

# Bericht für Projekt 2 im Masterstudiengang Informatik

Christian Blank

9. Mai 2015

## 1 Einführung

Die bisherigen Interaktionsmöglichkeiten mittels Tastatur, Maus und zweidimensionalen Gesten sind zwar sehr gut für die Arbeit mit zweidimensionalen Inhalten geeignet, aber nicht für Inhalte, die auch in der dritten Dimension dargestellt werden. Neben der Arbeit mit virtuellen Inhalten können dreidimensionale Gesten auch für die Kommunikation oder die Arbeit mit realen Objekten verwendet werden (vgl. Mixed Reality und Computer Supported Collaborative Work).

### 1.1 Motivation und Ziele

Ziel der Arbeit in der Gruppe  $I^2E$  ist die Entwicklung einer Mixed-Reality-Anwendung zur kollaborativen Konstruktion von Produkten und der Evaluierung dieser Anwendung. Verschiedene Teilnehmer können dabei zusammen an einem Produkt arbeiten, über das Produkt diskutieren und sich austauschen, auch wenn sie räumlich von einander getrennt sind. Das Konzept der freien Bewegung um den Tisch und damit auch um das Modell ist ein wichtiger Bestandteil des Projektes  $I^2E$ . Der Nutzer hat die Möglichkeit, das Modell von allen Seiten zu betrachten. Dies legt nahe, dass auch seine Interaktion mit dem System möglichst frei sein muss. Dabei wird bewusst auf ein Mapping von 2D-Eingabegeräten auf den virtuellen Raum verzichtet und es wird stattdessen eine direkte Interaktion angeboten. Der Nutzer hat die Möglichkeit die virtuelle Szene durch Gesten und reale Objekte, die beispielsweise als Werkzeuge oder Bauteile eingesetzt werden können, zu verändern. Diese Objekte werden nachfolgend als Tangibles bezeichnet.

Die Interaktion mit virtuellen Objekten soll auf die gleiche Weise möglich sein, wie die Arbeit mit Tangibles. Nutzer können virtuelle Objekte verschieben, anheben und greifen. Für diesen Zweck werden physikbasierte Gesten verwendet. Als

---

zweite Möglichkeit der Nutzung von Gesten zur Interaktion wird eine semantische Interpretation von Gesten durchgeführt. Somit ist es möglich, einzelne virtuelle Objekte und die gesamte Szene zu skalieren oder zu rotieren. In Kombination mit den physikbasierten Gesten wird dem Nutzer eine große Möglichkeit zur Interaktion geboten. Komplexere Interaktionsformen werden durch den Einsatz einer Menüführung vereinfacht. Damit soll vermieden werden, dass ein Nutzer sich komplizierte Bewegungsfolgen merken und durchführen muss, um eine gewünschte Aktion auszuführen.

Für dieses Ziel wurde in Projekt 1 ein prototypisches Objekttracking umgesetzt und ein ausführliches Konzept für eine Gestenerkennung erstellt. Für die Gestenerkennung wurden bereits einige wichtige Grundlagen implementiert.

Die Skeletterkennung mit der Kinect wird in vielen wissenschaftlichen Arbeiten behandelt und funktioniert unter guten Laborbedingungen zuverlässig. Für genauere Operationen wird jedoch eine gute Handerkennung benötigt, welche mit der Kinect problematisch zu realisieren ist. Aus diesem Grund wird ein zusätzliches Eingabegerät benötigt, mit dem die Hände der Nutzer erkannt werden können. Die Leap Motion bietet solch eine Funktionalität und arbeitet mit einem optischen Verfahren. Jedoch wird bei Devices, die sich auf die Erkennung der Hände spezialisiert haben, das restliche Skelett nicht ermittelt und neben den Händen im besten Fall noch die Ellenbogen zur Verfügung gestellt. Ideal wäre die gleichzeitige Verwendung einer Kinect oder einem Gerät mit ähnlicher Funktionalität zusammen mit einer Leap Motion. Für die Verarbeitung dieses Multisensor-Inputs wird die Sensorabstraktion Trame verwendet.

Die Abstraktionsschicht Trame ist in der Entwicklung weit fortgeschritten und unterstützt bereits verschiedene Sensoren (Leap Motion, Microsoft Kinect for Windows, Primesense Carmine). Neben einer prototypischen Implementierung in C++ steht auch eine Implementierung in C# zur Verfügung.

