Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 118

Gestensteuerung von Visualisierungen im Museumskontext

Rene Trefft

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Dr. Michael Krone
Beginn am:	12. 07. 2016
Beendet am:	11. 01. 2017
CR-Nummer:	H.5.2, I.3.6

Kurzfassung

Kostengünstige Technologien und Geräte wie die Microsoft Kinect oder Leap Motion haben eine berührungslose Mensch-Computer-Interaktion erstmals für die breite Masse verfügbar gemacht. Das Ziel dieser Interaktionsform ist es, eine natürliche Bedienung von Anwendungen zu ermöglichen, die ohne jegliche Vorkenntnisse auskommt. In dieser Masterarbeit wird MegaMol, eine Visualisierungssoftware für partikelbasierte Daten, um eine berührungslose Steuerung erweitert. Diese bietet sich an, da MegaMol zukünftig in Wissenschafts-Ausstellungen eingesetzt werden soll, in denen kein Personal zur Aufsicht und Unterstützung zur Verfügung steht. Die Steuerung wird sowohl mit der Kinect für Xbox One als auch Leap Motion realisiert. Es werden u. a. Handgesten implementiert, mit denen die Position der virtuellen Kamera verändert werden kann. Weiterhin wird unter Verwendung der 2D-Zeichenbibliothek NanoVG eine Benutzeroberfläche entwickelt, die eine Modifikation von beliebigen Parametern einer Visualisierung erlaubt. Die Interaktion erfolgt durch einen Cursor, der ebenso gestisch gesteuert wird. Mehrere Benutzer bzw. Hände werden unterstützt, sodass durch verschiedene Cursor gleichzeitig Parameteränderungen hervorgerufen werden können. Auch ist ein kombinierter Einsatz beider Geräte möglich. Zur Konfiguration der Benutzeroberfläche kommt das Datenformat XML zum Einsatz. Verschiedene Gestaltungsmöglichkeiten werden bereitgestellt. Es wird eine Schnittstelle für Geräte zur Bewegungssteuerung definiert, die von jedem unterstützten Gerät implementiert werden muss. Auf diese Weise kann die Steuerung zukünftig leicht um weitere Geräte ergänzt werden. In einer qualitativen Benutzerstudie werden die erzielten Ergebnisse abschließend evaluiert. Insbesondere werden dabei beide Geräte bzw. deren Implementierungen verglichen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

NUI	Natural User Interface
VISUS	Visualisierungsinstitut der Universität Stuttgart
UML	Unified Modeling Language
OpenGL	Open Graphics Library
XML	Extensible Markup Language
HP	Hewlett Packard
SDK	Software Development Kit
VIS	Institut für Visualisierung und Interaktive Systeme

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

Bereits seit vielen Jahren befasst man sich auf dem Gebiet der Mensch-Computer-Interaktion mit der Entwicklung von Natural User Interfaces (NUIs). Unter diesem Begriff werden Benutzerschnittstellen zusammengefasst, die eine Interaktion mit Computersystemen so natürlich wie möglich gestalten sollen [Chr12]. Neben der Sprachbedienung stellt die Verwendung von Gesten ein Schwerpunkt im Bereich der NUIs dar [Kra15]. Blickt man beispielsweise auf die Bedienungskonzepte von Smartphones und Tablets, so sind gestenbasierte NUIs allgegenwärtig. Eine grundlegende Technologie, die maßgeblich zum Erfolg dieser Geräte beigetragen hat, nennt sich Multi-Touch. Hierunter versteht man die gleichzeitige Erfassung mehrerer Fingerberührungen, sodass deutlich mehr intuitive Gesten entworfen werden können. Eine populäre Multi-Touch-Geste ist beispielsweise das voneinander wegbewegen zweier Finger, um ein Bild zu vergrößern.

Schon Jahrzehnte vor dem Durchbruch der Multi-Touch-Technologie durch das Apple iPhone im Jahr 2007 [Sch13] befasste man sich mit der räumlichen Erfassung von Händen und Fingern. Erste Prototypen, wie beispielsweise der Sayre Glove, entstanden in den siebziger Jahren [Kra15]. Viele dieser Lösungen kamen jedoch aufgrund der fehlenden Alltagstauglichkeit und hohen Anschaffungskosten vorwiegend im Umfeld von Forschung und Entwicklung zum Einsatz [Kra15].

Mit der Leap Motion, die seit Mitte 2013 kommerziell vertrieben wird [Lea13], existiert ein Gerät, dass auch für einen privaten Einsatz geeignet ist. Die Handhabung, dargestellt in Abbildung 1.1, gestaltet sich als sehr einfach, da der Benutzer lediglich eine Hand über das Gerät positionieren muss, worauf diese unmittelbar erfasst wird [Kra15]. Kontinuierlich werden daraufhin verschiedene Daten der Hand und deren Finger, insbesondere Position und Orientierung, berechnet und Anwendungen zur Verfügung gestellt [Kra15]. Ein physischer Kontakt mit der Hardware findet nicht statt, sodass eine natürliche Benutzbarkeit gewährleistet ist [Kra15].

Bereits Ende 2010 [Che10] wurde die Kinect veröffentlicht, die nicht nur Hände, sondern den gesamten Körper von Personen verfolgen kann. Daten zu einzelnen Fingern können im Gegensatz zur Leap Motion jedoch nicht bereitgestellt werden. Das Gerät wurde zur Steuerung der Spielekonsole Xbox 360 entwickelt. Entsprechende Spiele können so über Körperbewegungen bedient werden. Zusammen mit der Xbox One

Abbildung 1.1: Benutzung der Leap Motion [Loc15]

ist Ende 2013 [MH13] dann eine komplett überarbeitete Version des Geräts erschienen, die Kinect für Xbox One. Insbesondere wurde die integrierte Kamera deutlich optimiert, sodass umfangreichere Daten zuverlässiger bereitgestellt werden können. Wie auch bei der Leap Motion ist die Nutzung, veranschaulicht in Abbildung 1.2, sehr intuitiv: Benutzer positionieren sich in einem angemessenen Abstand vor dem Gerät, worauf dessen Bewegungen zur Interaktion eingesetzt werden. Häufig wird das Gerät über dem Fernseher aufgestellt. Ein Einsatz für Anwendungen am PC wird über einen Adapter ermöglicht.

Geräte zur Bewegungssteuerung haben vielfältige Anwendungsbereiche. Die Kinect bzw. Kinect für Xbox One wird beispielsweise auch zum virtuellen Anprobieren von Kleidung eingesetzt [Fit16]. Das Gerät verfolgt die Bewegungen des Benutzers, um in Echtzeit Kleidungsgegenstände auf dem Bild des Körpers zu platzieren. Ein beliebtes Beispiel für die Leap Motion ist das Spielen von virtuellen Instrumenten, z. B. einem Klavier [Lea16d], durch die entsprechenden Hand- bzw. Fingerbewegungen. Häufig werden Geräte dieser Art auch zum Steuern von Gegenständen (z. B. Robotern) oder virtuellen 3D-Modellen eingesetzt.

Abbildung 1.2: Benutzung der Kinect für Xbox 360 [Smi13]. Der Einsatz der zweiten Generation erfolgt auf die selbe Weise.

In dieser Arbeit soll ein Visualisierungs-Framework für Partikeldaten, MegaMol[1] genannt, um eine Gestensteuerung mittels Leap Motion und Kinect für Xbox One erweitert werden. Zukünftig soll die am Visualisierungsinstitut der Universität Stuttgart (VISUS) entwickelte Software in wissenschaftlichen Ausstellungen und Museen eingesetzt werden, in denen kein Aufsichtspersonal zur Verfügung steht. Hierbei hat sich eine berührungslose Steuerung als eine angemessene Lösung herausgestellt. Neben der Steuerung der virtuellen Kamera, die auf eine Visualisierung zeigt, sollen weiterhin beliebige Parameter der Visualisierung gestenbasiert modifiziert werden können. Für letzteres soll eine leicht anpassbare Benutzeroberfläche entwickelt werden. Die Benutzerinteraktion soll keine Vorkenntnisse benötigen und lediglich mit wenig Erklärungen oder Piktogrammen vermittelt werden können. Mittels einer Benutzerstudie soll die Gestensteuerung abschließend evaluiert werden. Insbesondere sollen hierbei die Implementierungen für die Leap Motion und Kinect für Xbox One verglichen werden.

[1] http://megamol.org

1.1 Gliederung der Arbeit

In Kapitel 2 werden zunächst Einführungen in verschiedene Themenbereiche gegeben, die für diese Arbeit relevant sind. Kapitel 3 gibt einen Überblick über verschiedene Arbeiten, in denen ebenfalls eine berührungslose Steuerung von 3D-Modellen realisiert worden ist. Kapitel 4 beschreibt die Konzeption von Kamerasteuerung, Benutzeroberfläche und weiterer Gesten, die in dieser Arbeit umgesetzt worden sind. Die technische Umsetzung erfolgt daraufhin in Kapitel 5. Nach Abschluss der Implementierung werden die Ergebnisse in Kapitel 6 evaluiert. Hier wird auch auf die Benutzerstudie eingegangen, die gegen Ende der Arbeit durchgeführt worden ist. Abschließend wird in Kapitel 7 die Arbeit zusammengefasst und Anregungen für Weiterentwicklungen gegeben.

1.2 Notationen

Im weiteren Verlauf der Arbeit werden häufig Richtungen oder Rotationen durch Achsen eines Koordinatensystems beschrieben. Es wird an dieser Stelle festgelegt, dass die X-Achse von links nach rechts, die Y-Achse von unten und nach oben und die Z-Achse aus der Tiefe nach vorne im Raum zeigt. Ein solches rechtshändiges Koordinatensystem wird sowohl von der Leap Motion als auch Kinect für Xbox One verwendet. Bei einem zweidimensionales Koordinatensystem (Ebene) wird die X-Achse von links nach rechts und die Y-Achse von unten nach oben definiert.

In Abschnitt 5 werden UML-Klassendiagramme zur Veranschaulichung des Entwurfs eingesetzt. Aus Gründen der Übersichtlichkeit werden Funktionen, die in einer Schnittstelle definiert worden sind, in deren Implementierungen nicht erneut aufgeführt. Weiterhin werden einige Funktionen ausgelassen, die im Rahmen anderer Funktionen erläutert oder als nicht wichtig eingestuft werden. Hilfsfunktionen werden generell nicht dargestellt.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vermittelt, die für das Verständnis der folgenden Kapitel erforderlich sind. Abschnitt 2.1 behandelt zunächst die Visualisierungssoftware, die um eine berührungslose Gestensteuerung erweitert wird. Die Abschnitte 2.2 und 2.3 gehen anschließend auf die beiden Geräte ein, mit denen die Steuerung realisiert wird. Damit die Daten dieser Geräte genutzt werden können, sind Filter erforderlich. Abschnitt 2.4 widmet sich diesem Thema. Die Benutzeroberfläche wird mittels einer Zeichenbibliothek realisiert. Diese wird in Abschnitt 2.5 vorgestellt.

2.1 MegaMol

MegaMol ist eine System-Software zur Forschung auf dem Gebiet der Visualisierung von Partikeldaten [GKM+15]. Es ist aus einem gemeinsamen Forschungsprojekt zwischen Biologen, Physikern, Materialwissenschaftlern und Visualisierungsexperten hervorgegangen, die alltäglich mit großen Datenmengen dieser Art arbeiten [GKM+15]. Durch flexible Datenstrukturen und einer modularen Architektur lässt sich MegaMol einfach an sich ändernde Forschungsfragestellungen anpassen [GKM+15]. Beispielsweise können Physiker Feststoffe untersuchen während Chemiker eine Analyse von Makromolekülen wie z. B. Proteinen durchführen [GKM+15]. MegaMol ist kein klassisches Visualisierungssystem, sondern ein Framework, das eine prototypische Entwicklung von partikel-basierten Visualisierungen erlaubt [GKM+15]. Insbesondere arbeitet es hardwarenah, sodass die Fähigkeiten von GPUs umfassend ausgenutzt werden [GKM+15]. Es ist jedoch nicht nur ein Rendering Toolkit, da es auch Funktionalitäten zur Datenverwaltung sowie wiederverwendbare Komponenten bereitstellt [GKM+15]. Entwickler können sich so auf die Implementierung von Visualisierungsalgorithmen fokusieren [GKM+15]. Gleichzeitig stehen bereits verschiedene Algorithmen zur Verfügung, was eine kurze Feedbackschleife zwischen Benutzer und Entwickler ermöglicht [GKM+15].

Die Architektur von MegaMol kann auf zwei Abstraktionsebenen betrachtet werden. Die obere Ebene, dargestellt in Abbildung 2.1, bezieht sich auf Komponenten, die durch ausführbare Programme oder dynamische Programmbibliotheken (DLLs) bzw.

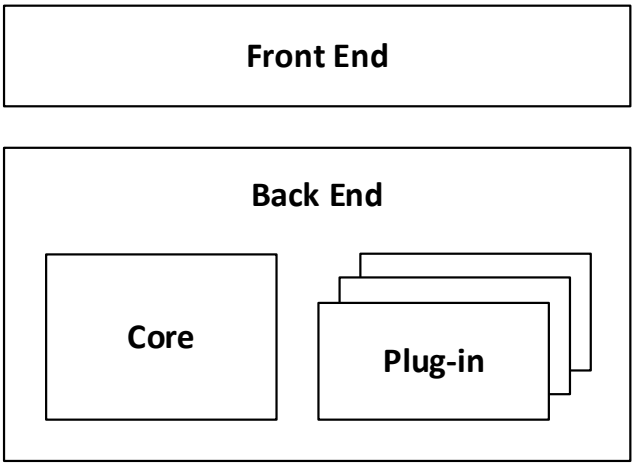

Abbildung 2.1: Architektur von MegaMol (obere Abstraktionsebene) nach [GKM+15]

gemeinsam genutzte Bibliotheken (SOs) repräsentiert werden [GKM+15]. MegaMol besteht aus einer Core-Komponente, Plug-ins bzw. Erweiterungen und einem Front-End. Die Core-Komponente stellt gemeinsam genutzte Verwaltungsfunktionalitäten bereit [GKM+15]. Hierzu gehört das Laden und die Verwaltung von Anwendungskonfigurationen und Plug-ins [GKM+15]. Weiterhin bildet sie die Schnittstelle zwischen Back-End und Front-End [GKM+15]. Auch stehen eine Reihe von Hilfsfunktionen zur Verfügung, z. B. zur Erstellung und Verwaltung von GPU-Ressourcen wie Shader, Texturen und Buffer-Objekten [GKM+15]. Die Benutzerschnittstelle wird durch das Front-End umgesetzt [GKM+15]. Als Hauptprogrammiersprache kommt C++ zum Einsatz [GKM+15]. Da die Core-Komponente eine C API bereitstellt, können Front-Ends in verschiedenen Sprachen realisiert werden [GKM+15]. Neben einigen domänenspezifischen Front-Ends existiert ein generisches C++ Front-End, MegaMol Console genannt, das mittels GLFW 3[2] und AntTweakBar[3] realisiert wurde [GKM+15]. Dieses Front-End wurde in der Arbeit verwendet. MegaMol ist auf Microsoft Windows und Linux lauffähig, sodass Open Graphics Library (OpenGL) als primäre Grafik-API verwendet wird [GKM+15].

Die untere Abstraktionsebene beschreibt die Anwendungslogik mittels funktionaler Komponenten, den Modulen und Calls, die durch Klassen repräsentiert sind. Module stellen die Bausteine der Anwendungslogik dar [GKM+15]. Ihre funktionale Granularität ist nicht global spezifiziert [GKM+15]. Ein Entwickler kann eine Funktionalität (z. B. ein Rendering-Algorithmus) folglich durch ein einzelnes Modul realisieren oder auf mehrere aufteilen, die miteinander verbunden sind [GKM+15].

Ein Modul wird durch einen Call mit einem anderen Modul verbunden [GKM+15]. Genauer gesagt wird hierbei ein Caller Slot mit einem Callee Slot verbunden [GKM+15].

[2] http://www.glfw.org
[3] http://anttweakbar.sourceforge.net

In Abbildung 2.3 sind Caller Slots durch rote, Callee Slots durch grüne Dreiecke visualisiert. Mehrere Caller Slots können auch mit dem selben Callee Slot verbunden werden [GKM+15]. Bei der Registrierung eines Caller bzw. Callee Slots in einem Modul muss der kompatible Call definiert werden [GKM+15]. Weiterhin muss bei einem Callee Slot angegeben werden, wie Aufrufabsichten auf Funktionen des Moduls abgebildet sind [GKM+15]. Diese Absichten sind im Call definiert [GKM+15]. Der Datenaustausch erfolgt auf folgende Weise: Zunächst holt sich das Caller-Modul das Objekt des Calls, dass mit dem Caller Slot verbunden ist [GKM+15]. Daraufhin werden Eingabedaten im Call-Objekt hinterlegt und schließlich eine oder mehrere Absichten aufgerufen [GKM+15]. Auf der Seite des Callees wird für jede Absicht die entsprechende Funktion aufgerufen [GKM+15]. Diese Funktion ruft die Eingabedaten aus dem Call-Objekt ab, verarbeitet diese und speichert abschließend die Ergebnisse wieder im Call-Objekt [GKM+15]. Der Caller kann schließlich auf die Ergebnisse im Call-Objekt zugreifen [GKM+15]. Dieser Ablauf entspricht dem Pull Pattern [GKM+15]. In Call-Objekten kann beliebige Logik enthalten sein [GKM+15]. Beispielsweise könnte man Calls auch zum Konvertieren von Daten verwenden [GKM+15].

Module und Calls werden in Plug-ins zusammengefasst. In dieser Arbeit wird ein neues Plug-in für die Gestensteuerung erstellt. Die Benutzeroberfläche wird als Modul, nicht als Front-End realisiert (siehe Abschnitt 5.3).

Module können Parameter besitzen, die über die API der Core zurückgegeben und verändert werden können. Für folgende Datentypen stehen entsprechende Parametertypen zur Verfügung: String, Integer, Float, Boolean und Enumeration. Weiterhin gibt es spezielle Typen für Vektoren, Dateipfade und ternäre Logik, die durch erstere Typen realisiert sind. In der ternären Logik gibt es drei Wahrheitswerte: Wahr, Falsch und Unbekannt. Zuletzt gibt es Button-Parameter, die Operationen auslösen. In einem Parameter wird ein Dirty-Flag gesetzt, sobald er verändert wird. Optional ist es auch möglich, Callback-Funktionen aufzurufen. Parameter erhalten durch Parameter Slots Namen und Beschreibung. Diese Parameter Slots werden schließlich in der Core registriert. Die MegaMol Console unterstützt eine Modifikation der genannten Parametertypen. Hierfür kommt AntTweakBar (siehe Abbildung 2.2) zum Einsatz.

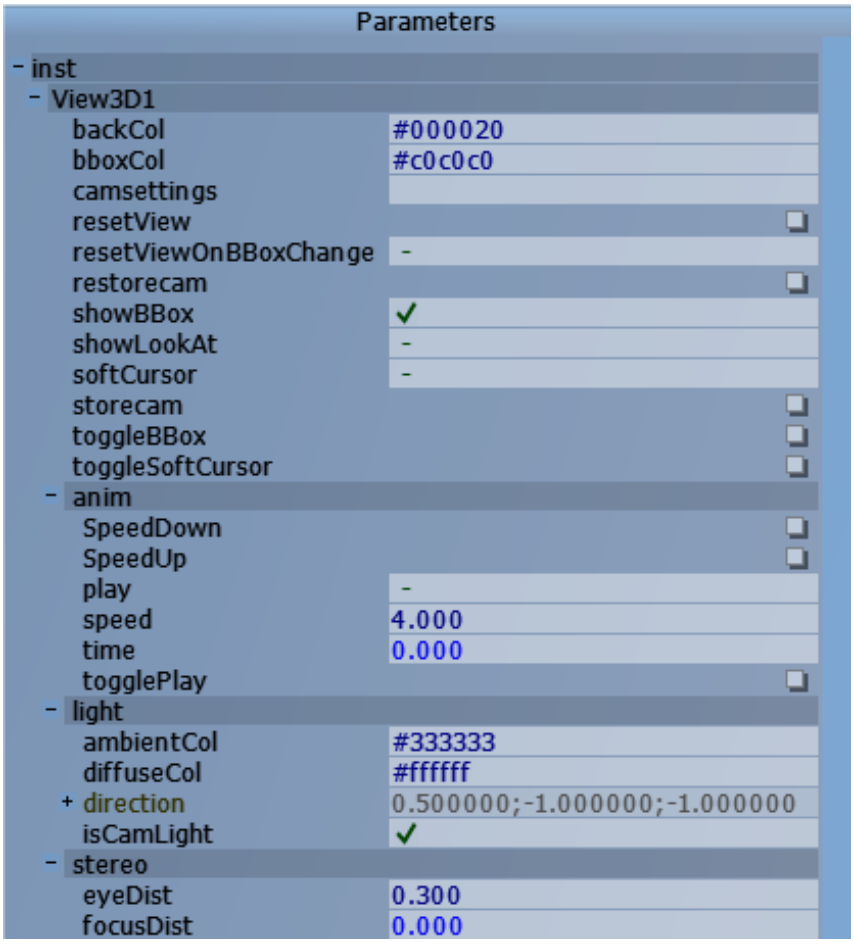

Abbildung 2.2: AntTweakBar der MegaMol Console (Ausschnitt). Dargestellt sind einige Parameter des Moduls „View3D“.

Module und Calls werden in einem Modulgraph zusammengesetzt, der für die Ausführung von MegaMol verwendet wird. Das Front-End weist die Core an, einen Modulgraph zu instanziieren [GKM+15]. Zur Erstellung von Modulgraphen kann der MegaMol Configurator verwendet werden, der in Abbildung 2.3 dargestellt ist.

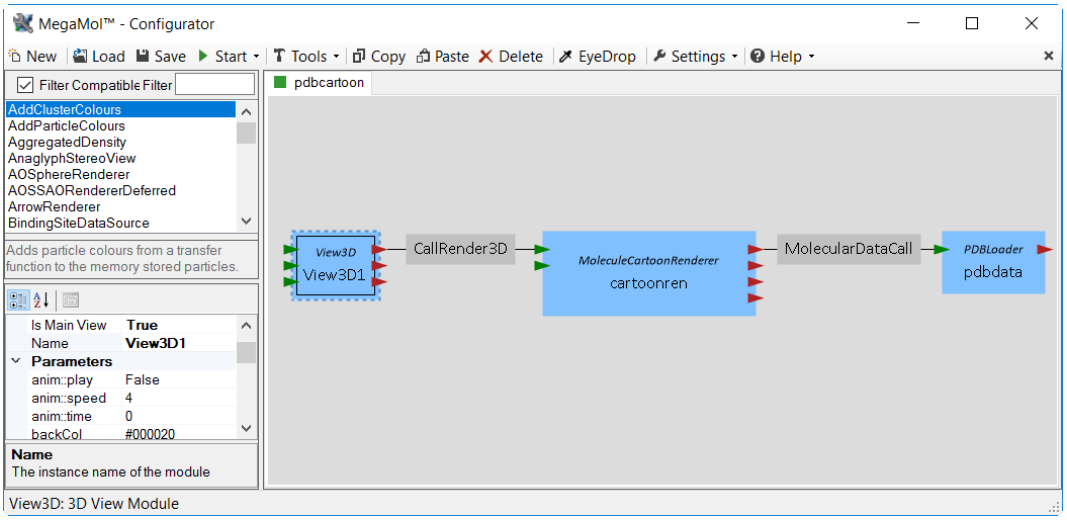

Abbildung 2.3: MegaMol Configurator. Module werden durch blaue Rechtecke, Calls durch graue Rechtecke repräsentiert. Oben links sind die verfügbaren Module aufgelistet. Unten links können die Parameter des aktuell selektierten Moduls verändert werden.

Damit Graphen erstellt und verändert werden können, müssen zunächst die Module und Calls der vorhandenen Plug-ins geladen werden. Der Configurator verwendet hierfür die MegaMol Console, die wiederum die Core anweist, eine State-Datei zu erzeugen, welche die entsprechenden Informationen enthält [GKM+15]. Ein einfacher Modulgraph ist aktuell geladen. Die Daten der Visualisierung werden aus einer lokal gespeicherten PDB-Datei geladen und daraufhin gerendert. Das View-Modul stellt eine Kameraansicht zur Verfügung, sodass die Visualisierung dargestellt werden kann. Die Steuerung der Kamera erfolgt mittels Maus und Tastatur. In dieser Arbeit wurde für Testzwecke mit einer PDB-Datei gearbeitet, welche die Daten einer Molekulardynamik-Simulation des Enzyms β-Lactamase[4] enthält. Die zugehörige Visualisierung ist in Abbildung 2.4 dargestellt. Gespeichert werden Modulgraphen in Projektdateien. Hierfür kommt die Auszeichnungssprache bzw. das Datenformat Extensible Markup Language (XML) zum Einsatz. Eine Projektdatei kann mehrere Modulgraphen definieren, die im Configurator in separaten Registerkarten dargestellt werden.

Die globale Konfiguration von MegaMol erfolgt in einer Datei „megamol.cfg“, die im „bin“-Verzeichnis abgelegt sein muss. Als Datenformat wird ebenfalls XML verwendet. Listing 2.1 zeigt beispielhaft Elemente der Konfigurationsdatei, die für diese Arbeit relevant sind. Für eine vollständige Beschreibung wird auf das Handbuch verwiesen, das über die MegaMol-Website heruntergeladen werden kann. Die Zeilen 3 und 4 definieren Plug-ins, die von der Core geladen werden. Üblicherweise haben diese

[4]Das Daten des Enzyms sind Bestandteil eines Pakets mit Beispieldaten, das von der MegaMol-Website heruntergeladen werden kann.

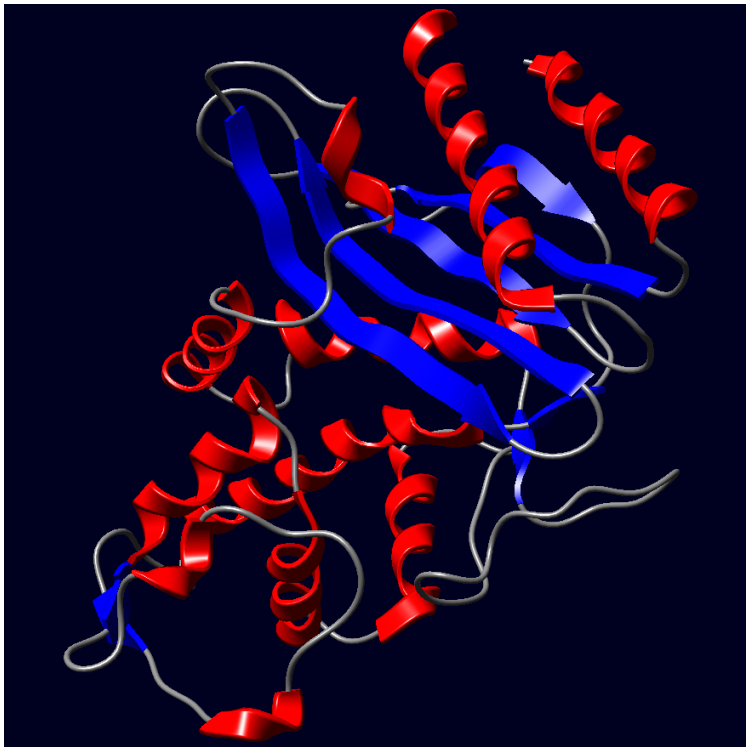

Abbildung 2.4: Visualisierung des Enzyms β-Lactamase mittels MegaMol

die Dateiendung „mmplg“. Generell müssen Pfade relativ zum „bin“-Verzeichnis angegeben werden. In Zeile 5 und 6 sind Ressource-Verzeichnisse angegeben. Diese Verzeichnisse werden in der Konfiguration der Benutzeroberfläche zum Auflösen von relativen Pfaden verwendet (siehe Abschnitt 4.2.3). Die AntTweakBar kann in Zeile 7 abgeschaltet werden. Das entsprechende Attribut muss hierfür auf `off` gesetzt werden. Für die Benutzerstudie wurde diese Änderung vorgenommen (siehe Abschnitt 6.3.1).

```
1 <MegaMol type="config" version="1.2">
2   <!-- ... -->
3   <plugin path="." name="protein.mmplg" action="include" />
4   <plugin path="." name="protein_calls.mmplg" action="include" />
5   <resourcedir path="../res/fonts" />
6   <resourcedir path="E:\Images" />
7   <set name="consolegui" value="on" />
8 </MegaMol>
```

Listing 2.1: Beispiel einer MegaMol-Konfigurationsdatei mit relevanten XML-Elementen

2.2 Leap Motion

Die Leap Motion ist ein Gerät zur räumlichen Erfassung von Händen und Fingern. Es wird hergestellt durch das amerikanische Unternehmen Leap Motion, Inc. und ist seit Juli 2013 [Lea13] erhältlich. Zuvor hatten bereits ausgewählte Entwickler die Möglichkeit, ein Vorseriengerät zu testen [Lea12a]. Auf diese Weise wollte der Hersteller Feedback einholen, gleichzeitig aber auch sicherstellen, das bereits zur Verkaufsbeginn möglichst viele Anwendungen aus verschiedenen Bereichen zur Verfügung stehen. Ursprünglich wurde das Produkt im Mai 2012 als „The Leap“ [Lea12b] angekündigt.

Das rechteckige Gerät, dargestellt in Abbildung 2.5, besitzt eine Größe von 80 mm x 30 mm x 11,25 mm [Lea15a] und wird über eine USB-Schnittstelle mit einem PC verbunden. Der Hersteller empfiehlt USB 3.0, jedoch kann es auch an einer USB 2.0-Buchse angeschlossen werden [Lea16c]. Nachdem das Gerät auf einem Tisch platziert wurde, können Hände über das Gerät bewegt werden. Diese werden sofort erkannt und stehen für eine Interaktion zur Verfügung. Der Hersteller stellt einen App Store zur Verfügung, aus dem kostenfreie wie auch kostenpflichtige Anwendungen für das Gerät bezogen werden können. Vorwiegend findet man hier Spiele. Es gibt jedoch u. a. auch Apps in denen man 3D-Modelle bewegt, Musikinstrumente spielt, zeichnet oder den PC bzw. die Maus steuert. Auf Grundlage einer Partnerschaft mit Hewlett Packard (HP) wurde das Gerät bereits in ein Notebook sowie eine Tastatur integriert.

Der technische Aufbau der Leap Motion ist in Abbildung 2.6 veranschaulicht. Die wesentlichen Komponenten stellen zwei monochrome CMOS-Sensoren sowie drei Infrarot-LEDs dar. Ein Tiefenbild wird konstruiert, indem die Bilder beider Sensoren analysiert werden. Bei dieser Analyse wird nach korrespondierenden Punkten in den Bildern gesucht und mittels Triangulation die räumliche Ausdehnung berechnet [Kis16]. Auf die selbe Weise funktioniert das räumliche Sehen des Menschen [Kis16]. Die LEDs dienen dazu, den Interaktionsraum auszuleuchten, damit die Sensoren ein möglichst klares Bild aufnehmen können [Kis16]. Um die Daten schnell an den PC

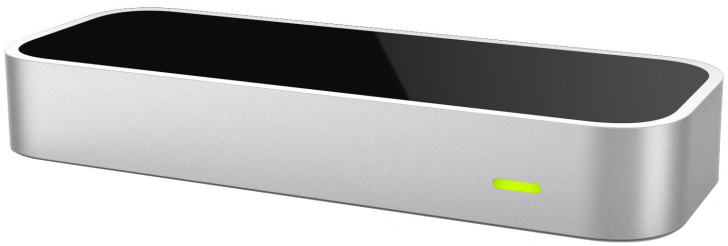

Abbildung 2.5: Leap Motion nach [Kur13]

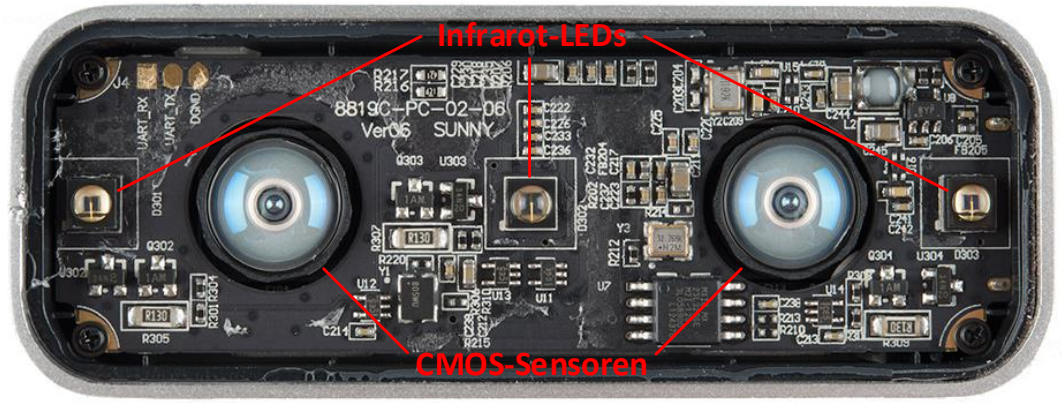

Abbildung 2.6: Obere Platine der Leap Motion nach [Spa13]. Die beiden Sensoren sind auf der darunter liegenden Platine angebracht.

zu übertragen, wurde ein USB 3.0-Controller verbaut [Spa13]. Die Verarbeitung der Daten erfolgt schließlich durch die Software der Leap Motion. Informationen zu den verwendeten Algorithmen hat der Hersteller bisher nicht veröffentlicht.

Abbildung 2.7 zeigt den Erfassungsbereich der Leap Motion. Die theoretische Genauigkeit des Geräts beträgt 0,01 mm [WBRF13]. Nach einer Studie von Weichert et al. [WBRF13], die Anfang 2013 durchgeführt wurde, erreicht es diesen Wert jedoch nicht. Die dort gemessene Genauigkeit von 0,7 mm[5] stellt jedoch immer noch einen sehr guten Wert dar, den beispielsweise die Kinect (erste und zweite Generation) nicht erreicht. Man muss zudem davon ausgehen, das sich die Genauigkeit mittlerweile

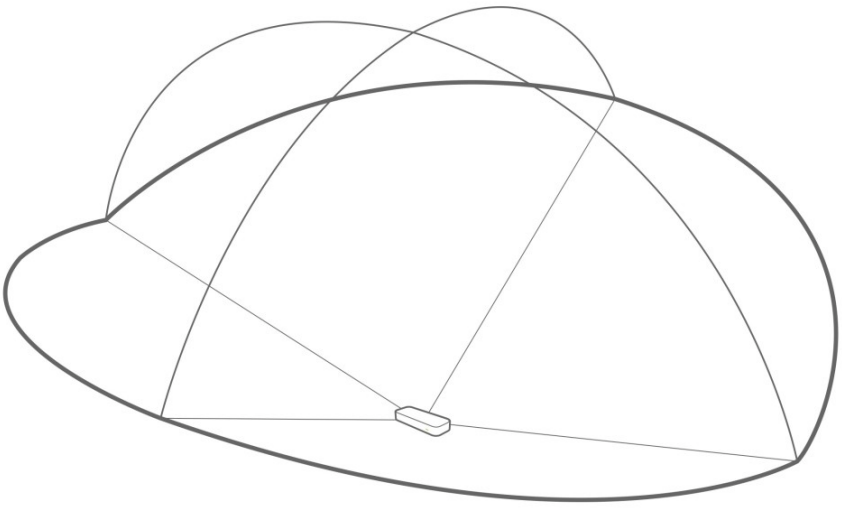

Abbildung 2.7: Erfassungsbereich der Leap Motion nach [Lea14a]. Auf der X-Achse beträgt der Winkel auf jeder Seite 150°. Für die Z-Achse sind es 120° auf jeder Seite. Es wird eine maximale Höhe von 80 cm unterstützt.

[5] Mittelwert aus den Genauigkeiten für einen statischen (<0,2 mm) und dynamischen Aufbau (1,2 mm).

deutlich verbessert hat. Die Software sowie Gerätefirmware waren damals noch in einem frühen Entwicklungszustand. Die Ergebnisse einer erneuten Durchführung wären daher interessant. Das Gerät liefert im Normalfall ca. 115 Bilder pro Sekunde zurück [Lea15b]. In der Vergangenheit gab es einen High-Speed-Modus, der jede Sekunde 300 Bilder zur Verfügung gestellt hat, sofern man mit einer geringeren Auflösung und damit auch Genauigkeit ausgekommen ist [Lea15b]. Diese Option wurde durch ein Softwareupdate entfernt. Falls die Leap Motion keine Hände erfasst hat, so wird die Bildwiederholrate auf 30 Bilder pro Sekunde gesenkt [Lea15b].

Die Software, die zur Nutzung des Geräts installiert werden muss, steht mittlerweile in der dritten Hauptversion zur Verfügung, die den Namen „Orion“ erhalten hat. Obwohl sich diese noch in einem Betastadium ist, wird sie ausdrücklich vom Hersteller empfohlen, falls man das Gerät unter Windows nutzt und „Tool Tracking“ (z. B. eines Stifts) nicht benötigt, wie es auch in dieser Arbeit der Fall gewesen ist. Alle anderen Nutzer müssen die Vorgängerversion, „V2“ genannt, einsetzen. Leap Motion App Home bezeichnet die Anwendung, in der Apps aufgerufen werden können, die über den App Store bezogen worden sind. Über das Leap Motion-Bedienfeld erfolgt die globale Konfiguration. Beispielsweise kann hier das Gerät abgeschaltet oder ein Robust-Modus aktiviert werden, der bei sehr hellen Lichtverhältnissen zum Einsatz kommt, welche die Erfassung stören. In diesem Fall wird die Zuverlässigkeit erhöht, gleichzeitig kommt es jedoch zu einer höheren Verzögerung und bei schnellen Bewegungen zu einem Verlust von Daten. Bei Problemen wie z. B. einem ständigen Springen von Positionsdaten, häufigen Unterbrechungen bei der Erfassung oder einem zu geringen Erfassungsbereich kann das Gerät neu kalibriert[6] werden. Über den Diagnose-Visualizer werden die Bewegungen der Hände und Finger dargestellt und weitere Gerätedaten (z. B. Bildwiederholrate) ausgeben. Auch können über das Bedienfeld Software- und Firmwareupdates eingespielt sowie Probleme an den Hersteller übermittelt werden.

