-Objekte nun aber direkt aufteilen zu können, wird ein neues Fenster benötigt. Gewiss wäre es denkbar im *DataPanel*, nachdem ein Objekt ausgewählt wurde, die Unterobjekte des Objekts anzuzeigen. Eine schnelle Navigation zwischen verschiedenen Objekten gleicher Kategorie wäre dann jedoch nicht mehr möglich. Dies führte dazu, dass ein komplett neues Menü im Design der bereits vorhandenen Menüs erstellt wurde.

Nun stellt sich die Frage, wie die Unterobjekte eines gewählten Objekts repräsentiert werden sollen. Die PartNet-Objekte sind hierarchisch angelegt, sodass auch im Programm eine hierarchische Präsentation sinnvoll erscheint.

Über einen Öffnen-Knopf könnte die nächste Hierarchieebene erreicht werden oder durch einen Zurück-Knopf die Vorherige. Eine Alternative wäre alle Unterobjekte gleichzeitig anzuzeigen, sodass über die Seiten navigiert werden kann. In beiden Fällen sind Bilder der einzelnen Unterobjekte ein hilfreiches Werkzeug, um sich im aktuellen Objekt zurechtzufinden. Die Bilder sowie die Namen der einzelnen Segmente werden durch die PartNet-Thumbnails zur Verfügung gestellt.

Je weniger Segmente ein Objekt besitzt, desto besser ist die Lösung über die Seiten. Andererseits überzeugt bei besonders detaillierten Objekten die hierarchische Lösung. Doch existiert auch der Fall, dass ein Objekt viele hierarchische Ebenen besitzt, diese jedoch wenige oder sogar nur einzelne Objekte enthalten. Es könnte sein, dass dies den Benutzer unnötig Zeit kosten würde.

Es besteht auch die Möglichkeit, die Objekte noch kleiner aufzuteilen, denn auch ein PartNet-Segment kann aus noch kleineren Objekten bestehen. Diese atomaren Objekte werden, wenn sie nicht auch gleichzeitig Teilobjekte sind, nicht einzeln greifbar oder annotierbar sein. Durch eine Aufteilung der Objekte in ihre kleinsten Bestandteile würde nämlich kein Mehrwert geschaffen. Diese Objekte sind weder annotiert noch können sie dies ordentlich werden. Ein Mensch würde die meisten dieser Objekte, wahrscheinlich als Splitter definieren. Die Möglichkeit wird zwar nicht realisiert, sie könnte aber leicht auch noch im Nachhinein integriert werden.

Im letzten Schritt soll das Platzieren der Objekte realisiert werden. Auch hier ist die Umsetzung bereits durch die gewünschte Einheitlichkeit vorgegeben. Ein PartNet-Objekt soll im Wesentlichen wie ein ShapeNet-Objekt mit Hilfe des *ObjectTabs* erstellt werden. Das bedeutet, das PartNet-Objekt wird im *DataPanel* ausgewählt und erstellt. Daraufhin wird es zum *ObjectTab* getragen und auf dem *Pointer* als GhostObject, analog zu den ShapeNet-Objekten, erstellt.

Neben den beschriebenen Optionen existieren noch drei weitere Ansätze. Einer dieser Ansätze wird im Abschnitt 5.3 näher behandelt, da dieser kombinierbar mit der gewählten Implementation ist. Er handelt von der Idee ein Schneidwerkzeug bereitzustellen, mit dem durch sämtliche Objekte geschnitten werden kann, um hierdurch neue Teilobjekte zu erzeugen.

Der nächste Ansatz wäre gewesen, ein bereits in der Szene vorhandenes ShapeNet-Objekt in der Szene auseinanderzunehmen. Dies könnte bewerkstelligt werden, in dem das Objekt gegriffen und durch eine bestimmte Geste oder per Knopfdruck ein Menü geöffnet wird. Natürlich würde sich das Menü nur öffnen, wenn Segmente für das passende ShapeNet-Objekt hinterlegt sind. Ansonsten könnte eine Fehlermeldung ausgegeben werden, damit sich der Benutzer nicht wundert, warum das Menü nicht erscheint.

Damit nicht beim Ersetzen des ShapeNet-Objekts durch das PartNet-Objekt die bereits verlinkten Annotationen verloren gehen würden, müssten die Objekte ein übergeordnetes Objekt besitzen, welches annotiert wird. Anschließend stünde die Frage im Raum, wie annotiert wird, wenn dem Objekt ein Segment abgetrennt wurde. Auch hier müsste eine eindeutige Regelung gefunden werden, wie eine *PartNetId*.

Nachteil dieses Ansatzes wäre, in einer sehr vollen Szene, in der Gegenstände nah bei einander liegen, müsste das gewünschte Objekt aus der eigentlichen Szene genommen werden. Ein Menü benötigt Platz und gerade in einer vollen Szene mit vielen Verbindungen zwischen Objekten und Text, kann es auch unübersichtlich sein. Des Weiteren ist durch den gegebenen Text von vornherein klar, welche Gegenstände in einer Szene gebraucht werden. Darum würde, wenn der Text ein Segment verlangt, immer die Segmentierung vonnöten sein. Außerdem kämen die PartNet-Kategorien nie zum Einsatz. Es wäre also nicht möglich nach diesen zu sortieren, da immer zuerst ein ShapeNet-Objekt erstellt werden würde. Da die Objekte trotzdem kategorisiert sind, wäre dieser Punkt aber zu vernachlässigen. Aufgrund der angeführten Argumente schien eine Auswahlmöglichkeit vor dem Erstellen des Objektes als sinnvoller.

Der nächste logische Ansatz wäre eine Kombination. So könnte der Benutzer sowohl zu Beginn des Erstellens als auch direkt in der Szene ein PartNet-Objekt segmentieren. Bei einem Fehler beim Erstellen des Objekts könnte so durch die Möglichkeit in der Szene das Objekt noch einmal bearbeiten zu können, dieser wieder ausgeglichen werden. Diese Variante ist die aufwendigste der bereits vorgestellten. Wahrscheinlich würde ein Nutzer sich immer für eine der beiden Erstellungsprozesse entscheiden und früher oder später den jeweils anderen nicht mehr in Betracht ziehen. Dennoch ist sie für die Nutzer der Erweiterung die komfortabelste.

Bei allen Varianten wäre die Beschaffung der Daten und das Endergebnis, dass segmentierte PartNet-Objekte in der Szene verfügbar sind, gleich.

4 Technische Umsetzung

Im Abschnitt 3 wurde erläutert, wie die Erweiterung realisiert werden kann. Dieses Kapitel befasst sich mit der tatsächlichen Umsetzung, wie die PartNet-Erweiterung im VAnnotatoR nun implementiert ist.

4.1 Ausgangslage

VAnnotatoR bietet bereits alle Funktionen für ShapeNet, die auch für PartNet nach Möglichkeit bereitgestellt werden soll. ShapeNet-Objekte können annotiert, platziert, bewegt, skaliert, gespeichert und entfernen werden. Die Daten für ShapeNet- sowie PartNet-Objekte können von der dazugehörigen Seite des Text Technology Labs (`shapenet.texttechnology lab.org/`, Abruf: 18.05.2021) bezogen werden. Im Gegensatz zu den ShapeNet-Objekten verfügen PartNet-Objekte derzeit nicht über Texturen. Auch existiert keine Suchfunktion für PartNet-Teilobjekte.

Unity bietet mit der Webrequest Funktion die Möglichkeit auf Webseiten zuzugreifen, die benötigten Inhalte lokal zu speichern und anschließend zu verwenden. Die ShapeNet-Daten werden somit, während die Applikation ausgeführt wird, heruntergeladen und daraufhin bereitgestellt. Anschließend können über ein Menü die jeweiligen Objekte ausgewählt und in einem weiteren Fenster erzeugt werden. Nun können die ShapeNet-Objekte skaliert und in der Szene platziert werden (Abrami, Henlein, Kett u. a. 2020). Zu guter Letzt bleibt nur die Verbindung der Wörter zu den einzelnen Objekten und das Speichern der Szene. Dies wird im Abschnitt 4.6 noch einmal im Detail für PartNet-Objekte behandelt. Diese Funktionen existieren, wie zuvor erwähnt, bereits für ShapeNet-Objekte. Für PartNet-Objekte sind sie nicht verfügbar. PartNet-Objekte sind lediglich als JSON-Dateien, Bilder und 3D-Objekte auf der Text Technology Lab Seite hinterlegt. Es gilt also diese Funktionen für PartNet zu realisieren.