Das Ziel von Projekt 2 ist die Auslagerung von Trame und der Objekterkennung in eigene Services und der Umsetzung der physikbasierte Gesten in der Gestenerkennung. Ebenso soll intensiv an der Gestenerkennung und den dafür benötigten Templates gearbeitet werden.

## 1.2 Aufbau

Der Bericht ist in vier Abschnitte unterteilt. In Abschnitt 1 wird kurz die Motivation angesprochen und es wird das Ziel von Projekt 2 vorgestellt. In Abschnitt 2 wird genauer auf die Umsetzung der Konzepte aus Projekt 1 eingegangen und der Stand zum Projektende ist in Abschnitt 3 zu finden. Das abschließende Fazit und ein Ausblick auf die folgende Masterarbeit wird in Abschnitt 4 gegeben.

## 2 Umsetzung

In diesem Abschnitt wird beschrieben, wie die einzelnen Komponenten, die in Projekt 2 bearbeitet wurden, umgesetzt sind. Auf das Objektracking wird an dieser Stelle nicht weiter eingegangen, da es bereits in Projekt 1 fertiggestellt und in Projekt 2 nur in einen Service ausgelagert wurde. Dieser Service wurde mithilfe der Network-Middleware von Malte Eckhoff umgesetzt.[Ec14]

### 2.1 Trame

Es wurde weiter an der Abstraktionsschicht **Trame**<sup>1</sup> gearbeitet. Trame basiert auf der Idee, dass eine Gestenerkennung nicht fest an einen spezifischen Sensor gekoppelt sein sollte[Ei03]. Die Abstraktionsschicht kommt nun nicht nur für die Gestenerkennung zum Einsatz, sondern wird auch für das Headtracking verwendet. Auch Jonathan Wischhusen benutzt Trame in seiner Bachelorarbeit und entwickelt aktiv an dem Konzept und dem Modell mit. Aufgrund der gewonnenen Erfahrungen konnten weitere Funktionalitäten eingebunden und Fehler behoben werden.

Das Headtracking wird von Raimund Wege verwendet, um die virtuellen Kameras für sein mobiles Display korrekt in der Szene platzieren zu können [We14]. Um die aktuelle Kopfposition des Nutzers zu erhalten muss man sich auf den Trame-Service einschreiben. Mithilfe der *RegisterForTrameMessage* kann sich im Service registriert werden. Der Subscriber erhält anschließend in regelmäßigen Abständen *SkeletonMessages*. In der *SkeletonMessages* ist das komplette Skelett von Trame abgebildet. Um die Kopfposition zu erhalten muss in dem Skelettbau nach dem Joint mit dem Type 21200 gesucht werden. Da der Baum anhand dieses Merkmals aufgebaut ist, kann dies sehr performant gelöst werden.

#### 2.1.1 Serialisierung

Für die Auslagerung wurde zunächst an der Serialisierung gearbeitet. Es ist nun möglich Skelette zu serialisieren und zu deserialisieren. Dabei stehen verschiedene Zielformate zur Auswahl.

- JSON
- Protobuf
- C#-Objektserialisierung

Die Zeit, die Trame für eine Serialisierung benötigt, ist aufgrund der einfachen Struktur sehr gering. Messungen haben ergeben, dass 8.928 Skelettobjekte pro Sekunde serialisiert werden können. Das bedeutet, dass die Serialisierung 0,112 ms in

<sup>1</sup> Das Git-Repository von Trame befindet sich unter <https://intergitlab.informatik.haw-hamburg.de/christian/trame>

---

Anspruch nimmt. Die Standardabweichung wurde bisher nicht ermittelt. Ist aber aus den gemessenen Daten leicht zu errechnen.