Zur Entwicklung kommt das Orion SDK zum Einsatz, das für die Programmiersprachen JavaScript, C#, C++, Java, Python, Objective-C sowie den Spiele-Engines Unity und Unreal Engine zur Verfügung steht [Lea16a]. Ein Service ruft die Daten vom Gerät ab, verarbeitet sie und stellt sie schließlich über zwei verschiedene APIs zur Verfügung [Lea16a]. Die native API wird vom C++ sowie Objective-C SDK über eine dynamische Programmbibliothek genutzt, bei den Sprachen C#, Java und Python kommen Wrapper-Bibliotheken zum Einsatz [Lea16a]. Grundsätzlich werden Daten nur der Anwendung bereitgestellt, die den Fokus besitzt[7] [Lea16a]. Hintergrundanwendungen müssen explizit den Zugriff auf Daten aktivieren, was jedoch über das

[6] https://developer.leapmotion.com/documentation/csharp/supplements/Leap_Application.html#recalibration

[7] Dies gilt nicht für Linux. Entsprechende Informationen werden von diesem Betriebssystem nicht bereitstellt.

Bedienfeld untersagt werden kann [Lea16a]. Das Java Script SDK verwendet die API eines WebSocket Servers, der vom Service gestartet wird und ebenfalls deaktiviert werden kann [Lea16a]. Als Datenformat kommt JSON zum Einsatz [Lea16a]. In dieser Arbeit wurde mit dem C++ SDK[8] gearbeitet (Version 3.1.3), da MegaMol (siehe Abschnitt 2.1) ebenfalls in dieser Sprache entwickelt worden ist. Im Folgenden soll nun auf die Daten eingegangen werden, die über das SDK abrufbar sind. Mit dem Orion SDK wurden einige Funktionen auf „deprecated“ gesetzt. Bis auf eine Ausnahme werden diese nicht behandelt.

Das SDK stellt zu einer Hand u. a. folgende Daten bereit: Handtyp (links/rechts), Position (Handfläche und Handgelenk), Richtung (Handfläche zu Finger sowie senkrecht zur Handfläche), Orientierung, Geschwindigkeit, Breite, Greifwinkel (z. B. zur Bestimmung einer flachen Hand bzw. Faust) und Abstand zwischen Daumen und Zeigefinger („pinch distance“). Jede erfasste Hand erhält eine ID, sodass sie eindeutig unterschieden werden kann. Abbildung 2.8 zeigt, wie das Gerät Finger repräsentiert. Zu einem Finger können Richtung (zur Fingerspitze), Länge, Breite sowie Position und Geschwindigkeit der Fingerspitze zurückgegeben werden. Weiterhin wird ausgegeben, ob der jeweilige Finger ausgestreckt ist. Für einen einzelnen Fingerknochen sind Position (Beginn, Mitte und Ende), Orientierung, Richtung (zum Knochenende), Breite und Länge definiert. Das Gerät wurde primär zur Erkennung von Händen bzw. Fingern entwickelt. Dennoch werden auch Informationen zum Vorderarm be-

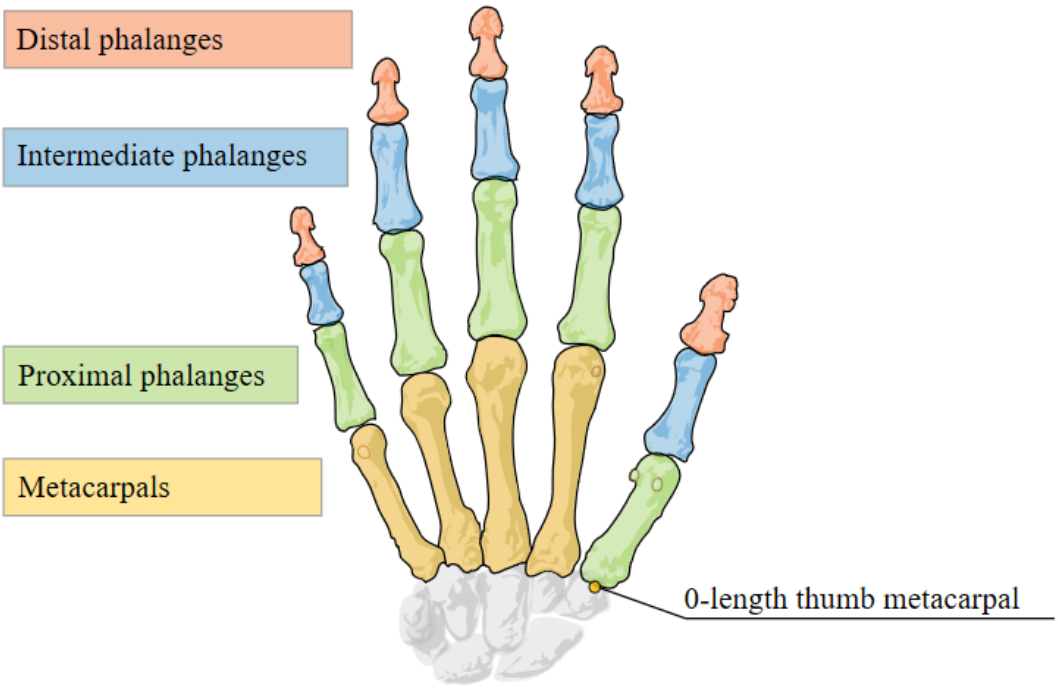

Abbildung 2.8: Fingermodelle der Leap Motion [Lea16a]

[8] Dokumentation: https://developer.leapmotion.com/documentation/cpp

reitgestellt: Position (Ellbogen, Mitte und Handgelenk), Richtung (zum Handgelenk), Orientierung und Breite. Falls Teile der Hand oder des Arms verdeckt sind, so werden stets Schätzungen durchgeführt. Diese erfolgen auf Grundlage eindeutiger Daten, vergangener Daten sowie interner Modelle, welche die typische Anatomie abbilden. Beliebig viele Hände können vom Gerät erfasst werden. Der Hersteller empfiehlt [Lea16a] jedoch, höchstens zwei Hände in den Erfassungsbereich des Geräts zu bewegen, damit Bewegungen mit maximaler Zuverlässigkeit erkannt werden.

Die Gestures API des SDK erlaubt eine Erkennung von Fingerbewegungen, die bestimmten Mustern folgen. Die erforderlichen Algorithmen sind bereits implementiert. Mit dem Orion SDK wurde die API als „deprecated“ definiert [Lea16a], kann jedoch immer noch verwendet werden. Da sie in dieser Arbeit zum Einsatz kommt, soll sie kurz erläutert werden. Vier verschiedene Bewegungsmuster werden unterstützt. Das erste Muster stellt eine Kreisbewegung dar [Lea16b]. Fortschritt sowie Mittelpunkt, Radius und Normalenvektor des Kreises, der in der Luft gezeichnet wird, können zurückgegeben werden. Eine lineare Bewegung in eine beliebige Richtung stellt eine Swipe-Geste dar [Lea16b]. Hierzu werden die Start- und aktuelle Position sowie die Richtung und Geschwindigkeit zurückliefert. Als Key Tap wird eine schnelle Bewegung nach unten verstanden, vergleichbar mit dem Drücken einer Taste auf dem Klavier [Lea16b]. Startposition sowie Richtung der Bewegung werden bereitgestellt. Zuletzt wird ein Screen Tap unterstützt, was eine schnelle Vorwärtsbewegung darstellt [Lea16b]. Ebenso werden Startposition und Richtung zur Verfügung gestellt.

Zuletzt besteht die Möglichkeit, auf die Bilder der beiden Sensoren zuzugreifen. Dies muss jedoch zunächst aktiviert werden [Lea16a]. Über das Bedienfeld kann das Abrufen von Bildern (zur Wahrung der Privatsphäre) generell verhindert werden [Lea16a].

Alle genannten Daten werden in einem Frame zusammengefasst. Der Leap Motion Service speichert die letzten 60 aufgezeichneten Frames [Lea16a]. Zum Abruf von Frames stehen zwei Möglichkeiten zur Verfügung: Falls die Anwendung kontinuierlich eine bestimmte Funktion aufruft, so kann in jedem Aufruf das aktuellste Frame vom SDK abgerufen werden. Die Alternative stellen Listener dar. Sobald eine neues Frame vorliegt, wird vom SDK eine Funktion `onFrame` ausgeführt, die das Frame übergeben wird. Der Entwickler kann die Verarbeitung des Frames wie gewünscht implementieren. Bei dieser Variante ist zu beachten, dass jeder Funktionsaufruf in einem separaten Thread erfolgt. Folglich muss sichergestellt werden, das Datenzugriffe stets threadsicher ausgeführt werden. Die MegaMol-Module, die in dieser Arbeit entwickelt worden sind, haben eine Render-Funktion zur Verfügung gestellt. Dementsprechend ist die Wahl auf die erste Option gefallen.

2.3 Kinect für Xbox One

Die Microsoft Kinect für Xbox One wurde im November 2013 [MH13] zusammen mit der Xbox One veröffentlicht, um ein Spielen auf der Konsole mittels Körperbewegungen zu ermöglichen. Es handelt um den Nachfolger der ersten Generation, Kinect genannt, der für die Xbox 360 entwickelt wurde und bereits Ende 2010 [Che10] erschienen ist.

Das Gerät, veranschaulicht in Abbildung 2.9, wird häufig über dem Fernseher platziert. Die Aufbau sollte in einer Höbe von 0,6 m bis 1,8 m [Mic16g] erfolgen. Personen werden am zuverlässigsten erkannt, wenn diese einen Abstand von etwa 1,4 m bis 1,8 m [Mic16g] einhalten. Die Breite wie auch Höhe des Erfassungsbereichs nimmt mit zunehmenden Abstand zu [Sme14] (siehe Tabelle 2.1). Zur Bestimmung der Breite kann folgende Formel verwendet werden:

$$2 * \text{Distanz} * tan(\frac{70{,}6°}{2})$$

Ein Benutzer, der sich z. B. 1,8 m zentriert vor der Kinect positioniert hat, kann sich also höchstens 1,27 m nach links oder rechts begeben. Das Gerät verfügt über einen proprietären Anschluss, sodass es nicht ohne Weiteres mit einem PC verbunden werden kann. Aus diesem Grund wurde im Juli 2014 [Kön14] die Kinect für Windows v2[9] veröffentlicht, die bezüglich der Hardware identisch ist, jedoch einen Adapter im Lieferumfang enthält [Mic15]. Dieser erlaubt den Anschluss an eine USB 3.0-Schnittstelle des PCs. Der Adapter verfügt über ein Netzteil, da über USB 3.0 lediglich 900 mA zur Verfügung gestellt wird, was für das Gerät nicht ausreichend ist. Im April 2015 [Mic15] hat Microsoft die Windows-Version dann jedoch wieder eingestellt, da der Adapter nun separat erworben werden konnte. Abbildung 2.10 zeigt die wesentlichen Komponenten der Kinect für Xbox One. Für die Konstruktion

Abbildung 2.9: Kinect für Xbox One nach [Sar13]. Das obere Teil enthält die RGB-Kamera, Infrarotkamera und den Infrarotprojektor (von links nach rechts) und ist kippbar. Unten befindet sich das Mikrofon-Array.

[9] Von der ersten Kinect-Generation ist ebenfalls eine Windows-Version erschienen (Kinect für Windows v1).

Abbildung 2.10: Hauptplatine der Kinect für Xbox One nach [Mia13]

des Tiefenbilds kommt die Time of Flight-Methode (ToF-Methode) zum Einsatz. Der Infrarotprojektor, versehen mit einem Bandpassfilter, sendet kurze Lichtsignale aus, die von Objekten (insbesondere Personen) reflektiert und von der Infrarotkamera schließlich wieder aufgenommen werden [Kis16]. Durch den Einsatz des Filters soll sichergestellt werden, das kein anderes Licht (mit einer abweichenden Wellenlänge) den Projektor verlassen kann [SLK15]. Die Zeit vom Aussenden bis zum Empfangen des Lichtsignals wird gemessen [Kis16]. Mit dieser Information und der Geschwindigkeit von Licht (ca. 300 000 $\frac{\text{km}}{\text{s}}$) kann die Distanz zwischen Gerät und Objekt bestimmt werden [Kis16]. Infrarotkamera als auch Farbkamera liefern 30 Bilder pro Sekunde zurück [Mic16f]. Bei dieser Frequenz stellt die Infrarotkamera eine Auflösung von 512 x 424 Pixel bereit, die Farbkamera liefert Full HD (1920 x 1080 Pixel) zurück [Mic16f]. Das Mikrofon-Array (siehe Abbildung 2.9), das sich aus vier einzelnen Mikrofonen zusammensetzt, erlaubt eine Steuerung der Xbox One mittels Sprachbefehlen [Mic16f; Mic16g]. Die Konsole kann zudem so konfiguriert werden, das das Mikrofon-Array auch im Standbymodus eingeschaltet bleibt [Mic16g]. Es ist nun ein Wecken durch Sprachkommandos möglich [Mic16g]. Bei der ersten Kinect-Generation wurde ein anderer Ansatz zur Tiefenmessung eingesetzt, der als Structured Light bezeichnet wird [Kis16]. Der Infrarotprojektor strahlt ein Punktemuster in den Raum aus, das durch Reflektion verzerrt und von der Kamera wieder aufgefangen wird [Kis16]. Auf Grundlage der Verzerrung wird mittels einem Triangulationsverfahren die Distanz zum Objekt errechnet [Kis16].

Zur Entwicklung von Anwendungen für die Kinect für Xbox One kommt das Kinect für Windows SDK 2.0[10] zum Einsatz. Es besteht die Wahl zwischen C++, C#, Visual Basic oder einer .NET-Programmiersprache, wobei mindestens Windows 8 benötigt wird [Mic16c]. Im Gegensatz zur Leap Motion gibt es noch ein Drittanbieter-SDK, libfreenect2[11] genannt, das sich jedoch lediglich als vorteilhaft erweist, wenn man das Gerät mit anderen Sprachen (z. B. Java oder Python) oder Betriebssystemen einsetzen möchte. Da zur Entwicklung Windows 10 verwendet wird und MegaMol unter C++ geschrieben wurde, ist die Wahl auf das offizielle SDK gefallen. Neben den Bibliotheken für die einzelnen Sprachen werden mehrere Anwendungen mitgeliefert.

[10] https://www.microsoft.com/en-us/download/details.aspx?id=44561
[11] https://github.com/OpenKinect/libfreenect2

Kinect Studio dient zum Ausgeben und Aufzeichnen der verschiedenen Daten des Geräts. Der Visual Gesture Builder erlaubt eine Erkennung von Gesten mittels den Maschinen-Lern-Algorithmen Adaptive Boosting (AdaBoost) und Random Forest Regression (RFR). Als Eingabedaten kommen Aufzeichnungen von Kinect Studio zum Einsatz. Über den SDK Browser können die zuvor genannten Programme aufgerufen sowie auf Beispielcode für verschiedene Sprachen zugegriffen werden. Auch kann der Configuration Verifier gestartet werden, mit dem überprüft wird, ob der PC die Systemvoraussetzungen erfüllt. Dieser ist jedoch nicht Bestandteil des SDKs und muss separat heruntergeladen[12] werden.

Über die API des SDK können verschiedene Arten von Frames abgerufen werden. Der Abruf kann wie auch bei der Leap Motion (siehe Abschnitt 2.2) durch das kontinuierliche Aufrufen einer Funktion oder Implementieren von Listener erfolgen. Auch hier ist die Entscheidung aus selbigen Grund auf die erste Variante gefallen. Ein `IBodyFrame` wird durch Tiefendaten erzeugt. Es enthält von maximal sechs Personen, die durch IDs identifiziert sind, jeweils Positions- und Orientierungsdaten zu 25 Körperteilen bzw. Joints (siehe Abbildung 2.11). Weiterhin ist zu jedem

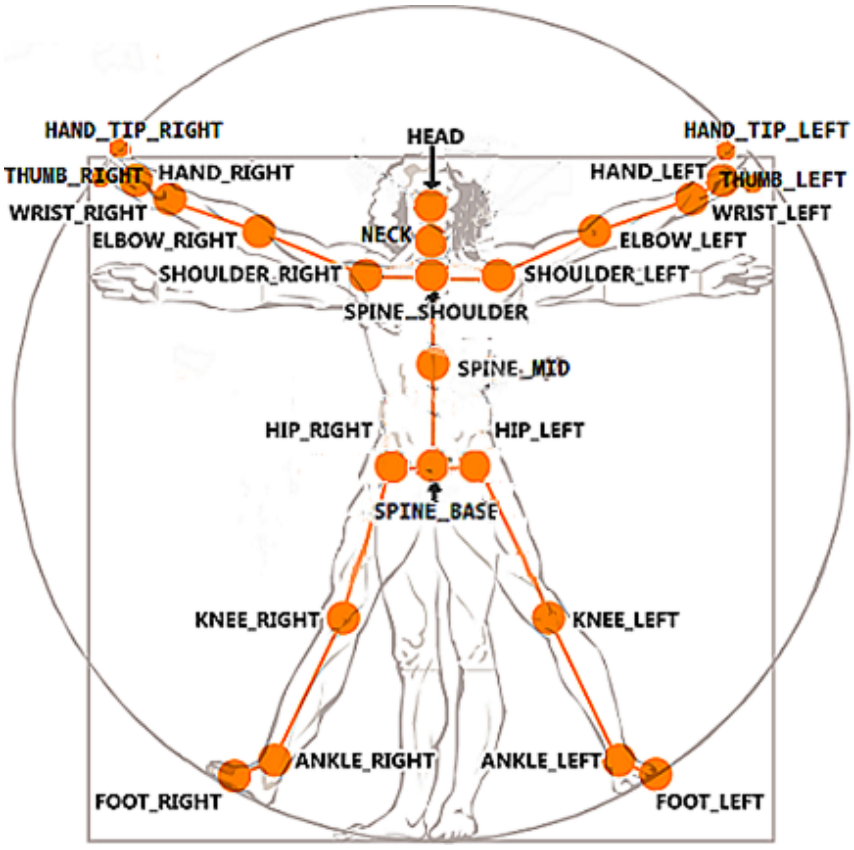

Abbildung 2.11: Körpermodell der Kinect für Xbox One [Mic16c]. Zu den angegebenen Körperteilen (orangene Kreise) werden Daten bereitgestellt.

[12] http://go.microsoft.com/fwlink/p/?LinkID=513889

Körperteil ein `TrackingState` definiert, der angibt, ob das Körperteil erfasst wurde bzw. nicht oder lediglich geschätzt worden ist. Von höchstens zwei Personen können Handzustände zurückgegeben werden. Ein Hand kann die Zustände „Open“, „Closed“, „Lasso“, „NotTracked“ und „Unknown“ besitzen. Der Lasso-Zustand wird gesetzt, falls die Hand geschlossen ist, jedoch Zeigefinger und Mittelfinger nach oben ausgestreckt sind [Mic16a]. Dieser Zustand wird erweitert durch einen Confidence-Wert, der eine Aussage darüber trifft, wie wahrscheinlich der Zustand korrekt ist. Mögliche Werte sind `High` und `Low`. Der Handzustand wird später in der Implementierung zum Einsatz kommen. Auch kann zurückgegeben werden, wie stark der Körper in alle Dimensionen geneigt ist. Falls sich eine Person teilweise außerhalb der Erfassungsbereich bewegt, so kann ausgegeben werden, auf welchem Rand sie sich befindet.

Da die übrigen Framearten nicht in der Arbeit verwendet werden, soll nur kurz auf sie eingegangen werden. Das `IBodyIndexFrame` verwendet ebenfalls Tiefendaten [Mic16c]. In diesem Bild werden die Pixel erkannter Personen von den übrigen Pixeln des Hintergrunds getrennt [Mic16c]. Ein `IInfraredFrame` ist ein Schwarz-Weiß-Bild der Infrarotkamera, dessen Helligkeit nicht von der Position oder Ausleuchtung des Raums beeinflusst ist [Mic16c]. Falls dieses Bild in einer höheren Qualität gewünscht ist, so kann ein `ILongExposureInfraredFrame` verwendet werden, das langzeitbelichtet ist [Mic16c]. Durch schnelle Bewegungen kann es jedoch zu einer Unschärfe kommen [Mic16c]. In einem `IDepthFrame` repräsentiert jedes Pixel die Distanz zum seinem nächstgelegenen Objekt [Mic16c]. Das `ColorFrame` ist das Bild der RGB-Kamera [Mic16c]. Der Ton des Mikrofon-Arrays kann durch `IAudioBeamFrames` zurückgegeben werden [Mic16c]. Ein `MultiSourceFrame` enthält Referenzen zu allen zuvor genannten Frames (Ausnahme `IAudioBeamFrame`) [Mic16c]. Über ein `IFaceFrame` können u. a. die Punkte des Gesichts, ein Rechteck, welches das Gesicht enthält sowie eine Reihe von Eigenschaften[13] bereitgestellt werden [Mic16c]. Gesichtseigenschaften sind beispielsweise ein geschlossenes Auge, ein Tragen einer Brille oder ein Wegschauen [Mic16c]. Dies erfolgt auf Grundlage der Infrarotdaten sowie RGB-Daten [Mic16c]. Zuletzt gibt es ein `IHighDefinitionFaceFrame`, das u. a. Gesichtsanimationen sowie die Gestalt des Gesichts erfasst [Mic16c]. Für letzteres sind 70 Parameter definiert, die jeweils mit -2 bis 2 gewichtet werden [Mic16c]. Die Gewichtung hängt je nach Parameter von der Gesichtsgröße oder von der Abweichung vom Durchschnitt ab [Mic16c].

Zennaro et al. [ZMM+15] haben in einer Evaluation die Genauigkeit beider Kinect-Generationen verglichen. Es hat sich gezeigt, dass im Nahbereich die Kinect für Xbox One zweimal so genau arbeitet. Nach 6 Metern war eine zehnmal höhere Präzision festzustellen. Weiterhin war die zweite Generation weniger anfällig für Kunst- und Sonnenlicht. Auch sollte nicht unerwähnt bleiben, dass die Auflösung der Farbkamera

[13] https://msdn.microsoft.com/en-us/library/microsoft.kinect.face.faceproperty.aspx

deutlich höher ist. Die Kameraqualität der ersten Generation kann mit einer Webcam verglichen werden [PP15]. In Tabelle 2.1 sind die wichtigsten technischen Daten beider Geräte gegenübergestellt.

Feature	**Kinect**	**Kinect für Xbox One**
RGB-Kamera (30 fps)	640 x 480	1920 x 1080
Horizontales Sichtfeld	62°	84,1°
Vertikales Sichtfeld	48,6°	53,8°
Tiefenkamera (30 fps)	320 x 240	512 x 424
Maximale Tiefe	4,0 m	4,5 m
Minimale Tiefe	0,8 m	0,5 m
Horizontales Sichtfeld	58,5°	70,6°
Vertikales Sichtfeld	45,6°	60°
Kippmotor	ja	nein
Körper	2	6
Körperteile (Joints)	20	25
USB	2.0	3.0
Kinect für Windows SDK	1.8 (mind. Windows 7)	2.0 (mind. Windows 8)

Tabelle 2.1: Vergleich beider Kinect-Generationen. Quellen der Daten: [Ash14; Mic16b; Mic16d; Mic16f; Sme14]

Im weiteren Verlauf der Arbeit wird die Kinect für Xbox One aus Gründen der Übersichtlichkeit nur noch als Kinect bezeichnet. Wird vom Kinect SDK gesprochen, so ist das Kinect für Windows SDK 2.0 gemeint.

2.4 Glättungsfilter

Positionen und andere Daten, die von Geräten zur Bewegungssteuerung bereitgestellt werden, zeichnen sich in Rohform durch ein Rauschen bzw. Zittern aus. Die Intensität dieses Rauschens hängt dabei nicht nur vom Gerät selbst ab, sondern wird auch durch Störungen verstärkt. Beide Geräte, die in dieser Arbeit zum Einsatz kommen, emittieren Infrarotlicht. Werden diese Geräte in Umgebungen mit sehr hellem Licht (insbesondere Sonnenlicht) eingesetzt, so wird das abgegebene Licht überstrahlt, was deren Funktionsweise behindert. Die berechneten Daten werden deutlich instabiler. Falls man den Erfassungsbereich des Geräts teilweise verlässt oder Körperteile, zu denen Daten bereitgestellt werden, verdeckt sind (z. B. durch andere Körperteile), so werden Schätzungen durchgeführt. Diese liefern jedoch in den meisten Fällen nur recht unzuverlässige Ergebnisse, was sich durch ein merkliches Zittern bemerkbar macht.

Das Zittern korreliert in der Regel mit der Zuverlässigkeit der Gestenerkennung. Ist das Zittern hoch, so werden Gesten häufiger nicht oder irrtümlich erkannt.

Zur Reduktion des Rauschens kommen sogenannte Glättungs- bzw. Stabilisierungsfilter zum Einsatz. Das Grundprinzip dieser Filter ist es, für die Bestimmung eines Werts zusätzlich die Datenhistorie miteinzubeziehen. Je mehr Daten aus der Vergangenheit jedoch mitverwendet werden, umso größer wird auch die Verzögerung (Latenz), bis der berechnete Wert mit dem aktuellsten übereinstimmt. Erfasste Körperbewegungen werden also ebenso zunehmend verzögert auf dem Bildschirm umgesetzt. Benutzerstudien haben gezeigt, das Verzögerungen von mehr als 100 ms wahrgenommen werden [Mic16e]. Ein Ansatz, diese Verzögerung wieder zu reduzieren, besteht darin, diese Filter auch zur Vorhersage von zukünftigen Daten zu nutzen [Mic16e]. Prognosen erfolgen jedoch ausschließlich auf Grundlage vergangener Daten, sodass nicht immer eine zufriedenstellende Genauigkeit erzielt werden kann [Mic16e]. Insbesondere trifft dies zu, falls ein Benutzer abrupt Körperbewegungen beginnt oder beendet [Mic16e]. Das Zittern nimmt hierbei kurzfristig zu [Mic16e]. Letztendlich muss bei der Wahl eines Filters und deren Parametrisierung ein Kompromiss zwischen Rauschreduktion und Verzögerung gefunden werden, der sich nach dem Einsatzbereich der Daten richtet [Mic16e]. In die Entscheidung sollte auch das jeweilige Körperteil bzw. dessen typische Bewegungsgeschwindigkeit miteinbezogen werden [Mic16e]. Ein Hand beispielsweise wird überwiegend schneller als die Wirbelsäule bewegt [Mic16e]. Dementsprechend fällt eine Verzögerung deutlich stärker auf.

Das SDK der ersten Kinect-Generation (Kinect für Windows SDK 1.8 oder niedriger) wendet auf alle zurückgelieferten Positionsdaten eine exponentielle Glättung zweiter Ordnung („Double Exponential Smoothing Filter“) an [Mic16e]. Für eine nähere Beschreibung der zugrundeliegenden Formeln wird auf den MSDN-Artikel [Mic16e] verwiesen. Die Parameter des Filters sind in erläutert. Bei der zweiten Kinect-Generation muss die Glättung der Daten vom Entwickler selbst durchgeführt werden [Sir14b]. Diese Änderung wurde vorgenommen, da die Daten der Kinect nun auch mehreren Anwendungen gleichzeitig zur Verfügung gestellt werden können [Sir14b]. Wie zuvor erwähnt, besitzt jede Anwendung unterschiedliche Anforderungen. Damit der identische Filter jedoch auch mit der zweiten Kinect eingesetzt werden kann, wurde der entsprechende Code in den MSDN-Foren [Sir14c] veröffentlicht. Es wurde entschieden, diesen Filter auch für die Gestensteuerung zu testen, da er sich bereits bei der ersten Kinect bewährt hat. Die Implementierung wurde leicht modifiziert. In der veröffentlichten Version müssen einer Instanz alle Positionsdaten eines Körpers übergeben werden, die daraufhin mit der selben Parametrisierung gefiltert werden. In der Arbeit werden jedoch lediglich Handpositionen verwendet. Aus diesen Gründen wurde der Code dahingehend verändert, das eine Instanz des Filters nur noch für ein Körperteil einer Person zuständig ist. Als Datentyp für Positionsdaten kommt die

„Vector“-Klasse aus der Hilfsbibliothek VISlib zum Einsatz. Auf diese Weise kann der Filter später auch mit der Leap Motion getestet werden. Ein weiterer Filter wurde implementiert, der auf dem einfachen gleitenden Mittelwert basiert („Simple Moving Average Filter“). Der aktuelle Wert wird bestimmt, indem der Mittelwert aus den zuletzt abgerufenen Daten gebildet wird. Die Fenstergröße legt fest, wie viele Elemente hierfür verwendet werden. Für die Speicherung der Datenhistorie wurde ein Ringpuffer implementiert. Welche Filter für die einzelnen Gesten ausgewählt worden sind und wie bei der Entscheidung vorgegangen worden ist, wird für die Leap Motion in Abschnitt 5.2.1, für die Kinect in Abschnitt 5.2.2 beschrieben.

2.5 NanoVG

NanoVG[14] ist eine in C geschriebene Zeichenbibliothek für 2D-Vektorgrafiken, die OpenGL zum Rendering einsetzt [Mon16]. Sie wird seit 2013 von Mikko Mononen entwickelt und steht unter einer Open Source-Lizenz zur Verfügung [Mon16].

Bevor mit dem Zeichnen von Grafiken begonnen werden kann, muss zunächst ein Zeichenkontext erstellt werden. Dies ist in Listing 2.2 dargestellt.

```
1 #include "nanovg.h"
2 #define NANOVG_GL3_IMPLEMENTATION
3 #include "nanovg_gl.h"
4 NVGcontext* vg = nvgCreateGL3(NVG_ANTIALIAS | NVG_STENCIL_STROKES);
```

Listing 2.2: Erstellen eines NanoVG-Zeichenkontexts

In diesem Fall wird die OpenGL 3.2 (Core Profile)-API zum Rendering gewählt. Daneben werden von NanoVG auch OpenGL 2.0, OpenGL ES 2.0 und OpenGL ES 3.0 unterstützt [Mon16]. Der Parameter `NVG_ANTIALIAS` aktiviert Antialiasing, durch `NVG_STENCIL_STROKES` werden (überlappende) Linien mit einer höheren Qualität gerendert [Mon16].

Die grundsätzliche Nutzung der API soll anhand von Listing 2.3 erläutert werden, in der erst ein Ringsegment und dann ein vollständiger Ring gezeichnet wird.

[14] https://github.com/memononen/nanovg

```
1  nvgBeginPath(vg);
2  nvgArc(vg, ringSegmentCenterX, ringSegmentCenterY, radius - ringSegmentWidth,
       startAngle, endAngle, NVG_CW);
3  nvgArc(vg, ringSegmentCenterX, ringSegmentCenterY, radius, endAngle, startAngle,
       NVG_CCW);
4  nvgClosePath(vg);
5  nvgFillColor(vg, color);
6  nvgFill(vg);
7
8  nvgBeginPath(vg);
9  nvgCircle(vg, ringCenterX, ringCenterY, radius);
10 nvgCircle(vg, ringCenterX, ringCenterY, radius - ringWidth);
11 nvgPathWinding(vg, NVG_HOLE);
12 nvgFillColor(vg, color);
13 nvgFill(vg);
```

Listing 2.3: Zeichnen eines Ringsegments und Rings mit NanoVG

In Zeile 1 wird für das Ringsegment mit einem neuen Pfad begonnen. Zeile 2 zeichnet den Pfad des inneren Bogens. Die Parameter `startAngle` und `endAngle` werden in Radiant (0 bis 2π) definiert und geben ausgehend vom Mittelpunkt an, wo der Bogen beginnt und endet. 0 Radiant befindet sich in der Mitte rechts, also in (ringSegmentCenterX + radius - ringSegmentWidth | ringSegmentCenterY). Durch `NVG_CW` wird angegeben, das der Bogenpfad im Uhrzeigersinn gezeichnet wird. Der Cursor, der den Pfad zeichnet, befindet sich nun am Ende des Bogens. In Zeile 3 wird der Pfad des äußeren Bogens entgegen dem Uhrzeigersinn gezeichnet. Der Cursor zeichnet dabei zunächst eine schmale Seite des Segments, da er zum Beginn des äußeren Bogens wandern muss. Die zweite schmale Seite wird schließlich in Zeile 4 gezeichnet, indem die aktuelle Position (Ende des äußeren Bogens) mit der ursprünglichen Startposition verbunden wird. Das Ringsegment ist nun vollständig definiert und wird schließlich in den Zeilen 5 und 6 mit einer Farbe gefüllt. Zeile 8 erstellt einen neuen Pfad für den Ring. In den Zeilen 9 und 10 werden die Kreispfade gezeichnet, die den Ring einschließen. Zeile 11 legt fest, das der innere Kreis ausgeschlossen wird. Der Ring ist nun definiert und wird abschließend eingefärbt.

NanoVG stellt noch eine Reihe von weiteren Möglichkeiten zur Verfügung, Formen und Pfade zu definieren und diese dann zu gestalten. Neben Kreise können auch (abgerundete) Rechtecke und Ellipsen gezeichnet werden. Pfade können durch Linien und Bögen, aber z. B. auch durch kubische und quadratische Bezierkurven definiert sein. Füllungen sind nicht nur in einer Farbe möglich, es werden auch lineare und radiale Farbverläufe unterstützt. Auch können Bilder geladen werden. Hierfür kommt die Bibliothek stb_image[15] zum Einsatz. Zum Rendern von Text kann zwischen

[15] https://github.com/memononen/nanovg/blob/master/src/stb_image.h

stb_truetype[16] (Standard) und FreeType[17] gewählt werden. Für eine vollständige Beschreibung aller Funktionalitäten wird auf die Header-Datei von NanoVG[18] verwiesen.

In dieser Arbeit wird die Benutzeroberfläche mittels NanoVG realisiert (siehe Abschnitt 5.3). Die Entscheidung ist auf diese Zeichenbibliothek gefallen, da sie (1) aktiv weiterentwickelt wird, (2) leichtgewichtig ist, (3) eine übersichtliche API besitzt, (4) OpenGL zum Rendern verwendet und bereits (5) Funktionen mitbringt, um Schriftarten und Bilder zu laden.

[16] https://github.com/memononen/nanovg/blob/master/src/stb_truetype.h
[17] https://www.freetype.org
[18] https://github.com/memononen/nanovg/blob/master/src/nanovg.h

3 Verwandte Arbeiten

Im Folgenden sollen Arbeiten vorgestellt werden, in denen eine Steuerung von dreidimensionalen Objekten mittels Gesten bereits umgesetzt worden ist. Es wird nur auf Anwendungen eingegangen, welche die Steuerung berührungslos mittels der Leap Motion oder Kinect (auch erste Generation) realisieren. Die Notwendigkeit eines weiteren Eingabegeräts (z. B. Tastatur) führt zum Ausschluss. Zu einer Anwendung soll angegeben werden (sofern vorhanden), mit welchen Gesten die Bewegung des Objekts bzw. der Kamera ermöglicht wird. Auch soll die Bedienung der Benutzeroberfläche erläutert werden, sofern diese ebenso ohne Berührungen erfolgt.

In der Leap Motion Software Orion sind u. a. bereits die Apps „Form & Function 3D“[19] sowie „Sculpting“[20] vorinstalliert. Mit ersterer App, die von einem Biologie-Studenten der Universität von Toronto [Lea14b] entwickelt wurde, kann die Anatomie des Herzens verschiedener Tiere kennengelernt werden. Zur Entwicklung wurde die Spiele-Engine Unity[21] eingesetzt [Lea14b]. Während der Benutzung der Anwendung ist kontinuierlich ein Modell der Hand sichtbar, das den Bewegungen der echten Hand folgt. Vergleichbar ist dies mit dem Handmodell, der im Diagnose-Visualizer der Leap Motion Software (siehe Abschnitt 2.2) sichtbar ist. Durch Bewegungen der flachen Hand parallel zur X- bzw. Y-Achse wird das 3D-Modell entsprechend rotiert, eine Zoom-Funktion ist nicht vorhanden. Streckt man nur den Zeigefinger aus, so erscheint ein grüner Cursor. Wenn der Benutzer den Cursor auf einen bestimmten Teil des Herzens bewegt und daraufhin die Hand zum Bildschirm hin bewegt, werden als Tooltip Informationen eingeblendet. Weiterhin können runde Schaltflächen ausgelöst werden, indem diese in einem bestimmten Bereich, der deutlich hervorgehoben ist, wieder verlassen werden. Es wird stets dargestellt, welche Geste (Kamera-, Cursor-Steuerung oder weder noch) gerade aktiv ist. In Sculpting, entwickelt von Leap Motion selbst, können Körper wie z. B. eine Kugel oder ein Zylinder deformiert werden. Verschiedene Materialien und Werkzeuge stehen zur Verfügung. Die Kamera wird wie zuvor durch eine flache Hand gesteuert, die Finger müssen nun jedoch gespreizt sein. Ein Vergrößern bzw. Verkleinern ist durch Bewegungen parallel zur Z-Achse möglich. Beim Ausstrecken des Zeigefingers erscheint ein runder durchsichtiger Cursor. Befindet sich dieser auf dem Modell, so können Deformierungen durch Bewegungen

[19] https://apps.leapmotion.com/apps/form-and-function-3d
[20] https://apps.leapmotion.com/apps/sculpting
[21] https://unity3d.com

zum Bildschirm hin umgesetzt werden. Wird der Cursor auf eine Schaltfläche bewegt, so wird er deutlich kleiner und mit maximaler Deckkraft dargestellt. Es erscheint ein ringförmiges Menü, in dem Segmente Menüeinträge repräsentieren. Zum Auslösen einer Aktion wird der Cursor einfach über das gewünschte Segment nach außen bewegt. Indem man zwei Hände voneinander weg bewegt, werden die Schaltflächen ausgeblendet. Die umgekehrte Geste blendet sie wieder ein.