Ein ShapeNet-Model ist die im VAnnotatoR verwendete Datenstruktur für ShapeNet-Objekte. Das Model verfügt über diverse Informationen wie physische Eigenschaften (z.B. das Volumen oder das Gewicht), Kategorien, in die das Objekt eingeordnet ist, und eine Identifikationsnummer (ID). Die wohl wichtigste Information für diese Arbeit ist jedoch die Angabe, ob ein ShapeNet-Modell über Teile verfügt. In Abbildung 4.1 gibt die **has_parts: true** Zeile an, dass in diesem Fall das Objekt mit dem Namen *Dresser* über Teile verfügt, also **has_parts: true** hat. Dies bedeutet, dass die PartNet-Datenbank unter der entsprechenden ID (in diesem Fall dca4c8bdab8bdfe739e1d6694e046e01) einen Eintrag aufweisen kann. Falls **has_parts: false** ist, gibt es kein PartNet-Objekt zu dem entsprechenden ShapeNet-Objekt.

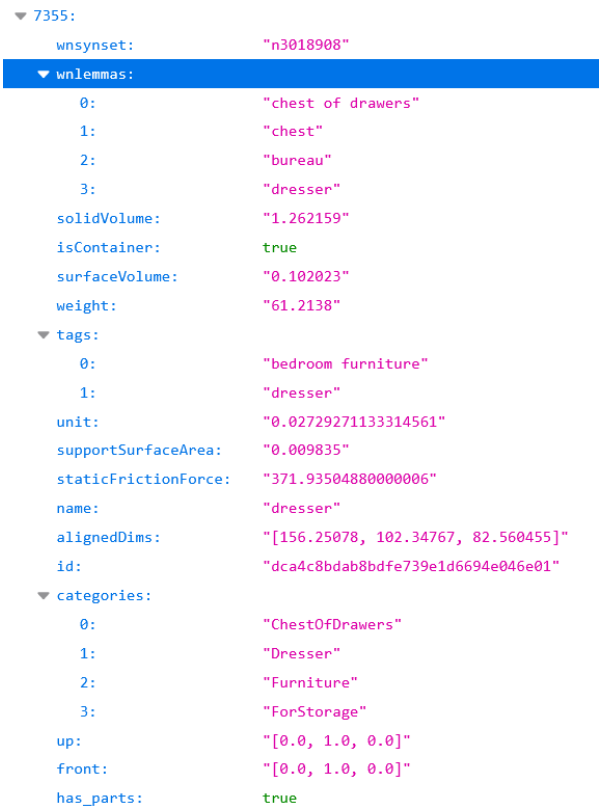


Abbildung 4.1: Zeigt beispielhaft ein ShapeNet-Objekt, welches auf der Text Technologie Lab Seite als JSON-Datei hinterlegt ist: der *Dresser*

4.2 Vorgehensweise

Um zu gewährleisten, dass der ShapeNet-Bereich unverändert funktioniert, wird zu Anfang ein komplett separater Bereich eröffnet, welcher im *DataBrowser* unter PartNet zu finden ist. Anschließend müssen die Daten in das Programm geladen werden. PartNet-Dateien sind größer als ShapeNet-Dateien, was daran liegt, dass die Größe dieser Dateien, maßgeblich von den 3D-Objekten bestimmt wird. Da PartNet aus vielen kleinen Objekten besteht, ist es logisch, dass diese Dateien größer sind. Im Gegensatz zu ShapeNet-Datensätzen, bei denen bei Programmstart sämtliche Daten geladen werden, werden die PartNet-Daten nur auf Abruf heruntergeladen. Dies bewirkt, dass keine langen Wartezeiten entstehen, bevor der *DataBrowser* oder ähnliches benutzt werden kann. Um eine hohe Kompatibilität gewährleisten zu können, werden die Daten in einzelnen Dictionaries abgespeichert, denn auch ShapeNet speichert sämtliche Daten in dazugehörige Dictionaries. Da sowohl die Art der Beschaffung als auch der Output möglichst identisch sein soll, macht diese Herangehensweise Sinn.

Weil jedes vollständige PartNet-Objekt auch als ShapeNet-Objekt vorliegt, teilen sich beide die eindeutige ID: Der *Dresser* hat beispielsweise die ID: dca4c8bdab8bdfe739e1d6694e046e01 und PartNet-Komponenten (ShapeNet 2019). Abbildung 4.1 zeigt die ShapeNet Repräsentation des *Dressers*. Die Abfrage, ob ein ShapeNet-Objekt auch ein PartNet-Objekt ist, ist somit

Dank der gemeinsamen ID und der Eigenschaft **has_parts** unproblematisch. Nun werden das bereits vorhandenen ShapeNet-Model erweitert und erhalten eine Information, das sogenannte PartNetFeature, von ihrem entsprechenden PartNet-Gegenstück. So sind die Daten zentral im ShapeNet-Objekt gespeichert und können in den verschiedenen Interfaces abgerufen werden.

Die PartNet-Objekte sollen des Weiteren so präsentiert werden, dass die einzelnen Teile erkennbar sind und auch jeweils erstellt werden können. Dies wird in einem *DataHierarchyPanel* gewährleistet (siehe Abschnitt 4.3).

4.3 Implementierung

Unity unterstützt zwar die Sprachen C#, JavaScript und Boo, in dieser Arbeit wird allerdings ausschließlich in C# programmiert. Dies liegt nicht nur daran, dass das Unity Manual für C# ausgelegt ist, sondern auch daran, dass das gesamte Projekt in C# gehalten ist. Auch wenn Unity fortlaufend weiterentwickelt wird, ist es nicht sinnvoll das Projekt auf jede neue Version zu aktualisieren. VAnnotatoR wird in der Unity Version 2019.3.0f6 (Unity3D 2019a, Abruf: 01.06.2021) entwickelt.

Da Ziel dieser Arbeit eine Erweiterung von ShapeNet um PartNet ist, ist das wichtigste Skript das *ShapeNetInterface*-Skript. Dieses dient dazu mit `http://shapenet.texttechnologylab.org/` zu kommunizieren und so die Daten von der Webseite in das Programm zu laden. Sobald das Programm ausgeführt wird, werden die benötigten Daten von der Webseite automatisch geladen und ShapeNet lädt sofort sämtliche nicht gecachten Daten runter. PartNet schließt sich dem an, lädt jedoch weder die einzelnen Objekte noch ihre Thumbnails herunter. Der Grund hierfür ist, dass dies schlichtweg sehr lange dauern würde, da die Daten wesentlich größer sind als die äquivalenten von ShapeNet. Stattdessen werden diese heruntergeladen, sobald der Benutzer das entsprechende PartNet-Objekt anklickt. Für einzelne Objekte ist das Herunterladen so schnell, dass der Benutzer kaum eine Verzögerung registrieren dürfte. Die heruntergeladenen Objekt-Strukturen werden in entsprechenden Datenstrukturen gespeichert. Diese sind: *PFeature*, *PStructure*, *PTaxanomy*, *PTCategories* und *ShapeNetModel*. Jedes PartNet-Objekt benötigt einen Eintrag in jeder dieser Datenstrukturen. Da die Daten im *ShapeNetInterface* heruntergeladen werden, werden sie dort auch befüllt. Anschließend werden die Daten im *ShapeNetModel*-Skript als Verweis hinterlegt.

Das *DataPanel* wird um die PartNet-Komponente erweitert. Im *ShapeNetInterface* wird über den Boolean partNet gesteuert, wie sich sämtliche Funktionen, die PartNet- und ShapeNet-Objekte enthalten, verhalten. Wenn sich der Benutzer nun im PartNet-Bereich befindet, werden alle Kategorien im *FilterPanel* zu PartNet-Kategorien gewechselt. Auch die Suchfunktion wird nun ausschließlich PartNet-Objekte ausgeben. Da die Suchfunktion jedoch auf ShapeNet-Objekte ausgelegt ist und über das Web funktioniert, gibt es keine Suche nach PartNet-Segmenten. Nachdem das erste Objekt im *DataPanel* angeklickt wurde, wird die Variable partNet auf true gesetzt. Nach einem weiteren klickt auf dasselbe Objekt, erscheint das *DataHierarchyPanel*. Dieser zusätzliche Klick ist leider notwendig, um das Objekt einzeln her-