### 2.1.2 Service

Trame wurde in einen eigenen Service<sup>2</sup> eingebunden und das Modell, TrameSkeleton<sup>3</sup>, ist nun eine eigenständiges Modul, das ohne Probleme in verschiedene Projekte eingebunden werden kann. Abhängigkeiten zu externen Libraries, bspw. für die Sensoren, müssen deshalb nicht aufgelöst werden, da das Modul TrameSkeleton nur von einer Geometry-Library abhängig ist, die Operationen für Matrizen, Vektoren und Quaternionen bereitstellt (Microsoft XNA Geometry). Dadurch kann das Modell ohne Unity oder anderen eigenen Projekten genutzt werden und die implementierten Algorithmen zur Suche von Skelettpunkten oder zum Glätten von Skelett-Streams stehen ebenfalls zur Verfügung. Die einzige weitere Anforderung ist die Sprache C# mit einer .NET-Version  $\geq 3.5$  oder einer kompatiblen Alternativimplementierung.

Trame besitzt nun auch eine Webservicekomponente und kann somit auch Daten über HTTP streamen. Für die Umsetzung wurde ein simpler HTTP-Server erstellt. Dieser Ansatz war jedoch nur eine Konzeptstudie und wird nicht weiter verfolgt. Die aktuelle Ausrichtung ist klar auf den Service in Verbindung mit der Middleware ausgerichtet.

### 2.1.3 Verschneidung von Sensordaten

Ein wichtiger Anwendungsfall für Trame wurde mit der Verbindung von Daten aus unterschiedlichen Sensoren erreicht. Ein Beispiel dafür ist in Abbildung 1 zu sehen. Das Skelett ist mit Trame erstellt und durch den *skeleton-viewer* visualisiert worden. Für die Vereinigung von verschiedenen Sensordaten werden die Daten zunächst in das gleiche Koordinatensystem umgerechnet und anschließend auf das Skelettmodell von Trame angewendet. Zu den Körperinformationen der Kinect können die Informationen zu den Händen, die durch die Leap Motion ermittelt wurden, hinzugefügt werden. Dabei muss aufgrund der relativen Koordinaten nicht auf die Position der Leap Motion geachtet werden. Die Orientierung der Joints für die Finger wird auf Basis der Handorientierung des Skelettes der Kinect ermittelt.

---

<sup>2</sup> <https://intergitlab.informatik.haw-hamburg.de/christian/trame/tree/master/csharp/TrameService>

<sup>3</sup> <https://intergitlab.informatik.haw-hamburg.de/christian/trame/tree/master/csharp/TrameSkeleton>

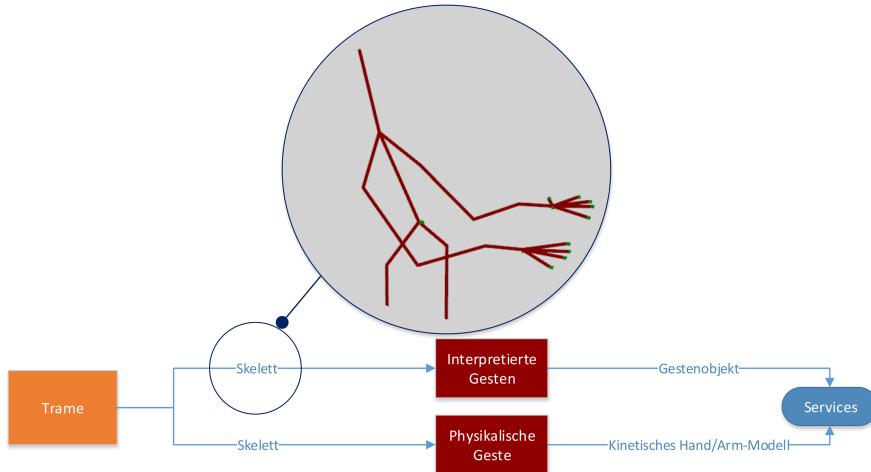


Abb. 1: Übersicht des Moduls für gestenbasierte Interaktion mit Device-Abstraktions und Gestenerkennung.

## 2.2 Physikbasierte Gesten

Um eine möglichst natürliche Interaktion mit virtuellen Objekten zu ermöglichen wurden physikbasierte Gesten eingebunden. Mithilfe von physikbasierten Gesten können Objekte auf gewohnte Weise bewegt werden. Ziel ist es, dass Nutzer virtuelle Objekte heben, greifen, verschieben und stoßen können. Abbildung 2 zeigt einen möglichen Usecase. Ein virtuelles Objekt wird auf einem Tisch verschoben und verhält sich dabei realistisch. Trifft die Hand auf das reale Objekt, werden beide Objekte mit der selben Bewegung verschoben.