Einige passende Apps findet man im App Store der Leap Motion. Viele davon sind kostenpflichtig. Im Folgenden sollen die übrigen Apps angesprochen werden. Mit Cyber Science Motion[22] kann ein menschlicher Schädel betrachtet und in einzelne Teile zerlegt werden. Das Steuern des Modells erfolgt durch Bewegungen der flachen Hand mit gespreizten Fingern in alle drei Dimensionen. Falls man den Zeigefinger ausgestreckt, erscheint ein Cursor-Ring. Wird dieser dann auf das Modell bewegt, so wird er ausgefüllt und wechselt seine Farbe. Als Tooltip wird eingeblendet, auf welchem Teil des Schädels man sich gerade bewegt. Für das Zerlegen und Zusammenführen des Schädels ist zusätzlich die Tastatur erforderlich. Beim ersten Start der Anwendung wird ein Tutorial gestartet. Hierbei müssen die genannten Gesten für eine definierte Strecke ausgeführt werden (Fortschritt durch Prozentzahl gegeben). Von Cyber Science Motion gibt es auch eine Version mit Tiermodellen[23]. Die Interaktion ist jedoch identisch, sodass sie nicht behandelt wird. Der Amadeus Travel Seeker[24] macht Vorschläge für Urlaubsziele und liefert hierzu die Flugpreise zurück. Mehrere Benutzer können definiert werden, die unterschiedliche Präferenzen haben (z. B. Kontinente und Aktivitäten). Unter Berücksichtigung der üblichen Eingabedaten wie Startflughafen, Reisezeitraum usw. werden die Preise für Flüge zu passenden Zielen auf einem Globus dargestellt. Die Rotation des Globus erfolgt durch eine Bewegung der flachen Hand in X- und Y-Richtung (gespreizte Finger). Für ein Heranzoomen um eine Stufe muss ein Finger einen Kreis im Uhrzeigersinn zeichnen. In der umgekehrten Richtung erfolgt dementsprechend das Herauszoomen. Ein Cursor ist außer bei einer Fausthaltung immer eingeblendet. Befindet sich der Cursor auf dem Globus und wird die flache Hand ausgeführt, so wird der Cursor mit mehreren Ringen visualisiert (Rotation des Globus aktiv). In allen anderen Fällen wird er nur durch einen Ring repräsentiert. Letzteres ist auch erforderlich, um eine Schaltfläche auszulösen. Um einen Flugpreis auf dem Globus zu selektieren, muss nur der Zeigefinger ausgestreckt sein. Eine Schaltfläche wird nach einer definierten Wartezeit angeklickt, was durch ein Füllen des Cursors visualisiert wird. Es wird generell eine Warnung eingeblendet, falls man sich zu nah an der Leap Motion befindet. Zuletzt findet man im App Store noch die OS X Anwendung „Molecules“, die wie MegaMol eine Visualisierung von molekularen Strukturen erlaubt. Da ein iMac zur Verfügung steht, wurde diese An-

[22] https://apps.leapmotion.com/apps/cyber-science-motion
[23] https://apps.leapmotion.com/apps/cyber-science-motion-zoology-trial
[24] https://apps.leapmotion.com/apps/travel-seeker

wendung ebenso berücksichtigt. Es kam hierbei das V2 SDK zum Einsatz. Wie bereits in Abschnitt 2.2 erwähnt wurde, ist das Orion SDK aktuell ausschließlich Windows-Nutzern vorbehalten. Die Rotation und das Zoomen der Visualisierung erfolgen wie bei „Cyber Science Motion“. Bewegt man zwei flache Hände in den Erfassungsbereich, so kann man durch Bewegungen beider Hände in eine bestimmte Richtung eine Kameraverschiebung parallel zur Visualisierung erreichen. Die Benutzeroberfläche ist auf eine Maus-Tastatur-Bedienung ausgerichtet.

Außerhalb des App Stores kann die Leap Motion beispielsweise auch in Google Earth genutzt werden. Hier kann ein virtuelles Flugzeug gesteuert werden, indem man dessen Ausrichtung mit der Hand imitiert [Neu14]. Neigt man die Hand nach oben, so geht das Flugzeug entsprechend in den Steigflug. Durch ein Senken hingegen nähert sich es zunehmend der Erdoberfläche. Ein Links- und Rechtskurve wird durch ein Neigen der Hand nach links bzw. rechts umgesetzt.

Weiterhin existiert ein Plug-in für die Animations- und Modellierungssoftware Autodesk Maya[25], das eine Unterstützung für die Leap Motion bereitstellt. Dieses erlaubt es beispielsweise die Positionsdaten der Hände auf die Freiheitsgrade eines Maya-Modells abzubilden [Neu14]. Durch entsprechende Handbewegungen kann man so mit dem Modell interagieren. Auch ist eine Navigation durch den Raum möglich, in dem die dreidimensionalen Modelle angeordnet sind [Neu14]. Handbewegungen in alle drei Dimensionen verändern die Position und Orientierung der virtuellen Kamera [Neu14]. Zuletzt können durch entsprechende Fingerbewegungen Modelle u. a. verformt oder verzogen werden [Neu14], ähnlich wie bei der zuvor vorgestellten App „Sculpting“.

Ploner [Plo12] hat in seiner Diplomarbeit ein Interaktionskonzept für eine Datenanalyse auf Powerwalls entwickelt, das vollständig auf berührungslose Gesten basiert. Powerwalls sind große, hochauflösende Displays, die durch Projektoren oder eine Vielzahl von LCD-Bildschirmen („Tiled LCD Panels“) realisiert werden. Zunächst werden zwei mögliche Anwendungsszenarien für eine entsprechende Gestensteuerung beschrieben. Im ersten Szenario geht es um eine Analyse von Eyetrackingdaten einer Studie. Das zweite Szenario behandelt die Präsentation von Forschungsergebnissen im Bereich der Visualisierung von Proteinen. Hierfür kommt MegaMol beispielhaft zum Einsatz. Anschließend wird ein Nah- und Fernbereich definiert, indem unterschiedliche Gesten zur Verfügung stehen sollen. In Nahbereich kann ein Nutzer die gesamte Fläche der Powerwall nicht mehr überblicken. Er führt hier Detailaufgaben an einer Visualisierung aus. Ein Nutzer ist durchgehend als halbtransparenter Schatten auf der Powerwall sichtbar. Dies erhöht die Orientierung und dient gleichzeitig als Menü-Schnittstelle. Um den Schatten herum werden Menü-Objekte platziert, die ein Nutzer

[25] http://area.autodesk.com/mayaleapplugin

auslösen kann. Es werden eine ganze Reihe von Gesten definiert, mit denen Ansichten gewechselt, verschoben, gelöscht, skaliert und dupliziert (Fernbereich) sowie der Inhalt von Visualisierungen modifiziert (Nahbereich) werden kann. Diese Gesten sind aus einer Pilotstudie hervorgegangen, an der vier Probanden teilgenommen haben. Zum Skalieren müssen beide Hände auf der Ansicht positioniert und anschließend voneinander weg bewegt bzw. zusammengeführt werden. Die zurückgelegte Strecke definiert die Intensität der Vergrößerung bzw. Verkleinerung. Generell beginnt im Fernbereich die Interaktion, falls die Hand für eine definierte Zeit nicht bewegt wird und sich zudem über der Hüfte befindet. Die Interaktion ist beendet, falls die Hand ebenso wieder für eine bestimmte Zeit ruht. Im Nahbereich ist die Interaktion aktiv, wenn der Nutzer die Hand geschlossen hält. Auf die weiteren Gesten wird nicht eingegangen, da sie für diese Arbeit nicht relevant sind. Abschließend wird prototypisch eine Analyseumgebung für Eyetracking-Daten implementiert, welche die beschriebenen Konzepte umsetzt. Zur Gestenerkennung kommt die erste Kinect-Generation zum Einsatz.

4 Konzeption

Dieses Kapitel erläutert die konzeptuelle Entwicklung der Gestensteuerung. Damit wird die Basis geschaffen für die praktische Umsetzung, die im darauffolgenden Kapitel 5 beschrieben ist. Auf die Kamerasteuerung wird in Abschnitt 4.1 eingegangen, Abschnitt 4.2 widmet sich der Benutzeroberfläche. Abschließend werden in Abschnitt 4.3 weitere Gesten behandelt, die im Rahmen dieser Arbeit realisiert worden sind.

Generell wurde festgelegt, das die Steuerung ausschließlich mittels Hand- bzw. Fingergesten umgesetzt werden sollte, da die Leap Motion ohnehin lediglich Hände erfassen kann. Zudem sind entsprechende Gesten weit verbreitet, insbesondere auf Touch-Geräten, sodass es einem Nutzer leichter fällt, sie auch in einem räumlichen Kontext anzuwenden. Blickt auf die Anwendungen, die in Abschnitt 3 vorgestellt worden sind, so sind ebenso ausschließlich Hände und Finger zum Einsatz gekommen.

4.1 Kamerasteuerung

Auf Touch-Geräten erfolgt das Rotieren eines Objekts in der Regel durch ein Ziehen eines Fingers über den Bildschirm. Die Drehrichtung wird durch die Richtung der Fingerbewegung, die Empfindlichkeit durch die zurückgelegte Distanz festgelegt. Es hat sich daher angeboten, diese Ansatz auf den räumlichen Kontext zu übertragen. Da keine Berührung mit einer Oberfläche erfolgt, musste zunächst eine Handhaltung gefunden werden, in der die Steuerung aktiviert ist. Da eine Faust nicht erlernt werden muss und zugleich das Gefühl gibt, die Visualisierung in den Händen zu halten, ist die Entscheidung letztendlich auf diese Handhaltung gefallen. Noch stärker ist dieses Gefühl ausgeprägt, wenn man eine Greifhand ausführt, die Hand also nicht flach, aber auch nicht geschlossen hält. Eine Greifhand wurde jedoch subjektiv als etwas anstrengender bewertet. Der entscheidende Ausschlussgrund war jedoch, dass eine zuverlässige Erkennung einer solchen Geste sehr aufwändig ist, insbesondere wenn die Geste auch in verschiedenen Orientierungen erkannt werden soll (Handfläche nicht parallel zum Gerät gerichtet). Da sowohl das SDK der Leap Motion als auch das der Kinect bereits Möglichkeiten bereitstellen, um eine Faust zu erkennen, wurde für beide Geräten dieselbe Handhaltung eingesetzt. Die Implementierung

der Fausterkennung wird für die Leap Motion in Abschnitt 5.2.1, für die Kinect in Abschnitt 5.2.2 behandelt. Eine Faustbewegung in X-Richtung lässt die virtuelle Kamera von MegaMol entgegen dem Uhrzeigersinn um die Y-Achse rotieren. Durch eine Bewegung in Y-Richtung bewegt sich die Kamera entgegen dem Uhrzeigersinn um die X-Achse. Handbewegungen in die entgegengesetzte Richtungen führen entsprechend zu Drehungen in die umgekehrten Richtungen.

Das Vergrößern und Verkleinern erfolgt auf Touchscreens üblicherweise durch das voneinander weg bewegen bzw. zusammenführen zweier Finger, einer Multi-Touch-Geste. Die Empfindlichkeit ergibt sich ebenfalls durch die Strecke, welche die Finger auf dem Bildschirm zurücklegen. Denkbar wäre es nun, diese Geste im räumlichen Kontext durch zwei Fausthaltungen umzusetzen. Letztendlich ist die Entscheidung dagegen ausgefallen. Da im Gegensatz zu Touchscreens eine dritte Dimension zur Verfügung steht, bietet sich an, diese zum Zoomen zu nutzen. Eine zweite Hand ist somit nicht erforderlich. Es wurde festgelegt, das eine Faustbewegung in Z-Richtung die Kamera entgegen der Z-Achse bewegt (Heranzoomen). Das Herauszoomen erfolgt durch die entgegengesetzte Bewegungsrichtung.

Ein Benutzer kann so die komplette Kamerasteuerung mit einer Hand ausführen, was als intuitiv und zugleich weniger ermüdend eingestuft worden ist. Zudem kann die andere Hand genutzt werden, um gleichzeitig einen Cursor der Benutzeroberfläche zu steuern (siehe Abschnitt 4.2.1). Die Geste zur Kamerasteuerung ist in Abbildung 4.1 veranschaulicht.

Wie empfindlich die Kamera auf eine Handbewegung mit einer bestimmten Geschwindigkeit reagiert, kann über das Front-End von MegaMol konfiguriert werden. Die Float-Parameter sind unter „View3DMotionController“ → „CameraControl“ zu finden. „RotationAroundXAxisSensitivity“ und „RotationAroundYAxisSensitivity“ betreffen die Rotationen um die X- bzw. Y-Achse. Die Empfindlichkeit des Vergrößerns und Verkleinerns wird über „TranslationInZDirectionSensitivity“ festgelegt. Die Konfiguration gilt für alle Geräte. Rotationen erfolgen immer zum Mittelpunkt der Bounding Box, unabhängig vom aktuell eingestellten Blickpunkt („look-at point“).

Die normale Maus-Tastatur-Bedienung der View erlaubt noch weitere Kamerabewegungen. Unter anderem sind auch Rotationen um die Z-Achse oder dem Mittelpunkt der Kamera selbst möglich. Diese Möglichkeiten wurden nicht in der Steuerung abgebildet, da sie sich als nicht notwendig herausgestellt haben. Zusätzliche Gesten würden die Komplexität der gestischen Schnittstelle und die Wahrscheinlichkeit für Fehlerkennungen erhöhen.

Die Kamera kann immer nur von einer Hand gesteuert werden. Falls ein zweiter Benutzer ebenfalls eine Faust macht, so wird dessen Handbewegung ignoriert. Öffnet

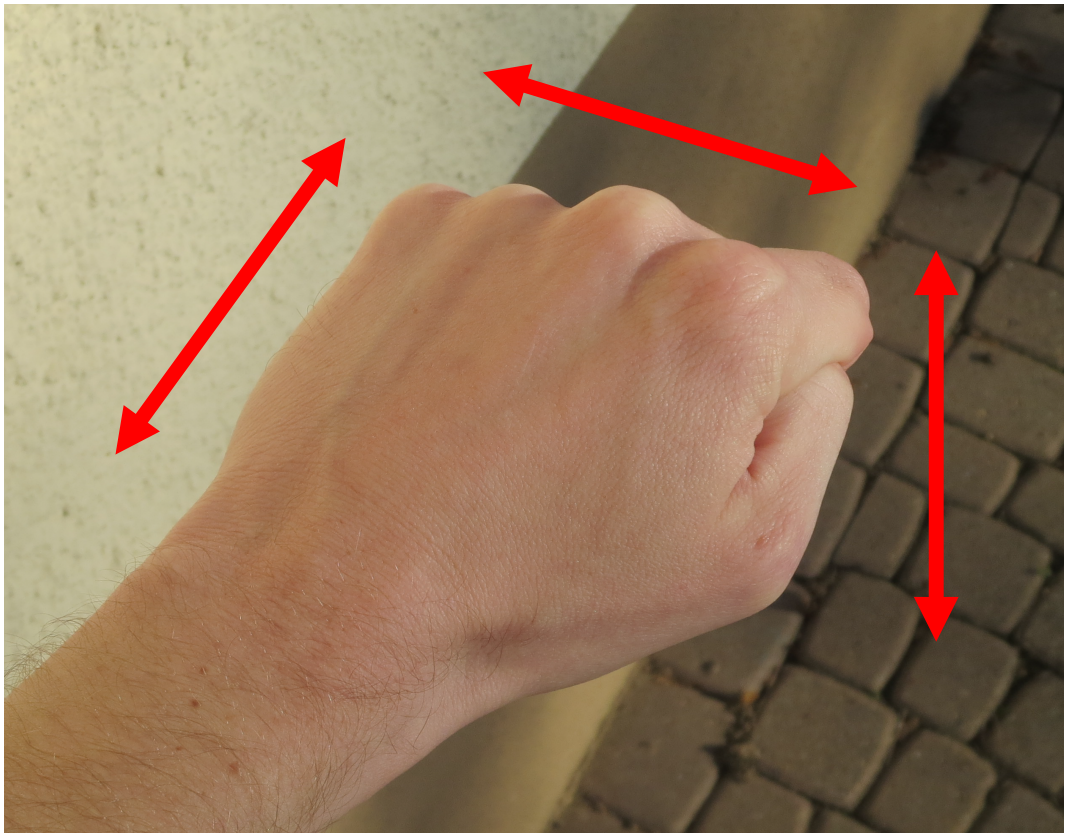

Abbildung 4.1: Handgeste zur Steuerung der virtuellen Kamera für Leap Motion und Kinect

der erste Nutzer seine Faust, so kann der zweite sofort die Steuerung übernehmen. Leap Motion und Kinect können auch gleichzeitig eingesetzt werden. In diesem Fall gilt diese Regel ebenso. Abschließend sollte noch erwähnt werden, dass sich bei der Kinect eine Hand immer mindestens 10 cm über der Hüfte befinden muss. Ist dies nicht mehr erfüllt, so wird die Steuerung sofort angehalten bzw. von einer anderen Person übernommen. Auf diese Weise soll vermieden werden, dass die Faust einer hängenden Hand die Kamerasteuerung unbewusst auslöst.

4.2 Benutzeroberfläche

In den folgenden Unterabschnitten wird die Konzeption der Benutzeroberfläche beschrieben. Das existierende Front-End erlaubt eine Anpassung aller Parameter, ist jedoch ausschließlich auf eine Verwendung mit Maus und Tastatur ausgerichtet. Abschnitt 4.2.1 widmet sich der gestenbasierten Cursor-Interaktion und den Widgets (Komponenten), welche die Benutzeroberfläche zu Verfügung stellt. Eine 3D-Position einer Cursor-Geste muss auf eine Cursor-Position in der Benutzero-

berfläche abgebildet werden. Wie dies umgesetzt wird, erläutert Abschnitt 4.2.2. Die Konfigurationsmöglichkeiten der Benutzeroberfläche werden in Abschnitt 4.2.3 beschrieben.

4.2.1 Cursor-Interaktion und Widgets

Die Interaktion mit der Benutzeroberfläche soll mittels einem Cursor erfolgen, da dieses Konzept jedem PC-Nutzer bekannt ist. Blickt man wieder auf Touch-basierte Geräte, so werden diese im einfachsten Fall durch eine Berührung mit dem Zeigefinger genutzt. Diese Tatsache hat zur Idee geführt, eine Hand mit ausgestrecktem Zeigefinger zur Steuerung eines Cursors einzusetzen. Durch Bewegungen in X- und Y-Richtung wird der Cursor bewegt, die Z-Koordinate wird verworfen.

Die Leap Motion wurde für eine Erfassung von Händen und Fingern entwickelt. Ihr internes Modell einer Hand (siehe Abschnitt 2.2) ist dementsprechend sehr detailliert. Das SDK liefert daher auch Informationen darüber zurück, welche Finger aktuell ausgestreckt sind. Aus diesem Grund wurde die Entscheidung getroffen, die genannte Handhaltung für dieses Gerät einzusetzen. Die Implementierung ist in Abschnitt 5.2.1 beschrieben. Abbildung 4.2 veranschaulicht die Geste.

Die Kinect besitzt ein deutlich einfacheres Handmodell (siehe Abschnitt 2.3), insbesondere wird nicht zwischen einzelnen Finger unterschieden. Folglich kann zumindest mittels Funktionen des SDK keine Erkennung einer Hand mit ausgestrecktem Zeigefinger realisiert werden. Es existieren verschiedene Bibliotheken und Werkzeuge [Bor12; Met16; Pte16; Ste12], die eigene Algorithmen zum Erkennen von Fingern implementieren. Dieses liefern jedoch lediglich Positionen und Konturen der Finger zurück. Explizite Funktionen, die angeben, ob Finger ausgestreckt sind, gibt es nicht. Zudem muss die Handfläche stets parallel zur Kinect ausgerichtet sein, damit überhaupt Fingerpositionen zurückgegeben werden. War in dieser Orientierung lediglich der Zeigefinger ausgestreckt, so wurden keine Positionsdaten zurückgeliefert. Letztendlich wurde daher die Entscheidung getroffen, für die Kinect eine andere Handhaltung festzulegen. Da zum Steuern der virtuellen Kamera eine Faust verwendet wird, hat es sich angeboten, für die Cursor-Steuerung die entgegengesetzte Handhaltung zu verwenden, die flache Hand. Diese kann ebenso wie die Faust problemlos mit dem SDK erkannt werden. Die Implementierung ist in Abschnitt 5.2.2 erläutert. Abbildung 4.3 veranschaulicht die Geste. Die flache Hand ist hier nahezu ganz nach oben orientiert. Wie sich später im Rahmen der Evaluation (siehe Abschnitt 6) zeigt, klappt die Erkennung ebenso problemlos, falls die Hand auch nur etwas aufgerichtet ist. Analog zur Faust muss die flache Hand mindestens 10 cm über der Hüfte ausgeführt werden, damit sie erkannt wird.

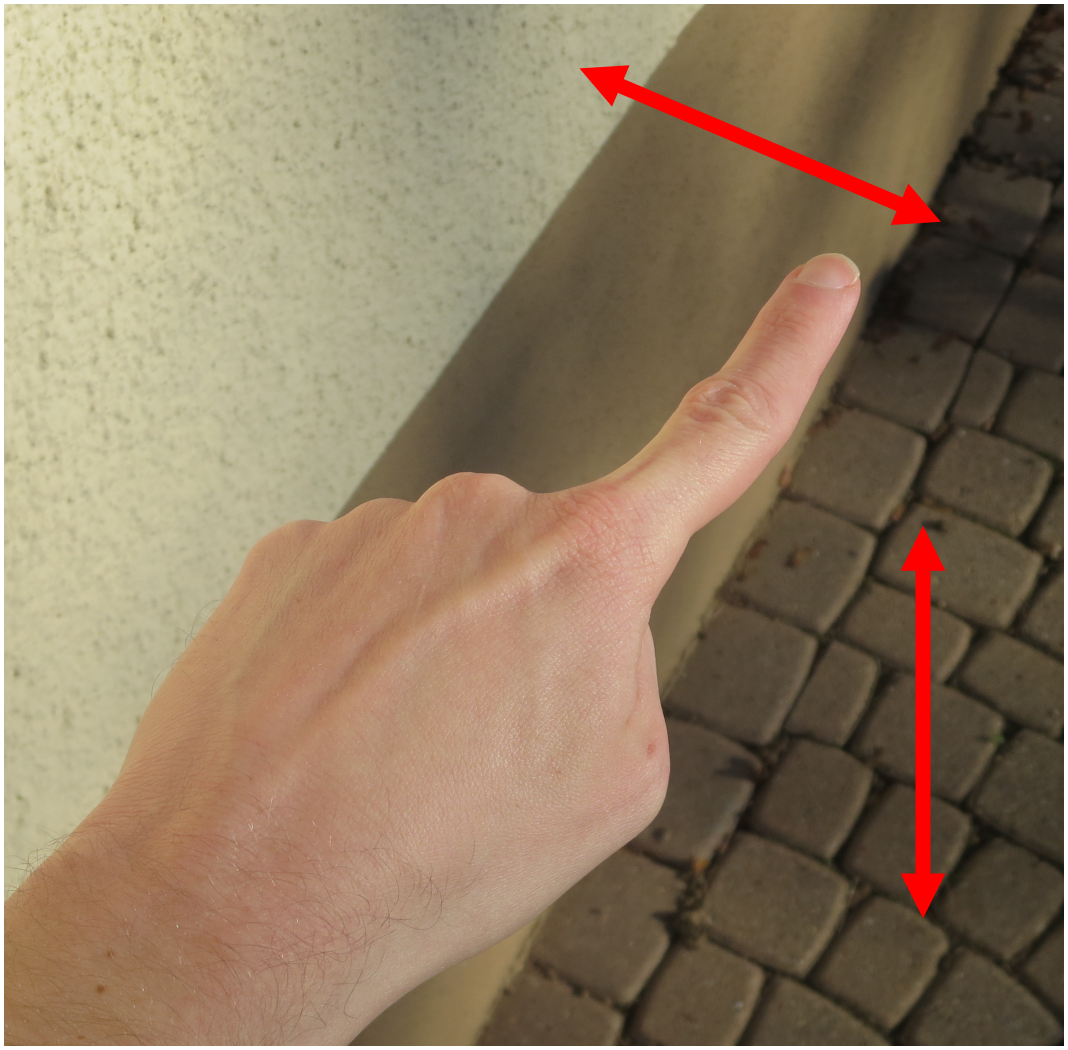

Abbildung 4.2: Handgeste zur Steuerung eines Cursors für Leap Motion

Entsprechend der Aufgabenstellung sollen beliebige Parameter der Visualisierung über die Benutzeroberfläche verändert werden können. Wie bereits in Abschnitt 2.1 beschrieben, unterstützt MegaMol verschiedene Parametertypen. Es wäre nun denkbar für jeden Typ ein separates Widget zu definieren wie es auch bereits im Front-End umgesetzt ist. Beispielsweise könnte ein Enumeration-Parameter über ein Liste repräsentiert werden. Ein Float-Parameter, der definierte Grenzen besitzt, könnte über ein Schieberegler visualisiert werden. Berücksichtigt man jedoch, das die Benutzeroberfläche in einer Ausstellung zum Einsatz kommen soll, in der kein unterstützendes Personal zur Verfügung steht, so ist es wenig sinnvoll, einen Benutzer mit einer Vielzahl von Widgets und Einstellmöglichkeiten zu überfordern. Nutzer sollten in einer minimalen Zeit Parameter der Visualisierung modifizieren können. Daher wurde entschieden, lediglich Schaltflächen anzubieten.

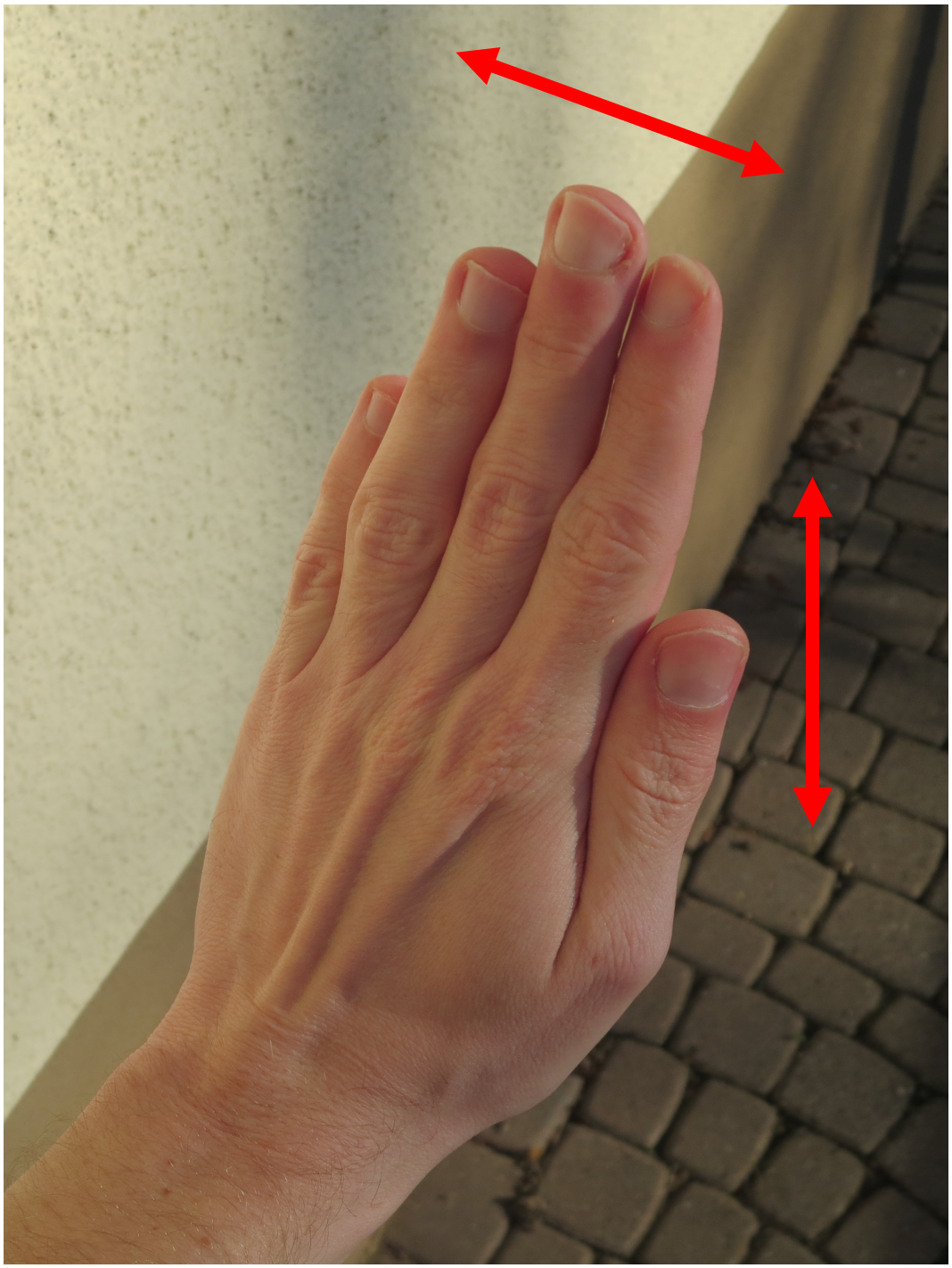

Abbildung 4.3: Handgeste zur Steuerung eines Cursors für Kinect

Jede Schaltfläche kann mit einer oder mehreren Aktionen verknüpft werden, die bei einem Klick eines Cursors ausgeführt werden. Ein Aktion weist einem Parameter einen definierten Wert zu.

Zunächst wurde ein Cursor als Kreis visualisiert. Ein Klick wurde durch eine schnelle, kurze Stoßbewegung der Hand mit ausgestrecktem Zeigefinger bzw. flachen Hand nach vorne ausgeführt. Veranschaulicht wurde dies durch ein kurzes Blinken des Cursors. Es hat sich jedoch herausgestellt, dass die Stoßbewegung häufig nicht wie vorgesehen ausgeführt wird. Entweder wurde die Geschwindigkeit und Länge der Bewegung nicht korrekt umgesetzt oder der Cursor ist während der Stoßbewegung verrutscht. Inspiriert von den Gestensteuerungen der Xbox 360 und Xbox One wurde daraufhin eine Wartezeit definiert, die ein Cursor auf einer Schaltfläche verweilen muss. Ist diese verstrichen, erfolgt der Klick. Abbildung 4.4 zeigt, wie der Cursor animiert ist. Das rote Ringsegment repräsentiert die verstrichene Zeit seit dem Betreten der Schaltfläche oder dem letzten Klick. Das weiße Segment veranschaulicht die Restzeit, bis die Schaltfläche ausgelöst wird. Befindet sich der Cursor auf keiner Schaltfläche, so bleibt der Cursor vollständig weiß. Selbiges gilt, falls sich der Cursor über zwei oder mehr Schaltflächen bewegt (nicht erlaubter Zustand). Durch die Animation soll ein selbständiges Erlernen der Interaktion ermöglicht werden. Die Farben wie auch Größe des Cursors können frei definiert werden. Die Breite des Rings ist fest vorgegeben und beträgt 35 Prozent des Durchmessers. Standardmäßig kann ein Cursor die Benutzeroberfläche nicht verlassen, er ist stets vollständig sichtbar. Dieses als „Clamping“ bezeichnete Verhalten ist z. B. auch vom Mauszeiger bekannt. Abschnitt 4.2.3 befasst sich mit der Konfiguration der Benutzeroberfläche. Im Gegensatz zur Kamerasteuerung, die zu jeder Zeit immer nur von einer Faust ausgeführt werden kann, können mehrere Cursor sichtbar sein. Nutzer können mit ihren Händen gleichzeitig zwei Cursor steuern. Selbiges gilt für Kamera und Cursor. Ebenso können auch Leap Motion und Kinect kombiniert eingesetzt werden.

Abbildung 4.4: Visualisierung eines Cursors der Benutzeroberfläche

4.2.2 Umrechnung auf Cursor-Position

Die räumliche Handposition, die von der Leap Motion bzw. Kinect bereitgestellt wird, muss auf eine Cursor-Position in der Benutzeroberfläche umgerechnet werden. Da die Benutzeroberfläche zweidimensional realisiert wurde, kann die Z-Koordinate verworfen werden. Ein absolutes Umrechnungsverfahren, das beispielsweise auch das Leap Motion SDK bereitstellt, stellt die Definition einer Interaction Box dar. Hierbei handelt es sich um ein Rechteck, das im Falle der Leap Motion ausgehend vom Mittelpunkt der Oberseite des Geräts definiert wird. Die Handposition kann so auf die normalisierte Position bezüglich der Interaction Box umgerechnet werden. Befindet sich die Hand z. B. genau in der Mitte des Bereichs, besitzen alle Koordinaten den Wert 0,5. Liegt die Hand außerhalb des Bereichs, wird dementsprechend für eine oder mehrere Koordinaten der Wert 0 festgelegt. Es ist darauf zu achten, dass die Interaction Box vollständig innerhalb des Erfassungsbereichs des Geräts liegt. Sowohl für die Leap Motion als auch Kinect wurde diese Methode umgesetzt. Für die Kinect erfolgt die Definition jedoch nicht ausgehend vom Gerät. Benutzer, die sich vor dem Gerät befinden, haben individuelle Bereiche, in denen sie ihre Hände bewegen. Es ist daher besser, relativ von einem Körperteil der jeweiligen Person auszugehen. Nach einigem Ausprobieren hat sich die Schulter als beste Lösung herausgestellt. Die Definition gilt für die linke Hand ausgehend von der Position der linken Schulter. Analog kann dies auf die rechte Hand übertragen werden. Die Geschwindigkeit des Cursors in X- und Y-Richtung ist durch die Höhe bzw. Breite der Interaction Box definiert.

Alternativ kann auch eine normalisierte Position in der Benutzeroberfläche definiert werden, von der aus der Cursor relativ bewegt wird. In diesem Fall muss die Geschwindigkeit der Cursors separat angegeben werden. Falls sich die Schaltflächen beispielsweise ausschließlich auf der linken Seite befinden, kann festgelegt werden, dass der Cursor zu Beginn immer in diesem Bereich erscheint.

Zur Konfiguration stehen im Front-End unter „View3DMotionController“ → „<Gerätename>“ → „CursorControl“ entsprechende Parameter zur Verfügung. Die Spezifikation einer Interaction Box erfolgt durch die untere linke Ecke sowie den Dimensionen. Längen werden grundsätzlich in Millimeter angegeben. Für die Leap Motion ist standardmäßig eine Interaction Box mit einer Breite von 400 und einer Höhe von 280 definiert, deren linke untere Ecke im Punkt (-200 | 200) liegt. Ebenso ist auch bei der Kinect die absolute Umrechnung voreingestellt. Die Interaction Box besitzt eine Breite und Höhe von 400, die untere linke Ecke befindet sich in (-200 | -200). Diese Konfiguration hat sich in eigenen Versuchen sowie der Studie (siehe Abschnitt 6.3) bewährt. Alle Teilnehmer konnten den Cursor über den kom-

pletten Bildschirm bewegen. Wechselt man auf das relative Umrechnungsverfahren, so beginnt der Cursor standardmäßig in der Mitte der Benutzeroberfläche.

4.2.3 Konfiguration

Da Konfigurations- und Projektdateien von MegaMol in XML definiert sein müssen (siehe Abschnitt 2.1), wurde die Entscheidung getroffen, ebenso dieses Datenformat für die Konfiguration der Benutzeroberfläche einzusetzen. Im Front-End befindet sich unter „XMLUIRenderer“ der String-Parameter „LoadUIConfigFromXMLFile“, dem der Pfad zur XML-Datei übergeben werden muss. Handelt es sich um einen relativen Dateipfad, so wird in allen Ressource-Verzeichnissen nach der entsprechenden Datei gesucht. Wie bereits erwähnt wurde, erfolgt die Definition der Ressource-Verzeichnisse in der MegaMol-Konfigurationsdatei. Sollte der relative Pfad für mehrere Ressource-Verzeichnisse gültig ist, so wird das zuerst in der MegaMol-Konfigurationsdatei definierte Ressource-Verzeichnis gewählt. Ist der Pfad hingegen absolut, wird er direkt verwendet. Eine gültige Konfiguration (XML-Datei und darin enthaltende Dateipfade gültig) überschreibt die aktuelle Konfiguration. Falls hingegen Fehler beim Laden auftreten, bleibt die derzeitige Konfiguration unberührt und die entsprechenden Fehlermeldungen können der Konsole entnommen werden. Während dem Start von MegaMol wird versucht die Datei „ui.xml“ zu laden.