unterzuladen. Dem *DataHierarchyPanel* wird mithilfe der ID des ausgewählten Objekts die Daten eben jenes Objekts übermittelt. Nun kann der *DataHierarchyPanel* sämtliche Segmentierungen des Objekts als Text und Bild anzeigen. Die einzelnen Objekte werden zwar hierarchisch im *PFeature*-Skript gespeichert, es wurde jedoch darauf verzichtet eine hierarchische Darstellung im *DataHierarchyPanel* zu benutzen (siehe Abschnitt 3). Links neben den einzelnen Segmenten erscheint nun ein Kästchen, welches markiert werden kann. Ist es markiert, so wird eine *PartNetID* im entsprechenden *ShapeNetModel*-Skript erzeugt. Diese wird beim Übergeben an den *ObjectTab* wieder in die einzelnen Bestandteile aufgeteilt. Mithilfe eines Dictionarys wird die aufgeteilte *PartNetID* den ausgewählten Objekten zugewiesen, sodass der *ObjectTab* die richtigen Objekte erstellt. Da keine weiteren Daten vorliegen, werden die Objekte mit denselben Daten wie die der entsprechenden ShapeNet-Objekte erstellt. Auch existieren keine Texturen für PartNet-Objekte. Die Funktion, die für die Zuteilung der Texturen zuständig ist, wurde um einen Sonderfall erweitert, der aktuell nichts bewirkt. Demnach ist eine nachträgliche Zuweisung von Texturen bereits vorbereitet. Die einzelnen Objekte, die im *ObjectTab* kreiert werden, werden unter einem *GatherObject* gesammelt. Dieses dient dazu, dass die einzelnen Objekte auch zusammen in der Szene bewegt, skaliert und rotiert werden können. Ihm wird auch das *InteractiveShapeNetObject*-Skript angehängt, damit Annotationen möglich sind.

Da das *DataPanel* nun die PartNet-Komponente und den Boolean partNet enthält, kann PartNet leicht von ShapeNet separiert werden. Für die Funktionen, die zwischen PartNet und ShapeNet unterscheiden müssen, wurde immer eine Abfrage und eine neue Variante für PartNet hinzugefügt. So bleiben die Funktionen von ShapeNet unberührt und funktionieren weiterhin wie zuvor.

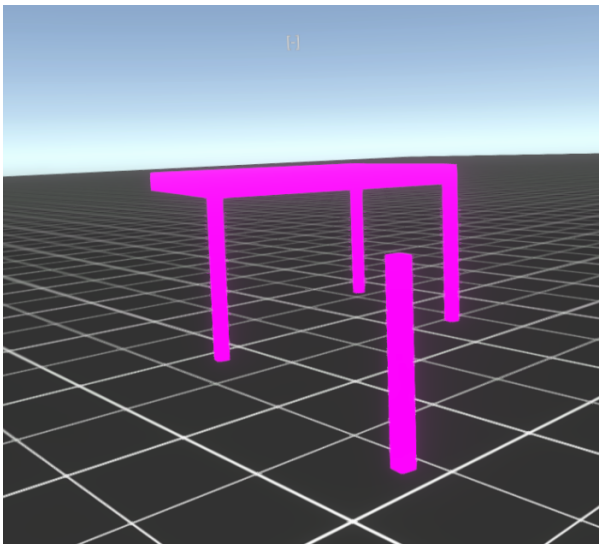


Abbildung 4.2: Ein Tisch dessen Bein mittels des neuen Annotationwerkzeugs abgetrennt und separat abgestellt wurde

4.4 Stolpersteine und Tests

Dieser Abschnitt handelt nicht, wie der Titel vielleicht vermuten lässt, von dem Kunstprojekt von Gunter Demnig. Stolperwege ist ein Teil des VAnnotatoRs und von seinem Projekt inspiriert. Vielmehr handelt es von den Problemen, auf die gestoßen, und den Tests, mit welchen sie aufgespürt wurden.

Zum Testen in Unity wird der Debugger, hauptsächlich aber die von Unity bereitgestellte Debug.Log Funktion, benutzt. Letztere gibt eine selbstgeschriebene Fehlermeldung auf der *Console* aus.

Alle Daten werden in Dictionaries gespeichert. Dies war anfangs kontraintuitiv, da die Ausgangsdateien JSON-Dateien sind und in diesen wird Sämtliches als Array gespeichert. Mit der Add-Funktion trat nun folgender Fehler auf: Selbst bei nur zwei Einträgen blieb der erste Eintrag aus. Die Lösung des Problems wurde allerdings mehr oder minder direkt auf der Microsoft Seite zu Dictionaries in den Hinweisen angegeben (Studios 2019, Abruf: 04.06.2021). Einzelne Einträge werden direkt überschrieben und somit kann auf die Add-Funktion verzichtet werden. Die Tests hierfür waren denkbar einfach; das Dictionary wurde über eine Foreach Schleife ausgegeben, um zu überprüfen, ob alle Einträge vorhanden sind.

Der nächste Widerstand bot sich beim Anlegen der Datenbank der PartNet-Objekte. Die PartNet-Objekte sind in Kategorien abgespeichert, jedoch nicht auf dieselbe Art und Weise wie die ShapeNet-Objekte. ShapeNet-Objekte haben eine Hauptkategorie, in der sie zu finden sind. Zusätzlich besitzen sie verschiedene Unterkategorien (engl. *subcategories*), sodass ein ShapeNet-Objekt z.B. eine Hauptkategorie und drei Unterkategorien besitzen kann. PartNet-Objekte hingegen sind so gespeichert, dass jedes Teilobjekt maximal in einer (Unter-)Kategorie zu finden ist. Meiner Meinung nach ist dies die bessere Vorgehensweise, obgleich ein Objekt gelegentlich in mehrere Kategorien passt. Das Problem entstand dadurch, dass die Entwickler der Datenbank bedauerlicherweise in verschiedenen Hauptkategorien einzelne Unterkategorien identisch benannt haben. So finden sich beispielsweise mehrere (Unter-)Kategorien mit dem Namen *foot*, beispielsweise einmal unter *dishwasher* und einmal unter *trash_can*. Die Kategorien sind zwar hierarchisch, demnach befinden sich die mehrfach benannten Kategorien nicht im gleichen Abschnitt, doch sind hierdurch bei der Anpassung an die ShapeNet Implementation Fehler aufgetreten, die dazu führten, dass das Programm nicht mehr funktionierte. Da nur mit einem Teil der Datenbanken sowohl von ShapeNet als auch von PartNet gearbeitet wird, konnte dieses Problem umgangen werden. Die Lösung war hierfür die Dictionaries zusammenzuschließen. Eleganter wäre jedoch, wenn die Kategorien nicht nur *foot* hießen, sondern z.B. *dishwasher_foot*, dann wäre die Zuordnung eindeutig. Alternativ könnte *foot* auch eine eigenständige Hauptkategorie mit neuen Unterkategorien werden. So würde keine Information verloren gehen.

Ein weiteres Problem war, dass viele der Kategorien leer sind, also keine Objekte enthalten. Diese leeren Kategorien führen zu unnötigem Speichern der Kategorien. Dass die Kategorien leer sind, ist wohl dem ShapeNetSem geschuldet (A. X. Chang u. a. 2015). Da ShapeNetSem

nur ein Teil von ShapeNet ist, wurden vermutlich fälschlicherweise Kategorien aus größeren Teildatenbanken übernommen, die für die aktuell in ShapeNetSem enthaltenen Objekte nicht nötig wären. Vielleicht sollten diese Kategorien aber auch eine Vorbereitung auf die Erweiterung von ShapeNetSem sein. Leere Kategorien sind zudem nicht benutzerfreundlich. Um dieses Problem zu vermeiden, wurde eine Abfrage eingeführt, welche vorab prüft, ob eine Kategorie leer ist, und falls dies der Fall ist, wird diese Kategorie ignoriert. In Abbildung 4.3 ist zu erkennen, dass die *subcategory foot* doppelt vorkommt und dass die *subcategory leg_base* zwar weitere *subcategories* hat, aber keine Objekte enthält und somit die folgenden Unterkategorien ebenfalls leer sind. Interessanterweise löst diese ist_leer-Abfrage auch das vorherige Problem, denn alle mehrfach benannten Kategorien sind zufallsbedingt leer.

Da das Dictionary, welches die Taxonomy speichert, direkt eine Fehlermeldung auswarf, dass der Key bereits vergeben war, benötigte es für den ersten Fehler, dass es zwei Kategorien mit dem gleichen Namen gab, keine weiteren Tests. Der zweite Fehler, der durch die leeren Kategorien entstand, wurde durch stichprobenartiges Auswählen der einzelnen Kategorien im *Filterpanel* entlarvt. Wurde eine leere Kategorie ausgewählt, stürzte das Programm ab. Nachdem der Fehler behoben war, wurden zum Vergleich mit den Daten auf `http://shapenet.texttechnologylab.org/partTaxonomy` die Dictionaries ausgegeben.