Um die Bewegung der Arme und der Hand in einer Physiksimulation nutzen zu können, wird aus der Bewegung eines Nutzers ein Colliderobjekt erstellt, welches anschließend, etwa von Unity, verwendet werden kann. Eine Collider wird dabei auf der Basis von drei aufeinander folgenden Skeletten berechnet und enthält Parameter für die Position, Rotation, Winkelgeschwindigkeit und -beschleunigung sowie lineare Geschwindigkeit und lineare Beschleunigung jedes zu simulierenden Knochens.

Bevor die Simulation beginnt, wird das Skelett instantiiert. Jeder Knochen erhält einen Identifier und wird durch seine Position, Rotation, Länge und seinen Durchmesser beschrieben. Die Masse für die Simulation in Unity wird auf 1 gesetzt. Die Position und Rotation wird anhand des Skelettes aus Trame bestimmt. Die Länge und der Durchmesser der Knochen bzw. Körperteile wurde aus einer Studie [JB04] zur Vermessung des menschlichen Körpers übernommen.

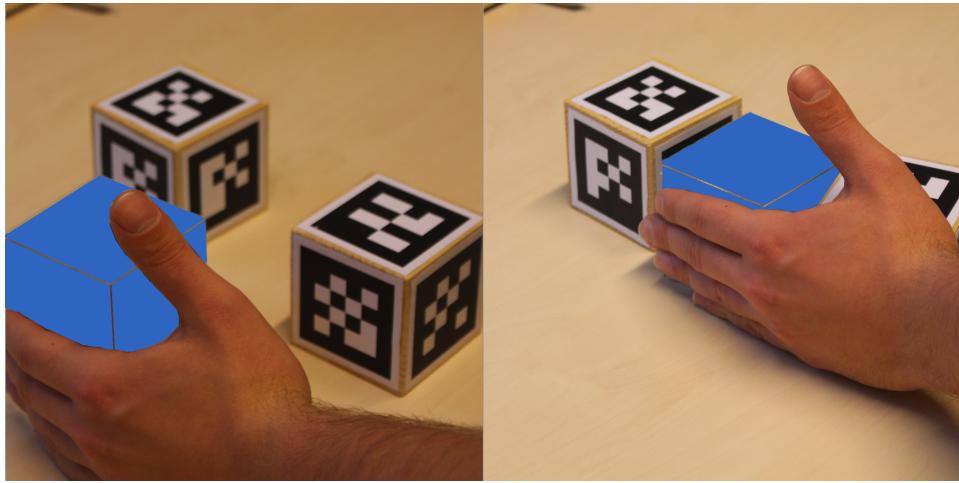


Abb. 2: Mockup für die Arbeit mit physikalischen Gesten. Das virtuelle Objekt (blauer Würfel) kann mit der Hand verschoben werden und verhält sich so, wie die realen Würfel.

### 2.3 Interpretierte Gesten

Nicht alle Interaktionen können über physikalische Gesten abgebildet werden. So ist es mit physikalischen Gesten nicht möglich ein Objekt beliebig zu skalieren oder Aktionen zu bestätigen. Eine Rotation eines virtuellen Objektes ist auch mit physikbasierten Gesten möglich, kann aber nicht so genau ausgeführt werden. Ein Mockup für die Rotation eines Objektes mit einer interpretierten Geste ist in Abbildung 3 zu sehen.



Abb. 3: Ein Beispiel für die Arbeit mit interpretierten Gesten. Die virtuelle Welle im Bild wird durch eine Geste rotiert.

Für die Erkennung der Geste wird ein Templatematching, das auf den Algorithmus von Kristensson [KNQ12] aufbaut, verwendet.

#### 2.3.1 Gestentemplates

Der Algorithmus des Templatematchings wird um die Einbeziehung der dritten Dimension und eine Erweiterung der Eingabepunkte erweitert. Ein Template besteht

dabei aus einer Abfolge von Punkten auf einer Ebene. Jeder Joint des Skelettes kann dabei eine eigene Abfolge haben. Die maximale Länge einer Abfolge sind drei Punkte. Das bedeutet eine minimale Verzögerung von 100ms für eine Geste. Dieser Wert wird in der Literatur als interaktiv anerkannt. Eine längere Geste setzt sich dann aus vielen Einzelgesten zusammen.