Im Folgenden sollen nun die Möglichkeiten der Konfiguration erläutert werden. Zur leichteren Verständlichkeit erfolgt dies anhand einer Beispiel-Konfiguration, die in Listing 4.1 dargestellt ist.

```
1  <MegaMol type="ui" version="1.0" xmlns="http://megamol.org/nui"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://megamol.org/nui ui.xsd">
4    <Fonts>
5      <Font id="opensans" src="C:/My Fonts/OpenSans-Regular.ttf"/>
6    </Fonts>
7    <Images>
8      <Image id="imgBucketRainbow" src="images/imgBucketRainbow.png"/>
9    </Images>
10   <Buttons>
11     <Button x="25" y="25" width="420" height="150">
12       <Backgrounds>
13         <Default>
14           <RadialGradient startColor="#7a7a7a" endColor="#1a1a1a" innerRadius="20"
                 outerRadius="270"/>
15         </Default>
16       </Backgrounds>
17       <Content iconLabelGap="30">
18         <ImageIcon image="imgBucketRainbow"/>
```

```
19        <Label font="opensans" size="80" text="Rainbow" color="#fff"/>
20      </Content>
21      <Actions>
22        <SetParameterValue name="cartoonren::color::coloringMode1" value="Rainbow"/>
23      </Actions>
24    </Button>
25    <Button x="2%" y="25" align="topRight" origin="topRight" width="20%"
26         height="150">
27      <RoundedCorners radius="20"/>
28      <Backgrounds>
29        <Default>
30          <Solid color="#d60000"/>
31          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
32        </Default>
33        <Hover>
34          <Solid color="#ff3838"/>
35          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
36        </Hover>
37        <Click>
38          <Solid color="#f66"/>
39          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
40        </Click>
41      </Backgrounds>
42      <Content>
43        <Label font="opensans" size="80" text="Reset"/>
44      </Content>
45      <Actions>
46        <SetParameterValue name="View3DMotionController1::resetView" value=""/>
47        <SetParameterValue name="cartoonren::color::coloringMode1"
48            value="Structure"/>
49      </Actions>
50    </Button>
51  </Buttons>
52  <Cursors radius="50">
53    <Cursor remainingColor="#fff" elapsedColor="#db0400"/>
54  </Cursors>
55</MegaMol>
```

Listing 4.1: Beispiel-Konfiguration der Benutzeroberfläche

Wie der Datei zu entnehmen ist, muss das Wurzelelement der XML-Datei `MegaMol` lauten. Die Attribute `type` und `version` müssen die Werte `ui` bzw. `1.0` besitzen. Bevor Schaltflächen definiert werden, sollten zunächst benötigte Schriftarten und Bilder angegeben werden. Dies erfolgt unter den Elementen `Fonts` und `Images`. In jedem `Font`- bzw. `Image`-Element muss eine ID sowie eine URI[26] angegeben werden, welche

[26] Gemäß XML Schema-Spezifikation sollen Dateireferenzen durch den Typ `xsd:anyURI` repräsentiert werden

auf die entsprechende Datei verweist. Die ID muss unter den definierten Schriften bzw. Bildern eindeutig sein. Ein syntaktisch korrekte absolute URI ist beispielsweise `file:///c:/path/to/image%201.png`. Es ist zu beachten, dass eine Reihe von Zeichen kodiert werden müssen. In diesem Beispiel wurde das Leerzeichen durch `%20` ersetzt. Dies gilt ebenso für eine relative URI, z. B. `../path/to/font%201.ttf`. Sollten normale Dateipfade angegeben werden, die nicht kodiert sind, so werden diese ebenfalls akzeptiert (Fehlertoleranz). Relative Referenzen werden mit den definieren Ressource-Verzeichnissen aufgelöst. Es werden TrueType-Schriften sowie Bilder in den Formaten PNG[27], GIF, JPEG[28], BMP[29], PSD[30], HDR (Radiance RGBE Format), PIC (Softimage PIC), PGM und PPM unterstützt [Mon16]. In obiger Datei wird eine Schriftart sowie ein Bild geladen. Die absolute URI der Schriftart ist ungültig, da kein Protokoll angegeben ist und das Leerzeichen nicht kodiert wurde. Dennoch wird sie akzeptiert.

Schaltflächen werden durch `Button`-Elemente repräsentiert. Position und Größe werden durch die Attribute `x`, `y`, `width` und `height` festgelegt. Werte können dabei absolut oder prozentual zu den aktuellen View-Dimensionen spezifiziert werden. Das Attribut `align` legt fest, welcher Punkt auf der Schaltfläche durch die Attribute `x` und `y` definiert ist. Erlaubte Werte sind `bottomLeft`, `middleLeft`, `topLeft`, `topCenter`, `topRight`, `middleRight`, `bottomRight`, `bottomCenter` und `middleCenter`. Der Ursprung des Koordinatensystems, in dem sich dieser Punkt befindet, wird durch `origin` festgelegt. Hier werden `bottomLeft`, `topLeft`, `topRight` und `bottomRight` akzeptiert. Die Attribute `align` und `origin` sind optional und besitzen standardmäßig jeweils den Wert `bottomLeft`. Die Beispielkonfiguration erstellt zwei Schaltflächen. Die erste Schaltfläche wird von unten links, die zweiten von oben rechts definiert. Weiterhin wird die X-Koordinate sowie die Breite der zweiten Schaltfläche prozentual angegeben.

Abgerundete Ecken werden durch das Element `RoundedCorners` umgesetzt. Der Radius wird in Pixel angegeben.

Eine Schaltfläche verfügt über drei verschiedene Zustände. Das Element `Default` in `Backgrounds` legt den Hintergrund fest, falls sich kein Cursor auf der Schaltfläche bewegt. Element `Hover` definiert den Hintergrund, falls sich mindestens ein Cursor auf der Schaltfläche bewegt. Wird ein Klick durch einen Cursor ausgelöst, so erscheint die Schaltfläche für 250 ms mit dem unter `Click` definierten Hintergrund. Falls kein Hover-Hintergrund definiert wurde, entspricht er dem Default-Hintergrund. Wird kein Click-Hintergrund angegeben, ist er mit dem Hover-Hintergrund identisch. Der

[27] 1-, 2-, 4- oder 8-bit pro Kanal
[28] Baseline und Progressiv
[29] 1-bit per Pixel und Lauflängenkodierung (RLE-Kodierung) werden nicht unterstützt
[30] 8 oder 16-bit pro Kanal

Default-Hintergrund muss definiert werden. Ein einfarbiger Hintergrund wird durch das Element `Solid` umgesetzt. Zur Farbdefinition können alle in der CSS Color Module Level 4 Spezifikation beschriebenen Hexadezimalnotationsarten[31] verwendet werden, also „#RRGGBB“ und „#RRGGBBAA“ sowie die Kurzschreibweisen „#RGB“ und „#RGBA“. Auch können die in der Spezifikation genannten Farbnamen[32] eingesetzt werden. Durch das Element `LinearGradient` kann ein linearer Farbverlauf (zwei Farben; von oben nach unten) festgelegt werden. Radiale Farbverläufe (zwei Farben; von der Mitte nach außen) erfolgen über das Element `RadialGradient`. Die Attribute `innerRadius` und `outerRadius` definieren durch zwei Radien einen Ring, in dem der Verlauf von `startColor` nach `endColor` stattfindet. Alle genannten Füllstile können auch kombiniert eingesetzt werden. Die Reihenfolge der Elemente gibt an, wie die Füllungen überlagert werden. In der Beispiel-Konfiguration wurde für die erste Schaltfläche ein radialer Farbverlauf festgelegt, der in allen Zuständen verwendet wird. Die zweite Schaltfläche besitzt unterschiedliche Hintergründe. Verschiedene Rottöne werden jeweils durch einen linearen weiß-schwarz Verlauf überlagert, der einen reduzierten Alpha-Wert besitzt. Das Ergebnis ist ein Verlauf, der von hellrot nach dunkelrot führt. Durch eine Änderung der einfarbigen Füllung kann die Grundfarbe des Verlaufs einfach angepasst werden.

Das Element `Content` legt den Inhalt der Schaltfläche fest, der aus Text und einem Icon bestehen kann. Ein Icon kann durch ein Bild oder einem Zeichen aus einer Schriftart repräsentiert werden. Text wird durch das Element `Label`, ein Zeichen-Icon durch `FontIcon` umgesetzt. Das Attribut `font` referenziert die ID einer zuvor definierten Schriftart. Die übrigen Attribute definieren Text, Schriftfarbe und Schriftgröße. Wird keine Farbe angegeben, wird schwarz („#000000FF“) verwendet. Für ein Bild-Icon, das durch das Element `ImageIcon` umgesetzt wird, muss die entsprechende ID im Attribut `image` definiert werden. Beim Einfügen des Bilds erfolgt keine Anpassung der Größe. Die Reihenfolge in der Text und Icon auf der Schaltfläche erscheinen (von links nach rechts), wird durch die Reihenfolge der Elemente festgelegt. Wird sowohl Text als auch ein Icon definiert, so wird das Attribut `iconLabelGap` berücksichtigt, was den Abstand festlegt. Ist es nicht vorhanden, wird 20 Pixel festgelegt. Falls mehrere `FontIcon`-, `ImageIcon`- oder `Label`-Elemente definiert sind, wird lediglich immer das erste berücksichtigt. Eine Kombination von `FontIcon` und `ImageIcon` ist nicht möglich. Der gesamte Inhalt wird horizontal und vertikal zentriert zur Mitte der Schaltfläche ausgerichtet. Im Beispiel besitzt die erste Schaltfläche ein Bild-Icon sowie Text (Abstand geändert), die zweite lediglich Text.

Unter `Actions` wird festgelegt, welche Operationen ausgeführt werden, wenn die Schaltfläche durch einen Cursor ausgelöst wird. Aktuell gibt es die Möglichkeit, ei-

[31] https://www.w3.org/TR/css-color-4/#hex-notation
[32] https://www.w3.org/TR/css-color-4/#named-colors

nem Parameter einen Wert zuzuweisen, was durch das Element `SetParameterValue` umgesetzt wird. Der Parametername muss einschließlich dem Modulnamen (in Projektdatei festgelegt) angegeben werden. In der Beispiel-Konfiguration verändern beide Schaltflächen die Einfärbung der Visualisierung. Die zweite Schaltfläche setzt weiterhin die Kamera zurück.

Die Cursor, die bei Ausführung der entsprechenden Handhaltungen erscheinen sollen, müssen unter dem Element `Cursors` explizit definiert werden. Radius wie auch Clamping gelten für alle Cursor und müssen nicht angegeben werden. Standardmäßig beträgt der Radius 30 Pixel, ein Verlassen der Benutzeroberfläche ist nicht möglich. Das Attribut `elapsedColor` eines `Cursor`-Elements definiert die Farbe des Ringsegments, welche die Zeit seit dem Betreten der Schaltfläche oder dem letzten Klick repräsentiert. Durch das Attribut `remainingColor` wird die Farbe des anderen Segments festgelegt, das die Restzeit bis zum Auslösen der Schaltfläche veranschaulicht. In dieser Farbe wird der Ring vollständig dargestellt, falls er sich auf keiner Schaltfläche oder mehreren Schaltflächen gleichzeitig befindet. Lediglich ein Cursor wurde im Beispiel definiert.

Formal ist die Konfiguration durch ein XML Schema definiert, das sich in Anhang A.1 befindet. Insbesondere kann dieses einem XML-Editor übergeben werden, was die Erstellung von Konfigurationsdateien erheblich vereinfacht.

4.3 Weitere Gesten

MegaMol soll zukünftig in einer Ausstellung zum Einsatz kommen, in der kein Personal zur Aufsicht zur Verfügung steht. In diesem Szenario erscheint es sinnvoll, dass nach einer definierten Zeit ohne eine Interaktion durch einen Nutzer die Kamera automatisiert wieder zurückgesetzt wird. Wünschenswert wäre es zudem, wenn diese Operation auch von einem Nutzer manuell ausgeführt werden könnte. Ein typischer Anwendungsfall wäre beispielsweise ein Nutzer, der noch nicht vollständig mit der Gestensteuerung vertraut ist und dementsprechend die Kameraposition so verändert, das die Visualisierung nicht mehr zu sehen ist. Selbiger Effekt kann aber auch durch eine fehlerhafte Erkennung einer Fausthaltung zustande kommen.

Wie bereits in Abschnitt 2.2 erläutert wurde, erlaubt das Leap Motion SDK eine Erkennung von verschiedenen Fingergesten. Eine Kreisbewegung eines Fingers stellt eine dieser Gesten dar. Ein typisches Icon, das eine Rückgängig-Funktionalität visualisiert, ist ein kreisförmig gebogener Pfeil, der entgegen den Uhrzeigersinn zeigt. Aus diesem Grund wurde die Entscheidung getroffen, das Zurücksetzen Kamera für die Leap Motion durch eine Kreisgeste umzusetzen. Der Nutzer bewegt einen Finger

(in der Regel den Zeigefinger) zweimal entgegen dem Uhrzeigersinn im Kreis, um die Kamera zurückzusetzen und damit die Visualisierung wieder vollständig sichtbar zu machen. Es hat sich herausgestellt, dass zwei vollständige Kreisbewegungen sicherstellen, dass die Geste nicht unbeabsichtigt erkannt wird und gleichzeitig der Aufwand für den Nutzer immer noch minimal ist.

Viele Visualisierungen verfügen über eine Animation, die in MegaMol abgespielt werden kann. Da eine Abspiel-Funktion üblicherweise durch ein Dreieck-Symbol visualisiert wird, das nach rechts zeigt, hat es sich angeboten, diese Funktionalität ebenfalls auf eine Kreisbewegung abzubilden. Ein Nutzer bewegt einen Finger zweimal im Uhrzeigersinn im Kreis, um die Animation zu starten. Wird daraufhin die selbe Geste nochmals ausgeführt, wird die Animation wieder angehalten. Auch kam die Idee auf, ein Klatschen für diese Funktionalität zu verwenden. Im offiziellen Leap Motion Forum wurde von einem Mitarbeiter des Herstellers ein Algorithmus zur Erkennung einer Klatschgeste veröffentlicht[33]. Für eine nähere Beschreibung, wie die Erkennung erfolgt, wird auf die referenzierte Webseite verwiesen. Der Algorithmus wurde in C# implementiert, da er für einen Einsatz mit der Spiele-Engine Unity vorgesehen wurde. Zunächst musste daher eine Portierung nach C++ vorgenommen werden. Es hat sich nach einigen Versuchen gezeigt, dass die Erkennung sehr unzuverlässig funktioniert. Wurde die Klatschbewegung bereits ausgeführt, wenn die Hände über das Gerät bewegt worden sind, so ist die Erkennung stets fehlgeschlagen. Dies lag daran, das das Gerät zunächst noch keine oder lediglich eine Hand erfasst hatte. Meist wurde das Klatschen erst nach dem zweiten (oder sogar dritten Versuch) erkannt nachdem beide Hände registriert worden sind. Generell hat die Erkennung deutlich besser geklappt, falls die Hände in der Ausgangsposition zum Klatschen parallel zum Gerät ausgerichtet waren. Selten war es notwendig, die Hände komplett aus dem Erfassungsbereich des Geräts zu entfernen, damit eine Erkennung überhaupt wieder klappt. Aus diesen Gründen wurde daher letztendlich die Kreisgeste zur Aktivierung bzw. Deaktivierung der Animation verwendet. Beide Kreisgesten sind in Abbildung 4.5 veranschaulicht.

Für die Kinect wurden im Rahmen dieser Arbeit keine expliziten Gesten für die genannten Funktionen realisiert. Es bleibt jedoch die Möglichkeit, Schaltflächen zu erstellen, welche die entsprechenden Button-Parameter auslösen.

[33] https://community.leapmotion.com/t/clap-hands-change-scene/5023/3

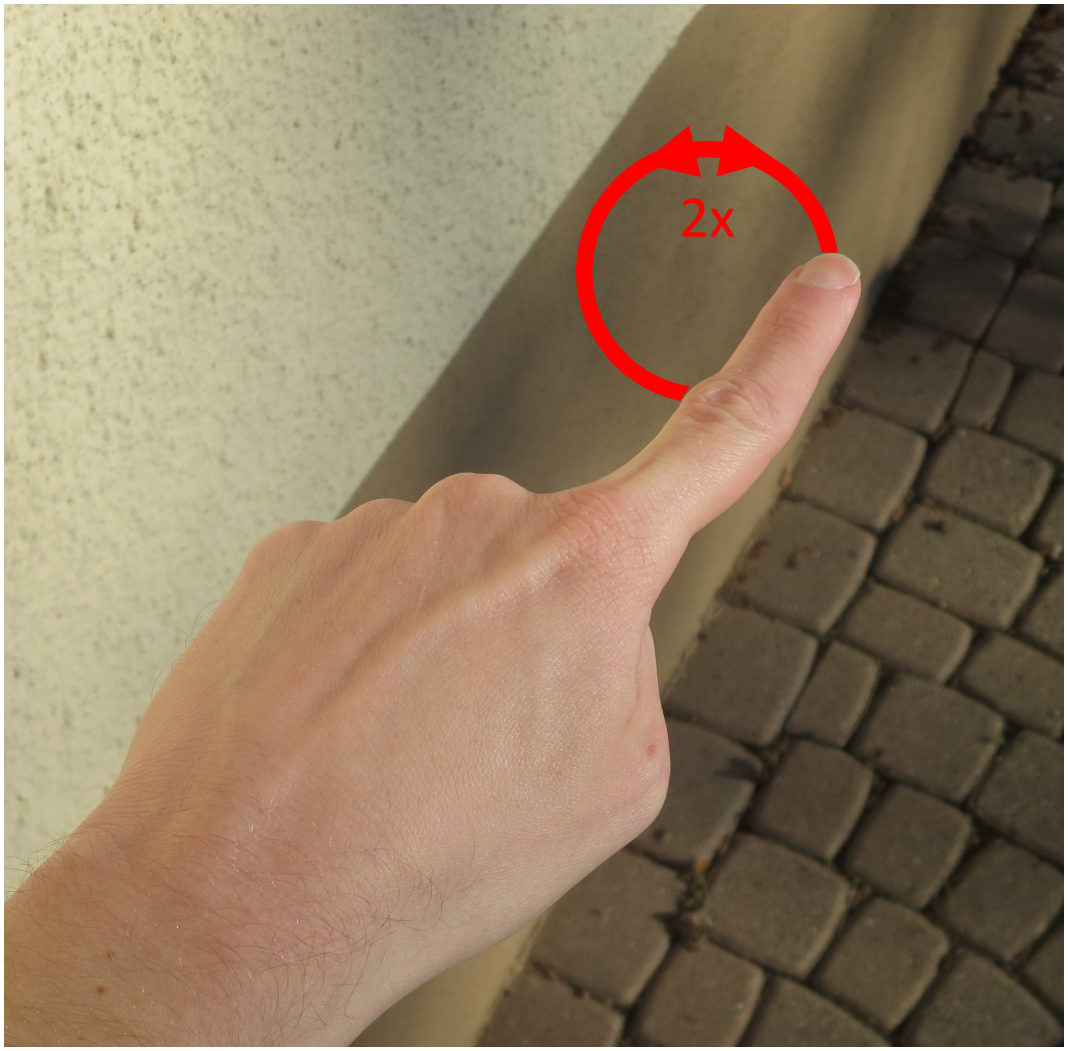

Abbildung 4.5: Fingergesten zum Zurücksetzen der Kamera und Aktivieren/Deaktivieren der Animation für Leap Motion

5 Entwurf und Implementierung

Nachdem im vorherigen Kapitel die Konzepte erläutert worden sind, widmet sich dieses Kapitel nun der technischen Umsetzung der Gestensteuerung. Wie bereits in Abschnitt 2.1 erläutert wurde, kann MegaMol flexibel durch Plug-ins erweitert werden. Für diese Arbeit wurde ein neues MegaMol-Plug-in „nui“ erstellt, dass alle entwickelten Komponenten zusammenfasst. In Abschnitt 5.1 wird zunächst ein Überblick über die Module und Calls dieses Plug-ins gegeben. Zuständigkeiten und Funktionalitäten werden kurz angesprochen. In den darauf folgenden Abschnitten werden die Module dann detailliert behandelt.

5.1 Überblick

Abbildung 5.1 zeigt einen Modulgraph, welcher die Module „View3DMotionController“ und „XMLUIRenderer“ sowie den Call „CallCursorPositionInput“ verwendet, die im Rahmen dieser Arbeit entwickelt worden sind.

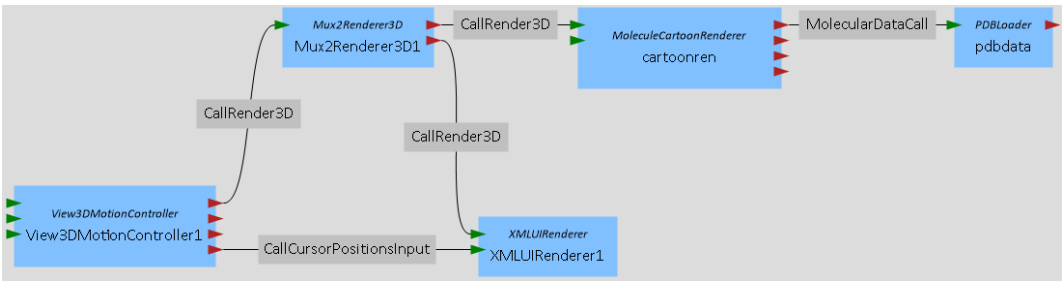

Abbildung 5.1: Modulgraph des MegaMol Configurator, der das Plug-in „nui“ nutzt

Das View-Modul „View3DMotionController“ verwaltet Geräte zur Bewegungssteuerung, realisiert den Zugriff auf deren Daten sowie die Erkennung von Gesten und ist für die Manipulation der Kamera verantwortlich. Wie bereits in Abschnitt 4.2.1 erläutert wurde, erfolgt die Interaktion mit der Benutzeroberfläche über einen Cursor. Wird eine entsprechende Geste erkannt, so erfolgt kontinuierlich eine Umrechnung von der räumlichen Position auf die zweidimensionale Position des Cursors (siehe Abschnitt 4.2.2). Letztere Positionsdaten werden über den Call „CallCursorPositionInput“ an das Modul „XMLUIRenderer“ übergeben, welches die Benutzeroberfläche implementiert.

Da das Modul als Renderer realisiert worden ist, wird ein Multiplexer-Modul benötigt, um es mit anderen Renderer-Modulen kombiniert einsetzen zu können. In der Abbildung wird ein Multiplexer mit zwei Ausgängen („Mux2Renderer3D“) verwendet, der das View-Modul mit den Renderer-Modulen „MoleculeCartoonRenderer“ und „XMLUIRenderer“ verbindet. Es existieren in MegaMol Multiplexer mit bis zu 10 Ausgängen. Die Interaktion mit den Geräten erfolgt im Wesentlichen aus Konsistenzgründen in einem View-Modul. Es existiert bereits das Modul „View3DSpaceMouse“, das die Kamerasteuerung mittels einer SpaceMouse von 3Dconnexion[34] ermöglicht. Die Benutzeroberfläche wurde erst gegen Ende der Entwicklung in ein separates Renderer-Modul ausgegliedert. Diese Entwurfsentscheidung wurde getroffen, damit die Geräte-Logik von der Benutzeroberfläche getrennt ist, wodurch Übersichtlichkeit gewonnen wird. Insbesondere jedoch kann die Benutzeroberfläche so mit beliebigen 3D View-Modulen eingesetzt werden. Zukünftige Module können eine Unterstützung für andere Arten von Eingabegeräten bereitstellen. Damit diese zur Steuerung eines Cursors genutzt werden können, sind meist nur geringe Anpassungen im View- oder XMLUIRenderer-Modul erforderlich.

5.2 Modul „View3DMotionController“

Der Entwurf des Moduls „View3DMotionController“ ist in Abbildung 5.2 veranschaulicht. Möglichst leicht sollen neue Geräte zur Bewegungssteuerung hinzugefügt werden können. Dementsprechend werden diese als separate Klassen repräsentiert (Erweiterungen), welche die gemeinsame Schnittstelle `IMotionController` implementieren müssen. Die Implementierungen für die Leap Motion und Kinect werden in den Abschnitten 5.2.1 und 5.2.2 separat behandelt.

Die Funktion `update` der Schnittstelle `IMotionController` ruft die aktuellen Daten des Geräts ab und führt daraufhin die Gestenerkennung aus. Zurückgeliefert wird eine Enumeration `UpdateResult`, an die der Aufrufer erkennen kann, ob eine (1) Aktualisierung der Daten stattgefunden hat, (2) zurzeit keine neuen Daten verfügbar sind, (3) eine Fehler aufgetreten ist oder (4) das Gerät nicht aktiv ist („not running“), d. h. aktuell keine Daten liefert oder überhaupt nicht mit dem Rechner verbunden ist. Falls ein Gerät nicht mehr aktiv ist oder ein Fehlerfall aufgetreten ist, sollte eine korrekte Implementierung alle gespeicherten Daten löschen. Bei einem aktiven Gerät, das noch keine neuen Daten besitzt, sollte dies hingegen nicht durchgeführt werden.

Wurden die Daten durch einen Aufruf von `update` aktualisiert, so kann über die Funktion `getCameraControllerPositions` für jede erkannte Handhaltung zur Ka-

[34] http://www.3dconnexion.de

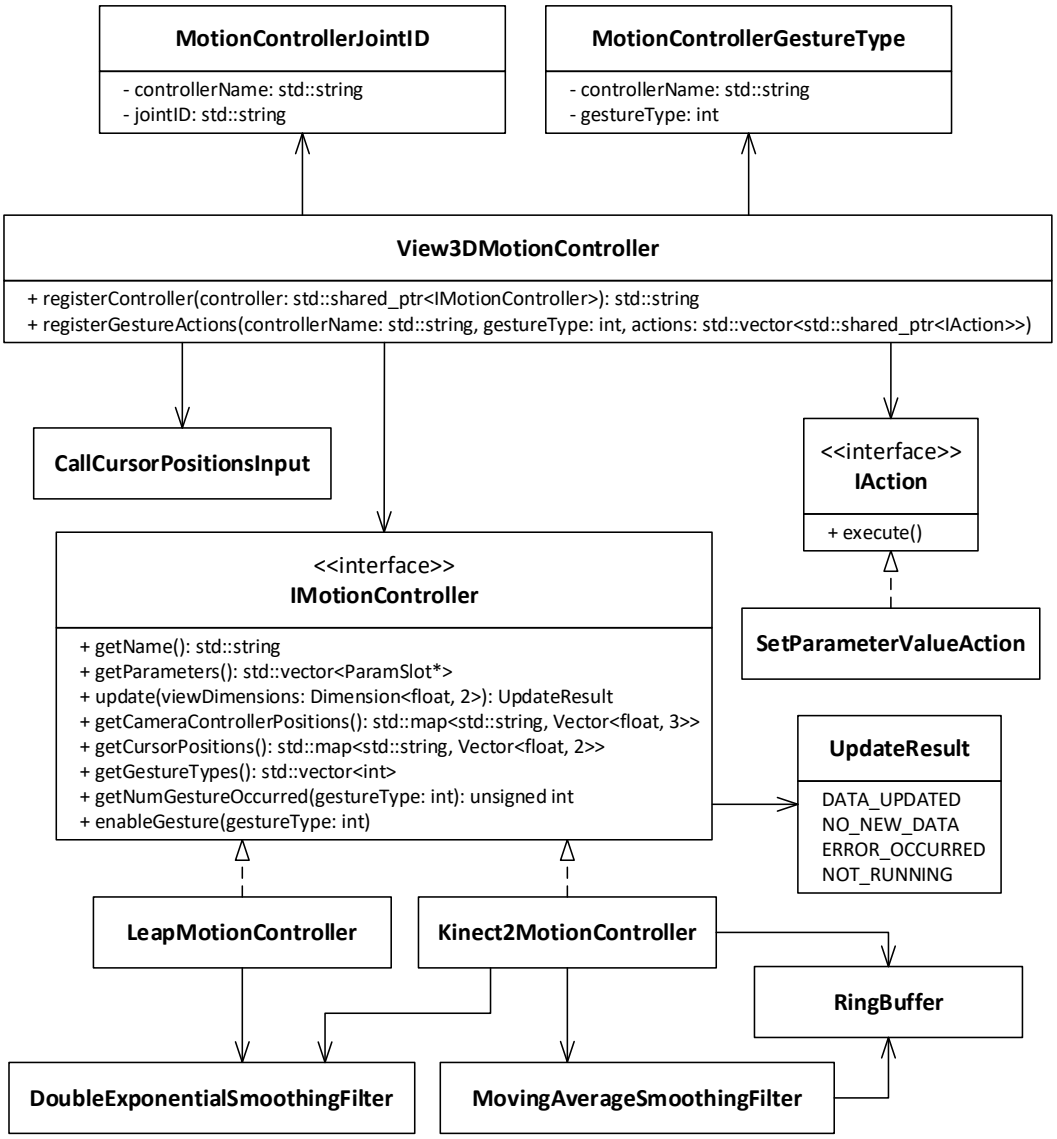

Abbildung 5.2: UML-Klassendiagramm des Moduls „View3DMotionController“

merasteuerung die aktuelle räumliche Position zurückgegeben werden. Als Einheit muss Millimeter verwendet werden, sodass in den Implementierungen ggf. Umrechnungen durchgeführt werden müssen. Eine Handhaltung zur Kamerasteuerung wird durch einen String identifiziert. Dieser String muss lediglich innerhalb dieser Funktion eindeutig sein. Der Aufrufer kann so später feststellen, ob der Nutzer die Handhaltung mit der linken bzw. rechten Hand weiterhin ausgeführt. Die Funktion `getCursorPositions` liefert für jede erkannte Handhaltung zur Cursor-Steuerung eine Position zurück. Hierbei handelt es sich jedoch nicht um die räumliche Position, sondern bereits um die zweidimensionale Position des Cursors (in Pixel). Für die Umrechnung sind die Dimensionen der View erforderlich, die der Funktion `update` übergeben werden. Da verschiedene Geräte u. U. unterschiedliche Umre-

chungsmethoden verwenden (siehe Abschnitt 4.2.2), kann die Umrechnung nicht vom Aufrufer durchgeführt werden. Wie auch zuvor wird eine String zur Identifizierung verwendet, die innerhalb der Funktion eindeutig sein muss. Falls ein Gerät keine Kamerasteuerung bzw. Cursor-Steuerung unterstützt, so entfällt die Implementierung der entsprechenden Funktion.

Optional können noch weitere Gesten implementiert werden. Für jeden Gestentyp muss eine Integer ID definiert werden, die über eine öffentliche Enumeration in der Implementierung bekannt gemacht werden sollte. Wird eine definierte ID der Funktion `getNumGestureOccurred` übergeben, so sollte zurückgegeben werden, wie häufig die jeweilige Geste erkannt worden ist. Genauer gesagt bedeutet dies, wie viele Gesten dieser Art im aktuellen Frame abgeschlossen worden sind. Damit dies jedoch möglich ist, muss die Erkennung der Geste zunächst aktiviert werden, was über die Funktion `enableGesture` erfolgt. Bei einigen Geräten wie z. B. der Leap Motion müssen Gesten, die vom SDK erkannt werden sollen, explizit aktiviert werden, da die Erkennung häufig mit einem hohen Aufwand verbunden ist. Diese Entscheidung wurde daher auch auf die Schnittstelle übertragen. Die Funktion `getGestureTypes` liefert die IDs von allen unterstützten Gesten zurück.

Weiterhin können Geräte-spezifische MegaMol-Parameter definiert werden, die von der Funktion `getParameters` als `ParamSlot`-Objekte zurückgeliefert werden müssen. Parameter können beispielsweise das Gerät ein- bzw. ausschalten oder die Umrechnung auf die Cursor-Position modifizieren. Der Aufrufer ist für die Registrierung der Parameter in MegaMol verantwortlich. Die sonst übliche Verwendung von Callback-Funktionen, die bei einer Parameter-Änderung aufgerufen werden, ist nicht möglich, da entsprechende Funktionen immer in der Hauptklasse eines Moduls definiert sein müssen. Stattdessen muss direkt mit den Werten gearbeitet werden, die in den `ParamSlot`-Objekten gespeichert sind (z. B. als `FloatParam`-Objekt).

Die Verwaltung der Geräte bzw. Implementierungen von `IMotionController` erfolgt in der Hauptklasse `View3DMotionController`. Mittels der Funktion `registerController` wird ein neues Gerät zur Bewegungssteuerung registriert, sodass es zur Kamerasteuerung und Cursor-Steuerung zum Einsatz kommt (sofern entsprechende Positionsdaten zurückgeliefert werden). Hierbei erfolgt auch die Registrierung der Geräteparameter in MegaMol.

Nach der Registrierung können mittels der Funktion `registerGestureActions` weitere Gesten eines Geräts aktiviert und mit Aktionen verknüpft werden. Aktionen können durch eine Implementierung der Schnittstelle `IAction` realisiert werden. Aktuell ist es möglich, den Wert eines beliebigen MegaMol-Parameters zu verändern (Klasse `SetParameterValueAction`). Der Parameter `gestureType` bezieht sich auf die ID des Gestentyps. Diese Funktion wird aktuell genutzt, um die Kreisgesten der

Leap Motion-Implementierung zu aktivieren und mit den entsprechenden Button-Parametern zu belegen (siehe Abschnitt 4.3).

Wird die Render-Funktion der Hauptklasse `View3DMotionController` aufgerufen, so werden zunächst die aktuellen Daten aller registrierten Geräte abgerufen. Handhaltungen werden durch `MotionControllerJointID`-Objekte geräteübergreifend identifiziert. Diese setzen sich aus dem Gerätenamen und der zuvor erwähnten String ID zusammen. Weitere Gesten werden durch `MotionControllerGestureType`-Objekte global identifiziert, die aus Gerätenamen und Gestentyp bestehen. Die Namen aller Geräte, die keine neuen Daten bereitgestellt haben, werden gespeichert, da sie im weiteren Verlauf benötigt werden.

Anschließend werden die Positionsdaten von Handhaltungen zur Kamerasteuerung verarbeitet. In Abschnitt 4.1 wurde bereits erwähnt, dass zu jeder Zeit immer nun eine Hand die Kamera steuern kann. Um die selbe Hand erneut identifizieren zu können, wird das zugehörige `MotionControllerJointID`-Objekt gespeichert. Da die Kamerabewegung relativ erfolgt, muss ebenso die Position gesichert werden. Sollte das gespeicherte `MotionControllerJointID`-Objekt unter den aktuellen Daten nicht mehr vorhanden sein, kann eine andere Hand die Kamerasteuerung übernommen. In diesem Fall wird die erste Handhaltung, die in der übergebenen Map `cameraControllerPositions` enthalten ist, verwendet. Die Hand, welche aktuell die Kamera steuert, darf jedoch keinesfalls verändert werden, falls das zugehörige Gerät aktiv ist, aber keine neuen Daten bereitgestellt hat. Dies wird bereits zu Beginn überprüft. Die eigentliche Manipulation der Kamera erfolgt über die Klasse `RelativeCursor3D`, die aus der Hilfsbibliothek VISlib stammt. Es wird der „ObjectControlMode“ verwendet, der die Kamera so bewegt, das der Benutzer annimmt, er würde die Bounding Box bewegen. Wie bereits erwähnt wurde, wird durch eine Handbewegung in X-Richtung die Kamera entgegen dem Uhrzeigersinn um die Y-Achse gedreht. Eine Bewegung in Y-Richtung rotiert die Kamera entgegen dem Uhrzeigersinn um die X-Achse. In Z-Richtung erfolgt eine Kamerabewegung entgegen der Z-Achse.

Im Anschluss daran erfolgt die Verarbeitung der Cursor-Positionen. Generell müssen Cursor-Positionen gespeichert haben, damit sie nicht verloren gehen, falls das zugehörige Gerät keine neuen Daten bereitgestellt hat. Positionsdaten von Geräten, die nicht mehr aktiv sind, werden hingegen gelöscht. Über den Call „CallCursorPositionInput“ werden die Positionen an die Funktion `updateCursorPositions` des Moduls „XMLUIRenderer“ (siehe Abschnitt 5.3) übergeben. Cursor-Positionen werden hier nicht mehr durch `MotionControllerJointID`-Objekte identifiziert. Stattdessen werden lediglich Strings verwendet. Diese Entscheidung wurde getroffen, damit die Benutzeroberfläche auch Cursor-Positionen unterstützt, die nicht vom

„View3DMotionController“-Modul bereitgestellt werden. Anders ausgedrückt soll die Benutzeroberfläche nicht an Geräte zur Bewegungssteuerung gebunden werden. `MotionControllerJointID`-Objekte werden zu Strings konvertiert, indem Gerätename und ID der entsprechenden Handhaltung konkateniert werden.

Abschließend werden für jede aufgetretene Geste die Aktionen ausgeführt, die zuvor über die Funktion `registerGestureActions` festgelegt worden sind.

5.2.1 Leap Motion-Implementierung

Die Unterstützung für die Leap Motion wird durch die Klasse `LeapMotionController` bereitgestellt. Zunächst wird in der Funktion `update` ein neues Frame über das Leap Motion SDK abgerufen. Da die API gleiche Frames mehrfach zurückliefert, wird der Zeitstempel des letzten Frame gespeichert.

Zur Erkennung der Faust wird „grabStrength“ verwendet, ein Wert, der für eine Hand vom SDK bereitgestellt wird. Er beträgt 1, falls die Hand eine Faust macht. Dieser Wert gilt ebenso, falls die Faust locker gehalten wird oder die Hand bereits etwas geöffnet ist. Wird die Hand weiter geöffnet, sinkt der Wert sehr schnell auf 0. Eine Alternative zur Erkennung einer Faust stellt der „grabAngle“-Wert dar, der einen größeren Wertebereich von 0 bis π besitzt. Bei einer leicht geöffneten Hand liegt dieser bei etwa 2,9 während „grabStrength“ noch 1 beträgt. Da eine leicht geöffnete Hand ausgeschlossen werden soll, wäre somit „grabAngle“ besser geeignet. Jedoch hat sich gezeigt, das „grabAngle“ häufiger falsche Werte besitzt. Bewegt man seine Faust beispielsweise mehrere Male zum Körper hin so kommt es sporadisch dazu, das „grabAngle“ bis auf 2,0 fällt während „grabStrength“ noch 1 beträgt (oder minimal darunter liegt). Streckt man seinen Zeigefinger aus, liefert „grabAngle“ einen Wert um 2,7, wohingegen „grabStrength“ 0 beträgt. Informationen zur Implementierung von „grabStrength“ und „grabAngle“ wurden vom Hersteller nicht veröffentlicht. Aus den genannten Gründen wurden daher schließlich entschieden, „grabStrength“ zur Erkennung der Faust einzusetzen.