Das größte Problem ist jedoch, dass im Gegensatz zu ShapeNet, PartNet nicht umfangreich annotiert ist (siehe Abschnitt 2.2). Ursprünglich wurde davon ausgegangen, dass die ShapeNet-Objekte und PartNet-Objekte sowohl über die gleiche Skalierung als auch Ausrichtung verfügen. Dies führte dazu, dass die in absoluten Zahlen kleineren PartNet-Objekte unskaliert (Skalierung = 1) zu groß sind, denn der Skalierungsfaktor der ShapeNet-Objekte ist immer zwischen 0 und 1 (siehe Abbildung 4.1, die *unit* ist der Skalierungsfaktor, in diesem Beispiel liegt er bei $\approx 0{,}027$). Das bedeutet das ShapeNet-Objekt wird vor dem Erstellen geschrumpft. Werden Objekte gleicher ID um den gleichen Faktor skaliert, so sind PartNet-Objekte viel zu klein bis teilweise nicht mehr erkennbar. Demnach muss noch ein passender Skalierungsfaktor gefunden werden. Dieser Skalierungsfaktor muss außerdem bekannt sein, bevor Segmente eines PartNet-Objekts platziert werden. Um einen Skalierungsfaktor zu errechnen, müssen die Bounds des PartNet-Objekts mit dem des ShapeNet-Objekts gleichgesetzt werden. Die Bounds legen einen minimal großen Quader um ein 3D-Objekt fest. Dieses besitzt den Abstand der auf den jeweiligen Achsen zueinander weit entferntesten Punkte als jeweilige Kantenlänge. Durch Streckung der Bounds der PartNet-Objekte auf die absolute Länge der Bounds der ShapeNet-Objekte sollte dies den Skalierungsfaktor ergeben. Die übergeordneten Quader für PartNet-Objekte und deren richtige Skalierungsfaktoren konnten zwar erstellt werden, doch fehlt es noch an einem geeigneten Speichersystem. Deshalb stimmt die Skalierung der PartNet-Objekte nur, wenn vorher das entsprechende ShapeNet-Objekt platziert wurde. Da dieses Problem den Ursprung in der Datenbank hat, wurde sich damit zwar auseinandergesetzt, gelöst wurde es jedoch nur teilweise. Meiner Meinung nach sollten alle 1.881 ShapeNet-Objekte, die über **has_parts** verfügen, platziert werden, um so die richtigen Bounds zu speichern. So wie die *unit* der einzelnen ShapeNet-Objekte für jedes Objekt unterschiedlich ist, ist die PartNetUnit auch nicht immer dieselbe für PartNet-Objekte unterschiedlicher ID. Anschließend wird der minimale Wert von Bounds.size mit der jeweiligen *unit* multipliziert. Dies ergibt die richtige PartNetUnit. Diese muss dann extern ge-

speichert werden, sodass sie nicht bei jedem Neustart des Programms neu errechnet werden muss. Jetzt kann auch noch das neue Zentrum der PartNet-Objekt errechnet werden, da dieses wie die Skalierung nicht mit dem des entsprechenden ShapeNet-Objekts übereinstimmt. Momentan stehen die errechneten Werte beim Laden einer Szene nicht mehr zur Verfügung.

ShapeNet-Objekte verfügen über einen FrameCube, dieser entspricht den skalierten Bounds. Wird ein PartNet-Objekt segmentiert, ist dieser FrameCube selbst für richtig skalierte und positionierte PartNet-Objekte logischerweise zu groß. Da jedoch zum Errechnen der Skalierung PartNet über eine Funktion verfügt, die die Bounds für segmentierte Objekte berechnet, kann der FrameCube leicht hieran angepasst werden. Dafür muss jedoch zuerst die Skalierung stimmen, sonst sind auch segmentierte Objekte zu klein.

Je nachdem welches PartNet-Objekt erstellt wurde, benötigte es keinen Test, denn die Objekte selbst sind winzig bis nicht mehr sichtbar.

Abbildung 4.3: Ausschnitt der PartNet Taxonomy

4.5 Dokumentation der Quelltexte

Viele der verschiedenen Skripte weisen Parallelen im Code auf. Um den Umfang dieser Arbeit nicht zu sprengen werden exemplarisch einige ausgewählt.

Der Code zum Herunterladen der JSON-Dateien ist vom Aufbau immer ähnlich, deshalb wird als Beispiel die PartNet Taxonomy herangezogen, um den Code zu erklären.

LoadPTaxonomy() ist eine Funktion des ShapenetInterface.cs. In der 3. Zeile wird der UnityWebRequest mit der zugehörigen Webseite vorbereitet und in der darauffolgenden Zeile gesendet. Die anschließende If-Anweisung prüft, ob es einen Netzwerkfehler gegeben hat. Falls dies nicht der Fall ist, wird die JSON-Datei heruntergeladen. Auch hier wird überprüft, ob die Datei die gewünschten Inhalte beinhaltet und falls ja, dann wird sie gespeichert (siehe Abschnitt 2.3). In Zeile 18 werden die Taxanomien erzeugt und direkt im Anschluss befüllt. Zeile 23 dient der Abfrage, ob die Taxanomie überhaupt ein Objekt besitzt. Wenn dies nicht der Fall ist, wird die zugehörige Kategorie nicht im *FilterPanel* angezeigt, weil dem Nutzer keine leeren Kategorien angezeigt werden sollen. Auch soll der Kategoriename nicht doppelt vorkommen. Damit keine Daten verloren gehen, wurden die Kategorien mit gleichen Namen zusammengeführt.

Wenn beide Voraussetzungen erfüllt sind, dann werden die Kategorien im entsprechenden Dictionary gespeichert. Dieses Dictionary kann nun wiederum an den *FilterPanel* übergeben werden, sodass die PartNet Kategorien angezeigt werden und diese auch die richtigen Objekte ausgeben, wenn sie ausgewählt wurden.

```
1  private IEnumerator LoadPTaxonomy()
2  {
3      request = UnityWebRequest.Get(GET_P_TAXO);
4      yield return request.SendWebRequest();
5      if (request.isNetworkError || request.isHttpError)
6          _pobjectTaxonomyError = request.error;
7      else
8      {
9          data = JsonMapper.ToObject(request.downloadHandler.text);
10         if (!data.Keys.Contains("success") ||
11         !bool.Parse(data["success"].ToString()) ||
12         !data.Keys.Contains("taxonomy"))
13         {
14             _pobjectTaxonomyError = "Downloading PartObject list failed.";
15             yield break;
16         }
17         objectList = data["taxonomy"]["taxonomy"];
18         pTaxonomys = new PTaxonomy[objectList.Count];
19         for (int i = 0; i < objectList.Count; i++)
20         {
21             pKeys = new List<string>(objectList[i].Keys);
22             pTaxonomys[i] = new PTaxonomy(pKeys[0], objectList[i]);
23             if (!pTaxonomys[i].leer &&
24             !PModelMainCategories.ContainsKey(pTaxonomys[i].CategorieName))
25             {
26                 PModelTaxonomies.Add(pTaxonomys[i].CategorieName, pTaxonomys[i]);
27                 PModelMainCategories.Add(pTaxonomys[i].CategorieName,
28                 InteractiveCheckbox.CheckboxStatus.AllChecked);
```

```
29              }
30          }
31      }
32  }
```