Die Templates werden in einem relativen, zweidimensionalen euklidischem Koordinatensystem definiert (0 - 1). Ein Beispiel dafür ist das in Abbildung 4 dargestellte Template zum Stauchen eines Objektes. Jede Hand besitzt in der Abbildung ihr eigenes Koordinatensystem. Die orangen Punkte und ihre Abfolge (1., 2., 3.) lassen erkennen, dass sich die beiden Hände aufeinander zu bewegen. Aus dem gezeigten Beispiel geht nicht hervor, welcher Joint für den Vergleich verwendet werden soll. In der Templateimplementierung ist dort der entsprechende Identifier hinterlegt. Die aktuelle Umsetzung erfordert die programmatische Definition eines Templates. Auf Dauer wäre eine Definition per DSL oder mittels eines grafischen Tools wünschenswert.

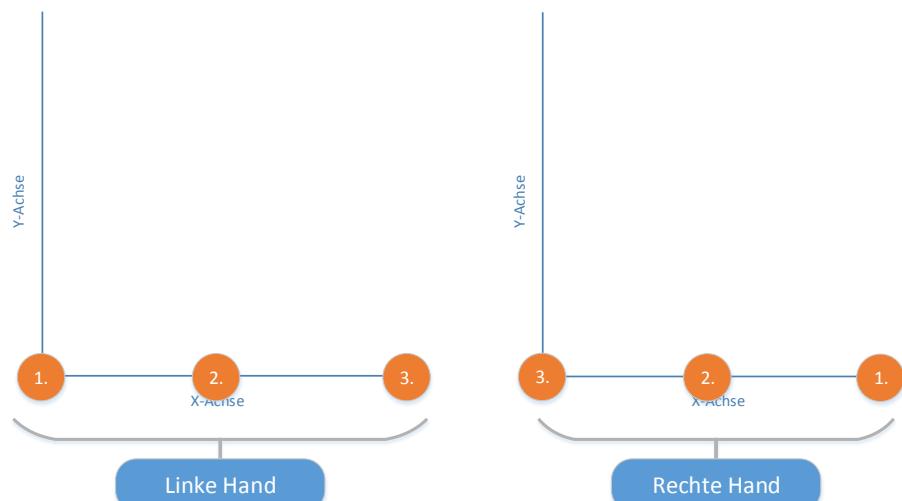


Abb. 4: Die grafische Darstellung eines Gestentemplates zum Stauchen eines virtuellen Objektes. Die Bewegung findet nur auf einer Linie statt.

Da das Template zweidimensional, aber die erlaubte Bewegung frei im dreidimensionalen Raum ist, muss eine Regelung für den Umgang mit der dritten Dimension gefunden werden. Die Ebene im Raum ist beliebig und wird durch den zweiten Punkt in der Abfolge bestimmt. Alle Punkte werden anschließend als Projektion auf diese Ebene betrachtet. Dadurch kann ein Template genutzt werden um eine Skalierung in eine Richtung zu ermöglichen.

---

## 3 Stand

Im Folgenden wird der aktuelle Entwicklungsstand für die einzelnen Komponenten aufgezeigt und es wird auf Probleme eingegangen, die sich bei der Entwicklung ergeben haben.

### 3.1 Trame und TrameService

In seiner aktuellen Entwicklung ist Trame und der TrameService stabil und Programmabstürze treten nicht mehr auf. In mehreren Testläufen konnte ohne Probleme mit dem Service gearbeitet werden, ohne ihn neu starten zu müssen. Der Service erkennt korrupte Verbindungen selbstständig und schließt diese über die Middleware. Allerdings liefert der Kinectsensor in manchen Situationen nach etwa 2700 Einzelframes keine weiteren Daten. Dieses Problem tritt unabhängig von dem verwendeten Computersystem und Sensor auf. Möglicherweise liegt der Fehler an dem verwendeten Treiber oder einer falschen Verwendung der API. Das Update auf eine neuere Version ist geplant und wird nicht viel Zeit in Anspruch nehmen, da die APIs der SDK-Versionen in vielen Bereichen kompatibel sind.