Zur Erkennung der Hand mit ausgestrecktem Zeigefinger wird bestimmt, welche Finger aktuell ausgestreckt sind. Es dürfen lediglich Zeigefinger oder Zeigefinger und Daumen ausgestreckt sein. Das SDK stellt für jeden Finger eine Funktion zur Verfügung, die hierfür einen Boolean zurückliefert. Da viele Nutzer in der ausgestreckten Zeigefingerhaltung auch den Daumen nach außen bewegen, wurde die Entscheidung getroffen, diesen auch zu akzeptieren. Eine andere Möglichkeit, einen ausgestreckten Finger zu erkennen, besteht darin, den Winkel zwischen Handfläche und dem entsprechenden Finger zu bestimmen. Das SDK stellt Funktionen bereit, welche für die Glieder bzw. Knochen von Finger Richtungsvektoren zurückliefern. Es

wurde in Versuchen die Richtung des Fingerendglieds (Distal Phalanx) verwendet, da hier der Winkel immer am größten ist. Nach einigem Experimentieren wurden Werte gefunden, die recht zuverlässig bestimmen, ob ein Finger ausgestreckt ist. Versuche mit diesem Ansatz wurden auch zur Erkennung der Faust durchgeführt. Als die selben Versuche jedoch mit anderen Personen wiederholt worden sind (mit abweichenden Handgrößen), ist die Zuverlässigkeit der Erkennung deutlich gesunken. Insgesamt hat sich der erste Ansatz als zuverlässiger herausgestellt. Ein andere Methode zur Erkennung definiert Distanzen zwischen Handfläche und Fingerspitze. Es wurde jedoch angenommen, dass dieser Ansatz noch stärker von der Anatomie der Hand abhängt, sodass er nicht weiter verfolgt wurde.

Nachdem eine Faust bzw. Hand mit ausgestrecktem Zeigefinger erkannt wurde, wird die Position der jeweiligen Handhaltung über die Schnittstelle bereitgestellt. Für die Faust wird die Position der Handfläche verwendet, die Position des Zeigefingers ist durch dessen Fingerspitze gegeben, wobei hier noch abschließend die Umrechnung zur Cursor-Position erfolgen muss. Wie bereits in Abschnitt 2.4 erläutert wurde, zeichnen sich Positionsdaten durch ein Rauschen aus, das mittels entsprechender Filter reduziert werden kann. Das Leap Motion SDK wendet einen solchen Filter bereits automatisch an. Die Intensität des Filters ist jedoch bewusst sehr gering gehalten, damit die nicht wahrnehmbare Latenz möglichst beibehalten wird. Zuerst wurde das Verhalten der Kamerabewegung ausschließlich mit diesem fest vorgegebenen Filter getestet. In diesem Fall war ein kontinuierliches Zittern wahrzunehmen, das deutlich stärker bei einer hohen Vergrößerung aufgetreten ist. Für die Position von Handfläche (und auch Fingerspitze eines beliebigen Fingers) stellt das SDK einen zusätzlichen Stabilisierungsfilter zur Verfügung, der daraufhin getestet wurde. Ein Zittern war nun auch bei einer hohen Vergrößerung (nahezu) nicht mehr wahrzunehmen, jedoch kam es zu einem Springen der Kamera, falls man die Hand entlang der Z-Achse bewegt hat. Dieses Verhalten wurde als nicht akzeptabel eingestuft. Als nächstes wurde der Double Exponential Smoothing Filter (siehe Abschnitt 2.4) getestet. Zu Beginn wurden die voreingestellten Filterparameter verwendet, die für die Kinect vorgesehen sind. Bezüglich der Latenz waren nahezu keine Unterschiede zum ersten Versuch wahrzunehmen. Ein Zittern war jedoch ebenso nur noch minimal bei einer sehr hohen Vergrößerung zu erkennen. Durch eine Veränderung der Parametrisierung konnte insgesamt kein besseres Ergebnis erzielt werden.

Bei der Suche nach einem passenden Filter für die Positionsdaten der Zeigerfingerspitze wurde analog vorgegangen. Zunächst wurde der implizit angewendete Filter getestet. Es war wie erwartet keine Verzögerung des Cursors bemerkbar, jedoch ein kontinuierliches, deutlich wahrnehmbares Zittern zu beobachten. Daraufhin wurde der zusätzliche Filter des SDK eingesetzt. Insbesondere bei Bewegungen in X-Richtung war das Verhalten nahezu verändert. Im Ruhezustand war jedoch kein Zittern des

Cursors wahrzunehmen. Bei der Anwendung des Double Exponential Smoothing Filters mit der Standardparametrisierung war ein leichtes Springen des Cursors beobachten. In der Ruheposition konnte ebenso ein Bewegung des Cursors festgestellt werden. Sukzessive wurde der Radius für die Reduktion des Zitterns erhöht. Nach einigen Iterationen wurde schließlich ein Wert gefunden, der das Springen nahezu vollständig eliminiert und gleichzeitig den Cursor nur minimal verzögert.

Die Position der Zeigefingerspitze muss in eine zweidimensionale Cursor-Position umgerechnet werden. Es wurden zwei Methoden implementiert, die bereits in Abschnitt 4.2.2 beschrieben worden sind.

Die Erkennung der Kreisgesten (siehe Abschnitt 4.3) erfolgt mittels der Gestures API des Leap Motion SDK. Sobald die API festgestellt hat, das ein Nutzer einen Kreis mit einem Finger zeichnet, wird der gerade ausgeführten Kreisgeste eine ID vergeben. Wurden nun zwei vollständige Kreise gezeichnet, so wird die Geste über die Schnittstelle zurückgeliefert (d. h. der Zähler erhöht) und die ID des SDK als abgeschlossene Kreisgeste in der Implementierung gespeichert. Beim nächsten Aufruf der `update`-Funktion wird überprüft, ob die ID einer aktuell ausgeführten Kreisgeste bereits abgeschlossen ist. Falls dies der Fall ist (d. h. die Kreisgeste besteht aus mehr als zwei Kreisen), wird die Geste übersprungen und somit verhindert, das sie ein weiteres Mal zurückgeliefert wird. Falls die Fingerbewegung im Kreis schließlich tatsächlich beendet ist, wird die ID wieder gelöscht. Über den Winkel zwischen dem Richtungsvektor des Fingers und dem Normalenvektor des Kreises, der von diesem Finger gezeichnet wird, kann ermittelt werden, in welcher Richtung die Kreisgeste ausgeführt wird. Ist der Winkel größer als $\pi/2$, so wird der Kreis entgegen dem Uhrzeigersinn gezeichnet. Andernfalls erfolgt die Ausführung im Uhrzeigersinn.

Es wurden Button-Parameter definiert, über welche die Leap Motion gestartet bzw. gestoppt werden kann. Hierbei wird der Service des SDK gestartet bzw. angehalten. Ist der Service nicht aktiv, erhalten auch andere Anwendungen keine Daten mehr vom Gerät. Da in dieser Klasse keine Callback-Funktionen definiert werden konnten, wird zu Beginn der `update`-Funktion geprüft, ob die entsprechenden Parameter verändert worden sind (Dirty-Flag gesetzt).

5.2.2 Kinect für Xbox One-Implementierung

Durch die Klasse `Kinect2MotionController` wird die Steuerung mittels der Kinect realisiert. Eine Instanz des `BodyFrameReader` liefert lediglich einmal das aktuelle Frame zurück. Liegt kein neues Frame vor, gibt die entsprechende Operation eine Fehlermeldung zurück. Ein Speicherung der letzten Frame ID ist somit nicht erforderlich.

Die Erkennung der flachen Hand als auch Faust erfolgt über den HandState des SDK, eine Enumeration, welche den Zustand „Open“, „Closed“, „Unknown“, „Lasso“, „NotTracked“ oder „Unknown“ besitzen kann. Eine Hand befindet sich im „Lasso“-Zustand, falls sie geschlossen ist, jedoch Zeigefinger und Mittelfinger ausgestreckt sind und nach oben zeigen [Mic16a]. Bei einer großen Hand, die deutlich geschlossen ist, wird der Zustand sporadisch auch erkannt, falls lediglich einer der beiden Finger ausgestreckt sind [Mic16a]. Für unsere beiden Handhaltungen wird dieser Zustand nicht benötigt.

Versuche haben gezeigt, dass die Kinect eine flache Hand deutlich besser erkennt, wenn auch nur ein Teil der offenen bzw. geschlossenen Handfläche zum Gerät gerichtet ist. Macht man beispielsweise eine genau waagrechte flache Hand, so bleibt der Zustand größtenteils durchgehend in „Unknown“. Bereits bei minimalen Bewegungen wird immer wieder für wenige Frames der Zustand „Open“ zurückgegeben. Bei schnelleren Bewegungen kommt es sporadisch auch vor, dass kurzzeitig „Closed“ zurückgeliefert wird. Richtet man die Hand langsam auf, so wird bereits nach kurzer Zeit durchgehend der Zustand „Open“ zurückgeliefert. Falls man den Handrücken genau parallel zur Kinect ausgerichtet, so konnten ständige Wechsel zwischen meist kurzen „Open“- und längeren „Unknown“-Phasen beobachtet werden. Bei beliebigen Bewegungen in dieser Haltung hat die Länge der „Unknown“-Phasen etwas zugenommen. Sporadisch haben auch kurzzeitige Wechsel zu „Lasso“ oder „Closed“ stattgefunden.

Bei der Faust hat sich in der waagrechten Haltung eine signifikant bessere Erkennung gezeigt. Ohne Bewegungen wurden häufig 30 Frames in Folge mit dem Zustand „Closed“ zurückgegeben. Umso schneller die Hand bewegt wird, umso häufiger sind „Unknown“-Phasen zu beobachten gewesen. Die Länge der Phasen hat dabei ebenso etwas zugenommen. Richtet man die geschlossene Handfläche genau parallel zur Kinect 2 aus, so wird nahezu durchgehend der korrekte Handzustand zurückgeliefert. Nur bei sehr schnellen Bewegungen wird selten für wenige Frames „Unknown“ ausgegeben. Sehr selten wird „Open“ bei einer Faust zurückgeliefert. In den Versuchen ist dies dann lediglich immer für ein Frame der Fall gewesen. Falls die Hand überhaupt nicht auf dem Tiefenbild zu erkennen war (z. B. in der Hosentasche), so wurde die Enumeration auf „NotTracked“ gesetzt. Auch konnte dieser Zustand häufig festgestellt werden, wenn der Arm nah am Körper hängen gelassen wurde. Alle genannten Beobachtungen gelten im Wesentlichen für die linke als auch rechte Hand.

In der ersten Version der Implementierung wurde ein HandState direkt auf die zugehörige Operation abgebildet. Ein „Unknown“ hat die Steuerung demnach sofort beendet. Das Ergebnis war nicht zufriedenstellend: Bei einer Faustbewegung kam es zu häufig zu kurzen Aussetzern der Kamerasteuerung. Bei einer Bewegung der flachen Hand war ein Flackern des Cursors wahrzunehmen. Wurde die Hand zusätzlich noch

genau waagrecht gehalten, so ist der Cursor nur selten erschienen. Eine Recherche hat ergeben, das der HandState lediglich auf der Bewertung eines einzelnen Frames beruht [Sir14a]. Um zuverlässig den aktuellen Zustand der Hand ermitteln zu können, müssen jedoch die Zustände mehrere Frames zusammen verwendet werden [Sir14a], d. h. ein Filter angewendet werden.

In den MSDN-Foren wird von einem Microsoft-Mitarbeiter vorgeschlagen [Sir14a], einen eigenen Handzustand einzuführen. Nach einer definierten Zahl von aufeinanderfolgenden, identischen Handzuständen, wird der eigene Zustand entsprechend angepasst. Dieser grundsätzliche Ansatz wurde weiter verfolgt. Der eigene Zustand wurde durch eine Enumeration repräsentiert, welche die Zustände „Open“, „Closed“ und „Unknown“ annehmen kann. Zur Speicherung der Datenhistorie wurde ein Ringpuffer implementiert. Aufgrund der Realisierung durch ein Klassen-Template können beliebige Daten verwaltet werden können. Auf Grundlage der zuvor beschriebenen Versuche musste nun für jeden der drei Zustände festgelegt werden, unter welchen Bedingungen der Wechsel erfolgt. Für einen Wechsel zum eigenen Zustand „Closed“ wurden 13 aufeinanderfolgende „Closed“-Zustände des SDK definiert. Bei einem etwas kleineren Wert kommt es bei schnellen Bewegungen der flachen Hand (insbesondere wenn die Handfläche nicht vollständig zur Kinect gerichtet ist) sporadisch dazu, dass die Steuerung der Kamera ausgelöst wird. Grundsätzlich stellen jedoch höhere Werte kein Problem dar, da wie zuvor erwähnt die Fausthaltung sehr zuverlässig von der Kinect erkannt wird. Für den Nutzer ist nahezu keine Verzögerung festzustellen, bis die Steuerung aktiviert ist. Es hat sich weiterhin gezeigt, dass bei einem nicht korrekten „Closed“-Zustand häufig die Tracking Confidence der Hand auf „Low“ gesetzt ist. Die Tracking Confidence des SDK gibt an, wie wahrscheinlich es ist, dass die Handdaten korrekt sind. Das SDK unterscheidet zwischen „High“ und „Low“. Auf Grundlage dieser Beobachtung konnte noch eine alternative Bedingung definiert werden, um in den eigenen Zustand „Closed“ zu wechseln: Lediglich nach sechs „Closed“-Zuständen des SDK, die durchgehend eine Tracking Confidence von „High“ aufweisen, wird gewechselt. Für eine Änderung in den eigenen Zustand „Open“ wurden lediglich zwei „Open“-Zustände des SDK definiert. Dies ist auch notwendig, da bei einer waagrechten flachen Hand, die ganz ruhig gehalten wird, nur selten mehr als zwei „Open“-Zustände in Folge zurückgeliefert werden. Der Wert konnte nicht auf eins reduziert werden, da andernfalls bei einer schnellen Faustbewegung in seltenen Fällen die Kamerasteuerung ausgefallen und der Cursor erschienen ist. Hierbei war die Tracking Confidence meist auf „High“ gesetzt, sodass diese Information nicht verwendet werden konnte. Das Wechseln zum eigenen Zustand „Unknown“ erfolgt, sobald einmalig der Handzustand „NotTracked“ vom SDK zurückgegeben wird und hierbei die Tracking Confidence mit „High“ festgelegt ist. In den Versuchen war letzteres im Zustand „NotTracked“ immer der Fall. Da in der Dokumentation jedoch

nichts über eine solche Korrelation gefunden werden konnte, wurde die Tracking Confidence als zusätzliche Bedingung beibehalten. Das Wechseln nach einer Zahl von „Unknown“-Zuständen durchzuführen war keine Alternative, da es so (insbesondere) bei einer waagrechten flachen Hand häufig zu einem Ausfall gekommen ist. Obwohl das SDK für hängende Arme meist den Zustand „NotTracked“ zurückliefert, wurde dennoch die Bedingung formuliert, dass sich eine Hand immer mindestens 10 cm über der Hüfte befinden muss. Ist dies nicht mehr gegeben, wird direkt in den eigenen Zustand „Unknown“ gewechselt. Die Steuerung wird folglich sofort beendet. Die Grenze wurde nicht höher festgelegt, damit der Nutzer in seinen Bewegungen nicht zu stark eingeschränkt wird.

Für die Position einer erkannten Faust wird die Handposition verwendet, die vom SDK bereitgestellt wird. Daneben kann auch die Position der Handspitze zurückgegeben werden. Es hat sich jedoch gezeigt, das diese Position bei einer Fausthaltung nur sehr unzuverlässig bestimmt werden kann, sodass ein stärkeres Zittern wahrzunehmen war. Generell gilt für die Kinect (jedoch nur für die zweite Generation), das Daten ohne einen Filter bereitgestellt werden. Für die Reduktion des Rauschens ist folglich vollständig der Entwickler verantwortlich.

Wie auch bei der Leap Motion wurde der Double Exponential Smoothing Filter getestet und Versuche mit den voreingestellten Parametern begonnen. Hierbei war eine Verzögerung kaum wahrnehmbar, jedoch ein leichtes Zittern bei einer hohen Vergrößerung zu erkennen. Dementsprechend wurde der Radius zur Reduktion des Zitterns etwas erhöht. Die Latenz war nun nach wie vor immer noch akzeptabel, das Zittern aber deutlich geringer. Eine weitere Erhöhung des entsprechenden Parameters hat das Zittern nochmals leicht reduziert, allerdings war nun die Verzögerung deutlich wahrnehmbar. Letztendlich wurde daher die zuvor verwendete Konfiguration beibehalten.

Auch für die Position der flachen Hand wird die Handposition des SDK genutzt. Die Position der Handspitze war bei einer offenen Hand ebenso deutlich instabiler. Erneut wurde mit dem Double Exponential Smoothing Filter begonnen. Mit der Standardparametrisierung konnte in den unteren Ecken ein starkes Springen des Cursors beobachtet werden, wenn die Hand genau waagrecht ausgerichtet war. Insbesondere ist dieses Springen bei einer Steuerung durch die linke Hand aufgetreten. Eine Verzögerung des Cursors war nicht wahrzunehmen. Daraufhin wurde der Radius für die Reduktion des Zitterns sukzessive erhöht. Das Springen des Cursors ist dabei jedoch nur langsam zurückgegangen. Als das Verhalten des Cursors weitestgehend zufriedenstellend war, konnte die Verzögerung des Cursors deutlich wahrgenommen werden, was als nicht hinnehmbar eingestuft worden ist. Versuche mit anderen Parametern des Filters haben ebenfalls kein akzeptables Ergebnis hervorgebracht.

Daraufhin wurde ein Simple Moving Average Filter (siehe Abschnitt 2.4) getestet. Bei einer Fenstergröße von 10 war die Latenz noch akzeptabel und das Springen des Cursors in den Ecken seltener und deutlich weniger ausgeprägt. Diese Konfiguration wurde schließlich beibehalten.

Zur Umrechnung auf die Cursor-Position stehen die selben zwei Verfahren wie für die Leap Motion zur Verfügung. Diese wurden bereits in Abschnitt 4.2.2 erläutert. Die Interaction Box wird nicht relativ zum Gerät, sondern ausgehend von der Schulter der jeweiligen Hand definiert. Es wäre nun denkbar, die Position der Schulter ständig zu aktualisieren, damit diese immer in der Mitte des Bereichs liegt. Versuche haben jedoch gezeigt, das häufig der Arm bzw. die Hand die Schulter verdeckt, insbesondere wenn der Arm gerade nach vorne gestreckt wird. Die Kinect 2 muss die Position nun schätzen, was meist nur sehr unzuverlässig klappt. Die Folge ist ein starkes Zittern des Cursor, was als nicht akzeptabel bewertet wurde. Daher wird die Schulterposition lediglich einmalig erhoben, wenn die flache Hand erkannt wird bzw. der Cursor erscheint. Falls ein Nutzer einen Cursor steuert und dabei seine aktuelle Position deutlich verändert, so sollte er den entsprechenden Arm kurz absenken, damit die Schulterposition neu erhoben wird.

Wie auch bei der Leap Motion wurden Button-Parameter definiert, mit denen das Gerät gestartet bzw. angehalten werden kann. Ist es angehalten, so gilt dies ausschließlich für MegaMol. Andere Anwendungen können nach wie vor Daten vom Gerät erhalten.

5.3 Modul „XMLUIRenderer“

Abbildung 5.3 veranschaulicht den Entwurf des Renderer-Moduls „XMLUIRenderer“, das die Benutzeroberfläche bereitstellt. Das Rendering erfolgt mittels der Zeichenbibliothek NanoVG, auf die bereits in Abschnitt 2.5 eingegangen worden ist. In der Hauptklasse `XMLUIRenderer` wird im Konstruktor der NanoVG-Zeichenkontext erzeugt. Es wird hierbei die OpenGL 3.2-Implementierung gewählt und Anti-Aliasing sowie Stencil Strokes (besseres Rendering von überlappenden Linien) aktiviert. Die Funktion `registerCursor` registriert einen Cursor in der Benutzeroberfläche, der erscheint, sobald ein Benutzer die entsprechende Handhaltung ausführt. Die Visualisierung und Funktionsweise des Cursors wurde in Abschnitt 4.2.1 beschrieben. Das Attribut `positionSourceID` der Klasse `Cursor` identifiziert den Ursprung einer Cursor-Position und gibt gleichzeitig an, ob der Cursor aktuell sichtbar ist. In diesem Fall ist der String nicht leer. Falls die ID vom Modul `View3DMotionController` stammt, so besteht diese aus dem Namen des Geräts und der ID der Handhaltung, welche den Cursor steuert (siehe Abschnitt 5.2). Das Attribut `button` zeigt auf die

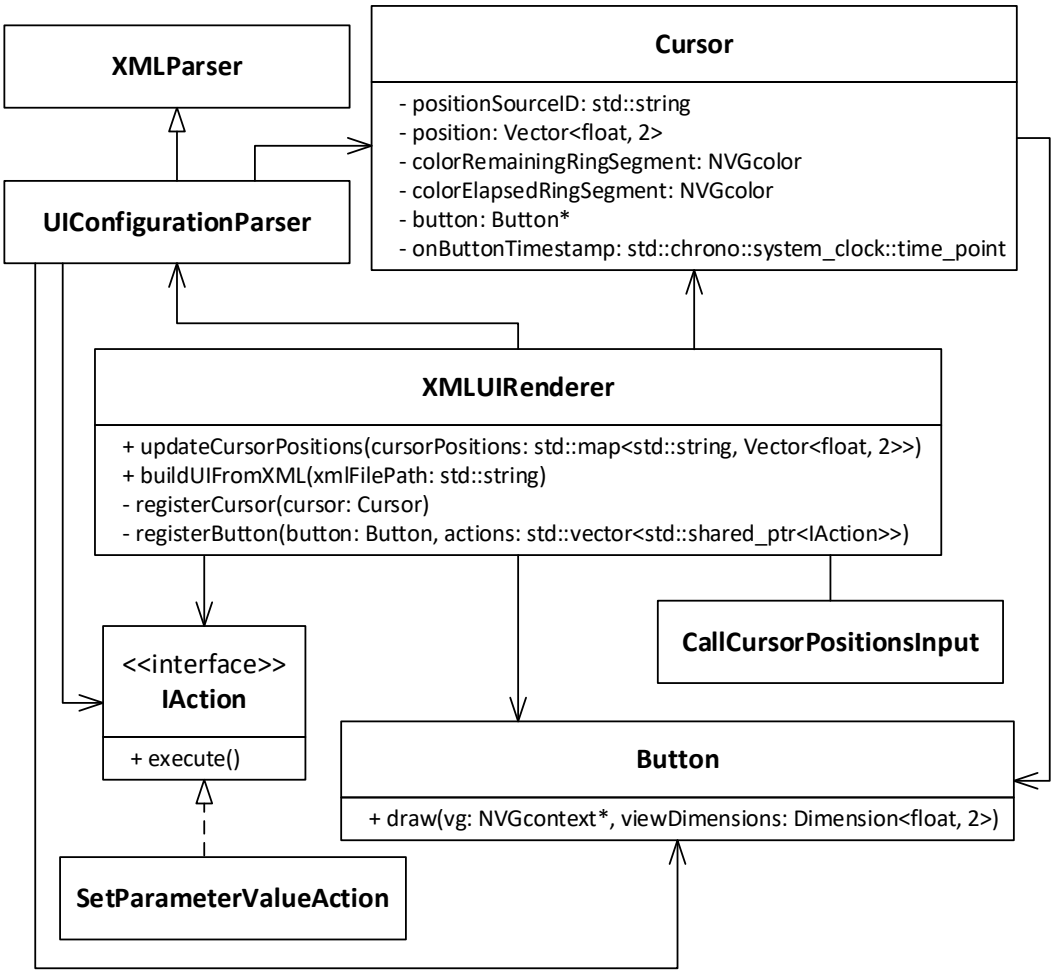

Abbildung 5.3: UML-Klassendiagramm des Moduls „XMLUIRenderer“

Schaltfläche, über die sich ein Cursor gerade bewegt. Auf diese Weise kann einfach der Zustand einer Schaltfläche verändert werden, wenn der Cursor beispielsweise einen Klick ausführt. Der Zeitstempel `onButtonTimestamp` gibt an, seit wann sich der Cursor auf der Schaltfläche befindet bzw. wann der letzte Klick erfolgt ist. Die übrigen Attribute sollten selbsterklärend sein. Die Reihenfolge, in der Cursor erscheinen, ist gegeben durch die Reihenfolge der Registrierungen.

Eine Schaltfläche wird über die Funktion `registerButton` erstellt. Hierbei müssen direkt die Aktionen übergeben werden, die bei einem Klick ausgeführt werden sollen. Es sollte erwähnt werden, dass es sich bei diesen Aktionen um Implementierungen der Schnittstelle `IAction` handelt. Diese können auch im Modul `View3DMotionController` mit Gesten verknüpft werden (siehe Abschnitt 5.2). Aktuell ist es möglich, einem Parameter von MegaMol einen Wert zuzuweisen. In der Klasse `Button` sind eine Reihe von Funktionen definiert, welche die Berechnung von Größe und Positionierung umzusetzen. Um ständige Neuberechnungen zu vermei-

den, werden diese Daten einmalig bestimmt und mit den Dimensionen der View gespeichert. Lediglich bei einer Änderung der Dimensionen wird eine erneute Berechnung notwendiger Daten durchgeführt. In der Funktion `draw` wird die Schaltfläche schließlich mittels NanoVG gezeichnet.

Werden Cursor-Positionen über den Call „CallCursorPositionsInput“ an die Benutzeroberfläche übergeben, so wird dabei die Funktion `updateCursorPositions` aufgerufen. Der Schlüssel der übergebenen Map stellt die zuvor angesprochene ID dar, die den Ursprung der Position identifiziert. Cursor-Positionen mit IDs, die aktuell in registrierten Cursors gespeichert sind (d. h. Cursor sind aktiv), werden in diesen aktualisiert. Sollte für einen aktiven Cursor keine neue Position bereitgestellt werden, so wird die in ihm gespeicherte ID gelöscht (d. h. ein leerer String zugewiesen), sodass der Cursor wieder frei ist und für eine neue ID zur Verfügung steht. Die übrigen Cursor-Positionen (mit nicht bekannten IDs) werden daraufhin den freien Cursor zugewiesen. Falls nicht genügend freie Cursor zur Verfügung stehen, so werden entsprechend Cursor-Positionen verworfen.

Das Laden einer Konfigurationsdatei erfolgt über die Funktion `buildUIFromXML`. Wie bereits erwähnt wurde, erfolgt der Aufruf durch einen String-Parameter im Front-End. Übergebene Dateipfade können relativ oder absolut sein. Ein relativer Pfad wird mit den Ressource-Verzeichnissen, die in der MegaMol-Konfigurationsdatei definiert sein, aufgelöst. Das erste Verzeichnis, in dem die Datei gefunden wurde, wird verwendet. Für das Parsen der Konfigurationsdatei kommt die Klasse `UIConfigurationParser` zum Einsatz, die in Abschnitt 5.3.1 separat behandelt wird. Nachdem dieser Prozess abgeschlossen ist, werden die Meldungen des Parsers auf der Konsole ausgeben. Hierbei kann es sich um Fehler handeln (z. B. XML-Dokument nicht wohlgeformt oder erforderliche Elemente nicht vorhanden), aber auch ignorierte Elemente bzw. Attribute werden ausgegeben. Letzteres stellt keinen Fehlerfall dar. Ist die Datei gültig, so wird zunächst die aktuelle Benutzeroberfläche gelöscht. Daraufhin werden die Schriftarten und Bilder geladen und schließlich die Schaltflächen und Cursor registriert. Auch wird der Cursor-Radius festgelegt und das Cursor-Clamping (Cursor darf Benutzeroberfläche nicht verlassen) aktiviert bzw. deaktiviert. Beide Parameter gelten für alle Cursor.

5.3.1 Klasse „UIConfigurationParser“

Die MegaMol-Konfigurationsdatei als auch MegaMol-Projektdateien werden aktuell mittels der Bibliothek Expat XML Parser[35] eingelesen. Aus Konsistenzgründen und

[35] http://expat.sourceforge.net

um die Einführung einer weiteren Bibliothek zu vermeiden, wurde die Entscheidung getroffen, ebenfalls diese Bibliothek zum Parsen von Konfigurationsdateien der Benutzeroberfläche zu verwenden. Die Core-Komponente von MegaMol stellt die Klasse `XMLParser` bereit, die das Einlesen mittels dieser Bibliothek realisiert. Ein eigener Parser wird erstellt, indem von dieser Klasse abgeleitet wird. Da die Core-Komponente jedoch als dynamische Programmbibliothek (DLL-Datei) realisiert ist, kann eine Vererbungshierarchie nur innerhalb des Projekts definiert werden. Dementsprechend musste die Klasse `XMLParser` einschließlich abhängiger Core-Klassen in das eigene Plug-in kopiert werden. Ebenso mussten auch die Header-Dateien und statischen Bibliotheken (LIB-Dateien) des Expat XML Reader übernommen werden. Abschließend wurde dann eine abgeleitete Klasse `UIConfigurationParser` erstellt.

Die Parser-Bibliothek Expat XML Parser verfügt über eine Push API [Coo99]. Ein XML-Dokument wird sequentiell gelesen und bei Ereignissen wie z. B. einem Start-Tag Funktionen aufgerufen (Callback-Prinzip), die der Entwickler festlegt [Coo99]. Der Vorteil dieses Ansatzes ist, das gelesene Daten sofort weiter verarbeitet oder auch übersprungen werden können [Coo99]. Nach Abschluss des Einleseprozesses muss keine Kopie des gesamten Dokuments im Speicher abgelegt werden [Coo99]. Der Speicherbedarf wird dadurch erheblich reduziert [Coo99]. Die Klasse `XMLParser` definiert bereits entsprechende Callback-Funktionen, die in der Kindklasse überschrieben werden können, sofern sie benötigt werden. In unserer Parser-Implementierung `UIConfigurationParser` werden Funktionen überschrieben, die bei dem (1) Wurzelelement, einem (2) Start-Tag, einem (3) End-Tag sowie nach (4) Abschluss des Einleseprozesses aufgerufen werden. Da der Expat XML Parser keine Validierung gegen ein XML Schema-Dokument unterstützt, muss ein Element in der eigenen Implementierung auf Korrektheit überprüft werden. Eine Prüfung auf Wohlgeformtheit hingegen findet statt. Falls der Parser erkennt, das eine Regel aus der XML-Spezifikation[36] verletzt ist, so wird der Parsing-Prozess nicht mehr fortgeführt. Ein nicht wohlgeformtes XML-Dokument besitzt z. B. nicht genau ein Wurzelelement oder enthält Tags, die nicht geschlossen sind. Auch wird eine weitere Verarbeitung verhindert, falls bereits das Wurzelelement nicht gültig ist, also z. B. nicht den korrekten Namen besitzt oder erforderliche Attribute fehlen. Falls es sonst Elemente oder Attribute gibt, die an der jeweiligen Position nicht erlaubt sind, so werden diese ignoriert. Ereignisse, die beim Lesen des Dokuments auftreten, sind zustandslos. Es besteht keine Beziehung zu den zuvor aufgetretenen Ereignissen. Dementsprechend können beim Aufruf der Callback-Funktionen keine Informationen übergeben werden, über welche die aktuelle Position in der Hierarchie des Dokuments bestimmt werden kann. Da entsprechende Positionsinformationen jedoch benötigt werden, musste in der eigenen Implementierung ein Zustand eingeführt werden.

[36] `http://www.w3.org/TR/xml` (Spezifikation von XML 1.0)

Falls ein erkannter Start-Tag an der aktuellen Position erlaubt ist, so wird in der Callback-Funktion überprüft, ob alle erforderlichen Attribute vorhanden sind und gültige Werte besitzen. Trifft dies zu, werden die entsprechenden Daten gespeichert. Bei einem Fehler hingegen ist die gesamte XML-Datei ungültig. Das Parsing wird jedoch fortgesetzt, um weitere Fehler ermitteln zu können, die dann später auf der Konsole ausgegeben werden. Weiterhin wird vermerkt, das ein Tag bereits verarbeitet wurde (nur falls der Tag nicht mehrfach vorkommen darf). Dadurch wird verhindert, das ein gleich benanntes Element erneut berücksichtigt wird, was ggf. zuvor gespeicherten Daten überschreiben würde. Auch kann so ermittelt werden, ob das Element überhaupt aufgetreten ist. Letzteres erfolgt in der Callback-Funktion, die für den End-Tag definiert ist. Ist ein erforderliches Element nicht vorhanden, so liegt ein Fehlerfall vor. In selbiger Callback-Funktion werden häufig aber auch die zuvor gespeicherten Daten zusammengefasst. Beispielsweise wird in der Funktion, die bei einem Button-End-Tag aufgerufen wird, das `Button`-Objekt erzeugt. Nach Abschluss des Parsing-Prozesses werden die definierten Referenzen auf Schriftarten und Bildern geprüft.

Die Möglichkeiten zur Konfiguration der Benutzeroberfläche wurden bereits in Abschnitt 4.2.3 beschrieben.

6 Evaluation und Vergleich

Nach Abschluss der Implementierung sollen die erreichten Ergebnisse evaluiert und verglichen werden. Einführend werden in Abschnitt 6.1 zunächst allgemeine Vor- und Nachteile der Geräte genannt, die sich aus der Spezifikation des jeweiligen Geräts ergeben oder in eigenen Versuchen aufgefallen sind. In Abschnitt 6.2 erfolgt anschließend eine persönliche Bewertung der Gesten im Hinblick auf die Erkennung durch die Geräte. Eigene Versuche wurden grundsätzlich bei Tageslicht in einem leicht abgedunkelten Raum durchgeführt. Eine zusätzliche Beleuchtung wurde nicht eingesetzt. Die Leap Motion wurde auf einem Schreibtisch platziert und im Sitzen genutzt. Die Kinect wurde in einer Höhe von 1,50 m aufgebaut. Es wurde ein Abstand von etwa 1,60 m zum Gerät eingenommen.

Eine objektive Beurteilung der erzielten Ergebnisse erfolgt abschließend durch die Benutzerstudie, die zum Ende der Arbeit durchgeführt worden ist. Hierbei wurden die Gesten auch unabhängig von den Geräten bewertet. Auf die Studie wird in Abschnitt 6.3 eingegangen.

6.1 Vor- und Nachteile der Geräte

Die Kinect wurde zur räumlichen Erfassung von Personen entwickelt. Dementsprechend besitzt sie einen signifikant größeren Erfassungsbereich als die Leap Motion, die zur Verfolgung von Händen und Fingern entwickelt worden ist. Wie bereits in Abschnitt 2.3 erwähnt wurde, können Personen, die sich in einem Abstand von 50 cm bis 4,5 m vor dem Gerät bewegen, erkannt werden. Dies stellt einen Vorteil dar, bedeutet gleichzeitig aber auch, dass u. U. Personen erkannt werden, die nicht mit der jeweiligen Anwendung interagieren möchten. Falls diese nun zufällig Körperbewegungen bzw. Gesten ausführen, die in der Anwendung definiert sind, so kann es zur unbeabsichtigten Ausführung von Aktionen kommen. Denkbar wäre in diesem Zusammenhang, das eine Person meint mit der Anwendung zu interagieren, dies aber tatsächlich durch eine andere Person unbewusst erfolgt. Erstere Person würde dann nach kurzer Zeit bemerken, das die Anwendung nicht wie gewünscht reagiert und von einer nicht nachvollziehbaren Steuerung ausgehen. Bei der Leap Motion hingegen wird die Hand explizit über das Gerät bewegt, damit sie erkannt wird.

Dementsprechend ist es wesentlich unwahrscheinlicher, das es zu entsprechenden Szenarien kommt. Ihr Erfassungsbereich stellt ungefähr eine invertierte Pyramide dar, die 80 cm in die Höhe geht (siehe Abschnitt 2.2). Experimente haben jedoch gezeigt, das in dieser Höhe zwar Hände und sogar Fingerbewegungen erkannt werden, dies jedoch keinesfalls zufriedenstellend funktioniert. Beispielsweise wird bei schnellen Hin- und Herbewegungen einer Faust in X-Richtung häufig ein Finger (meist Zeigefinger) kurzzeitig als ausgestreckt erkannt. Ebenso werden sporadisch Finger, die nicht mehr ausgestreckt sind, zunächst weiterhin als ausgestreckt erkannt. Das nach innen krümmen der Finger erfolgt verzögert oder häufig auch erst, wenn die Hand wieder etwas näher zum Gerät bewegt wird. In seltenen Fällen wurde die Hand kurzzeitig mit einer falscher Orientierung erkannt (meist um 180 Grad gedreht) oder ist komplett verloren gegangen. Falls man die Hand sehr schnell vollständig aus dem Erfassungsbereich des Geräts entfernt, so kommt es in seltenen Fällen dazu, das die Hand weiterhin bestehen bleibt. Erst nach wenigen Sekunden erkennt das Gerät, das die Hand nicht mehr vorhanden ist.

Da die Kinect vor einer Person aufgebaut wird, ist die Erkennung einer Hand, die sich genau über der anderen Hand befindet, ebenfalls möglich. Falls bei der Leap Motion hingegen eine Hand auch nur teilweise über die andere Hand positioniert wird, so verschwindet die obere Hand. Das Leap Motion SDK unterstützt unbegrenzt viele Hände. Es hat sich jedoch gezeigt, das bereits ab mehr als zwei Händen die Genauigkeit der Erkennung abnimmt. Dies wird auch vom Hersteller bestätigt [Lea16a]. Das Kinect SDK hingegen kann maximal 6 Körper verfolgen, jedoch nur von zwei Personen gleichzeitig die Zustände der Hände kontinuierlich ermitteln. Von der Leap Motion werden im Normalfall 115 Bilder pro Sekunde zurückgeliefert, die Kinect stellt lediglich 30 Bilder pro Sekunde zur Verfügung. Dementsprechend reagiert die Leap Motion spürbar schneller auf Bewegungen als die Kinect. Da die Kinect für die Spielekonsole Xbox entwickelt wurde, verfügt sie zusätzlich über eine 1080p-Farbkamera sowie ein Mikrofon. Die technischen Daten der Geräte wurden detailliert in den Abschnitten 2.2 und 2.3 behandelt.