Die Umwandlung der JSON-Datei zu der entsprechenden Datenstruktur wird am Beispiel des *PFStructure* Skripts erläutert. Um eine *PFStructure* zu erzeugen, wird JsonData und ein Dictionary benötigt. Die JsonData wird z.B. in der **LoadPTaxonomy** Funktion in Zeile 17 bereitgestellt und in Zeile 22 übergeben. Natürlich handelt es sich in Zeile 22 nicht um eine *PFStructure* sondern um eine *PTaxonomy*. Das übergebene Dictionary wird im *PFeature* Skript erstellt und an die *PFStructure* weitergereicht. Benötigt wird dies, um später aus der *PartNetID* (siehe Kapitel 4) die zu erstellenden Objekte abzuleiten. In den darauf folgenden Zeilen werden die Schlüssel der JSON-Datei gespeichert. Alle Objekte der aktuellen Struktur werden in Zeile 23 bis 28 in einem HashSet gespeichert. Außerdem werden dem übergebenen Dicitonary die Id der Struktur und das Objekte HashSet übergeben. Die Struktur Id dient zur Identifizierung eines Segments des PartNet-Objekts. Nun muss nur noch überprüft werden, ob die Struktur über weitere Unterstrukturen verfügt. Dies geschieht in Zeile 9. Ist das der Fall, so wird rekursiv eine weitere Struktur erstellt.

```
1  public PFStructure(JsonData json, Dictionary<int, HashSet<string>> idToObjs)
2      {
3          Ori_id = json.Keys.Contains("ori_id") ?
4          Convert.ToInt32(json["ori_id"].ToString()) : 0;
5          Name = json.Keys.Contains("name") ? json["name"].ToString() : "";
6          Text = json.Keys.Contains("text") ? json["text"].ToString() : "";
7          Id = json.Keys.Contains("id") ? Convert.ToInt32(json["id"].ToString()) : 0;
8
9          if (json.Keys.Contains("children"))
10         {
11             childrenList = json["children"];
12             Children = new PFStructure[childrenList.Count];
13             for (int i = 0; i < childrenList.Count; i++)
14             {
15                 Children[i] = new PFStructure(childrenList[i], idToObjs);
16             }
17
18         }
19         else  // Wenn die Struktur keine weiteren Unterstrukturen enthält
20         {
21             Children = null;
22         }
23         Objs = new HashSet<string>();
24         if (json.Keys.Contains("objs"))
25         {
26             for (int i = 0; i < json["objs"].Count; i++)
27                 Objs.Add(json["objs"][i].ToString().ToLower());
28         }
```

```
29          idToObjs.Add(Id, Objs);
30      }
```

Da im *ObjectTab* die Objekte erzeugt werden, muss auch dieser angepasst werden. In Zeile 1 wird über die *PartNetId* iteriert. Jedes Segment von PartNet besitzt eine eigene ID. Um die *PartNetId* zu erzeugen, werden alle Segment-IDs sortiert an die ShapeNet-ID geheftet. Die *PartNetId* ist durch „!“ separiert, sodass die verschiedenen IDs ausgelesen werden können. Diese werden im *ObjectTab* wieder getrennt, um so die einzelnen Objekte auszulesen. Im ShapeNetModel (hier _shapeNetObject) finden sich die Referenzen für die einzelnen atomaren Objekte, die ein Segment ausmachen. Diese werden in Zeile 9 geladen und erstellt. Zeile 11, sammelt alle Objekte ein und heftet sie an ein *gatherObject*. Dies erlaubt die Handhabung verschiedener Segmente als wären sie ein einzelnes Objekt.

```
1  for(int i = 1; i < idSplit.Length; i++)
2  {
3      foreach(string s
4      in _shapeNetObject.PartNet.idToObjs[Int32.Parse(idSplit[i])])
5      {
6          if (!gatherObject.OriObjs.ContainsKey(s))
7          {
8              GameObject obj =
9              ObjectLoader.LoadObject(path +
10             "\\" + s + ".obj", "part");
11             obj.transform.SetParent(gatherObject.Gatherer.transform);
12             if(!gatherObject.OriObjs.ContainsKey(s))
13                 gatherObject.OriObjs.Add(s, obj);
14             else
15                 continue;
16         }
17 }
```

4.6 Anwenderdokumentation

Im Folgenden wird davon ausgegangen, dass der Leser bereits Erfahrung im Umgang mit VAnnotatoR und in der Interaktion mit ShapeNet-Objekten besitzt. Auf detaillierte Erläuterung zur Bedienung der VR-Brille und -Controllern sowie das Starten des Programms wird daher verzichtet.

Im VAnnotatoR wird der *DataBrowser* wie gehabt über die linke Armbanduhr geöffnet. Im *DataBrowser* wird im *DataSpacePanel* ShapeNet ausgewählt. Daraufhin erscheinen drei neue Auswahlmöglichkeiten im *DataPanel*. Vor der PartNet-Erweiterung existierten hier nur zwei Auswahlmöglichkeiten; ShapeNet Models und ShapeNet Textures. Um PartNet-Objekte auswählen zu können, wird die neue Möglichkeit PartNet Models per Klick selektiert. PartNet-Objekte werden genau wie ShapeNet-Objekte verwendet. Der sich jetzt öffnende *FilterPanel*, zur Rechten, beinhaltet ausschließlich Kategorien des PartNets. Auch kann in der Such-Leiste nach Objekten gesucht werden. Wurde entschieden, welches Objekt in der Szene platziert

werden soll, hält man den Zeigefinger-Knopf gedrückt und der gewohnte kleine Kubus entsteht, welcher gegriffen werden kann. Dieser Kubus muss nun wie gehabt in den *ObjectTab* platziert werden, um das PartNet-Objekt in der Szene zu erstellen. Skalieren, Rotieren und Speichern der Objekte funktioniert wie bei den ShapeNet-Objekten.

Um einzelne Objekte aus dem PartNet-Objekt zu lösen, kann das Objekt im *DataPanel* anvisiert und angeklickt werden. Daraufhin erscheint das Objekt im *DataHierarchyPanel*, in dem eine Liste der Segmente des Objekts aufgeführt wird. Außerdem zeigt das *DataHierarchyPanel* an, welche Unterobjekte das Objekt besitzt, ohne dass es vorher platziert werden muss. An dieser Stelle sollte darauf hingewiesen werden, dass wenn das Objekt 0 im *DataHierarchyPanel* ausgewählt ist, sämtliche nachfolgenden Häkchen irrelevant sind. Das Objekt 0 ist immer das gesamte PartNet-Objekt mit allen Segmenten und somit äquivalent zum ShapeNet-Objekt. Wenn also nur ein einzelnes Segment ausgewählt werden soll, so muss das Häkchen bei Objekt 0 entfernt und beim gewünschten Segment ausgewählt werden. Demnach bewirkt der Select All-Knopf, dass alle Objekte außer das 0. nicht ausgewählt werden.

5 Ergebnisse

In diesem abschließenden Kapitel wird sich kritisch mit dem Erreichten auseinandergesetzt und die Resultate der erfolgten Evaluation aufgeführt (Abschnitte 5.1 und 5.2). Außerdem wird meine eigene Meinung und ein Ausblick in die Zukunft des VAnnotatoRs dargeboten (Abschnitte 5.3 und 5.4).

5.1 Evaluation

Die Evaluation soll darüber Auskunft geben, wie Benutzer mit der programmierten Erweiterung zurechtkommen. Aufgrund der aktuellen COVID-19 Situation kamen jedoch nur wenige in den Genuss, sie zu testen. Da eine VR-Brille benötigt wird, um die Erweiterung zu testen, konnte nur vor Ort getestet werden. Die Universität selbst konnte keine Möglichkeit bieten die Evaluation durchzuführen, weshalb nur Familie, Bekannte und die Arbeitsgruppe als Versuchspersonen in Frage kamen. Erstrebenswert wären unter normalen Bedingungen Versuchspersonen, die bereits Erfahrung mit VAnnotatoR oder zumindest im Umgang mit VR-Programmen besitzen. Dies war leider zum größten Teil nicht der Fall. Aus diesen Gründen ist fraglich, ob diese Evaluation aussagekräftig ist. Trotz allem war auch Kritik von wenigen wünschenswert, um einen neutralen Blick zu gewinnen.

Für diese Arbeit wurde ein UMUX-Fragebogen zur Evaluation verwendet (Finstad 2010a). Die vier Kriterien, die die Versuchspersonen bewertet haben, sind folgende:

1) The annotation tool's capabilities meet my requirements.
   Die Möglichkeiten des Annotationwerkzeuges genügen meinen Ansprüchen.

2) Using the annotation tool is a frustrating experience.
   Die Benutzung des Annotationwerkzeuges ist eine frustrierende Erfahrung.

3) The annotation tool is easy to use.
   Das Annotationwerkzeuges ist leicht zu bedienen.

4) I have to spend too much time correcting things with the annotation tool.
   Ich muss zu viel Zeit investieren, Sachen des Annotationwerkzeuges zu korrigieren.

Diese Kriterien zielten auf die Kernaspekte der Benutzerfreundlichkeit ab: Effektivität, Effizienz und Zufriedenheit. Hierbei wurde auf einer Likert Skala von eins bis sieben eine Wertung abgegeben, wobei eins *strongly disagree* und sieben *strongly agree* entsprach (Finstad 2010b). Bevor die Teilnehmer sich der eigentlichen Aufgabe stellen mussten, wurden sie von mir in einer kleinen Beispiel-Szene an VAnnotatoR und die neue Erweiterung herangeführt. Sobald alle Grundfunktionen bekannt waren, wurde jeder Teilnehmer gebeten eine leere Szene zu

befüllen. Damit am Ende eine vergleichbare Szene entstehen konnte, bekamen alle Teilnehmer die gleichen Sätze zum Umsetzen:

> Als die Ermittler den Raum betraten, bot sich ihnen ein seltener Anblick. Auf dem Tisch lagen verschiedene Klingen jeweils ohne Griff, der Mörder musste seine Waffen wohl selbst zusammenbauen. Zum Transport seines Werkzeuges hatte er eine Louis Vuitton Tasche benutzt. Der Henkel, der am Tatort gefunden wurde, passte zu jener seltenen Tasche in der Ecke, an dieser fehlte nämlich einer.