Im Verlauf der Zusammenarbeit mit Jonathan Wischhusen wurde das Modell des Skelettes angepasst um besser mit grafischen Umgebungen zusammen zu arbeiten. Zuvor wurden die Normalen des Gelenkes zur Rotation verwendet. In neueren Versionen verwendet Trame die absolute Orientierung des Gelenkes und kodiert diese als Quaternion.

Um nicht an Unity gebunden zu sein, wurde ein weiteres Projekt aufgesetzt, dass Bewegungen und Gesten von Trame respektive der Gestenerkennung darstellt und Daten zur Analyse bereitstellt, zu sehen in Abbildung 5. Die Visualisierung ist in einem frühen Stadium der Entwicklung und wird nur sporadisch weiterentwickelt. Aufgrund der Architektur des Systems und der Middleware kann die Visualisierung aber parallel zur Unity-Instanz mit der Gestenerkennung kommunizieren. Dadurch ist eine schnelle Auswertung der Ereignisse leicht umzusetzen, ohne dabei auf das aufwändige Debugging in Unity angewiesen zu sein.

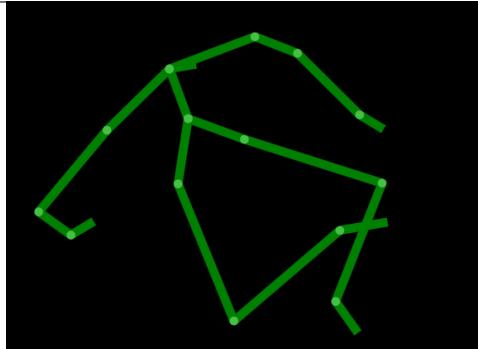


Abb. 5: Ausschnitt aus der Gestenvisualisierung. Die Person sitzt mit ausgebreiteten Armen und übereinander geschlagenem Bein vor dem Sensor auf einem Stuhl. Die Szene wird von der Decke aus betrachtet.

Für den TrameService wurde ein einfaches Konsolenframework<sup>4</sup> geschrieben, mit dem verschiedene Layouts schnell umgesetzt werden können. In Abbildung 6 ist ein Screenshot des Services zu sehen. Der Service sendet nur Nachrichten, wenn er ein neues Skelett erstellt hat und sich Applikationen bei ihm eingeschrieben haben.

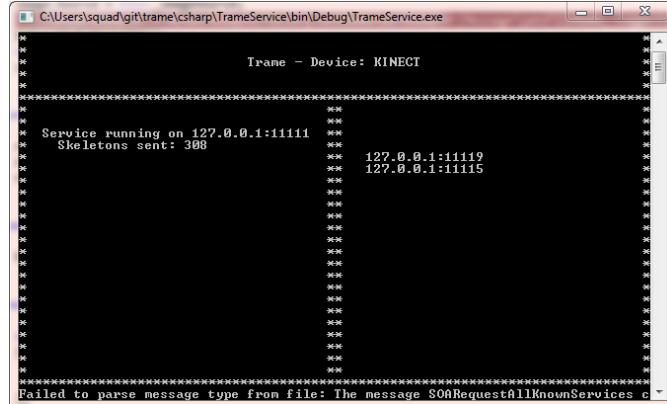


Abb. 6: Ansicht des TrameService als Konsolenanwendung. Im Kopfbereich wird angezeigt, welcher Adapter gerade genutzt wird. In diesem Fall ist es der Kinectadapter. Links wird die eigene Adresse mit IP und Port angegeben und es wird angezeigt, wie viele Nachrichten bereits versendet wurden. Rechts werden alle Subscriber mit IP und Port aufgelistet.

### 3.2 Gestensteuerung

Die Gestenerkennung besteht aus zwei unabhängigen Pipelines, wie in Abbildung 1 zu sehen ist. Die Pipeline für die physikalische Interaktion wurde zuerst bearbeitet und stellt Collider zur Verfügung. Die Rotation dieser Collider hat sich jedoch als

<sup>4</sup> <https://github.com/1blankz7/CoolKidsConsole>

---

fehleranfällig herausgestellt. In Abbildung 7 kann man sehen, dass die Repräsentanten für die Körperteile nicht korrekt rotiert werden. Der Fehler wird wahrscheinlich an einer der Rotationsmatrizen liegen, die für die Umrechnung der Koordinatensysteme verwendet wurden. Trotz des Fehlers ist es bereits möglich, mit virtuellen Objekten zu interagieren und bspw. einen Turm aus Würfeln umzustoßen.