Der Aufbau der Leap Motion ist im Vergleich zur Kinect deutlich einfacher. Dies liegt insbesondere an den geringen Abmessungen des Geräts (siehe Abschnitt 2.2), was es äußerst portabel macht. Üblicherweise wird die Leap Motion am PC verwendet. Das Gerät wird so auf dem Schreibtisch platziert, das der Nutzer mittig davor sitzt. Anschließend muss das Gerät lediglich über das mitgelieferte USB-Kabel mit dem PC verbunden werden. Es ist mindestens eine USB 2.0-Schnittstelle erforderlich. Für eine komfortable Nutzung kann der Unterarm (z. B. der Ellbogen) auf dem Tisch abgestützt werden. Die Kinect hingegen kann nicht direkt mit einem PC verbunden werden, es ist ein Adapter erforderlich. Das deutlich größere Gerät wird meist über dem Fernseher aufgebaut und danach etwas gekippt, damit Personen

vollständig verfolgt werden. Insgesamt kann somit die Einrichtung der Kinect als deutlich aufwändiger bewertet werden. In den Online Shops der Hersteller [Lea17; Mic17] wird die Leap Motion aktuell für 70 € verkauft, die Kinect kann für 100 € erworben werden. Der Adapter für den Anschluss der Kinect an den PC (auch Xbox One S) ist für 40 € verfügbar. Die angegebenen Preise wurden am 02.01.2017 erhoben.

Nutzt man die Kinect und Leap Motion im Stehen, so ist die Ermüdung des Arms bei der Kinect etwas höher. Wir gehen hierbei davon aus, dass beide Geräte in angewinkelten Armhaltungen genutzt werden, wobei der Arm bei der Kinect etwas weniger angewinkelt ist. Versuche haben gezeigt, dass eine ganz rechtwinklige Armhaltung bei der Kinect die Erkennung verschlechtert (Zittern nimmt zu). Die Aufbauhöhe der Leap Motion sollte ausgehend von der durchschnittlichen Ellbogenhöhe (hängender Arm) festgelegt werden, sofern eine feste Installation erfolgen soll. In der Benutzerstudie wird dies auch so umgesetzt (siehe Abschnitt 6.3.1). Es kann jedoch nie ausgeschlossen werden, das bei Personen, die von diesem Durchschnittswert stark abweichen, eine Nutzung nur erheblich erschwert möglich ist. Dies stellt einen grundsätzlichen Nachteil der Leap Motion dar. Geht man vom Regelfall aus, d. h. den Einsatz der Kinect im Stehen sowie der Leap Motion im Sitzen vor dem PC, so ist letzteres Gerät für eine längere Interaktion deutlich besser geeignet. Die Kinect verfügt an der Unterseite des Geräts über eine Befestigungsmöglichkeit. Diese wird z. B. von Halterungen genutzt, mit der das Gerät auf der Oberseite eines Fernsehers fest installiert werden kann. Für einen Einsatz im öffentlichen Raum ist jedoch eine einfache Befestigung meist nicht ausreichend. Hier müssen weitere Maßnahmen ergriffen werden. Eine Glasplatte sollte nicht zum Schutz vor Vandalismus eingesetzt werden, da diese das Infrarotlicht, das von den Geräten abgegeben wird, reflektiert.

Für eine Ausstellung in einem Museum wird die Kinect subjektiv als geeigneter bewertet. Zunächst besitzt sie einen deutlich größeren Erfassungsbereich, sodass auch eine Nutzung durch zwei Personen problemlos möglich ist. Die Leap Motion hingegen ist auf eine Person ausgelegt. Die meisten Besucher haben keine Vorkenntnisse mit dem Gerät, sodass sie als erstes den eingeschränkten Erfassungsbereich bemerken werden. Es müssen zunächst Erfahrungen gesammelt werden. Liegen diese vor, so gefällt dennoch der größere Erfassungsbereich deutlich besser, da man in seinen Bewegungen nicht eingeschränkt wird. Eine Person, die lediglich an der Kinect vorbei läuft, wird nicht immer erkannt. Passiert dies doch und befindet sich dabei eine Hand über der Hüfte, so kann es zu einem (kurzzeitigen) Erscheinen des Cursors kommen. Im Szenario einer Ausstellung wird dies jedoch als hinnehmbar eingestuft. Bei der Leap Motion kann es passieren, dass aufgrund der fest eingestellten Aufbauhöhe bei einzelnen Personen (insbesondere Kinder) eine Nutzung nicht oder nur eingeschränkt möglich ist. Für die Kinect stellen unterschiedliche Körpergrößen kein Problem dar.

Zur Zuverlässigkeit kann gesagt werden, dass bei der Leap Motion sehr selten auch Fehlerkennungen zu bemerken waren, falls die Hand direkt über dem Gerät positioniert war (optimale Position). Weiterhin reagiert das Gerät insgesamt empfindlicher auf Orientierungsänderungen. Zuletzt ist ein Vorteil der Kinect, das Hände übereinander positioniert werden können. Es ist denkbar, das einzelne Besucher diese Haltung der Hände ausführen.

6.2 Zuverlässigkeit der Gesten

In Versuchen hat sich gezeigt, dass die Fausthaltung zur Steuerung der Kamera zufriedenstellend von der Leap Motion erkannt wird, sofern man einige Regeln einhält bzw. Einschränkungen hinnimmt. Zunächst sollte eine Höhe von etwa 50 cm nicht überschritten werden. Ab dieser Höhe kommt es häufig vor, das das Gerät kurzzeitig einen Finger als ausgestreckt erkennt. Meist handelt es sich dabei um den Zeigefinger, sodass der Cursor erscheint. Selten kann es jedoch auch in einer niedrigeren Höhe zu einer irrtümlichen Erkennung eines ausgestreckten Zeigefingers kommen. Weiterhin sollte die Faust möglichst waagrecht (Handfläche zeigt nach unten) orientiert sein. Dadurch kann die Haltung in den meisten Positionen zuverlässig erkannt werden. Es konnte keine andere Orientierung gefunden werden, die insgesamt eine bessere Erkennung erzielt. Direkt über dem Gerät können bei einer Auf- und Abbewegung einer Faust, die um mehr als 45 Grad gedreht ist, häufige Aussetzer beobachtet werden. Wird selbiger Versuch mit einer vertikalen Faust durchgeführt, so konnten ebenso kontinuierliche Aussetzer wahrgenommen werden. Durch eine Verschiebung der Faust auf der X-Achse konnte die Erkennung jedoch deutlich verbessert werden. Grundsätzlich hat sich gezeigt, dass die Bewegung einer Faust, deren Handfläche oder Handrücken zum Gerät gerichtet ist, deutlich besser erkannt wird. Weiterhin konnten Aussetzer beobachtet werden, wenn die rechte (waagrechte) Faust weit nach links vom Gerät bewegt wurde (noch innerhalb des Erfassungsbereich). Analog konnte dies für die linke Faust festgestellt werden. Bei der rechten Faust waren die Aussetzer jedoch deutlich häufiger wahrzunehmen. Aus diesem Grund sollte man mit einer Faust möglichst nicht zu weit auf die gegenüberliegende Seite der Leap Motion kommen. Grundsätzlich empfiehlt es sich, mit der Fausthaltung direkt über dem Gerät oder (besser) etwas auf der Seite der jeweiligen Hand zu beginnen. Weiterhin sollte vermieden werden, den Daumen bei einer Fausthaltung auszustrecken. Der Daumen muss jedoch nicht innerhalb der Faust sein, es reicht aus, diesen auf die Faust aufzulegen. Bei einem ausgestreckten Daumen konnten insgesamt etwas häufiger kurzzeitige Aussetzer der Steuerung festgestellt werden. Ebenso konnten auch bei schnellen Bewegungen eine minimale Verschlechterung der Erkennung beobachtet werden.

Von der Kinect wird eine Fausthaltung mit einer minimalen Verzögerung erkannt. Diese Verzögerung kommt durch die Implementierung (siehe Abschnitt 5.2.1) zustande und wurde nicht als störend wahrgenommen. Falls der Daumen bei einer Faust ganz ausgestreckt ist, so ist es meist notwendig, die Hand einige Male hin- und herzubewegen, damit die Steuerung aktiviert wird. Dies gilt für die linke Hand. Bei der rechten Hand waren meist noch etwas mehr Bewegungen erforderlich. Falls die linke Faust um 180 Grad gedreht ist, d. h. die geschlossene Handfläche nach oben zeigt, so sind nur wenige Bewegungen notwendig, damit die Steuerung beginnt. Mit der rechten Hand müssen meist deutlich mehr Hin- und Herbewegungen ausgeführt werden. Manchmal klappt es auch erst nach einem zweiten Versuch. Wird nun zusätzlich der Daumen ausgestreckt, so kann auch mit Bewegungen die Steuerung nicht aktiviert werden. Dies gilt für die rechte und linke Hand. Da Nutzer nur sehr selten eine umgedrehte Faust machen, können die genannten Defizite jedoch weitestgehend vernachlässigt werden. Deutlich häufiger wird eine vertikale Faust ausgeführt. Für die linke Faust sind dieser Orientierung keine zusätzlichen Bewegungen erforderlich um die Steuerung zu aktivieren. Bei der rechten Faust ist selten ein kurzes Hin- und Herbewegen erforderlich. Streckt man nun den Daumen vollständig aus („Daumen hoch“-Geste), so mussten bei der linken Faust sehr viele Bewegungen ausgeführt werden, um die Steuerung zu aktivieren. Sporadisch hat es auch erst nach einem weiteren Versuch geklappt. Bei der rechten Faust ist eine Aktivierung der Steuerung nicht möglich gewesen. Orientiert man die Faust nun so, dass die Handfläche parallel zur Kinect zeigt, so wird auch mit ausgestrecktem Daumen bei beiden Händen die Steuerung ohne jegliche Bewegungen aktiviert. Es zeigt sich somit, das eine optimale Erkennung erzielt wird, falls die vollständige Handfläche vom Gerät wahrgenommen werden kann. Weiterhin kann aus den Beobachtungen abgeleitet werden, dass generell eine Faust der linken Hand besser erkannt wird und der Daumen möglichst nicht ausgestreckt werden sollte.

Nachdem eine Fausthaltung von der Kinect erkannt wurde, konnte in Versuchen keine Aussetzer der Kamerasteuerung festgestellt werden. Die Faust wurde dabei in alle Richtungen gedreht und gleichzeitig bewegt, wobei die Geschwindigkeit variiert wurde. Versuche wurden mit beiden Händen und auch mit ausgestrecktem Daumen durchgeführt.

Das Steuern eines Cursors der Benutzeroberfläche erfolgt bei der Leap Motion durch eine Hand mit ausgestrecktem Zeigefinger. Es hat sich gezeigt, dass diese Handhaltung bis in eine Höhe von etwa 55 cm zufriedenstellend erkannt wird. Wie auch bei der Faust, konnten jedoch auch unter dieser Grenze sporadisch Aussetzer wahrgenommen werden. Aussetzer zeigen sich dabei meist nicht nur durch ein kurzzeitiges Verschwinden des Cursors, sondern auch durch eine Veränderung der Kameraposition. Da der

Zeigefinger nicht mehr als ausgestreckt erkannt wird, wird die Handhaltung als Faust interpretiert.

Wurde die Handhaltung direkt über dem Gerät um die X-Achse gedreht (aus der Waagrechten), so kam es etwa 45 Grad zu einem dauerhaften Verschwinden des Cursors. Dies konnte für beide Drehrichtungen beobachtet werden. Die Position der Hand wurde dabei nicht verändert. In einem weiteren Versuch wurde anschließend eine Drehung um die Y-Achse durchgeführt. Hierbei wurde die linke Hand mit ausgestrecktem Zeigefinger so orientiert, das der Zeigefinger parallel zum Bildschirm steht und nach rechts zeigt. Als die Hand nun in Auf- und abbewegungen nach rechts bewegt wurde, so konnte eine kontinuierliche Zunahme an Aussetzern beobachtet werden. Der selbe Versuch wurde daraufhin mit der rechten Hand wiederholt. Die Hand wurde diesmal so ausgerichtet, dass der Zeigefinger nach links zeigt. Bei Auf- und Abbewegungen nach links war erneut eine Zunahme an Aussetzern wahrnehmbar. Abschließend wurde die Hand bei ständigen Auf- und abbewegungen um die Z-Achse gedreht. Hierbei konnte keine Zunahme an Aussetzern festgestellt werden. Aus den Ergebnissen dieser Versuche kann man ableiten, dass die Erkennung einer Hand mit ausgestrecktem Zeigefinger zuverlässiger klappt, falls der Zeigefinger besser vom Gerät wahrgenommen werden kann.

Falls die Geste sowohl von der linken als auch rechten Hand ausgeführt wurde, so konnte bei einem übereinander positionieren der Hände ein Ausfall des Cursor der oberen Hand beobachtet werden. Es hat dabei bereits gereicht, wenn die Hände nur teilweise überlagert worden sind. Als daraufhin die Hände wieder langsam voneinander weg bewegt worden sind, kam es häufig vor, dass der verschwundene Cursor nicht mehr erschienen ist. Die entsprechende Hand wurde vom Gerät nicht mehr erkannt. Erst nachdem die zugehörige Hand einige Male hin- und herbewegt worden ist, war der Cursor wieder sichtbar. Selten war es auch notwendig, die Hand komplett aus dem Erfassungsbereich des Geräts zu entfernen. Wurden die Arme nach einer Überlagerung überkreuzt, so war ebenso nur noch ein Cursor sichtbar. In seltenen Fällen konnte der zweite Cursor wieder sichtbar gemacht werden. Die Hände mussten dabei (überkreuzt) deutlich voneinander entfernt und die entsprechende Hand anschließend einige Male hin- und herbewegt werden. Wurde der Erfassungsbereich des Geräts bereits in überkreuzter Armhaltung betreten, so konnten häufig zunächst überhaupt keine Cursor beobachtet werden. Nach einigen Bewegungen der Hände war jedoch meist ein oder seltener beide Cursor sichtbar. Generell ist es in überkreuzter Armhaltung zu ständigen Aussetzern gekommen. Eine zufriedenstellende Steuerung von Cursor ist somit nicht möglich gewesen. Falls die Hände hingegen lediglich nebeneinander bewegt worden sind, so konnte keine Zunahme von Aussetzern festgestellt werden. Dies konnte auch beobachtet werden, falls der Abstand zwischen den Händen nur sehr gering war.

Bei höheren Bewegungsgeschwindigkeiten konnten grundsätzlich etwas mehr Aussetzer beobachtet werden. Durch ein zusätzliches Ausstrecken des Daumens konnte hingegen keine Verschlechterung der Erkennung beobachtet werden.

Für die Kinect wird eine flache Hand zum Steuern eines Cursors eingesetzt. Zunächst wurde die flache Hand genau waagrecht gehalten, wobei die Handfläche nach unten gezeigt hat. Sporadisch konnte beobachtet werden, das der Cursor auch nach wenigen Sekunden nicht erschienen ist. Meist wurde die Handhaltung dann als Faust interpretiert, sodass die Kamerasteuerung aktiviert worden ist. Bereits ein leichtes Anheben der Hand (Drehung entgegen dem Uhrzeigersinn um die X-Achse) oder manchmal auch ein Spreizen der Finger reichten aus, um den Cursor sofort erscheinen zu lassen. Falls die Handfläche parallel zum Gerät ausgerichtet war, so konnte wie erwartet eine problemlose Erkennung beobachtet werden. Im umgekehrten Fall, der Handrücken zeigt zur Kinect, war eine Erkennung möglich, falls die Finger gespreizt sind. Waren die Finger jedoch nicht gespreizt, so kam es auch durch beliebige Bewegungen (ohne die Orientierung zu verändern) nicht zu einem Erscheinen des Cursor. In einem weiteren Versuch wurde die Hand so ausgerichtet, das die schmale Seite bzw. Handkante parallel zur Kinect steht. In den meisten Fällen war der Cursor sofort sichtbar. Sporadisch kam es jedoch vor, dass der Cursor erst nach etwa zwei Sekunden erschienen ist oder auch nach längerer Zeit noch nicht sichtbar war. In letzterem Fall wurde die Hand leicht auf der X-Achse hin- und herbewegt, was den Cursor sofort auftauchen lassen hat. Selbiger Effekt konnte auch durch ein Spreizen der Finger erreicht werden. Alle beschriebenen Versuche wurden mit der linken und rechten Hand ausgeführt – die Ergebnisse waren identisch. Aus den Ergebnissen lässt sich ableiten, dass eine Erkennung der flachen Hand problemlos möglich, falls die Handfläche auch nur teilweise von der Kinect wahrgenommen werden kann. Auch wird die Erkennung durch ein Spreizen der Finger verbessert.

Als die Cursor-Steuerung aktiv war, konnte durch eine genau waagrechte Orientierung der flachen Hand (Finger nicht gespreizt; Handfläche zeigt nach unten oder oben) eine Aktivierung der Kamerasteuerung und damit ein Verschwinden des Cursors erreicht werden. Wie zuvor erwähnt, tritt dies jedoch nicht immer auf. Bewegt man die flache Hand und hält dabei ständig die waagrechte Orientierung ein, so kommt es sporadisch zu kurzzeitigen Kamerabewegungen. Generell sollte daher die flache Hand nicht in einer genau waagrechten Orientierung ausgeführt werden, sondern z. B. leicht angehoben werden. Falls dies dennoch gewünscht ist, so sollten die Finger gespreizt sein, da hierbei keine Aussetzer beobachtet werden konnten. Bei anderen Orientierungen der Hand konnten ebenso keine Aussetzer beobachtet werden.

Wurden mit beiden Händen Cursor gesteuert, so war es im Gegensatz zur Leap Motion ebenfalls möglich, bei Hände übereinander zu positionieren. Da die Kinect vor einer

Person aufgebaut ist, kann die Kamera nach wie vor beide Hände erfassen. Es ist jedoch zu beachten, das die Hände sich nicht zu Nahe kommen. In Versuchen hat sich gezeigt, dass Hände mindestens 15 cm voneinander entfernt sein müssen, damit sie von der Kinect erfasst werden. Wird diese Grenze unterschritten, verschwinden die Cursor. Die Leap Motion hingegen erkennt sogar noch nebeneinander liegende Hände, wenn sich deren Finger berühren. Bei einem Aufeinandertreffen der Handflächen verschwindet jedoch eine der beiden Hände. Auch ist es mit der Kinect möglich, Hände zu überkreuzen. Um Aussetzer zu vermeiden, muss der genannte Abstand dann jedoch auch jeweils zwischen Hand und Arm der anderen Hand eingehalten werden. Manchmal waren trotz Einhaltung des Abstands kurzzeitige Aussetzer zu beobachten.

Zuletzt sollte noch erwähnt werden, dass die Hände bei der Nutzung der Geräte möglichst nicht den Oberkörper berühren sollten, insbesondere nicht das Gesicht. Besonders die Leap Motion reagiert auf entsprechende Gesten, z. B. einem Anlegen von Daumen und Zeigefinger an die Nase. Diese Geste wird meist als Faust interpretiert, sodass es dann durch Kopfbewegungen zu einer Veränderung der Kameraposition kommt. Damit die Kinect nicht auf Handhaltungen von hängenden Händen reagiert, wurde als Mindesthöhe 10 cm über der Hüfte festgelegt.

6.3 Benutzerstudie

Um eine aussagekräftige Bewertung der erzielten Ergebnisse zu erhalten, wurde zum Abschluss des Arbeit eine qualitative Benutzerstudie durchgeführt. Die Teilnehmer sollten die gestenbasierte Interaktion mit MegaMol mittels einfacher Skalen subjektiv bewerten. Es ging dabei in erster Linie um einen Vergleich der Gesten, die zur Kamera- und Cursor-Steuerung für die Leap Motion und Kinect definiert worden sind. Die Gesten wurden bezüglich der Zuverlässigkeit der Erkennung, aber auch unabhängig von den Implementierungen bzw. Geräten bewertet. Abschnitt 6.3.1 beschreibt Organisation und Aufbau der Studie. Auf die Struktur und den Ablauf der Studie wird in Abschnitt 6.3.2 eingegangen. Die Präsentation und Analyse der Ergebnisse erfolgt schließlich in Abschnitt 6.3.3.

6.3.1 Organisation und Aufbau

Eine Einladung zur Teilnahme an der Benutzerstudie wurde über folgende Wege verbreitet:

- Verteiler studi-misc der Fachgruppe Informatik der Universität Stuttgart[37]. In dieser Mailingliste sind weitestgehend Studenten der Studiengänge Informatik und Softwaretechnik der Universität Stuttgart eingetragen.
- Mitarbeiter-Verteiler des Instituts für Visualisierung und Interaktive Systeme (VIS) der Universität Stuttgart.
- Facebook-Gruppe für Informatiker und Softwaretechniker der Universität Stuttgart[38].

Bestandteil dieser Einladung war ein Link zur einer Doodle[39]-Umfrage, über welche die Anmeldung erfolgt ist. In dieser Umfrage sollten alle Termine ausgewählt werden, an denen man für die Studie Zeit aufbringen kann. Mögliche Termine gab es jeweils am 30.11.2016 und 01.12.2016 durchgehend von 10 Uhr bis 17 Uhr (12 Uhr bis 13 Uhr ausgenommen). Die Studie wurde mit einer maximalen Länge von 40 Minuten angegeben. Einen Tag vor Beginn der Durchführung wurde um 16 Uhr die Umfrage geschlossen und den Teilnehmern ihren persönlichen Termin per E-Mail mitgeteilt. Für den Fall, dass es keine Verteilung gegeben hat, die allen angemeldeten Personen eine Teilnahme an der Studie möglich macht, so sollten die entsprechenden Personen ebenfalls darüber in Kenntnis gesetzt werden.

Die Studie wurde im Seminarraum 00.112 des VISUS-Gebäude der Universität Stuttgart (Allmandring 19, 70569 Stuttgart) durchgeführt. Der Raum wurde in drei Bereiche unterteilt. Der erste Bereich bestand aus einem Tisch, an dem einführende und abschließende Fragen gestellt worden sind. Abbildung 6.1 zeigt den Bereich, an dem die Leap Motion zur Interaktion mit MegaMol genutzt wurde. Um dem Anwendungsszenario einer Ausstellung möglichst nahe zu kommen, wurde festgelegt, dass ein Teilnehmer stehend die Leap Motion verwenden soll. Der Arm sollte bei einer Nutzung im Stehen möglichst rechtwinklig ausgerichtet sein und locker an der jeweiligen Körperseite anliegen oder minimal von dieser entfernt sein. Dadurch wird die Ermüdung so gering wie möglich gehalten. Nach DIN 33402 [DIN05] beträgt die durchschnittliche Ellbogenhöhe (Unterseite) für die Bevölkerung der Bundesrepublik Deutschland 106,75 cm[40]. Die Umrechnung auf die Cursor-Position erfolgt durch eine Interaction Box, welche eine Breite von 40 und eine Höhe von 28 besitzt. Die untere linke Ecke befindet sich im Punkt (-20 | 20) ausgehend vom Mittelpunkt der Oberseite des Geräts. Alle Angaben sind in Zentimeter. Damit bei einer genau rechtwinkligen Armhaltung der Cursor vertikal zentriert ist, ergibt sich mit den genannten Daten eine Aufbauhöhe (Oberseite des Geräts) von 106,75 cm – (28 cm / 2) – 20 cm = 72,75 cm.

[37] https://fius.informatik.uni-stuttgart.de/dienste/mailinglisten
[38] https://www.facebook.com/groups/unistuttgart
[39] http://doodle.com
[40] Mittelwert aus 95. Perzentil für Männer (117,5 cm) und 5. Perzentil für Frauen (96,0 cm)

Abbildung 6.1: Aufbau der Benutzerstudie für die Leap Motion

Dieser Wert wurde auf 80 cm aufgerundet, um zu berücksichtigen, das der ausgestreckte Zeigefinger etwas über der Höhe der Ellbogenunterseite liegt und die Person durch den Absatz der Schuhe ebenfalls leicht erhöht wird. Somit wurde der Tisch so eingestellt, das die Oberseite der Leap Motion bei dieser Höhe liegt. Die zuvor genannte Norm gibt ebenso an, dass die durchschnittliche Augenhöhe bei 158,25 cm[41] liegt. Der Tisch mit dem LCD-Monitor (29,8 Zoll) wurde so eingestellt, das die vertikale Mitte des Bildschirms etwa in dieser Höhe liegt.

Abbildung 6.2 zeigt die Konfiguration von MegaMol. Wie bereits erwähnt wurde, kam während der Entwicklung die Visualisierung des Enzyms β-Lactamase für Testzwecke zum Einsatz. Aus diesem Grund wurde die Entscheidung getroffen, diese auch in der Studie einzusetzen. Die Animation der Visualisierung wurde nicht aktiviert, damit die Teilnehmer nicht zwischen Animation und Zittern der Visualisierung unterscheiden müssen. Zur Intensität des Zitterns wurden Fragen in der Studie gestellt. Weiterhin wurde die Bounding Box deaktiviert, da sie in einer Ausstellung nicht verwendet wird. Über entsprechende Schaltflächen kann die Einfärbung des Enzyms verändert werden. Aktuell wird die Einfärbung „BlueRed“ verwendet. Die Schaltflächen unten rechts legen fest, wie das Rendering erfolgt. Aktuell ist „Normal“

[41] Mittelwert aus 95. Perzentil für Männer (173,5 cm) und 5. Perzentil für Frauen (143,0 cm)

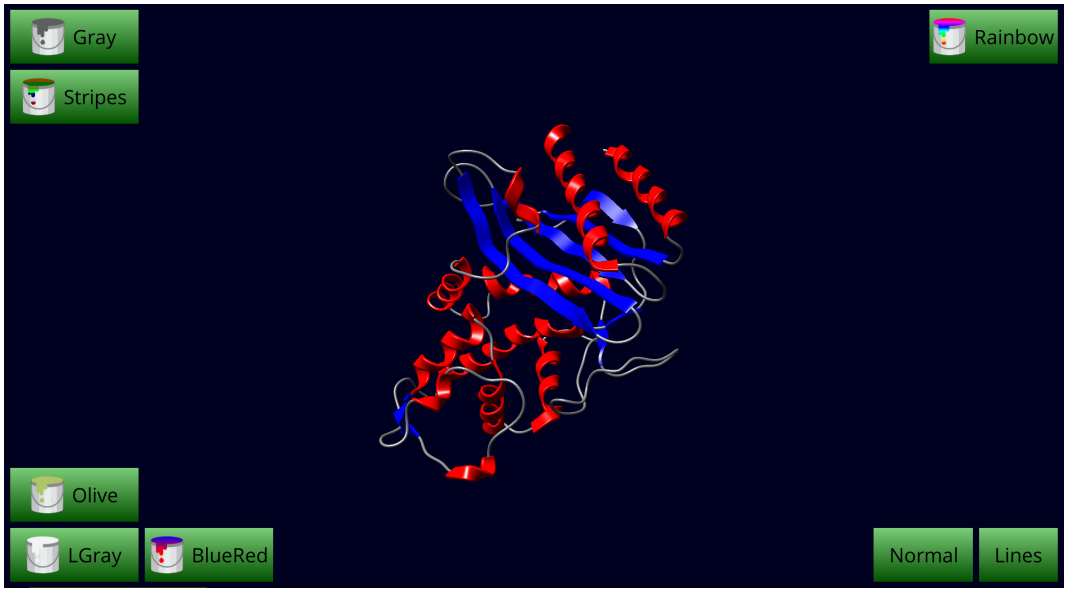

Abbildung 6.2: MegaMol-Konfiguration für die Benutzerstudie

aktiviert (Cartoon Hybrid Rendering). Durch einen Klick auf „Lines“ erfolgt die Darstellung des Enzyms durch Linien (Cartoon Lines Rendering). Die dargestellte Konfiguration war immer zu Beginn einer Durchführung eingestellt. In Anhang A.2 befindet sich die XML-Konfiguration zu dieser Benutzeroberfläche.

Die Leap Motion wurde über eine USB 3.0-Schnittstelle mit dem Notebook verbunden. Vom Tisch aus wurden Anweisungen gegeben, Fragen gestellt und beobachtet, wie der Teilnehmer sich verhält und ob es zu Aussetzern der Steuerung kommt. Zur Dokumentation wurde ein separates Notebook eingesetzt.

Abbildung 6.3 zeigt den Bereich des Raums, in dem die Interaktion mit der Kinect erfolgt ist. Der LCD-Monitor war das selbe Modell, dass auch bei der Leap Motion zum Einsatz gekommen ist. Auch wurde die Tischhöhe wieder so festgelegt, das die Bildschirmmitte bei etwa 158,25 cm liegt. Der durchschnittliche Nutzer blickt also bei waagrechter Kopfhaltung in die Mitte des Bildschirms. Die Kinect sollte in einer Höhe von 0,6 m bis 1,8 m aufgebaut werden. Je höher, desto besser. Dementsprechend wurde das Gerät auf dem Bildschirm platziert. Personen sollten möglichst vollständig von der Kinect erfasst werden. Daher musste das Gerät noch leicht nach unten gekippt werden (Anpassung des Kinect-Sensorwinkels[42]). Um zu gewährleisten, dass die Erfassung eines Teilnehmers stets unter gleichen Bedingungen erfolgt, wurde der Abstand zum Gerät fest vorgegeben. Für eine Person beträgt der Mindestabstand 1,4 m. Bei zwei Personen sind es 1,8 m. Da sich in eigenen Versuchen ein Abstand von 1,6 m bewährt hat, wurde dieser auch für die Studie festgelegt. Der entsprechende Abstand durch einen gelben Streifen markiert. Teilnehmer sollten sich zu Beginn mit ihren

[42] http://support.xbox.com/de-DE/xbox-one/accessories/angle-adjust-kinect

Abbildung 6.3: Aufbau der Benutzerstudie für die Kinect

Fußspitzen vor diesen Streifen stellen. In MegaMol wurde erneut die Visualisierung des β-Lactamase-Enzyms gezeigt. Die weitere Konfiguration war ebenso identisch.

Die Teilnehmer wurden wieder vom Tisch aus instruiert, befragt und beobachtet. Aufgrund des großen Erfassungsbereichs der Kinect wurde explizit darauf geachtet, das der Tisch nicht mehr auf dem Tiefenbild zu erkennen ist. Die Kinect war über den erforderlichen Adapter mit einer USB 3.0-Schnittstelle des Notebooks verbunden.

Damit kein Sonnenlicht die Benutzung der Geräte stört, waren die Fenster komplett abgedunkelt. Die Raumbeleuchtung war vollständig eingeschaltet.

6.3.2 Ablauf

Die Studie wurde in vier Abschnitte gegliedert. Für Fragen, die eine Bewertung als Antwort erwarten, wurde generell eine 5-stufige Skala eingesetzt. Die Differenzierbarkeit, die durch diese Skalenbreite ermöglicht wird, wurde für die Studie als angemessen eingestuft. Gleichzeitig ist ausgeschlossen, das Teilnehmer überfordert werden. Bis auf eine Ausnahme, die explizit erwähnt wird, wurde die Skala stets so orientiert, das ein höherer Wert für ein besseres Ergebnis steht.

Im einführenden Abschnitt wurden als erstes demografische Daten erhoben. Diese bestanden aus Alter, Geschlecht, Beruf und Händigkeit (Linkshänder/Rechtshänder). Mit letzterer Information sollte später bestimmt werden, ob ein Teilnehmer seine Haupthand auch für die Nutzung der Geräte einsetzt. Auch wurde gefragt, ob aktuell Einschränkungen bzw. Störungen des Bewegungsapparats vorliegen. Optional konnte angegeben werden, um welche Einschränkungen es sich konkret handelt. Anschließend sollte herausgefunden werden, welche Erfahrungen bzw. Kenntnisse der Teilnehmer bereits mit Geräten mitbringt, die eine berührungslose Steuerung ermöglichen. Folgende Fragen wurden hierfür gestellt:

1. Kannten Sie die Leap Motion bereits vor Beginn dieser Studie, rein vom Namen nach?
2. Haben Sie die Leap Motion bereits eingesetzt? (zutreffend falls Frage 1 bejaht)
3. Für was haben Sie die Leap Motion eingesetzt? (zutreffend falls Frage 2 bejaht)
4. Haben Sie bereits Anwendungen entwickelt, welche die Leap Motion nutzen? (zutreffend falls Frage 2 bejaht)
5. Wie gut hat Ihnen die Leap Motion insgesamt gefallen? (zutreffend falls Frage 2 bejaht)
6. Kannten Sie die Kinect bereits vor Beginn dieser Studie, rein vom Namen nach?
7. Haben Sie die Kinect bereits eingesetzt? (zutreffend falls Frage 6 bejaht)
8. Um welche Version bzw. Generation hat es sich gehandelt? (zutreffend falls Frage 7 bejaht)
    - Mögliche Antworten: Kinect für Xbox 360 / Kinect für Windows v1, Kinect für Xbox One / Kinect für Windows v2, unbekannt
9. Für was haben Sie die Kinect eingesetzt? (zutreffend falls Frage 7 bejaht)
10. Haben Sie bereits Anwendungen entwickelt, welche die Kinect nutzen? (zutreffend falls Frage 7 bejaht)
11. Wie gut hat Ihnen die Kinect für Xbox 360 bzw. Kinect für Windows v1 insgesamt gefallen? (zutreffend falls Version in Frage 8 genannt)
12. Wie gut hat Ihnen die Kinect für Xbox One bzw. Kinect für Windows v2 insgesamt gefallen? (zutreffend falls Version in Frage 8 genannt)
13. Haben Sie andere Geräte genutzt, die eine berührungslose Steuerung ermöglichen? Falls ja, welche?

Der zweite Abschnitt der Studie wurde an der Leap Motion durchgeführt. Dem Teilnehmer wurde mitgeteilt, sich zentriert vor das Gerät zu stellen, eine Hand jedoch zunächst noch nicht über das Gerät zu bewegen. Ihm wurde einführend erklärt, das er nun durch Bewegung einer Faust die 3D-Visualisierung, durch Bewegung einer Hand mit ausgestrecktem Zeigefinger einen gelben Cursor steuern wird. Er solle dabei durchgehend auf die Zuverlässigkeit beider Gesten achten. Die Zuverlässigkeit der Faust ist maximal, falls sich die Visualisierung nur dann bewegt, wenn die Faust bewegt wird und dabei keine Aussetzer der Steuerung zu bemerken sind. Die Zuverlässigkeit der Hand mit ausgestrecktem Zeigefinger ist maximal, falls der Cursor nur erscheint, wenn der Zeigefinger ausgestreckt ist und es dabei zu keinem (kurzzeitigen) Verschwinden des Cursors kommt. Diese Informationen lenken die Aufmerksamkeit des Teilnehmers auf Kriterien, die später für die Bewertung relevant sind.

Der Teilnehmer wurde nun instruiert, eine flache Hand über dem Gerät zu positionieren und daraufhin eine Faust zu machen. Generell wurden keine Vorgaben zur Orientierung einer Handhaltung gemacht. Selbiges gilt für die Armhaltung. Falls hierzu Fragen gekommen sind, so wurde dem Teilnehmer mitgeteilt, das er dies selbst entscheiden dürfe. Orientierung der Hand sowie Armhaltung wurden jedoch dokumentiert. Ebenso wurde festgehalten, welche Hand eingesetzt wurde und ob die Kamerasteuerung sofort aktiviert worden ist oder dies beispielsweise erst nach Bewegungen der Fall gewesen ist. Die Visualisierung sollte dann einige Male nach links und rechts, nach oben und unten und im Kreis bewegt werden. Dabei sollte mit der Geschwindigkeit der Handbewegung variiert werden. Falls Aussetzer aufgetreten sind, so wurde dies notiert. Dies gilt generell für alle Aufgaben. Anschließend sollte der Teilnehmer die Visualisierung einige Male vergrößern und verkleinern. In einer hohen Vergrößerung sollte die Visualisierung für wenige Sekunden gehalten werden, da hier das Zittern der Visualisierung deutlich stärker auftritt. Es wurde schließlich gefragt, wie das Zittern der Visualisierung bewertet wird. Daraufhin wurde der Teilnehmer angewiesen, seinen Zeigefinger auszustrecken. Falls der Cursor nicht sofort erschienen ist, so wurde dies notiert. Alle Schaltflächen der Benutzeroberfläche (siehe Abbildung 6.2) sollten daraufhin mit dem Cursor ausgelöst werden. Unten links sollte begonnen und im Uhrzeigersinn vorgegangen werden. Die Schaltflächen wurden in Gruppen zusammengefasst, da eine solche Anordnung typisch ist. Meist setzen die Schaltflächen einer Gruppe einen bestimmten Parameter auf verschiedene Werte. Die Gruppen wiederum wurden deutlich voneinander entfernt positioniert, damit Nutzer abwechselnd längere und kürzere Distanzen mit dem Cursor zurücklegen mussten. Es wurde festgehalten, falls ein Nutzer eine Schaltfläche nicht sofort getroffen hat oder von dieser „abgerutscht“ ist, was ein Indikator dafür ist, dass die Schaltflächengröße (oder auch die Cursor-Geschwindigkeit) verändert werden sollte. Weiterhin wurde

dokumentiert, ob sofort verstanden wird, wie eine Schaltfläche ausgelöst wird. Es kann daraus die Notwendigkeit abgeleitet werden, die visuellen Animationen noch ausdrucksstärker zu gestalten. Als der Nutzer an der Schaltfläche unten links angekommen ist, wurde er bezüglich des Zitterns des Cursors befragt. Nach Abschluss der Aufgabe wurde ihm mitgeteilt, dass er seine Hand wieder flach machen sollte und schließlich absenken kann. Der Teilnehmer musste dann die Zuverlässigkeit der Faust sowie Hand mit ausgestrecktem Zeigefinger für die Leap Motion bewerten. Unabhängig von Gerät bzw. Implementierung sollte er angeben, wie gut ihm die Hand mit ausgestrecktem Zeigefinger für eine Steuerung eines Cursors gefällt.