Der gewählte Text stellte sicher, dass alle Versuchspersonen das Auseinandernehmen und Platzieren von Teilobjekten mehrmals durchführen mussten.

Sowohl die Einverständniserklärung als auch der Fragebogen finden sich im Anhang dieser Arbeit (siehe Abschnitt 5.4).

5.2 Analyse

Dadurch, dass die meisten Versuchspersonen leider keinerlei Erfahrung im Umgang mit VAnnotatoR hatten, war es für sie schwer den Unterschied zwischen PartNet und ShapeNet festzustellen und daher nur die Erweiterung zu bewerten. Die VR-Neulinge hatten außerdem sichtliche Schwierigkeiten sich in der VR zurechtzufinden.

Getestet wurde von fünf Personen. Wie bereits erwähnt, reicht das nicht aus, um eine aussagekräftige Auswertung zu erstellen. Dennoch kann eine Tendenz abgelesen werden. Anhand der gesammelten Fragebögen wurde für jede Frage ein Tortendiagramm erzeugt. Desto dunkler das Grün desto positiver die Bewertung. Desto dunkler das Rot desto negativer. Eine neutrale Bewertung wird durch die Farbe Grau widergespiegelt. Abbildung 5.1 zeigt demnach ein überwiegend positives Ergebnis.

- Das erste Kriterium ergab einen Durchschnittswert von 5 „stimme eher zu“.
- Das zweite Kriterium ergab einen Durchschnittswert von 3 „stimme eher nicht zu“.
- Das dritte Kriterium ergab einen Durchschnittswert von 6 „stimme zu“.
- Das vierte Kriterium ergab einen Durchschnittswert von 2,6 zwischen „stimme nicht zu“ und „stimme eher nicht zu“.

Im Schnitt benötigten die Versuchspersonen 10 min und 44 Sek, um die ihnen gestellte Aufgabe zu lösen. Die schnellste Versuchsperson war nach 6 min und 4 Sek fertig, die langsamste nach 19 min und 13 Sek.

Am Rande: Da die Versuchspersonen freie Wahl hatten, wie detailliert sie die Szene gestalten, variieren die Zeiten selbstverständlich auch. Alle weiblichen Tester lagen zeitlich über dem Schnitt, während alle männlichen Tester darunter lagen. Ob dies am besseren zurechtfinden in der VR liegt oder daran, dass die Szenen der weiblichen Tester detaillierter waren, lässt sich anhand der Daten leider nicht erkennen.

Lediglich 20% der Befragten empfanden das Werkzeug als eine leicht frustrierende Erfahrung. Für 60% war die Erfahrung positiver Natur. Eines der wesentlichen Ziele, nämlich

die einfache Bedienung der Erweiterung, ist laut Evaluation gelungen. Das dritte Kriterium wurde ausschließlich positiv bewertet, 20% der Befragten wählten sogar die größtmögliche Punktzahl („stimme absolut zu“). Jedes Kriterium wurde durchgehend mit mindestens 60% positiv oder besser bewertet. Demnach ist die Erweiterung aus Sicht der Befragten, leicht zu bedienen und somit ein Erfolg.

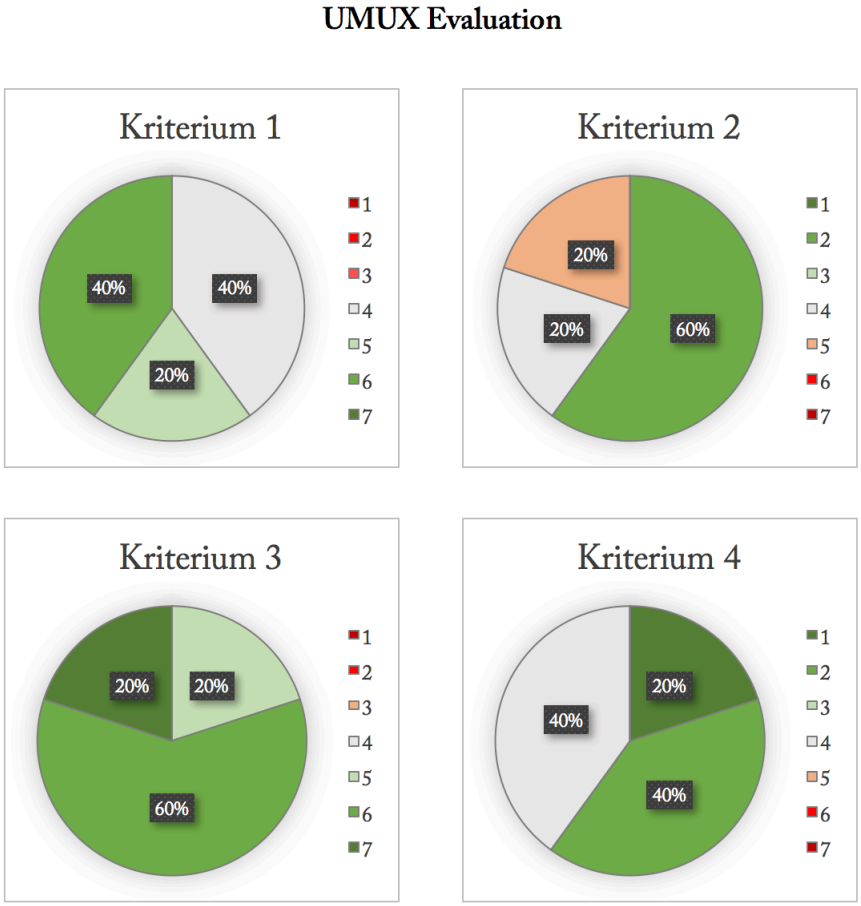


Abbildung 5.1: Auswertung des UMUX-Fragebogens aller Versuchspersonen

5.3 Weiterführende Arbeit

Zu Beginn stand die Frage im Raum, ob die Erweiterung fähig sein sollte einzelne Objekte zu zerlegen, unabhängig davon, ob dieses aus Einzelteilen besteht. Dies sollte über eine zerschneide Funktion geschehen, die so funktioniert, dass eine Ebene in ein Objekt gelegt wird, welche dann bestimmt, wo das Objekt zerteilt wird. Von dieser Idee wurde sich jedoch aus den nachfolgenden Gründen distanziert.

Was wäre der Nutzen einer solchen Funktion? Der Nutzer wäre dadurch in der Lage einen Schnitt zu simulieren, demnach könnte ein Text wie z.B. „Der Holzfäller spaltet Holz, um

anschließend ein Lagerfeuer mit den gewonnenen Scheiten zu errichten“, nun mithilfe des Tools annotiert und die entsprechende Szene hergestellt werden. Das gleiche Ergebnis kann auch erzielt werden, wenn das Holzscheit bereits in der Datenbank des VAnnotatoRs wäre. Einen Gegenstand so zu zerteilen ist zudem in der Realität ein eher seltenes unterfangen. Dieses System wäre des Weiteren äußerst umständlich, wenn beispielsweise ein einzelnes Blütenblatt mit einer Ebene von einer Blume getrennt werden soll oder generell bei sämtlichen Objekten, wo die Einzelteile relativ dicht beieinander sind. Natürlich erschließt dieses Tool auch ein neues interessantes Feld; das Schneiden einer Kiwi z.B. könnte Aufschluss über das Innenleben der Frucht bieten und neue Möglichkeiten der Annotation freilegen. Woraufhin die Schnittfläche automatisch grün eingefärbt werden könnte. Im System müsste jedoch demnach die innere Beschaffenheit aller Objekte hinterlegt sein, sodass je nach Schnitt (horizontal, vertikal, diagonal) eine akkurate Darstellung der Innenfläche geboten werden kann. Damit müsste viel Aufwand für die jeweiligen Spezialfälle betrieben werden. Zusätzlich wird anschließend noch ein System benötigt, in welchem die so neu erschaffenen Objekte gespeichert werden. Ohne das Speichern der neuen Objekte würde dieses Werkzeug wenig Sinn ergeben. Zum Speichern müsste dann abgewogen werden, wird das entstandene Objekt unter der gleichen ID mit Suffix gespeichert (oder entsteht jeweils eine ganz neue ID wie dies bei der *PartNetId* der fall ist (siehe Abschnitt 4.5)) und wie wird verfahren, wenn mehrere solcher Schnitte an demselben Objekt durchgeführt werden? Und noch grundlegender, wenn ein Objekt in zwei geteilt wird, werden die zwei Teile getrennt voneinander gespeichert? Schon alleine das Kategorisieren solcher Objekte ist eine nahezu unlösbare Aufgabe, wenn man bedenkt, dass selbst PartNet keine elegante Lösung für das Kategorisieren seiner Objekte gefunden hat. Außerdem ist eine Szene durch einen Text oder umgekehrt beschrieben. Damit fällt die Notwendigkeit eines Werkzeuges zum aktiven Erstellen in einer Szene weg. So würde nämlich aus einer passiven plötzlich eine aktive Szene entstehen. Die Szenen, die mithilfe des VAnnotatoRs erstellt werden, sind, auch wenn mit der Szene interagiert werden kann, passiver Natur. Des Weiteren wäre eine Implementation von PartNet redundant. Diese Funktion könnte nämlich auch ohne weiteres auf ShapeNet-Objekte angewandt werden. Demnach wäre dies eine eigenständige Arbeit.

Aus diesen Gründen erscheint dieser Ansatz wenig sinnvoll und würde des Weiteren deutlich den Rahmen dieser Arbeit sprengen. Daher ist meiner Meinung nach die elegantere Lösung die Datenbank der bereits vorhandenen Objekte um PartNet-Objekte zu erweitern, die bereits gespeichert und mit IDs versehen sind. Mit genügend Zeit und Arbeitsaufwand wäre gegebenenfalls eine Mischung aus beiden Zerlegungsmöglichkeiten ideal für den Nutzer. Vielleicht wäre auch eine mögliche Lösung dem Benutzer zu gestatten eigene Objekte zu erstellen und zu integrieren.

Auch sollte die Suchfunktion des *DataPanels* für PartNet-Segmente implementiert werden. Momentan funktioniert sie zwar für PartNet-Objekte, allerdings nur für ganze. Wird beispielsweise ein Tischbein gesucht, wird die Suchanfrage aktuell leer ausgehen.

5.4 Fazit

Die Ergänzung und Segmentierung der PartNet-Objekte ist erreicht (siehe Abbildung 4.2). Einzelne Segmente können im *DataHierarchyPanel* ausgewählt und anschließend platziert werden. Ebenso können für diese Segmente jegliche Aktionen ausgeführt werden, die auch für ShapeNet-Objekte existieren.

Der Grundstein für PartNet ist gelegt. Trotzdem ist diese Erweiterung nicht vollkommen. Ohne Texturen gestaltet sich der Gebrauch der Objekte zudem als schwer. Die fehlenden Texturen bewirken bei manchen Objekten, dass mit bloßem Auge überhaupt nicht unterschieden werden kann, ob dem Objekt Segmente fehlen. Des Weiteren muss eine generelle Speicherung der Skalierung für PartNet-Objekte erstellt werden, da beim Laden einer Szene diese Information nicht zur Verfügung steht. Deshalb werden geladene Objekte immer wieder zu klein erstellt.

Meiner Meinung nach sollte außerdem eine bessere und allgemeingültige Kategorisierung für alle Objekte gefunden werden. Da Datenbanken in der Regel immer wieder erweitert werden, ist es wichtig, dass neue Objekte sich nahtlos in bestehende Muster einfügen. Es sollte zumindest im Hinterkopf behalten werden, dass ein Werkzeug für Schnitte durch Objekte eine interessante Erweiterung ist. Neue Objekte zu kreieren kann gerade bei annotierbarem Vokabular, das für Sätze nicht zur Verfügung steht, einen Vorteil gegenüber anderer Text2Scene Programme geben.

Literaturverzeichnis

Gibson, William (1984). *Neuromancer / William Gibson.* eng. New York: Ace Books. ISBN: 0441569595.

Lakoff, George (1987). *Women, Fire and Dangerous Things: What Categories Reveal About the Mind.* Chicago: University of Chicago Press. ISBN: 978-0-226-46803-7.

Machover, C. und S.E. Tice (1994). „Virtual reality". In: *IEEE Computer Graphics and Applications* 14.1, S. 15–16. DOI: 10.1109/38.250913.

Nielsen, Jakob (1994). „Usability Inspection Methods". In: *Conference Companion on Human Factors in Computing Systems.* CHI '94. Boston, Massachusetts, USA: Association for Computing Machinery, S. 413–414. ISBN: 0897916514. DOI: 10.1145/259963.260531. URL: https://doi.org/10.1145/259963.260531.

Miller, George A. (Nov. 1995). „WordNet: A Lexical Database for English". In: *Commun. ACM* 38.11, S. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748.

Mazuryk, Tomasz und Michael Gervautz (1999). *Virtual Reality - History, Applications, Technology and Future.*

Liddy, Elizabeth D (2001). „Natural language processing". In:

Chowdhury, Gobinda G. (2003). „Natural language processing". In: *Annual Review of Information Science and Technology* 37.1, S. 51–89. DOI: https://doi.org/10.1002/aris.1440370103. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/aris.1440370103. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440370103.

Finstad, Kraig (2010a). „The Usability Metric for User Experience". In: *Interacting with Computers* 22.5. Modelling user experience - An agenda for research and practice, S. 323–327. ISSN: 0953-5438. DOI: https://doi.org/10.1016/j.intcom.2010.04.004. URL: https://www.sciencedirect.com/science/article/pii/S095354381000038X.

– (Mai 2010b). „The Usability Metric for User Experience". In: *Interacting with Computers* 22.5, S. 323–327. ISSN: 0953-5438. DOI: 10.1016/j.intcom.2010.04.004. eprint: https://academic.oup.com/iwc/article-pdf/22/5/323/1992916/iwc22-0323.pdf. URL: https://doi.org/10.1016/j.intcom.2010.04.004.

Furht, Borko (2011). *Handbook of Augmented Reality.* Springer Publishing Company, Incorporated. ISBN: 1461400635.

Chang, Angel, Will Monroe, Manolis Savva, Christopher Potts und Christopher D. Manning (2015). „Text to 3D Scene Generation with Rich Lexical Grounding". In: *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP).*

Chang, Angel X. u. a. (2015). „ShapeNet: An Information-Rich 3D Model Repository". In: *CoRR* abs/1512.03012. arXiv: 1512.03012. URL: http://arxiv.org/abs/1512.03012.

Hemati, Wahed, Tolga Uslu und Alexander Mehler (2016). „TextImager: a Distributed UIMA-based System for NLP“. In: *Proceedings of the COLING 2016 System Demonstrations*. Federated Conference on Computer Science und Information Systems. Osaka, Japan.

TTIC, Stanford University und Princeton University (2016). *ShapeNet Database*. https://shapenet.org/.

Mehler, Alexander, Giuseppe Abrami, Steffen Bruendel u. a. (Juli 2017). „Stolperwege - An App for a Digital Public History of the Holocaust“. In: DOI: 10.1145/3078714.3078748.

Song, Shuran u. a. (2017). „Semantic Scene Completion from a Single Depth Image“. In: *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*.

Abrami, Giuseppe und Alexander Mehler (2018). „A UIMA Database Interface for Managing NLP-related Text Annotations“. In: *Proceedings of the 11th edition of the Language Resources and Evaluation Conference, May 7 - 12*. LREC 2018. Miyazaki, Japan.

Ma, Rui u. a. (2018). „Language-driven synthesis of 3D scenes from scene databases“. In: *SIGGRAPH Asia 2018 Technical Papers*. ACM, S. 212.

Mehler, Alexander, Giuseppe Abrami, Christian Spiekermann und Matthias Jostock (Juli 2018). „VAnnotatoR: A Framework for Generating Multimodal Hypertexts“. In: DOI: 10.1145/3209542.3209572.

Mo, Kaichun u. a. (2018). „PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding“. In: *CoRR* abs/1812.02713. arXiv: 1812.02713. URL: http://arxiv.org/abs/1812.02713.

Spiekermann, Christian, Giuseppe Abrami und Alexander Mehler (2018). „VAnnotatoR: a Gesture-driven Annotation Framework for Linguistic and Multimodal Annotation“. In: *Proceedings of the Annotation, Recognition and Evaluation of Actions (AREA 2018) Workshop*. AREA. Miyazaki, Japan.

Abrami, Giuseppe, Christian Spiekermann und Alexander Mehler (2019). „VAnnotatoR: Ein Werkzeug zur Annotation multimodaler Netzwerke in dreidimensionalen virtuellen Umgebungen“. In: *Proceedings of the 6th Digital Humanities Conference in the German-speaking Countries, DHd 2019*. DHd 2019. Frankfurt, Germany.

Ballhaus, Werner (2019). *Studie: Deutscher Virtual-Reality-Markt wächst über die Nische hinaus, PwC Germany*. https://www.pwc.de/de/technologie-medien-und-telekommunikation/studie-deutscher-virtual-reality-markt-waechst-ueber-die-nische-hinaus.html.

Mehler, Alexander und Giuseppe Abrami (10.–11. Okt. 2019). „VAnnotatoR: A framework for the multimodal reconstruction of historical situations and spaces“. In: *Proceedings of the Time Machine Conference*. Dresden, Germany.

Mo, Kaichun u. a. (2019). *PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding*. https://partnet.cs.stanford.edu/.

ShapeNet, TexttechnologyLab (2019). *Dresser PartNet Feature, PartNet*. http://shapenet.texttechnologylab.org/getPartFeature?id=dca4c8bdab8bdfe739e1d6694e046e01.

Studios, Microsoft (2019). *Dictionary, Microsoft*. https://docs.microsoft.com/de-de/dotnet/api/system.collections.generic.dictionary-2.add?view=net-5.0.

TextTechnologyLab (2019a). *TextAnnotator*. http://www.textannotator.texttechnologylab.org/.

– (2019b). *TextAnnotator, TextTechnologyLab Goethe-Universität Frankfurt am Main*. https://www.texttechnologylab.org/applications/textannotator/.

TextTechnologyLab (2019c). *TextImager, TextTechnologyLab Goethe-Universität Frankfurt am Main*. https://www.texttechnologylab.org/applications/textimager/.
– (2019d). *VAnnotatoR, TextTechnologyLab Goethe-Universität Frankfurt am Main*. https://www.texttechnologylab.org/applications/vannotator.
Unity3D (2019a). *Unity3D, Scripting Manual*. https://docs.unity3d.com/2019.3/Documentation/ScriptReference/.
– (2019b). *unitywebrequest, Unity3D*. https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Networking.UnityWebRequest.html.
– (2019c). *WWW, Unity3D*. https://docs.unity3d.com/2019.3/Documentation/ScriptReference/WWW.html.
Abrami, Giuseppe, Alexander Henlein, Attila Kett und Alexander Mehler (2020). „Text2SceneVR: Generating Hypertexts with VAnnotatoR as a Pre-processing Step for Text2Scene Systems“. In: *Proceedings of the 31st ACM Conference on Hypertext and Social Media*. HT '20. Virtual Event, USA: Association for Computing Machinery, S. 177–186. ISBN: 9781450370981. DOI: 10.1145/3372923.3404791. URL: https://doi.org/10.1145/3372923.3404791.
Erl, Josef (2020). *Die Geschichte der Virtual Reality*. https://mixed.de/virtual-reality-geschichte/#:~:text=Basierend%20auf%20den%20Erkenntnissen%20aus,in%20der%20Forschung%20eingesetzt%20werden..
Wachowskis (2020). *The Matrix, Warner Bros. Entertainment Inc*. https://www.warnerbros.com/movies/matrix.
Abrami, Giuseppe, Alexander Henlein, Andy Lücking u. a. (Juni 2021). „Unleashing annotations with TextAnnotator: Multimedia, multi-perspective document views for ubiquitous annotation“. In:
Glick, Dave (2021). *LitJSON*. https://litjson.net/.
Go, Culture to (2021). *Augmented Reality im Museum*. http://culture-to-go.com/mediathek/augmented-reality-im-museum/.
Malavida (2021). *Yobi3D*. https://yobi3d.de.malavida.com/webapps/.
Trimble, Inc (2021). *3D Warehouse*. https://3dwarehouse.sketchup.com/.
Unity3D (2021). *Unity3D, official website*. https://unity.com/.
Verdu, Mike (2021). *From Bear to Bull: How Oculus Quest 2 Is Changing the Game for VR*. https://www.oculus.com/blog/from-bear-to-bull-how-oculus-quest-2-is-changing-the-game-for-vr/.

Tobias Felden
E-Mail: s9645211@stud.uni-frankfurt.de

Nutzerstudie zur Annotation Teil-bezogener Objekte Informationen

UMUX-Fragebogen

VP _____

Zur Auswertung des Annotationstools wird die *Usability Metric for User Experience* (**Finstad:2010**) verwendet. UMUX umfasst die folgenden vier Fragen, die Sie bitte per Ankreuzen jeweils eines Feldes beantworten.

1. The annotation tool's capabilities meet my requirements.

1	2	3	4	5	6	7
Strongly Disagree						Strongly Agree

2. Using the annotation tool is a frustrating experience.

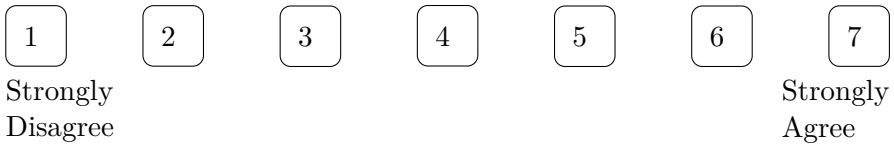
1	2	3	4	5	6	7
Strongly Disagree						Strongly Agree

3. The annotation tool is easy to use.

1	2	3	4	5	6	7
Strongly Disagree						Strongly Agree

[bitte wenden]

4. I have to spend too much time correcting things with the annotation tool.



Tobias Felden
E-Mail: s9645211@stud.uni-frankfurt.de

Nutzerstudie zur Annotation Teil-bezogener Objekte Informationen

UMUX-Fragebogen Unterschrift

Hiermit bestätige ich, an der Nutzerstudie zur Annotation rhetorischer Relationen teilgenommen und den UMUX-Fragebogen ausgefülllt zu haben.

Datum, Unterschrift des Probanden:

Tobias Felden
E-Mail: s9645211@stud.uni-frankfurt.de

Nutzerstudie zur Annotation Teil-bezogener Objekte Informationen

Einverständniserklärung für Probanden

Texttechnologie, Goethe-Universität Frankfurt

Frankfurt am Main, 29. Juli 2021

Beschreibung der Studie

Die Nutzerstudie zur Annotation Teil-bezogener Objekte Informationen evaluiert die Performanz des entwickelten Interfaces anhand einer Beispielannotation. Die Studie beinhaltet einen Eingewöhnungsdurchgang, eine Beispielannotation sowie einen kurzen Fragebogen. Es geht allein um die Verwendung des Annotationstools. Die Interpretation von rhetorischen Relationen zwischen Textsegmenten ist nicht Teil der Studie, weshalb die Annotation anhand bereits nach rhetorischen Relationen analysierter Texte geschieht.

Während der Annotation werden Click-Ereignisse und die benötigte Zeit aufgezeichnet. Der Teilnehmer wird nicht gefilmt oder auf andere Weise aufgezeichnet.

Im Anschluss wird der Fragebogen anonym ausgefüllt. Es werden keine personenbezogenen Daten erhoben.

Die in der Studie gesammelten Daten und die Fragebogenergebnisse werden im Rahmen der Bachelorarbeit von Tobias Felden ausgewertet und nicht an Dritte weitergeleitet.

Vielen Dank für Ihre Teilnahme an dieser Studie!

Name und Vorname des Versuchsteilnehmers: _____

Ich nehme freiwillig an der Studie teil und wurde über den Inhalt, die Vorgehensweise und die Risiken der Studie in verständlicher Form aufgeklärt. Darüber hinaus habe ich eine Kopie der Versuchsteilnehmerinformationen erhalten. Meine Fragen wurden ausreichend und verständlich beantwortet. Ich hatte genügend Zeit, mich gegen eine Teilnahme an der Studie

zu entscheiden und willige hiermit freiwillig in diese ein. Ich kann meine Einverständniserklärung jederzeit mit Wirkung für die Zukunft widerrufen.

Die aufgenommen Daten werden gespeichert und statistisch ausgewertet. Ich erkläre mich damit einverstanden, dass die im Rahmen des Experiments erhobenen Daten im Rahmen wissenschaftlicher Publikationen veröffentlicht und z. B. auf Tagungen verwendet werden dürfen.

Datum, Unterschrift des Probanden

Name des Versuchsleiters: _____

Ich bestätige, dass alle Details der Untersuchung erklärt, dem o.g. Probanden mitgeteilt und von ihm verstanden wurden.

Datum, Unterschrift des Versuchsleiters