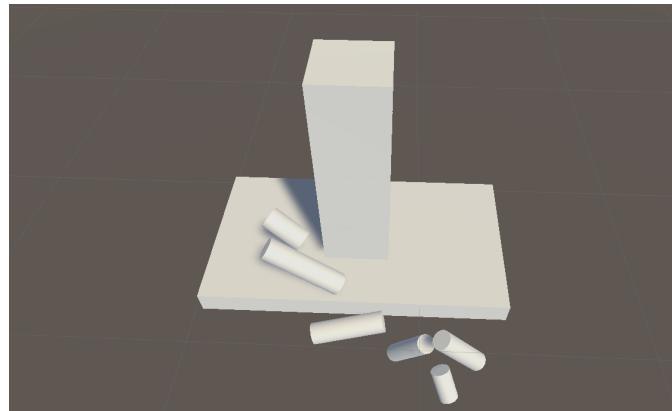


Abb. 7: Die Visualisierung in Unity stellt den Arbeitstisch mit einem Turm aus drei Würfeln mit einer Kantenlänge von 0,3m dar. Der linke Arm bewegt sich auf den Turm zu und die Hand kann den untersten Stein verschieben.

Die Umsetzung des Templatematching beschränkt sich derzeit noch auf die Implementierung des Basisalgorithmus und der Verwendung von einem einfachen Template. Da nun feststeht, wie die Templates definiert werden sollen und welche Gesten zunächst unterstützt werden, können die noch fehlenden Templates in kurzer Zeit erstellt werden. Im Folgenden werden die unterstützten Gesten aufgeführt.

- Zoom - vergrößert eine Szene
- Pinch - verkleinert die Szene wieder
- Rotate - lässt die gesamte Szene rotieren
- Select - ermöglicht die Auswahl eines oder mehrerer Objekte oder einer Aktion
- Confirm - bestätigt eine Aktion
- Dismiss - löscht eine Aktion

Es wird zwischen Aktionen, Objekten und der Szene unterschieden. Die Szene ist zusammengesetzt aus allen virtuellen Objekten und kann in ihrer Betrachtungsgröße und ihrer Rotation geändert werden. Einzelne Objekte in der Szene können ausgewählt werden, um nähere Informationen über sie zu erfahren. Aktionen können komplexere Operationen, wie das Starten einer Simulation sein.

### 3.3 Allgemein

Die Gruppe hat sich dafür entschieden, das Weltkoordinatensystem zu ändern. Das neue Koordinatensystem ist das gleiche System, dass auch von Unity und der PCL verwendet wird. Die anstehenden Änderungen sind noch nicht vollständig umgesetzt.

## 4 Fazit

Vergleicht man die Ziele aus Abschnitt 1 und die Ergebnisse aus 3 fällt auf, dass nicht alle Ziele im Rahmen von Projekt 2 erreicht werden konnten. Das lag zum einen an dem unerschätzten Aufwand, der in die Lösung einiger auftretender Probleme investiert werden musste und auch an einigen Stolpersteinen, die zu Anfang nicht ersichtlich waren. Zum anderen wurden zusätzliche Funktionalitäten entwickelt, die zu Beginn von Projekt 2 noch nicht geplant waren. Es ist wahrscheinlich, dass durch die Nutzung dieser Funktionen im weiteren Verlauf jedoch Zeit eingespart werden kann.

### 4.1 Ausblick

Mit den aktuellen Ergebnissen aus Projekt 2 kann weiter an der Gestenerkennung gearbeitet werden. Das Ziel ist es, dieses Modul prototypisch umzusetzen und anschließend die eigene Masterarbeit zu beginnen. Die Ausrichtung der Masterarbeit soll einen klaren Fokus auf die Usabilitystudie der Gestenerkennung besitzen.

Trame wird derzeit noch in einem privaten Git-System genutzt. Zum Abschluss des nächsten Semesters soll es als Opensource-Software veröffentlicht werden. Zuvor müssen aber noch einige Kinderkrankheiten entfernt werden und die Struktur des Projektes muss überarbeitet werden.