Nachdem bisher lediglich eine Hand verwendet wurde, soll jetzt eine Interaktion mit beiden Händen erfolgen. Wie auch zuvor wurden zunächst allgemeine Erläuterungen zum Ablauf gegeben und darauf hingewiesen, auf was geachtet werden sollte. Konkret wurde dem Teilnehmer mitgeteilt, dass er nun durch Bewegung der Faust und Hand mit ausgestrecktem Zeigefinger Visualisierung und Cursor gleichzeitig steuern wird. Anschließend werden zwei Cursor (gelb und weiß) gesteuert indem die Zeigefinger beide Hände ausgestreckt werden. Durchgehend soll wieder auf die Zuverlässigkeit beider Gesten geachtet werden. Abschließend soll dann jedoch die Interaktion mit beiden Händen insgesamt bewertet werden.

Der Teilnehmer wurde angewiesen, beide Hände nacheinander flach über dem Gerät zu positionieren. Mit einer Hand sollte daraufhin eine Faust gemacht werden, von der anderen Hand nur den Zeigefinger ausgestreckt werden. Es wurde dokumentiert, mit welcher Hand die jeweilige Handhaltung ausgeführt worden ist und ob Kamerasteuerung und Cursor-Steuerung sofort aktiviert worden sind. Der Teilnehmer sollte die Visualisierung daraufhin beliebig steuern und gleichzeitig auf eine Schaltfläche in einer oberen und unteren Ecke klicken. Aussetzer wurden notiert. Diese wären hier insbesondere denkbar, falls der Nutzer sich für eine Schaltfläche entscheidet, die auf der Seite der Faust liegt (Überlagerung der Hände). Anschließend wurde der Teilnehmer instruiert, auch den Zeigefinger seiner Fausthand auszustrecken. Mit einer Hand sollte nun auf die Schaltfläche unten links geklickt werden, die andere Hand sollte die Schaltfläche oben rechts auslösen. Dasselbe musste daraufhin mit den Schaltflächen in den übrigen beiden Ecken wiederholt werden. Danach wurde dem Teilnehmer mitgeteilt, beide Hände wieder flach zu machen und abzusenken. Wie angekündigt, musste abschließend die Zuverlässigkeit mit zwei Händen bewertet werden.

Im dritten Abschnitt der Studie wurde dann die Kinect für eine Interaktion mit MegaMol eingesetzt. Dementsprechend wurde der Teilnehmer gebeten, sich vor den gelb markierten Streifen zu stellen (siehe Abbildung 6.3). Die Kinect sollte sich genau vor ihm befinden. Da der Ablauf dieses Abschnitts weitestgehend dem vorherigen

entspricht, sollen im Folgenden lediglich auf wesentliche Unterschiede eingegangen werden.

Während die Schaltflächen im Uhrzeigersinn angeklickt werden mussten (flache Hand), sollte der Teilnehmer nun die Wartezeit bis zum Auslösen einer Schaltfläche bewerten. Die Wartezeit ist unabhängig von den Geräten definiert. Für die Bewertung wurde eine abweichende Orientierung der Skala eingesetzt. Der niedrigste Wert steht für eine viel zu kurze, der höchste Wert für eine viel zu lange Zeit. Dementsprechend repräsentiert der mittlere Wert 3 die optimale Wartezeit. Nachdem die Fausthaltung bei der Kinect ein weiteres Mal zum Einsatz gekommen ist, sollte der Teilnehmer nun angeben, wie gut ihm die Faust zur Steuerung der Kamera unabhängig vom Gerät gefällt. Ebenso sollte die Interaktion mit der Benutzeroberfläche (animierter Cursor-Ring) bewertet werden. Da die Kinect eine Überlagerung von Händen erlaubt, wurde bei der Steuerung von zwei Cursor eine Aufgabe eingeführt, in der die Schaltflächen oben rechts und unten rechts angeklickt werden mussten. Bei Fragen, die für die Kinect ein weiteres Mal gestellt worden sind (z. B. Zuverlässigkeit der Geste zur Steuerung eines Cursor), wurde dem Teilnehmer seine bereits abgegebene Antwort mitgeteilt.

Abschnitt 4 bildete den Abschluss der Studie. Der Teilnehmer sollte angeben, welches der beiden Geräte ihm bezüglich der Zuverlässigkeit insgesamt besser gefallen hat. Ebenso sollte er das Gerät benennen, bei dem ihm die Steuerung insgesamt eher zugesagt hat, unabhängig von der Zuverlässigkeit. Die Gestensteuerung soll zukünftig in einer wissenschaftlichen Ausstellung eingesetzt werden. Aus diesem Grund wurde der Teilnehmer nach dem Gerät gefragt, welches seiner Meinung nach besser geeignet ist für dieses Anwendungsszenario. Begründungen zu den getroffenen Entscheidungen für ein bestimmtes Gerät waren explizit erwünscht. Abschließend gab es noch die Gelegenheit, Kritik und Verbesserungsvorschläge zum Ablauf der Studie zu äußern.

Die Erläuterungen, Anweisungen und Fragen, die an einen Teilnehmer gerichtet worden sind, waren durch einen Ablaufplan vorgegeben. Für jeden Teilnehmer wurde ein separater Plan eingesetzt, da in diesem auch gleichzeitig die Antworten erfasst worden sind. Das Ausfüllen des Dokuments wurde durch ein PDF-Formular realisiert. Um die Aussagekraft der Studie zu erhöhen, wurden die Ablaufpläne so zusammengestellt, dass bei der einen Hälfte der Teilnehmer zuerst Abschnitt 2, bei der anderen Hälfte zuerst Abschnitt 3 durchgeführt werden. Jeder Ablaufplan wurde mit einer Nummer versehen. Ein Zettel, der zu Beginn einer Durchführung gezogen werden musste, hat den Ablaufplan für den jeweiligen Teilnehmer festgelegt. Für den Fall, das die Zahl der angemeldeten Personen ungerade ist, sollte ein zusätzlicher Ablaufplan vorbereitet werden. Der Zufall entscheidet dann, welches der beiden Abläufe häufiger vorkommt.

6.3.3 Ergebnisse und Auswertung

Über die Doodle-Umfrage haben sich insgesamt 10 Personen für die Benutzerstudie angemeldet. Jeder Person konnte ein Termin zugewiesen werden. Eine Person ist zum vereinbarten Termin nicht erschienen, sodass letztendlich 9 Personen (3 davon weiblich) die Studie absolviert haben. Die Teilnehmer lagen im Altersbereich von 25 bis 40 Jahren, wobei fast alle Personen zwischen 25 und 35 Jahren alt waren. Kein Teilnehmer war Linkshänder. Bis auf zwei Ausnahmen haben ausschließlich wissenschaftliche Mitarbeiter teilgenommen, die zuvor Informatik oder Softwaretechnik studiert hatten. Alle Teilnehmer haben angegeben, das sie die Kinect vom Namen nach kennen. Lediglich 3 Personen haben noch nie eine Kinect genutzt. Die übrigen 6 Teilnehmer haben bis auf eine Ausnahme ausschließlich die Kinect für Xbox 360 verwendet. Von dieser Gruppe wiederum haben alle das Gerät für Spiele auf der Konsole genutzt. Die Hälfte hat jedoch weiterhin angegeben, auch bereits Anwendungen für den PC entwickelt zu haben. Die Kinect für Xbox 360 wurde vorwiegend mit 4 Punkten bewertet. Ungefähr die Hälfte der Teilnehmer haben bereits die Leap Motion für Anwendungen am PC genutzt. Nur eine Person hat bereits Anwendungen entwickelt, die das Gerät nutzt. Zwei Personen kannten das Gerät überhaupt nicht. Die Leap Motion wurde ebenfalls überwiegend mit 4 Punkten bewertet. Als gefragt wurde, ob bereits andere Geräte für eine berührungslose Steuerung genutzt worden sind, haben einige Teilnehmer die Wiimote genannt, den Spielecontroller der Nintendo Wii. Die Interaktion erfolgt hier jedoch nicht berührungslos, da das Gerät stets in den Händen gehalten werden muss. Ein Teilnehmer hat angegeben, dass er bereits mit Produkten von Qualisys[43] und Optitrack[44] gearbeitet hat. Ein anderer Teilnehmer hat ein Gerät getestet, das wie die Leap Motion auf die Erfassung der Hände ausgerichtet ist, jedoch mittels mehrerer Farbkameras funktioniert. Kein Teilnehmer hat angegeben, das aktuell Beschwerden vorliegen, die seine Beweglichkeit einschränken.

Als Teilnehmer angewiesen worden sind, eine flache Hand über die Leap Motion zu positionieren, so haben alle ihre rechte Hand (Haupthand) eingesetzt und diese parallel zum Gerät orientiert. Die Teilnehmer haben dabei ihren Arm angewinkelt, jedoch nicht ganz rechtwinklig. Ein Teilnehmer hat seinen Arm zunächst nahezu ganz ausgestreckt, jedoch kurze Zeit später ebenfalls etwas angewinkelt. Hierfür hat er sich näher an das Gerät begeben. Die Fausthaltung wurde bei allen Teilnehmern sofort erkannt. Jeder Teilnehmer hat die Faust so orientiert, das die geschlossene Handfläche nach unten gerichtet war. Bei den anschließenden Kamerabewegungen ist bei einigen Teilnehmern kurzzeitig der Cursor erschienen, als die Hand weit nach oben oder nach links bewegt worden ist. Insbesondere letzteres wurde bereits in

[43] http://www.qualisys.com
[44] http://www.optitrack.com

Abschnitt 6.2 beobachtet. Eine Faust darf nicht zu weit auf die gegenüberliegende Seite kommen. Als daraufhin die Visualisierung vergrößert und verkleinert worden ist, haben einige Teilnehmer ihren Arm ganz ausgestreckt oder sehr an den Körper heran bewegt. Hierbei war manchmal ebenso kurzzeitig der Cursor wahrzunehmen. Bei einem Teilnehmer waren jedoch auch an einer normalen Position kurzzeitige Aussetzer zu beobachten. Das Zittern der Visualisierung wurde von den meisten Teilnehmern mit 4 Punkten bewertet (höhere Punktzahl entspricht geringerem Zittern). Beim Ausstrecken des Zeigefingers war bei allen Teilnehmern der Cursor sofort sichtbar. Als die Nutzer daraufhin die Schaltflächen anklicken mussten, kam es bei einigen Teilnehmern insbesondere in der oberen linken Ecke zu Aussetzern. Hierbei wurde auch die Visualisierung leicht bewegt. Ein anderer Teilnehmer hat einen Aussetzer produziert indem er den Zeigefinger weit nach unten gerichtet hat. Das Zittern des Cursor wurde von fast allen Teilnehmern mit 5 Punkten bewertet. Beim Absenken der Hand kam es bei zwei Teilnehmern zu einer leichten Veränderung der Kamera (Heranzoomen). Ein Teilnehmer hat in der Bewegung nach unten eine Faust gemacht, wodurch die Kameraposition deutlich verändert worden ist. Die Zuverlässigkeit der Fausthaltung wurde meist mit 4 Punkten bewertet. Selbiges gilt für die Handhaltung mit ausgestrecktem Zeigefinger. Letztere Geste wurden von den Nutzern als intuitiv eingestuft, sodass sie unabhängig von der Zuverlässigkeit 4 oder 5 Punkte vergeben haben. Ein Teilnehmer hat lediglich mit 2 Punkten bewertet, da er das durchgehende Ausstrecken des Zeigefingers als anstrengend empfindet.

Als Faust und Hand mit ausgestrecktem Zeigefinger gleichzeitig ausgeführt werden mussten, haben bis auf eine Ausnahme alle Teilnehmer ihre rechte Hand für die Faust verwendet. Kamerasteuerung sowie Cursor-Steuerung waren sofort aktiv. Zwei Teilnehmer haben sich dafür entschieden, Schaltflächen anzuklicken, die sich auf der Seite der Faust befinden. Ersterer Teilnehmer hat die obere Schaltfläche gewählt, sodass die Hand mit ausgestrecktem Zeigefinger über die Faust hinweg bewegt worden ist. Es ist dabei zu einem dauerhaften Ausfall des Cursors gekommen. Der andere Teilnehmer, der sich für die untere Schaltfläche entschieden hat, konnte diese hingegen auslösen. Die Kamerasteuerung ist nur für einen Moment ausgefallen als die Faust durch die andere Hand verdeckt worden ist. Bei den übrigen Teilnehmern, welche die Arme nicht überkreuzt hatten, gab es weitestgehend keine Aussetzer. Bei einem Teilnehmer konnte jedoch beobachtet werden, das der Cursor nicht immer sofort dem Zeigefinger folgt, wenn Faust und Hand mit ausgestrecktem Zeigefinger nah nebeneinander positioniert sind. Als nun auch der Zeigefinger der anderen Hand ausgestreckt worden ist, waren bei zwei Teilnehmern einige Handbewegungen erforderlich, um den zweiten Cursor erscheinen zu lassen. Beim Steuern der beiden Cursor zu den Schaltflächen in den Ecken waren bei etwa der Hälfte der Teilnehmer kurze Aussetzer oder auch ein Springen von Cursor zu beobachten. Eine Person

hat die Arme überkreuzt wodurch es zum Ausfall des oberen Cursors gekommen ist. Während dem Absenken der beiden Hände waren bei zwei Teilnehmern wieder leichte Kamerabewegungen wahrzunehmen. Die Zuverlässigkeit der Interaktion mit zwei Händen wurde von den meisten Teilnehmern abschließend mit 3 bis 4 Punkten bewertet.

Auch an der Kinect haben sich alle Teilnehmer dazu entschieden, ihre rechte Hand für die einhändige Interaktion einzusetzen. Auffällig war, das zu Beginn nur drei Teilnehmer eine etwas angewinkelte Armhaltung eingenommen haben, die auf Dauer zu einer geringeren Ermüdung führt. Alle anderen Teilnehmer haben ihren Arm nahezu komplett ausgestreckt. Die Fausthaltung, die (zunächst) durchgehend waagrecht ausgeführt worden ist, wurde bei allen Teilnehmern sofort erkannt. Während der anschließenden Ausführung der Kamerabewegungen (auch Vergrößern und Verkleinern) kam es bei einer Person zu einem kurzzeitigen Erscheinen des Cursor. Bei allen anderen Personen waren keine Aussetzer zu beobachten, auch als einzelne Teilnehmer dann die Faust aufgerichtet haben (Handfläche zeigt nach links). Das Zittern der Visualisierung wurde von den meisten Teilnehmern mit 3 Punkten bewertet. Ein Teilnehmer hat angegeben, das das Zittern in einer sehr hohen Vergrößerung eher etwas störend ist. Als daraufhin die Hand flach gemacht werden musste, haben alle Teilnehmer ihre Hand genau waagrecht gehalten und etwa die Hälfte weiterhin ihre Finger gespreizt. Falls letzteres nicht erfolgt ist, waren meist kurze Bewegungen notwendig, damit der Cursor erschienen ist. Waren die Finger hingegen gespreizt, so waren Bewegungen seltener erforderlich. Die Kamerasteuerung war dabei nach wie vor aktiv. Entsprechende Beobachtungen konnten bereits in eigenen Versuchen (siehe Abschnitt 6.2) gemacht werden. Während dem Auslösen der Schaltflächen waren bei einigen Personen sporadisch kurze Aussetzer wahrzunehmen (Kamerasteuerung war aktiv). Die Hand wurde hierbei waagrecht gehalten, die Finger waren nicht gespreizt. Zwei Teilnehmer haben schließlich selbständig bemerkt, das es durch ein Anheben der Hand zu keinem Aussetzer mehr kommt. Ein anderer Teilnehmer, der zuvor noch nicht die Leap Motion genutzt hatte, hat auf der ersten Schaltfläche sofort eine Faust gemacht, da er angenommen hat, das auf diese Weise die Schaltfläche angeklickt wird. Auf den Schaltflächen links unten (gegenüberliegende Seite der Hand) war häufig ein deutliches Springen des Cursor zu beobachten. Insbesondere aus diesem Grund wurde das Zittern des Cursor meist nur mit 3 Punkten bewertet. Beim Absenken der Hand kam es lediglich bei einer Person zu einer Kamerabewegung. Diese hat hierbei jedoch eine Faust gemacht. Die Zuverlässigkeit der Faust wurde weitestgehend mit 5 Punkten bewertet. Für die Zuverlässigkeit der flachen Hand wurden überwiegend 3 Punkte vergeben. Hier wäre vermutlich ein deutlich besseres Ergebnis erzielbar gewesen, falls man den Teilnehmern vorab mitgeteilt hätte, das die Hand angehoben werden müsse (oder zumindest die Finger gespreizt). Dies hätte ebenso das Zittern

des Cursor deutlich reduziert. Unabhängig vom Gerät wurde die flache Hand von den meisten Teilnehmern mit 4 Punkten bewertet. Nutzer haben angegeben, das die Geste etwas weniger intuitiv als der ausgestreckte Zeigefinger ist.

Bei einer gleichzeitigen Ausführung von Faust und flacher Hand konnte ebenso sporadisch beobachtet werden, das der Cursor erst nach wenigen Bewegungen erschienen ist. Die entsprechende Hand wurde dabei genau waagrecht gehalten, die Finger waren meist nicht gespreizt. Das dies jedoch häufiger als zuvor auftritt, konnte nicht festgestellt werden. Die Fausthaltung, die von allen Teilnehmern mit der rechten Hand ausgeführt worden ist, wurde stets sofort erkannt. Als daraufhin Schaltflächen angeklickt worden sind, konnten auf der Seite der flachen Hand keine Aussetzer festgestellt werden. Mehrere Teilnehmer haben Schaltflächen auf der Seite der Faust ausgelöst. Bei einem dieser Teilnehmer kam es zu einem Ausfall des Cursor, da sich die Hände zu Nahe gekommen sind. Als schließlich zwei Cursor gesteuert worden sind, haben einige Teilnehmer Schaltflächen auf der selben Seite gleichzeitig angeklickt. Generell waren keine Aussetzer zu beobachten. Beim Absenken der beiden Hände konnte bei keinem Teilnehmer eine Kamerabewegung beobachtet werden. Die Zuverlässigkeit der Interaktion mit zwei Händen wurde von den Teilnehmern meist mit 4 Punkten bewertet.

Die Fausthaltung, die sowohl bei der Leap Motion als auch Kinect verwendet wird, wurde unabhängig von den Geräten durchgehend mit 4 oder 5 Punkten bewertet. Ein Teilnehmer hat als Kritikpunkt angegeben, das man beim Absenken einer Hand dazu neigt, diese zu schließen. Man muss explizit darauf achten, die Hand flach zu halten, um eine Kamerabewegung zu vermeiden. Von zwei Teilnehmern kam die Idee, eine Greifhand zum Steuern der Kamera zu verwenden, da man so noch mehr das Gefühl erhält, mit der Visualisierung zu interagieren. Andere Teilnehmer wiederum haben betont, das die Faust sehr intuitiv ist. Man hat das Gefühl, die Visualisierung in der Hand zu halten. Die Interaktion mit der Benutzeroberfläche durch den animierten Cursor-Ring hat den meisten Benutzern ebenfalls gut gefallen. Überwiegend wurden 4 Punkten vergeben. Einige Teilnehmer haben jedoch angemerkt, dass es besser wäre, wenn man eine Schaltfläche aktiv auslösen könnte. Eine Geste, die hierfür genannt worden ist, war eine schnelle Vor- und Zurückbewegung des Zeigefingers. Von einem anderen Teilnehmer kam die Idee, einen Klick durch ein kurzzeitiges Zusammendrücken von Daumen und Zeigefinger auszulösen. Für die Kinect wurde vorgeschlagen, einen Klick durch ein schnelles Schließen und Öffnen der Hand umzusetzen. Letztere Geste würde jedoch kurzzeitig die Kamerasteuerung aktivieren. Man müsste folglich eine kurze Verzögerung festlegen, bis die Kamera bewegt werden kann, sofern die Faust zur Steuerung der Kamera beibehalten werden soll. Ein weiterer Teilnehmer hat jedoch darauf hingewiesen, das es bei einer solchen Klickbewegung passieren kann, das man von einer Schaltfläche abrutscht, sofern der Cursor ungünstig

positioniert ist. Durch eine weitere Geste wird zudem die Komplexität der Steuerung erhöht. Die Wartezeit, bis eine Schaltfläche ausgelöst wird, wurde von den meisten Teilnehmern mit 3 Punkten (optimal) bewertet. Zwei Teilnehmer waren der Meinung, das es etwas zu lange dauert und haben dementsprechend 4 Punkte vergeben.

Die Mittelwerte aller zuvor erwähnten Bewertungen, die während der Benutzung der Geräte abgegeben worden sind, sind in den Abbildungen 6.4 und 6.5 veranschaulicht.

Im letzten Abschnitt der Studie wurde nach dem Gerät gefragt, dass bezüglich der Zuverlässigkeit insgesamt am besten ist. 5 von 9 Teilnehmern haben sich für die Kinect entschieden, sodass eine klare Meinung nicht abgeleitet werden kann. Eine mögliche Ursache für dieses Ergebnis ist, das die zu erledigenden Aufgaben von jedem Teilnehmer nicht genau gleich ausgeführt worden sind. Als beispielsweise mit der Leap Motion die Kamera gesteuert werden sollte, so haben einige Teilnehmer ihre rechte Hand sehr weit auf die linke Seite bewegt. Hierbei sind ggf. Aussetzer aufgetreten. Andere Teilnehmer wiederum haben ihre Hand nicht dorthin bewegt, sodass ein anderer Eindruck von der Zuverlässigkeit vermittelt worden ist. Ein anderes Beispiel ist das Überlagern von Händen, das ebenfalls nicht von jedem Teilnehmer getestet worden ist. Auch wurde es jedem Teilnehmer erlaubt, eigene kurze Versuche mit dem

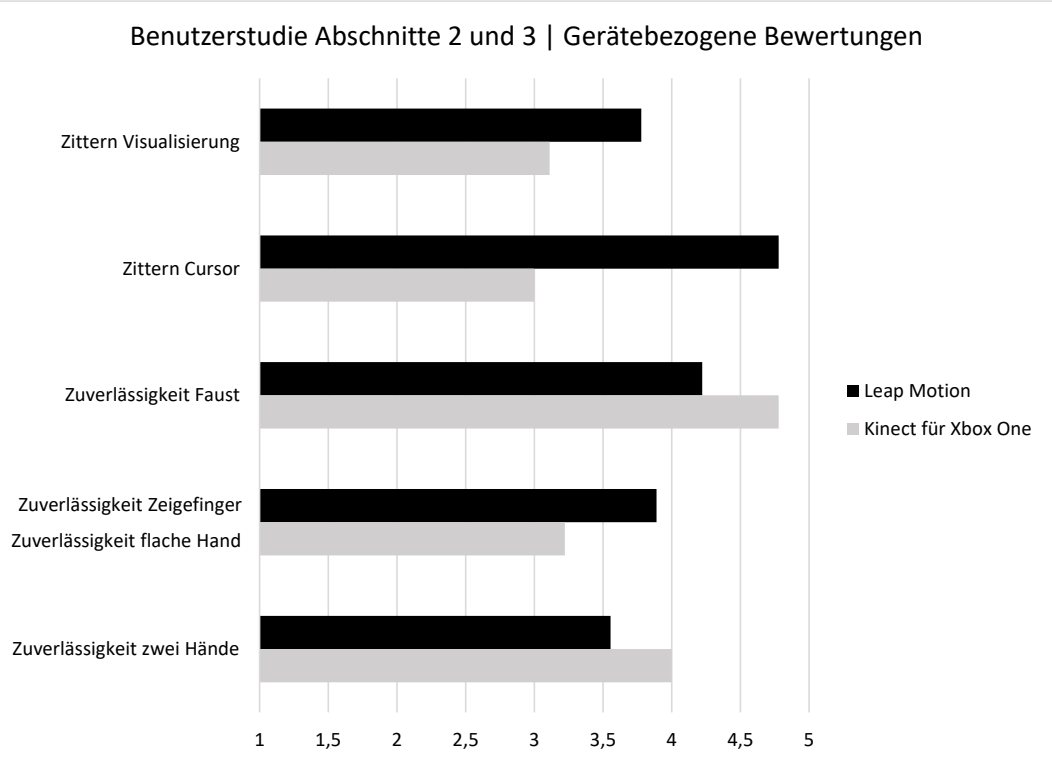

Abbildung 6.4: Ergebnisse der Benutzerstudie Abschnitte 2 und 3 – gerätebezogene Bewertungen

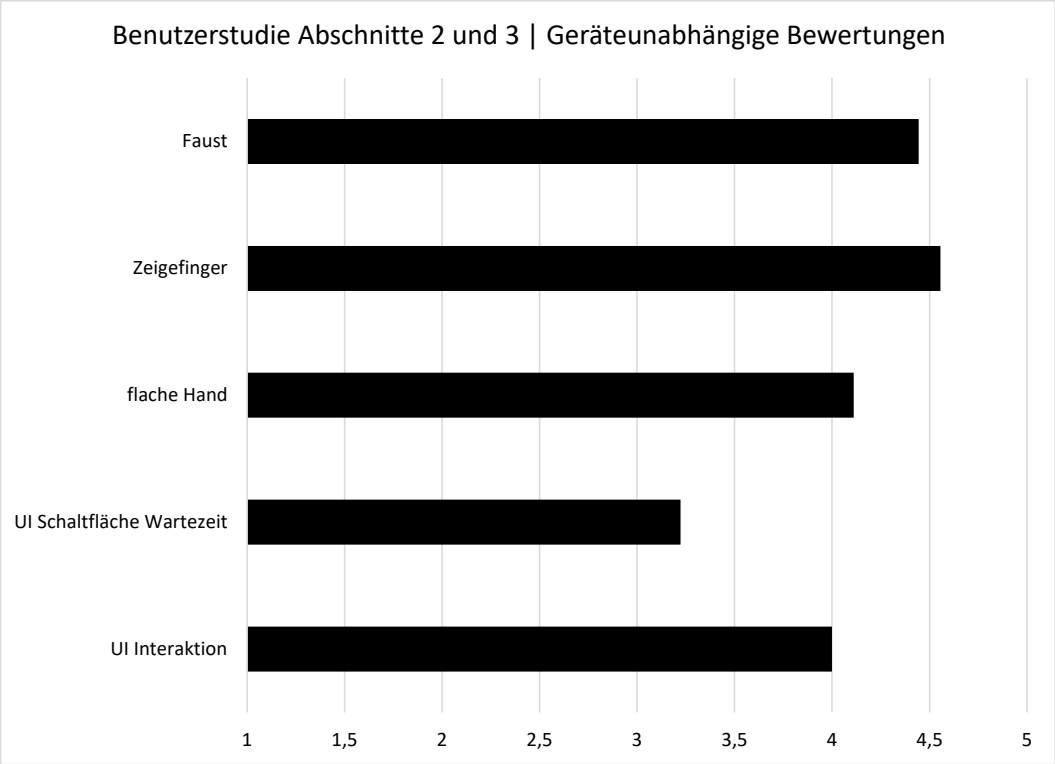

Abbildung 6.5: Ergebnisse der Benutzerstudie Abschnitte 2 und 3 – geräteunabhängige Bewertungen

Gerät durchzuführen. Weiterhin war ebenso nicht vorgegeben, wie die Handhaltungen exakt ausgeführt werden sollten. Beispielsweise haben einzelne Nutzer stets darauf geachtet, das bei der Kinect die flache Hand möglichst waagrecht ausgeführt wird. Dies hatte jedoch häufig zur Folge, dass es öfters zu kurzen Aussetzern gekommen ist. Zwei Teilnehmer haben erkannt, dass ein leichtes Anheben der Hand das Auftreten von Aussetzern verhindert. Beide Teilnehmer haben die Kinect als zuverlässigeres Gerät gewählt. Zuletzt haben vermutlich auch die unterschiedlichen Anatomien der Hände die Erkennung beeinflusst. Die zweite Frage, bei welchem Gerät die Steuerung eher zusagt, hat hingegen ein eindeutiges Ergebnis erzielt. 8 von 9 Teilnehmer haben die Leap Motion gewählt. Alle 8 Teilnehmer haben angegeben, das ihnen die Zeigefingergeste besser gefällt. Mehrere Personen haben weiterhin mit dem etwas höheren Komfort der Leap Motion argumentiert. Ein Teilnehmer hat die niedrigere Verzögerung der Leap Motion erwähnt. Als nach der Eignung für eine Ausstellung in einem Museum gefragt worden ist, sind die Meinungen dann jedoch wieder auseinander gegangen. 5 von 9 Teilnehmer haben sich für die Leap Motion entschieden. Alle 5 Teilnehmer haben darauf hingewiesen, das bei der Leap Motion eine Hand explizit über das Gerät bewegt werden muss. Ihr Erfassungsbereich ist deutlich kleiner als der von der Kinect. Dementsprechend ist es nahezu ausgeschlossen, dass die Hand einer Person erfasst wird, die sich lediglich in der Nähe des Geräts befindet. Ein Argument

für die Kinect war der deutlich höhere Bekanntheitsgrad, sodass eine Einweisung in das Gerät meist nicht erforderlich ist. Weiterhin wurde auch die Möglichkeit, Hände überlagern zu können, als ein Vorteil hervorgehoben.

Im Hinblick auf die Zuverlässigkeit der Gesten kann zusammengefasst gesagt werden, dass beide Geräte etwa gleich gut in der Studie abgeschnitten haben. Bei der Faust-haltung kam es bei der Leap Motion immer wieder zu kurzzeitigen Aussetzern, falls Personen sich den Rändern des Erfassungsbereichs genähert haben, insbesondere jedoch dem linken Rand. Alle Personen haben die Faust mit der rechten Hand ausgeführt, sodass dieser stets gegenüberliegend war. Die Kinect hat hier aufgrund ihres deutlich größeren Erfassungsbereichs keine Probleme gezeigt, sodass die Zuverlässigkeit der Faust für die Kinect höher bewertet worden ist. Als der Cursor durch die Leap Motion gesteuert worden ist, so waren ebenfalls insbesondere auf der linken Seite Aussetzer zu beobachten. Viele Teilnehmer haben daher für die Hand mit ausgestrecktem Zeigefinger die gleiche Punktzahl vergeben. Bei der Kinect waren häufig kurze Bewegungen der Hand erforderlich, bis der Cursor erschienen ist. Der Grund hierfür war, dass Personen ihre Hand waagrecht gehalten haben. War die Hand während dem anschließenden Steuern des Cursors ebenso noch genau waagrecht orientiert und zudem die Finger nicht gespreizt, so kam es sporadisch zu Aussetzern. Die flache Hand wurde somit etwas niedriger bewertet als die Hand mit ausgestrecktem Zeigefinger der Leap Motion. Hätte man jedoch die Teilnehmer zu Beginn angewiesen, die flache Hand anzuheben und diese Orientierung beizubehalten, so wäre die Bewertung vermutlich deutlich besser ausgefallen.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Steuerung für das Visualisierungs-Framework MegaMol entwickelt, die es erlaubt, durch berührungslose Gesten mit Visualisierungen zu interagieren. Die Steuerung wurde mit zwei Hardware-Lösungen realisiert, der Leap Motion sowie Kinect für Xbox One. Beide Geräte erzeugen ein dreidimensionales Bild der Hand bzw. des Körpers und liefern hierzu Positionsdaten zurück.

Zukünftig soll MegaMol in wissenschaftlichen Ausstellungen eingesetzt werden, in denen kein Personal zur Verfügung steht. In diesem Anwendungsszenario soll die Gestensteuerung zum Einsatz kommen. Dementsprechend musste bei der Wahl der Gesten berücksichtigt werden, das diese einfach auszuführen sind und mit minimalen Aufwand erlernt werden können. Gemäß Aufgabenstellung sollte eine Geste definiert werden, welche die virtuelle Kamera von MegaMol steuert. Letztendlich wurde entschieden, die Kamerasteuerung durch eine Faustbewegung in alle drei Dimensionen umzusetzen. Die Fausthaltung muss nicht erlernt werden und vermittelt das Gefühl, die Visualisierung festzuhalten, was als passend eingestuft worden ist. Für beide Geräte wurde dieselbe Geste verwendet, da die SDKs bereits eine Erkennung der Faust erlauben. Es kann immer nur eine Faust die Kamera steuern. Ist die Kamerasteuerung aktiv und wird eine weitere Faust erkannt, so wird diese ignoriert.

Weiterhin wurde eine Benutzeroberfläche entwickelt, die eine Modifikation von Parametern der Visualisierung erlaubt und ebenso gestenbasiert gesteuert wird. Zur Cursor-Steuerung wurde für die Leap Motion – inspiriert von Touch-Geräten – eine Hand mit ausgestrecktem Zeigefinger gewählt. Die Kinect verfügt über ein deutlich einfacheres Handmodell, sodass ihr SDK keine Möglichkeiten vorsieht, um zwischen Fingern unterscheiden zu können. Die Entscheidung ist auf eine flache Hand gefallen, da sich diese deutlich von der Faust abgrenzt. Eine Erkennung durch das SDK ist möglich. Durch Bewegungen in X- und Y-Richtung wird der Cursor gesteuert. Parameter werden verändert, indem Schaltflächen von Cursor ausgelöst werden. Hierzu muss ein Cursor für eine definierte Wartezeit auf der Schaltfläche verbleiben, was durch eine Animation des Cursors visualisiert wird. Die Konfiguration der Benutzeroberfläche wird in XML definiert. Schaltflächen können mit Text und einem Icon (Bild oder Zeichen aus Schriftart) versehen werden. Es stehen verschiedene Gestaltungsmöglichkeiten zur Verfügung. Neben Schaltflächen müssen auch Cursor explizit angegeben werden. Falls mehr Hände die genannten Handhaltungen ausführen als

Cursor definiert sind, so werden diese ignoriert. Leap Motion und Kinect können auch kombiniert eingesetzt werden.

Für die Leap Motion wurden weitere Gesten realisiert. Über zwei Kreisbewegungen eines Fingers entgegen dem Uhrzeigersinn wird die Kameraposition zurückgesetzt. Beispielsweise wäre ein Einsatz dieser Geste denkbar, falls die Kameraposition so verändert worden ist, das die Visualisierung nicht mehr zu erkennen ist. Dies kann z. B. durch einen Nutzer erfolgt sein, der noch nicht mit Gestensteuerung vertraut ist. Die Ausführung der Geste im Uhrzeigersinn startet bzw. stoppt die Animation einer Visualisierung. Zwei Kreisbewegungen stellen sicher, das die entsprechenden Aktionen nicht unbeabsichtigt ausgelöst werden. Die Erkennung der Kreisgesten erfolgt über Algorithmen des SDK.

Im Rahmen der Implementierung wurden zwei MegaMol-Module erstellt. Ein View-Modul realisiert die Interaktion mit den Geräten, die Erkennung von Gesten sowie die Kamerasteuerung. Es wurde eine Schnittstelle für Geräte zur Bewegungssteuerung definiert. Auf diese Weise kann die Steuerung zukünftig leicht um neue Geräte erweitert werden. Die Benutzeroberfläche wurde durch ein Renderer-Modul realisiert. Positionsdaten werden über einen MegaMol-Call an dieses Modul übergeben. Schaltflächen und Cursor werden mittels der Bibliothek NanoVG gezeichnet. Zum Parsen von Konfigurationen wird die Expat XML Parser-Bibliothek verwendet. Diese wird bereits in MegaMol eingesetzt, z. B. für Projektdateien.

Eigene Versuche haben gezeigt, dass die Fausthaltung schlechter von der Leap Motion erkannt wird, falls die Hand insbesondere weit nach oben oder auf die gegenüberliegende Seite bewegt wird. Direkt über dem Gerät ist eine Drehung um die Z-Achse nur eingeschränkt möglich. Generell gilt, das die Faust umso besser erkannt wird, je mehr das Gerät die Handfläche oder den Handrücken wahrnehmen kann. Ist der Daumen ausgestreckt, so konnten etwas häufiger Aussetzer beobachtet werden.

Die Kinect hat wesentlich unempfindlicher auf Drehungen um die Z-Achse reagiert, wobei die rechte Hand etwas schlechter abgeschnitten hat. Das Ausstrecken des Daumens hat die Erkennung verschlechtert. Trotz ausgestrecktem Daumen war die Erkennung problemlos möglich, falls die Handfläche parallel zur Kinect ausgerichtet worden ist. Die genannten Beobachtungen gelten jedoch nur für die Aktivierung der Kamerasteuerung. War diese dann aktiv, so konnten trotz Orientierungsänderungen und einem Ausstrecken des Daumens keine Aussetzer festgestellt werden.

Die Hand mit ausgestrecktem Zeigefinger wurde von der Leap Motion schlechter erkannt, falls der Zeigefinger nicht mehr vollständig vom Gerät wahrgenommen werden konnte. Dies hat sich beispielsweise gezeigt falls der Zeigefinger der linken Hand parallel zum Bildschirm und zum Gerät orientiert und dann nach rechts bewegt

worden ist. Eine Drehung um die Z-Achse hat die Erkennung nicht verschlechtert. Falls die Geste von beiden Händen ausgeführt worden ist, so ist bei einem Überlagern der Hände (auch nur teilweise) zu einem Ausfall des Cursors der oberen Hand gekommen.

Von der Kinect wurde eine flache Hand problemlos erkannt, falls diese nicht genau waagrecht orientiert war, sondern leicht angehoben worden ist. In der waagrechten Orientierung kam es sporadisch vor, das die Handhaltung als Faust interpretiert worden ist. Falls jedoch die Finger gespreizt worden sind und die Kamerasteuerung bereits aktiv war, so konnten auch bei einer waagrecht ausgerichteten Hand keine Aussetzer beobachtet werden. Falls mit der Kinect zwei Cursor gesteuert worden sind, so ist es bei einem Überlagern der Hände zu keinen Ausfällen gekommen, sofern stets darauf geachtet worden ist, das sich die Hände nicht zu Nahe kommen.