Um die Gestenerkennung in einer Studie verwenden zu können, müssen die beiden Teilbereiche vollständig implementiert und ausgiebig getestet werden. Für die interpretierten Gesten sollen weitere Templates erstellt werden und der Algorithmus muss an das eigene Konzept angepasst werden. Für die Umsetzung wird sich an Elementen der funktionalen Programmierung bedient (Pure Functions), die eine gute Testbarkeit von Algorithmen ermöglichen. Für die Collider, die bei der physikalischen Interaktion zum Einsatz kommen, müssen die Rotationen korrekt berechnet und es muss Unity als mögliche Fehlerquelle ausgeschlossen werden.

### 4.2 Risiken

Einige der in den bisherigen Ausarbeitungen genannten Risiken sind in Projekt 2 eingetreten. So gab es immer wieder Schwierigkeiten bei der Verfügbarkeit externer Komponenten. Viele Probleme traten auf, weil nicht genügend über Details

---

gesprochen wurde. Als Beispiel sind dort die unterschiedliche Interpretation der Maßeinheiten zu nennen, die in den Nachrichten genutzt wird (Meter oder Millimeter).

Neue Risiken konnten während der Bearbeitung des Projektes jedoch nicht festgestellt werden. Zudem wurden einige Arbeiten erledigt, die beispielsweise vor einer großen Abhängigkeit durch externe Komponenten schützen sollen und die technische Umsetzung nicht beeinträchtigen.

## Literaturverzeichnis

- [Ec14] Eckhoff, Malte: Middleware - Mixed Reality. Bericht, HAW Hamburg, Immersive Interactive Environments, 2014.
- [Ei03] Eisenstein, Jacob; Ghandeharizadeh, Shahram; Golubchik, Leana; Shahabi, Cyrus; Yan, Donghui; Zimmermann, Roger: Device independence and extensibility in gesture recognition. In: Virtual Reality, 2003. Proceedings. IEEE. IEEE, S. 207–214, 2003.
- [Jb04] Jürgens, H. W.: Erhebung anthropometrischer Maße zur Aktualisierung der DIN 33 402. Teil 2. Forschungsbericht, Bundesanstalt für Arbeitsschutz und Arbeitsmedizin, Postfach 17 02 02 44061 Dortmund, 2004.
- [KNQ12] Kristensson, Per Ola; Nicholson, Thomas; Quigley, Aaron: Continuous Recognition of One-handed and Two-handed Gestures Using 3D Full-body Motion Tracking Sensors. In: Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces. IUI '12, ACM, New York, NY, USA, S. 89–92, 2012.
- [We14] Wege, Raimund: Mobile Wrapped Prototyping for Mixed-Realities. Bericht, HAW Hamburg, Immersive Interactive Environments, 2014.

## Abbildungsverzeichnis

1	Übersicht des Modules für gestenbasierte Interaktion mit Device-Abstraktions und Gestenerkennung. . . . .	5
2	Mockup für die Arbeit mit physikalischen Gesten. Das virtuelle Objekt (blauer Würfel) kann mit der Hand verschoben werden und verhält sich so, wie die realen Würfel. . . . .	6
3	Ein Beispiel für die Arbeit mit interpretierten Gesten. Die virtuelle Welle im Bild wird durch eine Geste rotiert. . . . .	6
4	Die grafische Darstellung eines Gestentemplates zum Stauchen eines virtuellen Objektes. Die Bewegung findet nur auf einer Linie statt. . . . .	7

5	Ausschnitt aus der Gestenvisualisierung. Die Person sitzt mit ausgebreiteten Armen und übereinander geschlagenem Bein vor dem Sensor auf einem Stuhl. Die Szene wird von der Decke aus betrachtet. . . . .	9
6	Ansicht des TrameService als Konsolenanwendung. Im Kopfbereich wird angezeigt, welcher Adapter gerade genutzt wird. In diesem Fall ist es der Kinectadapter. Links wird die eigene Adresse mit IP und Port angegeben und es wird angezeigt, wie viele Nachrichten bereits versendet wurden. Rechts werden alle Subscriber mit IP und Port aufgelistet. . . . .	9
7	Die Visualisierung in Unity stellt den Arbeitstisch mit einem Turm aus drei Würfeln mit einer Kantenlänge von 0,3m dar. Der linke Arm bewegt sich auf den Turm zu und die Hand kann den untersten Stein verschieben. . . . .	10