Gegen Ende der Arbeit wurde eine Benutzerstudie durchgeführt, in der 9 Teilnehmer die Gestensteuerung evaluiert haben. Die Steuerung wurde mit einer Hand und beiden Händen getestet. Zur Bewertung wurde stets eine 5-stufige Skala (1–5 Punkte) eingesetzt, bei der bis auf eine Ausnahme eine höhere Punktzahl ein besseres Ergebnis repräsentiert hat. Für die Leap Motion wurde die Zuverlässigkeit der Faust sowie Hand mit ausgestreckten Zeigefinger von den meisten Teilnehmern mit 4 Punkten bewertet. Unter anderem waren Aussetzer auf der linken Seite und oben zu beobachten. Alle Teilnehmer haben ihre rechte Hand eingesetzt. Die Interaktion mit zwei Händen wurde meist mit 3 bis 4 Punkten bewertet. Beim Steuern der beiden Cursor sind bei einigen Teilnehmern Aussetzer aufgetreten. Auch war manchmal ein Springen eines Cursors zu beobachten. Ein paar Teilnehmer haben die Hände überlagert wodurch es zu einem Ausfall eines Cursors bzw. der Kamerasteuerung gekommen ist. Für die Kinect wurde die Fausthaltung mit 5 Punkten bewertet. Es war bis auf eine Ausnahme kein Ausfall der Kamerasteuerung zu beobachten. Die flache Hand wurde meist nur mit 3 Punkten bewertet. Es waren zu Beginn häufig kurze Bewegungen notwendig um den Cursor erscheinen zu lassen. Als der Cursor dann gesteuert wurde, waren sporadisch Aussetzer feststellbar. Hierbei wurde die Hand waagrecht gehalten und die Finger waren nicht gespreizt. Die Interaktion mit zwei Händen wurde mit 4 Punkten bewertet. Wie zuvor waren sporadisch Bewegungen erforderlich um den Cursor erscheinen zu lassen. Ein Überlagern der Hände hat weitestgehend keine Probleme verursacht. Bezüglich der Zuverlässigkeit haben beide Geräte somit insgesamt ein ähnliches Ergebnis erzielt. Die Beobachtungen aus der Studie decken sich mit denen aus den eigenen Versuchen.

Das Zittern der Visualisierung wurde bei der Leap Motion mit 4 Punkten, bei der Kinect mit 3 Punkten bewertet. Für den Cursor waren die Ergebnisse 5 und 3 Punkte. Dementsprechend hat die Leap Motion besser abgeschnitten. Unabhängig von den

Implementierungen wurde die Fausthaltung meist mit 4 bis 5 Punkten bewertet. Die Hand mit ausgestrecktem Zeigefinger hat das selbe Ergebnis erzielt, für die flache Hand haben die meisten Teilnehmer mit 4 Punkten abgestimmt. Erstere Handhaltung wurde als intuitiver eingestuft. Auch sollte die Interaktion mit der Benutzeroberfläche unabhängig von den Geräten bewertet werden. Hier haben die Teilnehmer überwiegend 4 Punkte vergeben. Die Wartezeit, bis ein Klick erfolgt, wurde vorwiegend mit 3 Punkten bewertet. In diesem Fall war die Skala so orientiert, das die Mitte (3 Punkte) die optimale Zeit repräsentiert.

Im Ausstellungsszenario ist es aktuell erforderlich, dass die verfügbaren Gesten z. B. auf einem Blatt separat erläutert werden. Benutzerfreundlicher wäre es, wenn eine entsprechende Einführung direkt in MegaMol integriert wäre. Man könnte beispielsweise Animationen der einzelnen Gesten entwerfen, die in einer Ecke der Benutzeroberfläche eingeblendet werden. Auch wäre ein separates Tutorial denkbar, das Schritt für Schritt die Möglichkeiten der Steuerung erklärt.

Da die Benutzeroberfläche von mehreren Personen gleichzeitig genutzt werden kann, ist es wichtig, das jede Person klar erkennen kann, mit welchem Element sie gerade interagiert. Cursor können daher in unterschiedlichen Farben repräsentiert werden. Bei der Kinect könnte man zusätzlich noch ein Bild der entsprechenden Person einblenden und die jeweilige Hand in der Farbe des Cursors umranden. In [Pte14] werden Personen aus dem Bild der Farbkamera freigestellt. Hierfür werden die RGB-Daten und Tiefendaten kombiniert eingesetzt. Zur Erkennung der Handkonturen kann z. B. die Bibliothek [Pte16] eingesetzt werden. Weiterhin wäre es denkbar, Gestenindikatoren (Icons) auf der Benutzeroberfläche einzublenden, die angeben, welche Geste gerade von welchem Nutzer erkannt wird.

Nutzt man die Kinect, so erscheint der Cursor, wenn man die Hand öffnet und 10 cm über die Hüfte bewegt. Zukünftig könnte man noch weitere Bedingungen definieren, um sicherzustellen, das eine Interaktion nicht unbewusst erfolgt. Beispielsweise könnte man festlegen, das die Handfläche immer parallel zur Kinect ausgerichtet oder der Arm um einen bestimmten Prozentsatz nach vorne gestreckt sein muss. Für letzteres müsste man eine Kalibrierung einführen, in der die Armlänge des Nutzers ermittelt wird, was im Szenario einer Ausstellung jedoch nicht optimal wäre. Generell wäre es besser, wenn man den Bewegungsspielraum des Nutzers nicht eingeschränkt und stattdessen eine Geste definiert, die einmalig ausgeführt werden muss, damit mit der Interaktion begonnen werden kann. Denkbar wäre es z. B. ein Winken festzulegen. Positioniert sich eine neue Person vor dem Gerät, so erscheint eine Animation, die darauf hinweist, diese Geste auszuführen.

Wie zuvor erwähnt, löst ein Cursor eine Schaltfläche aus, wenn diese für eine bestimmte Zeit nicht verlassen wird. Die hierbei verwendete Cursor-Animation wird

als selbsterklärend eingestuft. Dennoch könnte es einzelne Nutzer geben, die auch nach mehreren Berührungen mit Schaltflächen nicht verstanden haben, wie der Klick umgesetzt wird. Zukünftig wäre es daher denkbar, noch einen entsprechenden Tooltip in die Benutzeroberfläche zu integrieren, der in einem solchen Fall eingeblendet wird. NanoVG stellt Funktionen bereit, mit denen (mehrzeilige) Textfelder definiert werden können.

In den meisten Fällen werden auf der Benutzeroberfläche Schaltflächen definiert, die zwischen mehrere Modi wechseln. In der Benutzerstudie gab es z. B. Schaltflächen, welche die Einfärbung des Moleküls verändert haben. Der selbe Parameter wurde dabei auf verschiedene Werte gesetzt. Es kann jedoch aktuell nicht verdeutlicht werden, in welchem Modus man sich gerade befindet. Denkbar wäre es daher, einen neuen Schaltflächentyp einzuführen (Toggle-Schaltfläche), der zwei Zustände besitzt (aktiv bzw. nicht aktiv) und abhängig von diesem unterschiedlich visualisiert ist. Alle Toogle-Schaltflächen, die den selben Parameter verändern, würden dann einer Gruppe zugewiesen werden. Innerhalb der Gruppe ist immer genau eine Schaltfläche im Zustand aktiv.

Zuletzt wäre es insbesondere für das Ausstellungsszenario denkbar, interessante Punkte (oder Bereiche) an der Visualisierung definieren zu können. Bewegt der Nutzer seinen Cursor auf einen solchen Punkt, so werden in einem Tooltip nähere Informationen eingeblendet und ggf. der entsprechende Teil der Visualisierung hervorgehoben.

A Anhang

A.1 XML Schema zur Konfiguration der Benutzeroberfläche

Konfigurationsdateien der Benutzeroberfläche (siehe Abschnitt 4.2) sind durch das in Listing A.1 dargestellte XML Schema definiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified
    " targetNamespace="http://megamol.org/nui" xmlns:nui="http://megamol.org/nui">
  <xs:element name="MegaMol">
    <xs:complexType>
      <xs:all>
        <xs:element name="Buttons" type="nui:tButtons" minOccurs="0"/>
        <xs:element name="Fonts" type="nui:tFonts" minOccurs="0"/>
        <xs:element name="Images" type="nui:tImages" minOccurs="0"/>
        <xs:element name="Cursors" type="nui:tCursors" minOccurs="0"/>
      </xs:all>
      <xs:attribute name="type" use="required" fixed="ui"/>
      <xs:attribute name="version" use="required" fixed="1.0"/>
    </xs:complexType>
    <xs:key name="fontID">
      <xs:selector xpath="nui:Fonts/nui:Font"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="fontIDRef" refer="nui:fontID">
      <xs:selector xpath="nui:Buttons/nui:Button/nui:Content/nui:FontIcon | nui:
          Buttons/nui:Button/nui:Content/nui:Label"/>
      <xs:field xpath="@font"/>
    </xs:keyref>
    <xs:key name="imageID">
      <xs:selector xpath="nui:Images/nui:Image"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="imageIDRef" refer="nui:imageID">
      <xs:selector xpath="nui:Buttons/nui:Button/nui:Content/nui:ImageIcon"/>
      <xs:field xpath="@image"/>
    </xs:keyref>
  </xs:element>
  <xs:complexType name="tButtons">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Button" type="nui:tButton"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="tButton">
    <xs:all>
      <xs:element name="RoundedCorners" type="nui:tRoundedCorners" minOccurs="0"/>
      <xs:element name="Backgrounds" type="nui:tBackgrounds"/>
      <xs:element name="Content" type="nui:tContent" minOccurs="0"/>
      <xs:element name="Actions" type="nui:tActions" minOccurs="0"/>
    </xs:all>
```

```
43      <xs:attribute name="x" use="required" type="nui:tAbsAndPercentPosDecimal"/>
44      <xs:attribute name="y" use="required" type="nui:tAbsAndPercentPosDecimal"/>
45      <xs:attribute name="align" type="nui:tAlign" default="bottomLeft"/>
46      <xs:attribute name="origin" type="nui:tOrigin" default="bottomLeft"/>
47      <xs:attribute name="width" use="required" type="nui:tAbsAndPercentPosDecimal"/>
48      <xs:attribute name="height" use="required" type="nui:tAbsAndPercentPosDecimal"/>
49    </xs:complexType>
50    <xs:simpleType name="tAlign">
51      <xs:restriction base="xs:string">
52        <xs:enumeration value="bottomLeft"/>
53        <xs:enumeration value="middleLeft"/>
54        <xs:enumeration value="topLeft"/>
55        <xs:enumeration value="topCenter"/>
56        <xs:enumeration value="topRight"/>
57        <xs:enumeration value="middleRight"/>
58        <xs:enumeration value="bottomRight"/>
59        <xs:enumeration value="bottomCenter"/>
60        <xs:enumeration value="middleCenter"/>
61      </xs:restriction>
62    </xs:simpleType>
63    <xs:simpleType name="tOrigin">
64      <xs:restriction base="xs:string">
65        <xs:enumeration value="bottomLeft"/>
66        <xs:enumeration value="topLeft"/>
67        <xs:enumeration value="topRight"/>
68        <xs:enumeration value="bottomRight"/>
69      </xs:restriction>
70    </xs:simpleType>
71    <xs:complexType name="tRoundedCorners">
72      <xs:attribute name="radius" use="required" type="nui:tAbsPosDecimal"/>
73    </xs:complexType>
74    <xs:complexType name="tBackgrounds">
75      <xs:all>
76        <xs:element name="Default" type="nui:tBackgroundType"/>
77        <xs:element name="Hover" type="nui:tBackgroundType" minOccurs="0"/>
78        <xs:element name="Click" type="nui:tBackgroundType" minOccurs="0"/>
79      </xs:all>
80    </xs:complexType>
81    <xs:complexType name="tBackgroundType">
82      <xs:choice minOccurs="1" maxOccurs="unbounded">
83        <xs:element name="Solid" type="nui:tSolidFill"/>
84        <xs:element name="LinearGradient" type="nui:tLinearGradientFill"/>
85        <xs:element name="RadialGradient" type="nui:tRadialGradientFill"/>
86      </xs:choice>
87    </xs:complexType>
88    <xs:complexType name="tSolidFill">
89      <xs:attribute name="color" use="required" type="nui:tHexColor"/>
90    </xs:complexType>
91    <xs:complexType name="tLinearGradientFill">
```

```
92    <xs:attribute name="startColor" use="required" type="nui:tHexColor"/>
93    <xs:attribute name="endColor" use="required" type="nui:tHexColor"/>
94  </xs:complexType>
95  <xs:complexType name="tRadialGradientFill">
96    <xs:attribute name="startColor" use="required" type="nui:tHexColor"/>
97    <xs:attribute name="endColor" use="required" type="nui:tHexColor"/>
98    <xs:attribute name="innerRadius" use="required" type="nui:tAbsPosDecimal"/>
99    <xs:attribute name="outerRadius" use="required" type="nui:tAbsPosDecimal"/>
100 </xs:complexType>
101 <xs:complexType name="tContent">
102   <xs:all>
103     <xs:element ref="nui:Icon" minOccurs="0"/>
104     <xs:element name="Label" type="nui:tText" minOccurs="0"/>
105   </xs:all>
106   <xs:attribute name="iconLabelGap" type="nui:tAbsPosDecimal" default="20"/>
107 </xs:complexType>
108 <xs:element name="Icon" abstract="true"/>
109 <xs:element name="FontIcon" substitutionGroup="nui:Icon" type="nui:tText"/>
110 <xs:element name="ImageIcon" substitutionGroup="nui:Icon" type="nui:tImage"/>
111 <xs:complexType name="tText">
112   <xs:attribute name="font" type="xs:string" use="required"/>
113   <xs:attribute name="size" type="nui:tAbsPosDecimal" use="required"/>
114   <xs:attribute name="color" type="nui:tHexColor" default="#FFFFFFFF"/>
115   <xs:attribute name="text" type="xs:string" use="required"/>
116 </xs:complexType>
117 <xs:complexType name="tImage">
118   <xs:attribute name="image" use="required" type="xs:string"/>
119 </xs:complexType>
120 <xs:complexType name="tActions">
121   <xs:sequence>
122     <xs:element name="SetParameterValue" type="nui:tSetParameterValueAction"
          minOccurs="0" maxOccurs="unbounded"/>
123   </xs:sequence>
124 </xs:complexType>
125 <xs:complexType name="tSetParameterValueAction">
126   <xs:attribute name="name" type="xs:string" use="required"/>
127   <xs:attribute name="value" type="xs:string" use="required"/>
128 </xs:complexType>
129 <xs:complexType name="tFonts">
130   <xs:sequence>
131     <xs:element name="Font" type="nui:tFile" minOccurs="0" maxOccurs="unbounded"/>
132   </xs:sequence>
133 </xs:complexType>
134 <xs:complexType name="tImages">
135   <xs:sequence>
136     <xs:element name="Image" type="nui:tFile" minOccurs="0"
137       maxOccurs="unbounded"/>
138   </xs:sequence>
139 </xs:complexType>
```

```
140   <xs:complexType name="tFile">
141     <xs:attribute name="id" type="xs:string" use="required"/>
142     <xs:attribute name="src" type="xs:anyURI" use="required"/>
143   </xs:complexType>
144   <xs:complexType name="tCursors">
145     <xs:sequence>
146       <xs:element name="Cursor" type="nui:tCursor" minOccurs="0" maxOccurs="
          unbounded"/>
147     </xs:sequence>
148     <xs:attribute name="radius" type="nui:tAbsPosDecimal" default="30"/>
149     <xs:attribute name="clamp" type="xs:boolean" default="true"/>
150   </xs:complexType>
151   <xs:complexType name="tCursor">
152     <xs:attribute name="remainingColor" type="nui:tHexColor" use="required"/>
153     <xs:attribute name="elapsedColor" type="nui:tHexColor" use="required"/>
154   </xs:complexType>
155   <xs:simpleType name="tAbsAndPercentPosDecimal">
156     <xs:restriction base="xs:string">
157       <xs:pattern value="([0-9]+\.?[0-9]*|[0-9]*\.[0-9]+)%?"/>
158     </xs:restriction>
159   </xs:simpleType>
160   <xs:simpleType name="tAbsPosDecimal">
161     <xs:restriction base="xs:decimal">
162       <xs:minInclusive value="0"/>
163     </xs:restriction>
164   </xs:simpleType>
165   <xs:simpleType name="tHexColor">
166     <xs:restriction base="xs:string">
167       <xs:pattern value="#([0-9a-fA-F]{3,4}|[0-9a-fA-F]{6}|[0-9a-fA-F]{8})"/>
168     </xs:restriction>
169   </xs:simpleType>
170 </xs:schema>
```

Listing A.1: XML Schema zur Konfiguration der Benutzeroberfläche

A.2 Konfiguration der Benutzeroberfläche für die Benutzerstudie

Listing A.2 zeigt die Konfiguration der Benutzeroberfläche, die für die Benutzerstudie (siehe Abschnitt 6.3) entworfen worden ist. Die Parameter, welche durch die Schaltflächen verändert werden, gehören zum Modul „MoleculeCartoonRenderer“. In der Projektdatei muss als Modulname „cartoonren“ definiert sein. Die Schriftart Open Sans wurde von Google Fonts[45] heruntergeladen. Die Vorlage für die Farbeimer-Icons stammt von Flaticon[46].

```
<?xml version="1.0" encoding="UTF-8"?>
<MegaMol type="ui" version="1.0" xmlns="http://megamol.org/nui" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://megamol.org/nui
    ui.xsd">
  <Buttons>
    <Button x="15" y="15" align="bottomLeft" origin="bottomLeft" width="310"
        height="130">
      <Backgrounds>
        <Default>
          <Solid color="#059C02FF"/>
          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
        </Default>
        <Hover>
          <Solid color="#08FB04FF"/>
          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
        </Hover>
        <Click>
          <Solid color="#70FD6DFF"/>
          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
        </Click>
      </Backgrounds>
      <Content iconLabelGap="20.0">
        <ImageIcon image="imgBucketLightGray"/>
        <Label font="opensans" size="65" text="LGray"/>
      </Content>
      <Actions>
        <SetParameterValue name="cartoonren::color::coloringMode1" value="BFactor"/>
      </Actions>
    </Button>
    <Button x="15" y="160" align="bottomLeft" origin="bottomLeft" width="310"
        height="130">
      <Backgrounds>
        <Default>
          <Solid color="#059C02FF"/>
          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
```

[45] https://fonts.google.com/specimen/Open+Sans
[46] http://www.flaticon.com/free-icon/bucket_137064

```
34        </Default>
35        <Hover>
36          <Solid color="#08FB04FF"/>
37          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
38        </Hover>
39        <Click>
40          <Solid color="#70FD6DFF"/>
41          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
42        </Click>
43      </Backgrounds>
44      <Content iconLabelGap="20.0">
45        <ImageIcon image="imgBucketOlive"/>
46        <Label font="opensans" size="65" text="Olive"/>
47      </Content>
48      <Actions>
49        <SetParameterValue name="cartoonren::color::coloringMode1" value="Chain"/>
50      </Actions>
51    </Button>
52    <Button x="15" y="160" align="topLeft" origin="topLeft" width="310"
53         height="130">
54      <Backgrounds>
55        <Default>
56          <Solid color="#059C02FF"/>
57          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
58        </Default>
59        <Hover>
60          <Solid color="#08FB04FF"/>
61          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
62        </Hover>
63        <Click>
64          <Solid color="#70FD6DFF"/>
65          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
66        </Click>
67      </Backgrounds>
68      <Content iconLabelGap="20.0">
69        <ImageIcon image="imgBucketStripes"/>
70        <Label font="opensans" size="65" text="Stripes"/>
71      </Content>
72      <Actions>
73        <SetParameterValue name="cartoonren::color::coloringMode1" value="Residue"/>
74      </Actions>
75    </Button>
76    <Button x="15" y="15" align="topLeft" origin="topLeft" width="310" height="130">
77      <Backgrounds>
78        <Default>
79          <Solid color="#059C02FF"/>
80          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
81        </Default>
82        <Hover>
```

```
83          <Solid color="#08FB04FF"/>
84          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
85        </Hover>
86        <Click>
87          <Solid color="#70FD6DFF"/>
88          <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
89        </Click>
90      </Backgrounds>
91      <Content iconLabelGap="20.0">
92        <ImageIcon image="imgBucketGray"/>
93        <Label font="opensans" size="65" text="Gray"/>
94      </Content>
95      <Actions>
96        <SetParameterValue name="cartoonren::color::coloringMode1" value="Element"/>
97      </Actions>
98    </Button>
99    <Button x="15" y="15" align="topRight" origin="topRight" width="310"
100         height="130">
101     <RoundedCorners radius="0"/>
102     <Backgrounds>
103       <Default>
104         <Solid color="#059C02FF"/>
105         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
106       </Default>
107       <Hover>
108         <Solid color="#08FB04FF"/>
109         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
110       </Hover>
111       <Click>
112         <Solid color="#70FD6DFF"/>
113         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
114       </Click>
115     </Backgrounds>
116     <Content iconLabelGap="20.0">
117       <ImageIcon image="imgBucketRainbow"/>
118       <Label font="opensans" size="65" text="Rainbow"/>
119     </Content>
120     <Actions>
121       <SetParameterValue name="cartoonren::color::coloringMode1" value="Rainbow"/>
122     </Actions>
123   </Button>
124   <Button x="15" y="15" align="bottomRight" origin="bottomRight" width="190"
          height="130">
125     <Backgrounds>
126       <Default>
127         <Solid color="#059C02FF"/>
128         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
129       </Default>
130       <Hover>
```

```
131         <Solid color="#08FB04FF"/>
132         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
133       </Hover>
134       <Click>
135         <Solid color="#70FD6DFF"/>
136         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
137       </Click>
138     </Backgrounds>
139     <Content iconLabelGap="20.0">
140       <Label font="opensans" size="65" text="Lines"/>
141     </Content>
142     <Actions>
143       <SetParameterValue name="cartoonren::renderingMode" value="Cartoon Lines"/>
144     </Actions>
145   </Button>
146   <Button x="220" y="15" align="bottomRight" origin="bottomRight" width="240"
          height="130">
147     <Backgrounds>
148       <Default>
149         <Solid color="#059C02FF"/>
150         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
151       </Default>
152       <Hover>
153         <Solid color="#08FB04FF"/>
154         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
155       </Hover>
156       <Click>
157         <Solid color="#70FD6DFF"/>
158         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
159       </Click>
160     </Backgrounds>
161     <Content iconLabelGap="20.0">
162       <Label font="opensans" size="65" text="Normal"/>
163     </Content>
164     <Actions>
165       <SetParameterValue name="cartoonren::renderingMode" value="Cartoon Hybrid"/>
166     </Actions>
167   </Button>
168   <Button x="340" y="15" align="bottomLeft" origin="bottomLeft" width="310"
          height="130">
169     <Backgrounds>
170       <Default>
171         <Solid color="#059C02FF"/>
172         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
173       </Default>
174       <Hover>
175         <Solid color="#08FB04FF"/>
176         <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
177       </Hover>
178
```

```
179         <Click>
180           <Solid color="#70FD6DFF"/>
181           <LinearGradient startColor="#FFFFFF78" endColor="#00000078"/>
182         </Click>
183       </Backgrounds>
184       <Content iconLabelGap="20.0">
185         <ImageIcon image="imgBucketBlueRed"/>
186         <Label font="opensans" size="65" text="BlueRed"/>
187       </Content>
188       <Actions>
189         <SetParameterValue name="cartoonren::color::coloringMode1"
190             value="Structure"/>
191       </Actions>
192     </Button>
193   </Buttons>
194   <Fonts>
195     <Font id="opensans" src="fonts/OpenSans-Regular.ttf"/>
196   </Fonts>
197   <Images>
198     <Image id="imgBucketRainbow" src="images/bucket-rainbow.png"/>
199     <Image id="imgBucketBlueRed" src="images/bucket-bluered.png"/>
200     <Image id="imgBucketLightGray" src="images/bucket-lightgray.png"/>
201     <Image id="imgBucketOlive" src="images/bucket-olive.png"/>
202     <Image id="imgBucketGray" src="images/bucket-gray.png"/>
203     <Image id="imgBucketStripes" src="images/bucket-stripes.png"/>
204   </Images>
205   <Cursors radius="40" clamp="true">
206     <Cursor remainingColor="#ffff3d" elapsedColor="#db0400"/>
207     <Cursor remainingColor="#fff" elapsedColor="#db0400"/>
208   </Cursors>
209 </MegaMol>
```

Listing A.2: Konfiguration der Benutzeroberfläche für die Benutzerstudie

Literaturverzeichnis

[Ash14] J. Ashley. *Quick Reference: Kinect 1 vs Kinect 2*. 5. März 2014. URL: http://www.imaginativeuniversal.com/blog/post/2014/03/05/quick-reference-kinect-1-vs-kinect-2.aspx (zitiert auf S. 28).

[Bor12] G. Borenstein. *FingerTracker*. 2012. URL: http://makematics.com/code/FingerTracker (zitiert auf S. 40).

[Che10] J. Chen. *Microsoft Xbox 360 Kinect Launches November 4*. Hrsg. von G. M. Group. 14. Juni 2010. URL: http://gizmodo.com/5563148/microsoft-xbox-360-kinect-launches-november-4 (zitiert auf S. 9, 24).

[Chr12] P. Christensson. *TechTerms – NUI Definition*. 26. Juli 2012. URL: http://techterms.com/definition/nui (zitiert auf S. 9).

[Coo99] C. Cooper. *Using Expat*. Hrsg. von O'Reilly Media, Inc. 1. Sep. 1999. URL: http://www.xml.com/pub/1999/09/expat (zitiert auf S. 66).

[DIN05] DIN Deutsches Institut für Normung e. V. *DIN 33402-2: Ergonomie – Körpermaße des Menschen – Teil 2: Werte*. Dez. 2005. URL: http://www.din.de/wdc-beuth:din21:84092742 (zitiert auf S. 76).

[Fit16] Fitnect Interactive. *3D Virtual fitting dressing room / mirror*. 2016. URL: http://www.fitnect.hu (zitiert auf S. 10).

[GKM+15] S. Grottel, M. Krone, C. Müller, G. Reina, T. Ertl. „MegaMol – A Prototyping Framework for Particle-Based Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 21.2 (Feb. 2015), S. 201–214. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2350479 (zitiert auf S. 13–17).

[Kis16] F. Kistler. „Full Body Interaction – Design, Implementation, and User Support". PhD dissertation. Augsburg University, Faculty of Applied Computer Science, 2016 (zitiert auf S. 19, 25).

[Kön14] P. König. *Zweite Kinect für Windows jetzt im Handel*. Hrsg. von Heise Medien. 16. Juli 2014. URL: https://www.heise.de/newsticker/meldung/Zweite-Kinect-fuer-Windows-jetzt-im-Handel-2261356.html (zitiert auf S. 24).

[Kra15] K. Krause. „Leap Motion: Qualität von Präzisionsgriffen“. Magisterarb. Technische Hochschule Mittelhessen, 8. Apr. 2015. URL: http://digdok.bib.thm.de/volltexte/2015/5026 (zitiert auf S. 9).

[Kur13] F. Kurzmaier. *Bewegungssteuerung Leap Motion im Test*. Hrsg. von Macwelt. 25. Juli 2013. URL: http://www.macwelt.de/produkte/Bewegungssteuerung-Leap-Motion-im-Test-8102108.html (zitiert auf S. 19).

[Lea12a] Leap Motion, Inc. *Leap Motion Invites Developers Worldwide to Imagine and Create the Future; 26,000 From 140+ Countries Apply*. 31. Juli 2012. URL: https://www.leapmotion.com/news/leap-motion-invites-developers-worldwide-to-imagine-and-create-the-future-26-000-from-140-countries-apply (zitiert auf S. 19).

[Lea12b] Leap Motion, Inc. *Leap Motion Unveils World's Most Accurate 3-D Motion Control Technology for Computing*. 21. Mai 2012. URL: https://www.leapmotion.com/news/leap-motion-unveils-world-s-most-accurate-3-d-motion-control-technology-for-computing (zitiert auf S. 19).

[Lea13] Leap Motion, Inc. *Leap Motion Launches World's Most Accurate 3-D Motion Control Technology for Computing*. 22. Juli 2013. URL: https://www.leapmotion.com/news/leap-motion-launches-world-s-most-accurate-3-d-motion-control-technology-for-computing (zitiert auf S. 9, 19).

[Lea14a] Leap Motion, Inc. *How Does the Leap Motion Controller Work?* 9. Aug. 2014. URL: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work (zitiert auf S. 20).

[Lea14b] Leap Motion, Inc. *Leap Motion Community – Thread „Form & Function 3D: Leap Motion Application for Undergraduate Anatomy Education“*. 29. Apr. 2014. URL: https://community.leapmotion.com/t/form-function-3d-leap-motion-application-for-undergraduate-anatomy-education/1047 (zitiert auf S. 33).

[Lea15a] Leap Motion, Inc. *Leap Motion Community – Thread „Dimensions Leap motion“*. 6. März 2015. URL: https://community.leapmotion.com/t/dimensions-leap-motion/2626 (zitiert auf S. 19).

[Lea15b] Leap Motion, Inc. *Leap Motion Community – Thread „Sample period / Frames frequency“*. 22. Aug. 2015. URL: https://community.leapmotion.com/t/sample-period-frames-frequency/3281 (zitiert auf S. 21).

[Lea16a] Leap Motion, Inc. *C++ Orion SDK Documentation*. 2016. URL: https://developer.leapmotion.com/documentation/cpp (zitiert auf S. 21–23, 69).

[Lea16b] Leap Motion, Inc. *C++ V2 SDK Documentation*. 2016. URL: https://developer.leapmotion.com/documentation/v2/cpp (zitiert auf S. 23).

[Lea16c] Leap Motion, Inc. *Leap Motion Support Article „What are the system requirements?“* 18. Dez. 2016. URL: https://support.leapmotion.com/hc/en-us/articles/223783668 (zitiert auf S. 19).

[Lea16d] Leap Motion, Inc. *Virtual Piano For Beginners*. 2016. URL: https://apps.leapmotion.com/apps/virtual-piano-for-beginners (zitiert auf S. 10).

[Lea17] Leap Motion, Inc. *Store Europa*. 2. Jan. 2017. URL: http://store-eur.leapmotion.com (zitiert auf S. 70).

[Loc15] T. Lock. *5 stunningly futuristic gaming technologies of 2015*. Hrsg. von TechSpartan.co.uk. 9. Sep. 2015. URL: http://www.techspartan.co.uk/features/futuristic-gaming-technologies (zitiert auf S. 10).

[Met16] Metrilus GmbH. *Metrilus Aiolos Finger Tracking*. 2016. URL: http://www.metrilus.de/blog/portfolio-items/aiolos (zitiert auf S. 40).

[MH13] S. Miles, R. Henderson. *Xbox One release date and everything you need to know*. Hrsg. von Pocket-lint. 8. Nov. 2013. URL: http://www.pocket-lint.com/news/121177-xbox-one-release-date-and-everything-you-need-to-know (zitiert auf S. 10, 24).

[Mia13] MiastoGier.pl. *Xbox One i Kinect - zdjęcia przedstawiające sprzęt od wewnątrz*. 22. Mai 2013. URL: http://www.miastogier.pl/wiadomosc,23042.html (zitiert auf S. 25).

[Mic15] Microsoft Corporation. *Kinect for Windows Product Blog – Microsoft to consolidate the Kinect for Windows experience around a single sensor*. 2. Apr. 2015. URL: https://blogs.msdn.microsoft.com/kinectforwindows/2015/04/02/microsoft-to-consolidate-the-kinect-for-windows-experience-around-a-single-sensor (zitiert auf S. 24).

[Mic16a] Microsoft Corporation. *MSDN Library – Kinect Body Tracking*. 2016. URL: https://msdn.microsoft.com/de-de/library/dn799273.aspx (zitiert auf S. 27, 60).

[Mic16b] Microsoft Corporation. *MSDN Library – Kinect Constants*. 2016. URL: https://msdn.microsoft.com/en-us/library/hh855368.aspx (zitiert auf S. 28).

[Mic16c] Microsoft Corporation. *MSDN Library – Kinect for Windows SDK Documentation*. 2016. URL: https://msdn.microsoft.com/de-de/library/dn799271.aspx (zitiert auf S. 25–27).

[Mic16d] Microsoft Corporation. *MSDN Library – Kinect for Windows Sensor Components and Specifications*. 2016. URL: https://msdn.microsoft.com/library/jj131033.aspx (zitiert auf S. 28).

[Mic16e] Microsoft Corporation. *MSDN Library – Skeletal Joint Smoothing White Paper*. 2016. URL: https://msdn.microsoft.com/en-us/library/jj131429.aspx (zitiert auf S. 29).

[Mic16f] Microsoft Corporation. *Windows Dev Center – Kinect-Hardware*. 2016. URL: https://developer.microsoft.com/de-de/windows/kinect/hardware (zitiert auf S. 25, 28).

[Mic16g] Microsoft Corporation. *Xbox Support*. 2016. URL: http://support.xbox.com (zitiert auf S. 24, 25).

[Mic17] Microsoft Corporation. *Store Deutschland*. 2. Jan. 2017. URL: https://www.microsoftstore.de (zitiert auf S. 70).

[Mon16] M. Mononen. *GitHub – NanoVG*. 2016. URL: https://github.com/memononen/nanovg (zitiert auf S. 30, 47).

[Neu14] A. Neupert. „Natürliche Benutzerschnittstellenkonzepte“. Magisterarb. Institut für Visualisierung und Interaktive Systeme, Universität Stuttgart, 6. Mai 2014. URL: http://digdok.bib.thm.de/volltexte/2015/5026 (zitiert auf S. 35).

[Plo12] N. Ploner. „Gestensteuerung für Powerwall-basierte Visualisierungen“. Diplomarbeit. Institut für Visualisierung und Interaktive Systeme, 9. Juli 2012 (zitiert auf S. 35).

[PP15] D. Pagliari, L. Pinto. „Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors“. In: *Sensors* 15.11 (2015), S. 27569–27589. ISSN: 1424-8220. DOI: 10.3390/s151127569. URL: http://www.mdpi.com/1424-8220/15/11/27569 (zitiert auf S. 28).

[Pte14] V. Pterneas. *Background removal using Kinect 2 (green screen effect)*. 11. Apr. 2014. URL: http://pterneas.com/2014/04/11/kinect-background-removal (zitiert auf S. 94).

[Pte16] V. Pterneas. *Finger Tracking using Kinect v2*. 24. Jan. 2016. URL: http://pterneas.com/2016/01/24/kinect-finger-tracking (zitiert auf S. 40, 94).

[Sar13] S. Sarkar. *Xbox One includes a new Kinect sensor*. Hrsg. von Polygon. 21. Mai 2013. URL: http://www.polygon.com/2013/5/21/4341770/xbox-one-kinect (zitiert auf S. 24).

[Sch13] T. Schlegel. *Multi-Touch – Interaktion durch Berührung*. Springer-Verlag, 2013. ISBN: 978-3-642-36112-8 (zitiert auf S. 9).

[Sir14a] C. Sirignano. *MSDN Forums – Thread „body.HandLeftState hand-state false positives“*. 2014. URL: https://social.msdn.microsoft.com/Forums/en-US/4ebc1cb6-694f-45d4-9e8a-36d63c8feec1/bodyhandleftstate-handstate-false-positives (zitiert auf S. 61).

[Sir14b] C. Sirignano. *MSDN Forums – Thread „How to reduce Jitter in Kinect v2.0 sdk“*. 2014. URL: https://social.msdn.microsoft.com/Forums/en-US/f3bc6904-d07a-43d5-8657-1ac8cd147f95/how-to-reduce-jitter-in-kinect-v20-sdk (zitiert auf S. 29).

[Sir14c] C. Sirignano. *MSDN Forums – Thread „Joint Smoothing code“*. 2014. URL: https://social.msdn.microsoft.com/Forums/en-US/045b058a-ae3a-4d01-beb6-b756631b4b42/joint-smoothing-code (zitiert auf S. 29).

[SLK15] H. Sarbolandi, D. Lefloch, A. Kolb. „Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect“. In: *CoRR* abs/1505.05459 (2015). URL: http://arxiv.org/abs/1505.05459 (zitiert auf S. 25).

[Sme14] R. Smeenk. *Kinect V1 and Kinect V2 fields of view compared*. 11. März 2014. URL: http://smeenk.com/kinect-field-of-view-comparison (zitiert auf S. 24, 28).

[Smi13] R. Smith. *Kinect Sesame Street TV & Nat Geo TV Get Season 2*. Hrsg. von TrueAchievements. 9. Jan. 2013. URL: https://www.trueachievements.com/n11923/kinect-sesame-street-tv--nat-geo-tv-get-season-2 (zitiert auf S. 11).

[Spa13] SparkFun Electronics. *Leap Motion Teardown*. Juni 2013. URL: https://learn.sparkfun.com/tutorials/leap-motion-teardown (zitiert auf S. 20).

[Ste12] S. Stegmüller. *Codeplex – Candescent NUI*. 23. Okt. 2012. URL: https://candescentnui.codeplex.com (zitiert auf S. 40).

[WBRF13] F. Weichert, D. Bachmann, B. Rudak, D. Fisseler. „Analysis of the Accuracy and Robustness of the Leap Motion Controller“. In: *Sensors* 13.5 (2013), S. 6380. ISSN: 1424-8220. DOI: 10.3390/s130506380. URL: http://www.mdpi.com/1424-8220/13/5/6380 (zitiert auf S. 20).

[ZMM+15] S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, E. Menegatti. „Performance evaluation of the 1st and 2nd generation Kinect for multimedia applications“. In: *2015 IEEE International Conference on Multimedia and Expo (ICME)*. Juni 2015, S. 1–6. DOI: 10.1109/ICME.2015.7177380 (zitiert auf S. 27).

Alle URLs wurden zuletzt am 10. 01. 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift