HOCHSCHULE
ANHALT University
of Applied Sciences

FB6 - EMW
Fachbereich Eletrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen
Campus Köthen

Hausarbeit

zum Berufspraktikum

Jincheng Li

Vorname Nachname

Maschinenbau, 2018, 4067849

Studiengang, Matrikel, Matrikelnummer

Thema:
Virtual PlayDoh: Knete in VR simulieren

Prof. Dr. Tümler Johannes

Hochschulmentor(in)

03.11.2022 – 09.03.2023

Berufspraktikum von - bis

20. 04. 2023

Abgabe am

Kurzfassung

Virtual Reality (VR) hat das Potenzial, unsere physischen Erfahrungen zu erweitern und uns in virtuelle Welten zu versetzen. Eine interessante Anwendung von VR ist die Simulation von Knete - ein Material, das oft mit kreativem Spielen und kindlicher Entwicklung in Verbindung gebracht wird. Virtual PlayDoh es den Benutzern ermöglicht, digitale Knete zu formen und zu manipulieren. Das Programm verwendet spezielle Controller, die die Bewegungen und Gesten des Benutzers verfolgen und in Echtzeit auf den virtuellen Raum übertragen. Der Benutzer kann die Knete formen, ziehen, drücken und drehen, um verschiedene Formen und Texturen zu erstellen.

Eine der großen Vorteile von Virtual PlayDoh ist, dass es keine Unordnung gibt, die mit der Verwendung von echter Knete verbunden ist. Der Benutzer kann seine Kreationen speichern, bearbeiten und später wiederherstellen. Darüber hinaus kann der Benutzer verschiedene Farben und Texturen auswählen, um seine Kreationen noch weiter zu personalisieren.

Virtual PlayDoh hat auch Anwendungen im Bildungsbereich. Kinder können mit der Anwendung kreativ sein und gleichzeitig ihre feinmotorischen Fähigkeiten verbessern. Lehrer können die Anwendung nutzen, um Konzepte wie Formen, Textur und Farben zu demonstrieren.

Insgesamt bietet Virtual PlayDoh eine interessante und unterhaltsame Möglichkeit, die Vorteile von VR zu nutzen und die Kreativität und die feinmotorischen Fähigkeiten zu fördern.

Danksagung

Vielen Dank an Professor Johannes Tümler für die Gelegenheit, dieses interessante Thema zu erforschen, und für seine wertvolle Anleitung und Unterstützung während des gesamten Prozesses.

Mein besonderer Dank gilt auch meinen Kollegen im Labor, deren Gesellschaft und Hilfe die treibende Kraft hinter meiner Beharrlichkeit waren.

Ich möchte auch meiner Familie und meinen Freunden danken, die mich in dieser Zeit unterstützt und ermutigt haben.

Ohne die Hilfe und Unterstützung dieser Menschen wäre diese Arbeit nicht möglich gewesen.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Arbeit selbständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Köthen, 20.04.2023 Li. Jincheng

Ort, Datum Unterschrift der Autorin

Sperrvermerk

Sperrvermerk: ja ☐ nein ☒

Der Inhalt der Arbeit darf Dritten ohne Genehmigung der/des (Bezeichnung des Unternehmens) nicht zugänglich gemacht werden. Dieser Sperrvermerk gilt für die Dauer von X Jahren.

Ort, Datum Unterschrift der Autorin

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
APP	Application Portability Profile
AR	Augmented Reality
HMD	Head Mounted Displays
KI	künstlicher Intelligenz
MR	Mixed Reality
PC	Personalcomputer
SDK	Software Development Kit
UE	Unreal Engine
VR	Virtual Reality
xR	Extended Reality
XRIT	XR Interaction Toolkit

1. Einleitung

1.1. Motivation

Das Ziel dieser Arbeit ist es, eine Simulation des Knetens in Virtual Reality (VR) zu entwickeln und zu evaluieren, um die Möglichkeit zu schaffen, Kneten in einer virtuellen Umgebung zu erleben. Virtual Reality ermöglicht es, reale Erfahrungen in einer simulierten Umgebung zu schaffen. Die VR-Technologie kann dazu genutzt werden, um das Kneten in einer virtuellen Umgebung zu erleben und das Erlebnis zu optimieren.

Die Simulation des Knetens in VR kann auch als pädagogisches Werkzeug in der Bildung von Kindern genutzt werden. Durch die Nutzung dieser Technologie können Kinder ihre Feinmotorik und Kreativität verbessern, indem sie digitale Knete formen und manipulieren. Darüber hinaus kann die Simulation des Knetens in Schulen und Bildungseinrichtungen als Lehrmittel verwendet werden, um den Schülern ein besseres Verständnis von Geometrie und Formen zu vermitteln. Die Schüler können virtuelle Knete verwenden, um geometrische Formen zu erstellen und ihr Verständnis von räumlichen Konzepten zu vertiefen. Insgesamt bietet die Simulation des Knetens in VR eine neue und aufregende Möglichkeit, die Bildung von Kindern zu fördern und ihr kreatives Potenzial zu entfalten.

1.2. Zielsetzung

Die Ziele für die Weiterentwicklung von VR-Simulationsanwendungen für Knete können wie folgt formuliert werden:

- Verbesserung der Benutzererfahrung: Eine der Hauptziele ist es, die Benutzererfahrung zu verbessern, um ein realistischeres, immersiveres und intuitiveres Erlebnis zu bieten. Dies kann durch die Verbesserung von Grafiken, Physiksimulationen, Steuerungsoptionen und haptischem Feedback erreicht werden, um den Benutzern ein beeindruckenderes und ansprechenderes Knete Erlebnis zu bieten.
- Erweiterung der pädagogischen und bildungsbezogenen Funktionen: Ein weiteres Ziel ist die Weiterentwicklung der pädagogischen und bildungsbezogenen Funktionen von VR-Simulationsknete. Dies kann durch die Integration von Lerninhalten, Anleitungen, Übungen und Feedback-Mechanismen erreicht werden, um den Nutzern ein verbessertes Lern- und Bildungserlebnis zu bieten. Ziel ist es, VR-Simulationsknete als pädagogisches Werkzeug zu etablieren, das in Bildungseinrichtungen, Trainingsprogrammen und pädagogischen Aktivitäten eingesetzt werden kann.

- Förderung der sozialen Interaktion: Ein weiteres Ziel ist die Verbesserung der sozialen Interaktionsmöglichkeiten innerhalb von VR-Simulationsknete-Anwendungen. Durch die Integration von Multiplayer- und Kollaborationsfunktionen können Benutzer miteinander interagieren, zusammenarbeiten und sich austauschen, um das soziale und pädagogische Potenzial der App zu erweitern. Ziel ist es, den Nutzern eine sozialere und kooperative Umgebung zu bieten, in der sie gemeinsam Knete Prozesse erforschen und gestalten können.
- Kombination von virtueller Realität und erweiterter Realität: Ein weiteres Ziel ist die Kombination von VR-Simulationsknete mit erweiterter Realität, um ein reichhaltigeres und realistischeres Erlebnis zu schaffen. Ziel ist es, die Grenzen zwischen virtueller und realer Welt aufzuheben und den Benutzern ein noch immersiveres und erweitertes Erlebnis zu bieten. Dies könnte durch die Integration von AR-Funktionen ermöglicht werden, die es den Benutzern ermöglichen, Knete in ihrer realen Umgebung zu platzieren, zu manipulieren und mit ihr zu interagieren.
- Verbesserung der technischen Leistung: Ein weiteres Ziel ist die kontinuierliche Verbesserung der technischen Leistung von VR-Simulationsknete-Anwendungen. Dazu gehören die Optimierung von Grafiken, die Verbesserung von Leistung und Stabilität, sowie die Unterstützung von verschiedenen VR-Hardware-Plattformen, um die Zugänglichkeit und Benutzerfreundlichkeit zu verbessern.
- Fortschrittliche Knete-Modellierung: Ein weiteres Ziel ist die Weiterentwicklung von fortschrittlichen Knete-Modellierungstechniken, um realistischere und vielfältigere Knete-Modelle zu erstellen. Dies könnte die Integration von unterschiedlichen Texturen, Materialien und Farben in die Knete-Modelle, sowie die Möglichkeit zur Modifikation von Knete-Eigenschaften in Echtzeit beinhalten, um den Benutzern mehr kreative Freiheit zu ermöglichen.
- Nutzerfeedback und Iteration: Ein weiteres Ziel ist die kontinuierliche Einholung von Nutzerfeedback und die Iteration basierend auf diesem Feedback. Durch die Integration von Feedback-Mechanismen und die Aktualisierung der Anwendung entsprechend den Bedürfnissen und Anforderungen der Benutzer kann eine kontinuierliche Verbesserung und Weiterentwicklung von VR-Simulationsknete erreicht werden.

Zusammenfassend ist das Hauptziel der Weiterentwicklung von VR-Simulationsknete-Anwendungen die Verbesserung der Benutzererfahrung, die Erweiterung der pädagogischen Funktionen, die Förderung der sozialen Interaktion, die Kombination von VR und AR, die Verbesserung der technischen Leistung, die Berücksichtigung von Barrierefreiheit, die Fortschrittliche Knete-Modellierung und die kontinuierliche Verbesserung basierend auf Nutzerfeedback und Iteration.

1.3. Vorgehensplan

In dieser Arbeit wird eine Virtual-Reality-Anwendung unter dem Namen "Virtual PlayDoh" konzeptionell vorgeschlagen und entwickelt. Der Vorgehensplan für die Umsetzung der Arbeit umfasst folgende Schritte:

1. Recherche: Eine umfassende Literaturrecherche wird durchgeführt, um den aktuellen Stand der Forschung im Bereich Virtual Reality und Kneten zu ermitteln. Hierbei werden auch bereits existierende VR-Anwendungen zum Thema Kneten untersucht, um mögliche Ansätze und Verbesserungen zu identifizieren.
2. Konzeptentwicklung: Auf Basis der Rechercheergebnisse wird ein Konzept für die Virtual-Reality-Anwendung "Virtual PlayDoh" entwickelt. Dabei werden die Anforderungen an die Anwendung, wie z.B. Interaktionsmöglichkeiten, Benutzerfreundlichkeit und Immersion, berücksichtigt.
3. Implementierung: Die Anwendung wird unter Verwendung von Unity implementiert. Dabei werden die entwickelten Konzepte in die Anwendung integriert und die Funktionsfähigkeit der Anwendung getestet.
4. Evaluation: Um die Funktionsfähigkeit und Wirksamkeit der Anwendung zu evaluieren, werden Benutzertests durchgeführt. Hierbei werden verschiedene Aspekte wie Benutzerfreundlichkeit, Immersion und Interaktion untersucht, sowie die Auswirkungen auf die körperliche und geistige Gesundheit der Nutzer.
5. Ergebnisbewertung: Auf Basis der Ergebnisse aus der Evaluation werden die Stärken und Schwächen der Anwendung analysiert und Verbesserungspotenziale aufgezeigt. Es wird bewertet, inwiefern die Anwendung die gesteckten Ziele erreicht hat und wie sie in Zukunft weiterentwickelt werden kann.
6. Zusammenfassung: Die Arbeit wird mit einer Zusammenfassung der Ergebnisse abgeschlossen, die die wichtigsten Erkenntnisse und Schlussfolgerungen hervorhebt. Es werden auch mögliche Anwendungsgebiete und zukünftige Entwicklungen aufgezeigt.

2. Grundlagen

2.1. Virtual Reality

Virtual Reality (VR) ist eine Technologie, die es ermöglicht, eine computergenerierte Umgebung zu schaffen, die eine immersive und interaktive Erfahrung bietet. Im Gegensatz zu herkömmlichen Medien wie Film oder Fernsehen können Nutzer in VR-Anwendungen aktiv in die simulierten Welten eingreifen und interagieren. VR-Technologie wird in vielen Bereichen eingesetzt, wie beispielsweise in der Unterhaltungsindustrie, im Bildungssektor und in der Medizin.

Im Bereich der Unterhaltung werden VR-Anwendungen häufig für Spiele und Simulationen genutzt, um den Nutzern eine immersive Erfahrung zu bieten. Im Bildungsbereich können VR-Anwendungen genutzt werden, um komplexe Sachverhalte und Zusammenhänge visuell und interaktiv darzustellen und somit das Lernen zu erleichtern. In der Medizin wird VR-Technologie genutzt, um chirurgische Eingriffe zu simulieren oder Patienten mit Phobien zu therapieren.

VR-Anwendungen werden durch die Verwendung von Head-Mounted-Displays (HMDs), Sensoren und Tracking-Systemen erzeugt. Diese Geräte ermöglichen es dem Nutzer, in eine virtuelle Umgebung einzutauchen und mit ihr zu interagieren. Durch die Kombination von VR-Technologie mit haptischen Feedback-Systemen, wie beispielsweise Handschuhen oder Controllern, können Nutzer auch das Gefühl haben, physisch in der simulierten Umgebung zu sein und mit den virtuellen Objekten zu interagieren.

Neben kabelgebundenen oder mobilen Head-Mounted Displays gewinnen im Bereich der VR weitere Geräte wie Haptikgeräte, Controller, Westen, omnidirektionale Laufbänder, Tracking-Technologien sowie optische Scanner für gestenbasierte Interaktion an Bedeutung. Die meisten dieser Technologien sind bereits präzise und robust genug, um für den professionellen Einsatz und wissenschaftliche Experimente verwendet zu werden [Ch16].

2.2. Play-Doh

Play-Doh ist eine weiche, formbare Masse, die oft in bunten Farben erhältlich ist und speziell für kreative und spielerische Aktivitäten entwickelt wurde. Die Knete besteht in der Regel aus einfachen Zutaten wie Wasser, Mehl, Salz, Lebensmittelfarben und gegebenenfalls Duftstoffen, die sicher und ungiftig sind, um eine sichere Verwendung für Kinder zu gewährleisten.

Die Konsistenz von Play-Doh ist weich, geschmeidig und leicht formbar, was es zu einem idealen Material für kreative Projekte macht. Es lässt sich leicht kneten, rollen, formen, schneiden und modellieren, um eine Vielzahl von Formen, Figuren, Designs und Texturen zu erstellen. Die Knete

behält ihre Form, ohne zu zerbröckeln oder auszuhärten, was es den Benutzern ermöglicht, ihre Kreationen zu verfeinern und immer wieder neu zu gestalten.

Play-Doh ist auch in verschiedenen Farben erhältlich, was es den Benutzern ermöglicht, bunte und lebendige Kreationen zu erstellen. Die Knete lässt sich leicht mischen, um neue Farben zu erzeugen, und ermöglicht es den Benutzern, ihrer Kreativität freien Lauf zu lassen und eigene Farbkombinationen zu erstellen.

Ein weiteres Merkmal von Play-Doh ist seine nicht permanente Natur, was bedeutet, dass es nicht aushärtet oder dauerhaft wird. Dies ermöglicht es den Benutzern, ihre Kreationen zu verändern, neu zu gestalten und immer wieder von vorne zu beginnen, was ein hohes Maß an Flexibilität und Experimentierfreude bietet.

Play-Doh wird oft für kreative Aktivitäten wie Modellieren, Formen, Basteln, Kunsthandwerk und sogar als sensorisches Spielzeug oder therapeutisches Hilfsmittel verwendet. Es ist bekannt für seinen spielerischen und kreativen Charakter, der die Fantasie und die motorischen Fähigkeiten von Kindern und Erwachsenen fördert und ihnen ermöglicht, ihrer Kreativität Ausdruck zu verleihen und einzigartige Kunstwerke oder Modelle zu erstellen.

Eine Studie ergab, dass Handaktivitäten mit Spielknete die Fingerfertigkeit und die Hand-Augen-Koordination bei Kindern deutlich verbessern[Pa18]. Beim Spielen mit Spielknete werden häufig verschiedene Fingerbewegungen wie Drücken, Kneten, Reiben und Schneiden ausgeführt, die die Fingerfertigkeit und Koordination fördern.

Darüber hinaus hilft Spielknete den Kindern, die Kontrolle über die Handmuskeln zu entwickeln. Beim Spielen mit Spielknete wenden die Kinder mit ihren Fingern verschiedene Kräfte an, um die Form und Textur der Spielknete zu verändern, und trainieren so die Kontrolle und Koordination der Handmuskeln.

Diese feinmotorischen Fähigkeiten sind für viele Aktivitäten im täglichen Leben der Kinder unerlässlich, wie z. B. Schreiben, Zeichnen, Umgang mit Werkzeugen und Selbstversorgung. Durch die Beschäftigung mit Spielknete können Kinder diese wichtigen feinmotorischen Fähigkeiten in der Freizeit trainieren und entwickeln, was sich positiv auf ihre Entwicklung und ihr Lernen auswirkt.

3. Stand der Forschung und Technik

3.1. Anwendungsbereich von VR

Virtual Reality (VR) hat in den letzten Jahren stark an Bedeutung gewonnen und wird in verschiedenen Anwendungsbereichen eingesetzt. Einer der wichtigsten Anwendungsbereiche von VR ist die Unterhaltungsindustrie. VR-Brillen werden immer beliebter und ermöglichen es Benutzern, in virtuelle Welten einzutauchen und interaktive Erfahrungen zu machen. Diese Technologie wird auch in der Gaming-Industrie immer häufiger eingesetzt, um den Spielern ein immersiveres und realistischeres Spielerlebnis zu bieten.

Neben der Unterhaltungsindustrie gibt es viele weitere Anwendungsbereiche von VR. In der Medizin wird VR beispielsweise zur Behandlung von Angststörungen und zur Schmerzlinderung eingesetzt. Durch VR können Patienten in eine virtuelle Umgebung eintauchen und lernen, mit ihren Ängsten und Schmerzen umzugehen. Auch in der Ausbildung und im Training wird VR immer häufiger eingesetzt, um realistische und sichere Lernumgebungen zu schaffen. So können beispielsweise Piloten in einer VR-Umgebung gefährliche Flugsituationen simulieren und lernen, wie sie darauf reagieren können, ohne dabei reale Risiken einzugehen.

In der Architektur und im Design kann VR ebenfalls eingesetzt werden, um virtuelle Modelle von Gebäuden und Produkten zu erstellen und zu präsentieren. Dadurch können Kunden und Benutzer ein realistisches Gefühl für das Endprodukt bekommen, bevor es tatsächlich gebaut oder produziert wird. Auch in der Stadtplanung und im Verkehrswesen kann VR eingesetzt werden, um Verkehrsflüsse zu simulieren und zu optimieren.

Insgesamt gibt es viele Anwendungsbereiche von VR, die sich ständig weiterentwickeln und erweitern. Durch die immer fortschrittlicheren Technologien und die wachsende Nachfrage nach immersiven Erlebnissen wird die Bedeutung von VR in Zukunft sicherlich weiter zunehmen.

3.2. Simulieren der Knete in VR

Obwohl es keine spezifischen Forschungsprojekte zur Simulation von Knete gibt, kann die Technologie der Echtzeitverformung auch auf die Simulation von Knete angewendet werden. Mit der Verwendung von Echtzeitverformung in der virtuellen Realität können Benutzer interaktive und realistische Erfahrungen mit Knete machen, indem sie die Knete formen und verformen und die Auswirkungen in Echtzeit betrachten können. Echtzeitverformung kann dazu beitragen, die Benutzerinteraktion in VR-Spielen und Anwendungen zu verbessern. Es ermöglicht den Benutzern, Objekte in der virtuellen Welt zu greifen, zu bewegen und zu manipulieren, als ob sie physisch vorhanden wären.

Eine Lösung für das Problem der Verformung in Echtzeit ist Echtzeit-Verformung von Strukturen mit Hilfe von Finiten Elementen und neuronalen Netzen in Anwendungen der virtuellen Realität. Der Ansatz kombiniert die Finite-Elemente-Methode und Berechnungen mit Neuronalen Netzen, um dem Benutzer die Möglichkeit zu geben, interaktive Formänderungen vorzunehmen und die daraus resultierenden Verformungsänderungen in einer virtuellen Umgebung zu betrachten [HCS06].

In der Medizin kann Echtzeitverformung verwendet werden, um chirurgische Simulationen durchzuführen. Die Chirurgen können virtuelle Organe manipulieren und trainieren, um ihre Fähigkeiten zu verbessern, bevor sie echte Operationen durchführen [Al15, Ho12].

In der Architektur und im Ingenieurwesen kann die Echtzeitverformung verwendet werden, um virtuelle Prototypen von Gebäuden und Maschinen zu erstellen und zu manipulieren. Dies kann helfen, Fehler in einem frühen Stadium des Designsprozesses zu erkennen und zu korrigieren.

Insgesamt bietet die Echtzeitverformung in der virtuellen Realität eine Reihe von Anwendungen und hat das Potenzial, die Art und Weise zu verändern, wie wir mit virtuellen Welten interagieren und sie erleben.

4. Analyse der Problemstellung

4.1. Bedarfsanalyse

Die Simulation von Knete in VR birgt einige Herausforderungen. Eine Herausforderung besteht darin, das Verhalten des Benutzers in der realen Welt präzise zu erfassen und sinnvoll in die virtuelle Welt zu übertragen. Ein weiteres Problem ist die präzise Steuerung des Modells der Knete, um es entsprechend den Benutzeraktionen realistisch zu verformen. Zudem ist die Integration von Benutzerverhalten und Echtzeitverformung der Knete erforderlich, um ein insgesamt realistisches Knet-Simulationserlebnis zu gewährleisten. Zusätzlich ist es von großer Bedeutung, eine intuitive und benutzerfreundliche Benutzerschnittstelle zu entwickeln, um den Benutzern die einfache Steuerung des Knetens in der virtuellen Umgebung zu ermöglichen.

4.2. Auswahl von Entwicklungswerkzeugen

Bei der Entwicklung von Software zur Simulation von Knete in der virtuellen Realität ist es wichtig, die Anforderungen an verschiedene Entwicklungswerkzeuge zu berücksichtigen. Dieser Abschnitt behandelt die Auswahl von Hardware, Software, Entwicklungskits und Anwendungsschnittstellen, die für das Projekt relevant sind.

4.2.1. Hardware

Die Auswahl der richtigen Hardware ist ein entscheidender Schritt bei der Entwicklung von Programm zur Simulation von Knete in der virtuellen Realität. Es sind verschiedene Aspekte zu berücksichtigen, darunter die Leistungsfähigkeit der Hardware, die Tracking-Funktionen, die Benutzerfreundlichkeit und die Kosten.

Eine wichtige Hardware-Komponente ist die VR-Brille oder das Headset, das von den Benutzern getragen wird, um in die virtuelle Umgebung einzutauchen. Hierbei ist es wichtig, ein VR-Headset auszuwählen, das eine hohe Auflösung, ein weites Sichtfeld, ein komfortables Tragegefühl und präzises Tracking der Kopfbewegungen bietet, um ein immersives und realistisches Knete-Simulationserlebnis zu gewährleisten.

Darüber hinaus ist auch die Auswahl der Eingabegeräte von großer Bedeutung. Diese können beispielsweise Handcontroller oder Gestensteuerungssysteme sein, die es den Benutzern ermöglichen, die Knete in der virtuellen Umgebung zu formen und zu manipulieren. Die Eingabegeräte sollten präzise, ergonomisch und intuitiv zu bedienen sein, um ein realistisches haptisches Feedback und eine authentische Interaktion mit der Knete zu ermöglichen.

Zudem sollte auch die Leistungsfähigkeit des Computers oder der Spielkonsole, auf der die VR-Anwendung läuft, berücksichtigt werden. Eine leistungsfähige Hardware mit ausreichendem Prozessor, Arbeitsspeicher und Grafikkarte ist erforderlich, um eine flüssige Darstellung und Echtzeitverformung der Knete in der virtuellen Umgebung zu gewährleisten.

Die Kosten sind ebenfalls ein wichtiger Faktor bei der Auswahl der Hardware. VR-Headsets und Eingabegeräte können in verschiedenen Preisklassen erhältlich sein, und es ist wichtig, ein ausgewogenes Verhältnis zwischen den Kosten und den Anforderungen des Projekts zu finden.

Insgesamt ist die Auswahl der richtigen Hardware von großer Bedeutung, um eine optimale Performance, Benutzerfreundlichkeit und Immersion bei der Simulation von Knete in der virtuellen Realität zu gewährleisten. Es ist wichtig, die Anforderungen des Projekts sowie die verfügbaren Ressourcen und Budgets sorgfältig zu analysieren, um die besten Hardware-Optionen auszuwählen.

Die in diesem Projekt verwendete Hardware wurde vom XR-Labor der Hochschule Anhalt zur Verfügung gestellt. Die technischen Daten der für die Tests verwendeten Computer und VR-Geräte sind in Tabelle 1 und 2 aufgeführt [Mi21].

CPU	Intel(R) Core(TM) i7-8700 CPU @ 3.20 GHz
GPU	NVIDIA GeForce RTX 2080
RAM	64,0 GB
Betriebssystem	Windows 10 Pro

Table 1 Technische Daten des Computers

Display	Zwei 2.89-Zoll LCDs mit Pulse Backlight
Auflösung	2160 x 2160 pro Auge
Sichtfeld	114 Grad
Augenabstand	63 mm, +/- 8 mm durch Software-Justierung
Bildwiederholfrequenz	90 Hz
Sensoren	Gyroskop, Beschleunigungssensor, Magnetometer
Tracking	Inside-Out-Tracking mit zwei Kameras
Audio	- abnehmbare integrierte Kopfhörer - integriertes Mikrofon - 3,5 mm Kopfh¨orer/Mikrofon Buchse

Table 2 Technische Daten der PC-VR Brille (HP Reverb)

4.2.2. Software

Die Wahl der richtigen Software für die Entwicklung von Knetsimulationen in der virtuellen Realität ist ein weiterer wichtiger Aspekt. Es gibt verschiedene VR-Entwicklungstools auf dem Markt, die es Entwicklern ermöglichen, VR-Anwendungen zu erstellen, darunter Unity, Unreal Engine und viele mehr. Diese Tools bieten eine Vielzahl von Funktionen und Werkzeugen zur Erstellung von 3D-

Modellen, zur Implementierung von Interaktionen, zur Handhabung von Physik und zur Integration von VR-Hardware.

Unity ist eine benutzerfreundliche Engine, die eine schnelle Entwicklung ermöglicht und eine breite Palette von Plattformen unterstützt. Sie bietet eine umfangreiche Bibliothek von Assets und Plugins, die die Entwicklung erleichtern und beschleunigen können. Unity eignet sich gut für die Erstellung von Anwendungen mit geringer bis mittlerer Komplexität und ist ideal für Entwickler, die schnell Prototypen erstellen möchten.

Unreal Engine ist bekannt für ihre hohe Grafikqualität und ihre leistungsstarke Physik-Engine. Sie ist ideal für die Erstellung von anspruchsvollen VR-Anwendungen mit hoher Interaktivität und Realismus. Unreal Engine hat auch eine große Community, die die Entwicklung erleichtert und Unterstützung bietet.

In der folgenden Tabelle 3 werden die Unterschiede zwischen Unity und Unreal in Bezug auf Funktionalität, Programmiersprache, Lernfreundlichkeit usw. vergleichen [De22, Pr20, Sc17, Hu21].

Features	Unity Engine	Unreal Engine
Tutorials und Dokumente	Viele Tutorials und Dokumente online verfügbar (insbesondere für EntryLevel-Entwickler)	Grundlegende Tutorials und Dokumente online verfügbar
Programmiersprache	C# und JavaScript Kodierung	C++ Kodierung und Blaupausen (visuelle Programmierung)
Physik Simulation	Ja	Ja
Renderingverwaltung	Ja	Ja
Laden von Inhalten	Assetbundles vorhanden	Gesonderte Implementierung nötig
Quellcode	Proprietär	Open-Source
Verwendete VR-Brille Kompatibel	Ja	Ja
Einsteigerfreundliches Programm	Ja	Nein

Table 3 Unterschiede zwischen Unity Engine und Unreal Engine

Die Auswahl der richtigen Software ist entscheidend, um die gewünschten Funktionalitäten und die gewünschte Realitätstreue in der Knete-Simulation zu erreichen. Es ist wichtig, verschiedene Optionen zu evaluieren, die den Anforderungen des Projekts und den Fähigkeiten des Entwicklungsteams gerecht werden, und die ausgewählte Software entsprechend in den Entwicklungsprozess zu integrieren. Basierend auf den persönlichen Programmierungsfähigkeiten und der Arbeitsumgebung wird in diesem Projekt die Unity Engine verwendet, um eine Knetsimulationsanwendung zu erstellen und zu produzieren.

4.2.3. SDK und API

VR-Geräte verschiedener Hersteller verwenden unterschiedliche Software-Entwicklungskits (SDKs) und Anwendungsprogrammierschnittstellen (APIs), die die Art und den Prozess der Entwicklung von VR-Programmen beeinflussen können. Entwickler müssen sich mit verschiedenen SDKs und APIs vertraut machen und lernen, wie sie diese verwenden, um VR-Anwendungen zu schreiben.

Die Wahl des SDKs zur Entwicklung von VR-Anwendungen hängt von den spezifischen Anforderungen und Fähigkeiten des Entwicklers ab. Hier sind einige gängige VR-Entwicklungskits, aus denen Entwickler je nach Bedarf eines oder mehrere auswählen können:

SteamVR	SteamVR ist eine von Valve entwickelte VR-Plattform, die eine Reihe von Entwicklungswerkzeugen und APIs bietet, um Entwicklern bei der Erstellung von VR-Anwendungen für die SteamVR-Plattform zu helfen.
Oculus SDK	Das Oculus SDK ist ein SDK, das speziell für Oculus VR-Geräte entwickelt wurde und Entwicklern bei der Erstellung optimierter Oculus VR-Anwendungen hilft. Es bietet umfangreiche Werkzeuge und Ressourcen, einschließlich Funktionen wie Kopfverfolgung, Handsteuerung und Audio-Rendering.
OpenVR	OpenVR ist eine Open-Source-VR-Entwicklungsplattform, die verschiedene VR-Geräte und Engines unterstützt. Es bietet eine Reihe von APIs und Werkzeugen, um Entwicklern bei der Erstellung von Anwendungen für verschiedene VR-Geräte zu helfen.
XR Interaction Toolkit	Das XR Interaction Toolkit ist eine Sammlung von Tools und Komponenten, die von Unity zur Verfügung gestellt werden, um die Entwicklung von interaktiven Anwendungen in der erweiterten Realität (AR), virtuellen Realität (VR) und gemischten Realität (MR) zu erleichtern. Mit dem Toolkit können Entwickler verschiedene Interaktionsmethoden implementieren, wie z.B. Handgesten, Blicksteuerung und physische Interaktionen, um die Benutzererfahrung zu verbessern. Es ist plattformübergreifend und kann auf verschiedenen Geräten und Plattformen wie Oculus Rift, HTC Vive, Windows Mixed Reality und Magic Leap verwendet werden.

Table 4 Unterschiede zwischen unterschiedliche SDKs

Für dieses Praktikum wurden die Unity-Engine und das XR Interaction Toolkit (XRTK) auf der Grundlage der von der Arbeitsumgebung unterstützten Hardwaregeräte, der Programmierkenntnisse des Entwicklers und der Entwicklungskosten ausgewählt.

5. Programmieranforderungen

5.1.1. Modellierung und Verformung der Knete

Die Simulation von Knete in der virtuellen Realität erfordert ein Modell mit realistischen Verformungseigenschaften. Da der Entwickler in diesem Projekt erstmals an einer Modellverformung arbeitet, wurde als anfänglicher Prototyp für die Knete die mit Unity mitgelieferte Kugel gewählt, um das Projekt auf einfache und intuitive Weise zu testen.

Die Verformung wird durch Manipulation des Netzes auf der Oberfläche des Modells mithilfe einer physikbasierten Simulation erreicht. Dabei wird ein Eckpunkt im Oberflächennetz des Modells erkannt, um zu prüfen, ob er einen Trigger für das Handmodell im Programm auslöst. Ist dieser Trigger aktiv, wird die Form des Knetmodells durch Änderung der Position der Eckpunkte verändert.

Die Modellierung und Verformung der Knete stellen eine technische Herausforderung dar, da sie realistisch und interaktiv sein muss, um ein immersives VR-Erlebnis zu gewährleisten. Aus diesem Grund sind sorgfältige Programmierung und Tests erforderlich, um die gewünschte Funktionalität und Realitätstreue in der Knete-Simulation zu erreichen.

5.1.2. Erfassung der Benutzerinteraktion

Um dem Benutzer die Manipulation der Knete in VR zu ermöglichen, ist es notwendig, den Controller zu überwachen, um Informationen über die Handaktionen des Benutzers zu erhalten, wie z. B. Knopfdrücke und Auslöser. In diesem Projekt wird das XR Interaction Toolkit (XRTK) Entwicklungskit zur Überwachung des Controllers verwendet. Das Toolkit enthält zwei Arten von Komponenten zur Erfassung von Benutzereingaben: aktionsbasierte Komponenten und gerätebasierte Komponenten.

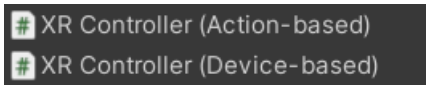

Abbildung 1 Zwei Eingabesysteme

Diese beiden Komponenten beziehen sich auf die beiden Eingabesysteme in Unity VR: das Input System und das XR Input Subsystem. Das XR Input Subsystem ist das Eingabesystem, das von der gerätebasierten Komponente des XRTK-Toolkits verwendet wird, um das Mapping zwischen Aktionen in XR und physischen Geräten (wie z. B. die Knöpfe an VR-Controllern) zu handhaben und Geräte von verschiedenen XR-Herstellern anzupassen. Dieses Eingabesystem ist einfacher zu verwenden, aber die Bindung von Aktionen an Gerätetasten kann nicht geändert werden.

Das Input System ist das neue universelle Eingabesystem von Unity, mit dem die Zuordnung zwischen Aktionen und Tasten frei konfiguriert werden kann, was es flexibler macht. Das Input System wird von der aktionsbasierten Komponente des XRTK-Toolkits verwendet.

Die Verwendung der aktionsbasierten Komponente des XRTK-Toolkits, die auf dem Input System-Eingabesystem basiert, ermöglicht den Entwicklern, das Programm besser an verschiedene Eingabesysteme anzupassen und flexibler bei der Zuordnung von Tasten zu sein. Dies wird es ermöglichen, zukünftige Änderungen oder Updates in den Eingabesystemen effizienter zu bewältigen und die Benutzererfahrung insgesamt zu verbessern.
Die gerätebasierte Komponente des XRTK-Toolkits hingegen bietet eine einfachere Verwendung, da sie bereits voreingestellte Mapping-Funktionen für physische Geräte wie VR-Controller bereitstellt. Dies ermöglicht eine schnellere Implementierung, jedoch ist die Bindung von Aktionen an Gerätetasten nicht veränderbar. Daher wurde im Rahmen dieses Projekts die Entscheidung getroffen, die aktionsbasierte Komponente zu verwenden, um größere Flexibilität und Anpassungsfähigkeit zu gewährleisten [He22].
Durch die Kombination von XRTK-Toolkit und Input System-Eingabesystem ist das Programm in der Lage, die Benutzerinteraktion präzise und effektiv zu erfassen, um eine realistische und immersivere VR-Erfahrung für die Benutzer zu bieten. Die Auswahl der richtigen Komponenten zur Erfassung der Benutzerinteraktion ist entscheidend für die Funktionalität und Benutzerfreundlichkeit der VR-Anwendung.

5.1.3. Echtzeit-Kombination von Benutzerinteraktion und Knete- Verformung

Um die Benutzerinteraktion und die Knete-Verformung in Echtzeit zu kombinieren, werden die Betriebsdaten des Benutzers und die Verformungsmethode der Knete kontinuierlich miteinander synchronisiert. Hierfür wird ein Coroutine-Programm verwendet, das in Echtzeit die 3D-Position des Handmodells erfasst, wenn eine Verformung erforderlich ist. Anhand der Handposition des Benutzers wird die Beziehung zwischen der Handposition und dem Verformungszustand der Knete berechnet. Diese Berechnungen werden dann auf das Knetmodell angewendet, um die gewünschte Verformung in Echtzeit zu erzeugen.
Es ist auch möglich, die Textur und Farbe der Knete zu ändern, um die interaktive Erfahrung des Benutzers zu verbessern. Dies kann beispielsweise verwendet werden, um visuelle Effekte hinzuzufügen, die die Verformung der Knete betonen oder die Benutzerinteraktion weiter verstärken. Durch die Echtzeit-Kombination von Benutzerinteraktion und Knete-Verformung wird dem Benutzer eine immersive und realistische Erfahrung geboten, bei der er die Knete in VR interaktiv verformen und gestalten kann. Die Synchronisation von Benutzerinteraktion und Knete-Verformung in Echtzeit stellt sicher, dass die Verformung der Knete unmittelbar und reaktionsschnell auf die Handaktionen des Benutzers erfolgt und eine nahtlose und intuitive Benutzererfahrung gewährleistet wird.

6. Lösungsansatz

6.1. Kneten Objekt

Das Spielobjekt ist das wichtigste Konzept im Unity Editor. Jedes Objekt im Spiel (von Charakteren und Sammelgegenständen bis hin zu Lichtquellen, Kameras und Effekten) ist ein Spielobjekt. Das Spielobjekt selbst kann jedoch keine Aktionen ausführen [Un21]. Der Programmierer muss dem Spielobjekt durch Komponenten Eigenschaften zur Verfügung stellen, damit es zu einem Charakter, einer Umgebung oder einem Effekt werden kann.

Die Spielobjekte in diesem Praktikumsprojekt umfassen Lichtquellen, Strahlen, Hände, Knete und andere Objekte, wobei das Knete-Spielobjekt am wichtigsten ist. Die erforderlichen Eigenschaften für dieses Objekt werden durch die folgenden Komponenten bereitgestellt:

Komponenten	Beschreiben
Transform	Die Komponente Transform definiert die Position, Rotation und Skalierung des Spielobjekts. Es bestimmt die Position, Ausrichtung und Größe des Spielobjekts in der Szene. In diesem Praktikum wird das Transform-Komponente verwendet, um die Position und Rotation des Knetmodells zu kontrollieren.
Mesh Filter	Mesh Filter ist eine Komponente in Unity, die für das Rendern des Aussehens von 3D-Modellen verantwortlich ist. Es muss zusammen mit dem Mesh Renderer verwendet werden, der das Rendern des Aussehens definiert (z.B. Material, Beleuchtung, Schatten usw.). Der Mesh Filter benötigt eine Referenz auf ein Mesh-Objekt, das die zu rendernden Vertices, Dreiecke und andere geometrische Daten enthält. In diesem Praktikum verwenden wir grundlegende geometrische Formen in Unity (z.B. Kugeln), um die Grundform der Knete darzustellen, und verformen das Mesh-Objekt zur Laufzeit durch die Änderung der Vertices, um die Verformung der Knete zu erreichen.
Mesh Renderer	Die Komponente Mesh Renderer wird verwendet, um 3D-Modelle zu rendern. Es wandelt die Dreiecksflächen des Modells in sichtbare Grafiken um. In diesem Praktikum wird das Mesh Renderer-Komponente verwendet, um das Aussehen des Knetmodells zu rendern.
Mesh Collider	Die Komponente Mesh Collider wird verwendet, um Kollisionen zu erkennen. Es ermöglicht es dem Spielobjekt, mit anderen Spielobjekten zu interagieren und auf diese Weise auf die Eingabe des Benutzers zu reagieren. In diesem Praktikum wird das Mesh Collider-Komponente verwendet, um Kollisionen mit der Hand zu erkennen und so die Benutzerinteraktion in Echtzeit mit der Kneteverformung zu kombinieren.
Rigidbody	Die Komponente Rigidbody wird verwendet, um physikalische Simulationen zu implementieren. Es gibt dem Spielobjekt Masse, Geschwindigkeit und Kraft und ermöglicht so die physikalische Interaktion in der Szene. In diesem Praktikum wird Die Komponente Rigidbody verwendet, um die physikalischen Eigenschaften der Knete zu simulieren, wie z.B. Elastizität und Gewicht.
XR Grab Interactable	XR Grab Interactable ist eine Komponente im XRTK-Toolkit, das die Interaktionsfunktionen für VR-Controller implementiert. Es ermöglicht es einem Spielobjekt, von einem VR-Controller gegriffen, bewegt und platziert zu werden. Im Rahmen dieses Praktikums wird das XR Grab Interactable

	verwendet, um die Interaktionsfunktionen für das Knetmodell zu implementieren und es dem Benutzer zu ermöglichen, es mit einem VR-Controller zu manipulieren.

Table 5 Komponenten von Knete-Gameobjekt

6.2. Unterscheidung der Kraftzustände der Knete

Um den Effekt der Kneten Simulation in VR zu realisieren, müssen die verschiedenen Kraftzustände berücksichtigt werden, denen die Knete ausgesetzt ist. Im echten Leben ist Knete Kräften wie Dehnung und Kompression ausgesetzt, und unterschiedliche Kraftzustände können zu Veränderungen der Form der Knete führen. In diesem Projekt wird ein VR-Joystick verwendet, um Informationen über die Aktionen des Benutzers zu erhalten, sodass die auf die Knete ausgeübte Kraft durch die Joystick-Tasten gesteuert wird. Der Benutzer kann die Joystick-Tasten oder Gesten verwenden, um die Kraftanwendung auf die Knete zu simulieren.

6.2.1. Druckzustand

Der Druckzustand der Knete kann durch Drücken der X-Taste des Handgriffs erreicht werden. In diesem Zustand erfasst das Programm die Eckpunkte auf dem Knete-Netz in einem bestimmten Bereich in der Nähe des Handmodells, so dass diese Eckpunkte dem Handmodell in Richtung des Zentrums des Knetmodells folgen können.

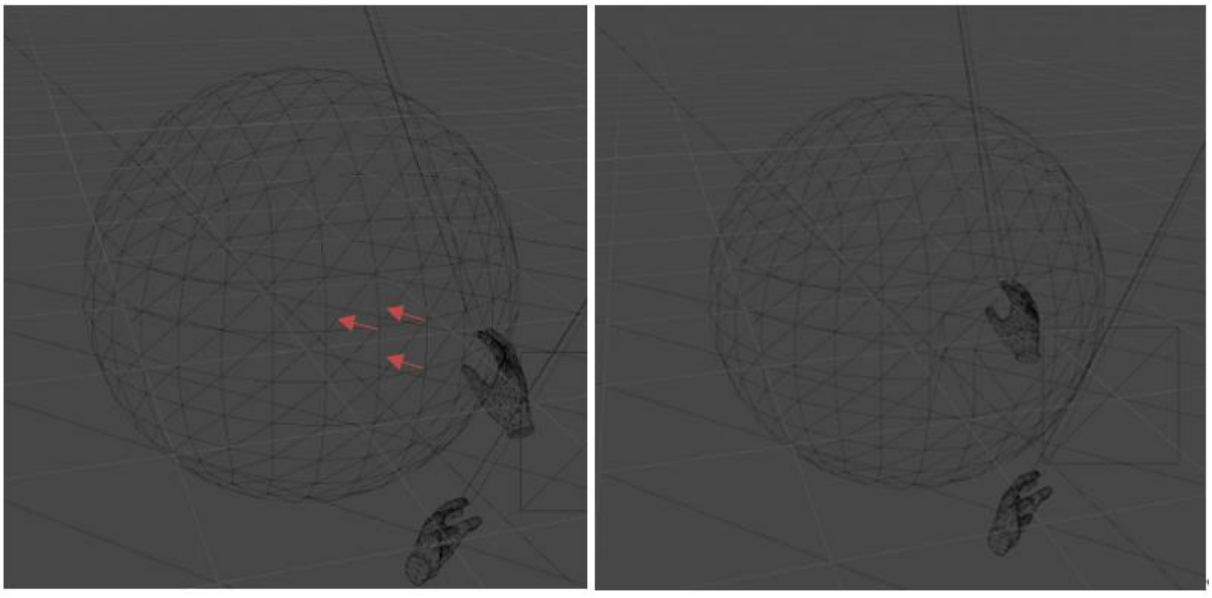

Abbildung 2 Druckzustand von Knete-Model

Konkret berechnet das Programm im drückbaren Zustand zunächst den Abstand zwischen dem Handmodell und dem Zentrum der Knete und bestimmt anhand der Differenz zwischen den beiden Abständen im vorderen und hinteren Bereich, ob sich das Handmodell der Knete nähert. Wenn sich das Handmodell der Knete näher bewegt, berechnet das Programm die Position der Eckpunkte, die

entsprechend dem Abstand und der Bewegungsrichtung zwischen dem Handmodell und der Knete verschoben werden müssen, sodass die Eckpunkte auf dem Knete-Netz näher an die Mitte der Knete heranrücken und so den Druckeffekt erzielen. Wenn sich das Handmodell nicht mehr näher an die Knete bewegt, hört das Programm auf, die Veränderung des Knete-Netzes zu berechnen und beendet den Zustand. Im drückbaren Zustand kann der Benutzer den Zustand beenden, indem er die X-Taste am Griff loslässt und das Knetmodell in einen nicht drückbaren Zustand zurückbringt.

6.2.2. Zugzustand

Der Zugzustand wird durch gleichzeitiges Drücken der X- und Grab-Tasten des Handgriffs erreicht. In diesem Zustand erfasst das Programm die Eckpunkte auf dem Netz der Knete in einem bestimmten Bereich in der Nähe des Handmodells, so dass diese Eckpunkte dem Handmodell in eine Richtung weg vom Zentrum des Knetmodells folgen können.

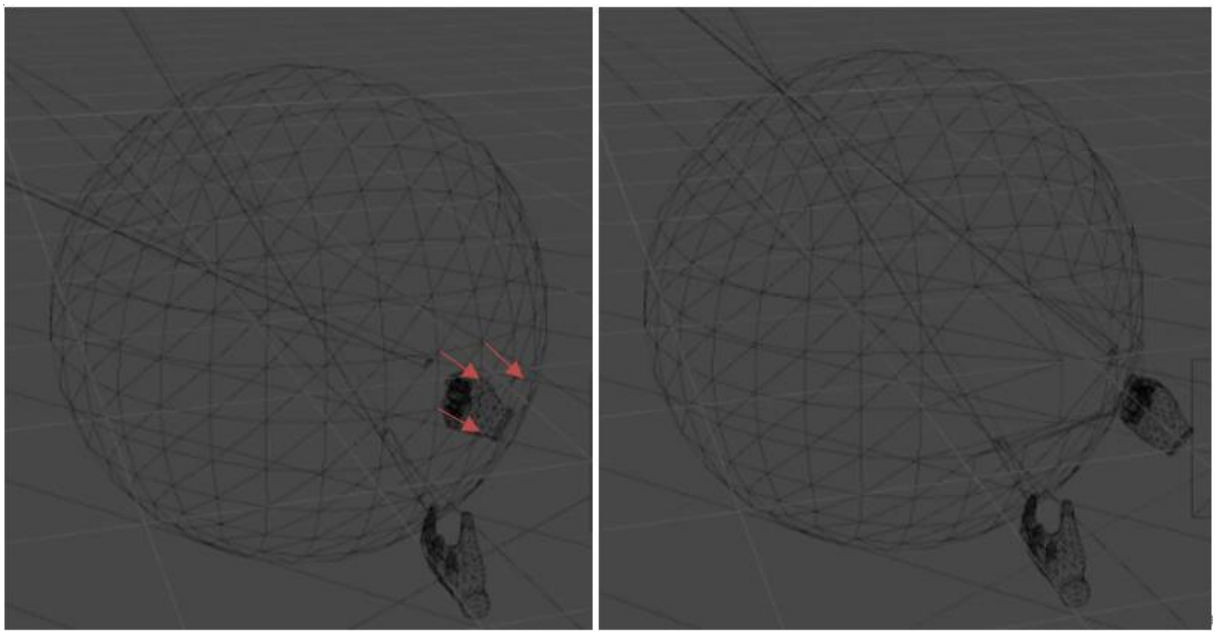

Abbildung 3 Zugzustand von Knete-Model

Konkret berechnet das Programm im streckbaren Zustand zunächst den Abstand zwischen dem Handmodell und dem Zentrum der Knete und bestimmt anhand der Differenz zwischen den beiden Abständen im vorderen und hinteren Rahmen, ob sich das Handmodell auf die Knete zubewegt. Wenn sich das Handmodell nicht auf die Knete zubewegt, berechnet das Programm die Position der Eckpunkte, die verschoben werden müssen, anhand des Abstands und der Bewegungsrichtung zwischen dem Handmodell und der Knete, so dass sich die Eckpunkte auf der Knete Netz vom Knete Mittelpunkt wegbewegen, um den Zugseffekt zu erzielen. Wenn sich das Handmodell nicht mehr von der Knete wegbewegt, hört das Programm auf, die Bewegung der Knete zu berechnen und beendet den Zustand. Im streckbaren Zustand kann der Benutzer den Zustand beenden, indem er die Tasten

X und Greifen am Griff loslässt, um das Knetmodell in einen nicht streckbaren Zustand zurückzubringen.

6.3. Formänderung der Knete

Die Form der Knete wird durch das Mesh Filter-Modul gesteuert, das ein Mesh-Objekt enthält, welches die Gitterpunkte, Dreiecke und Normalenvektoren der Punkte des Knetmodells enthält. Diese Gitterdaten definieren das dreieckige Gitternetzwerk der Knete Oberfläche und können durch die Änderung der Gitterdaten verändert werden. In der Echtzeit-Simulation müssen die Gitterdaten kontinuierlich aktualisiert werden, um die Verformung der Knete zu simulieren. Im Rahmen dieses Praktikums können wir die Form der Knete durch folgende Schritte ändern:

a) Um das Mesh-Objekt zu erhalten, kann der folgende Code verwendet werden, um das aktuelle Mesh-Objekt des Clay-Game-Objekts abzurufen:

```
1. Mesh mesh = gameObject.GetComponent<MeshFilter>().mesh;
```

b) Die Netzwerkdaten können direkt durch Ändern der Position der Netzwerkdatenpunkte geändert werden, um die Form von Knete zu ändern. Der folgende Code zeigt, wie der erste Punkt um eine Einheit nach oben verschoben wird:

```
1. Vector3[] vertices = mesh.vertices;
2. vertices[0] += Vector3.up;
3. mesh.vertices = vertices;
```

c) Um das Netzwerk zu aktualisieren, kann der folgende Code verwendet werden, um Unity mitzuteilen, die Netzwerkdaten zu aktualisieren:

```
1. mesh.RecalculateNormals();
2. mesh.RecalculateBounds();
```

In Echtzeit-Simulationen kann die Form von Knete durch Gesten des Benutzers geändert werden. Wenn der Benutzer Knete mit seiner Hand greift, können die Positionen der Knete-Vertices und die Positionen der Hand berechnet werden. Die Ergebnisse der Berechnungen können dann verwendet werden, um die Positionen der Knete-Vertices zu ändern und somit die Form der Knete zu verändern. Um diese Funktionalität zu implementieren, ist es erforderlich, ein zusätzliches Skript zur Steuerung hinzuzufügen. In diesem Projekt wurde das Skript "Kneten Mesh Kontrolle" entwickelt, um die Kontrolle über das Modellnetz zu ermöglichen.

6.4. Skript Kneten-Mesh-Kontrolle

Das Skript ermöglicht es dem Benutzer, die Form der Knete in einer 3D-Szene interaktiv zu steuern und dynamische Veränderungen zu erzielen. Wenn der Benutzer das Handmodell bewegt, verformt sich die Knete entsprechend, sodass das Handmodell natürlicher mit der Knete interagieren kann. Dieses Skript ermöglicht es dem Benutzer, einfach Knetmodelle mit einer Vielzahl von Formen zu erstellen.

6.4.1. Flussdiagramm der Hauptfunktion

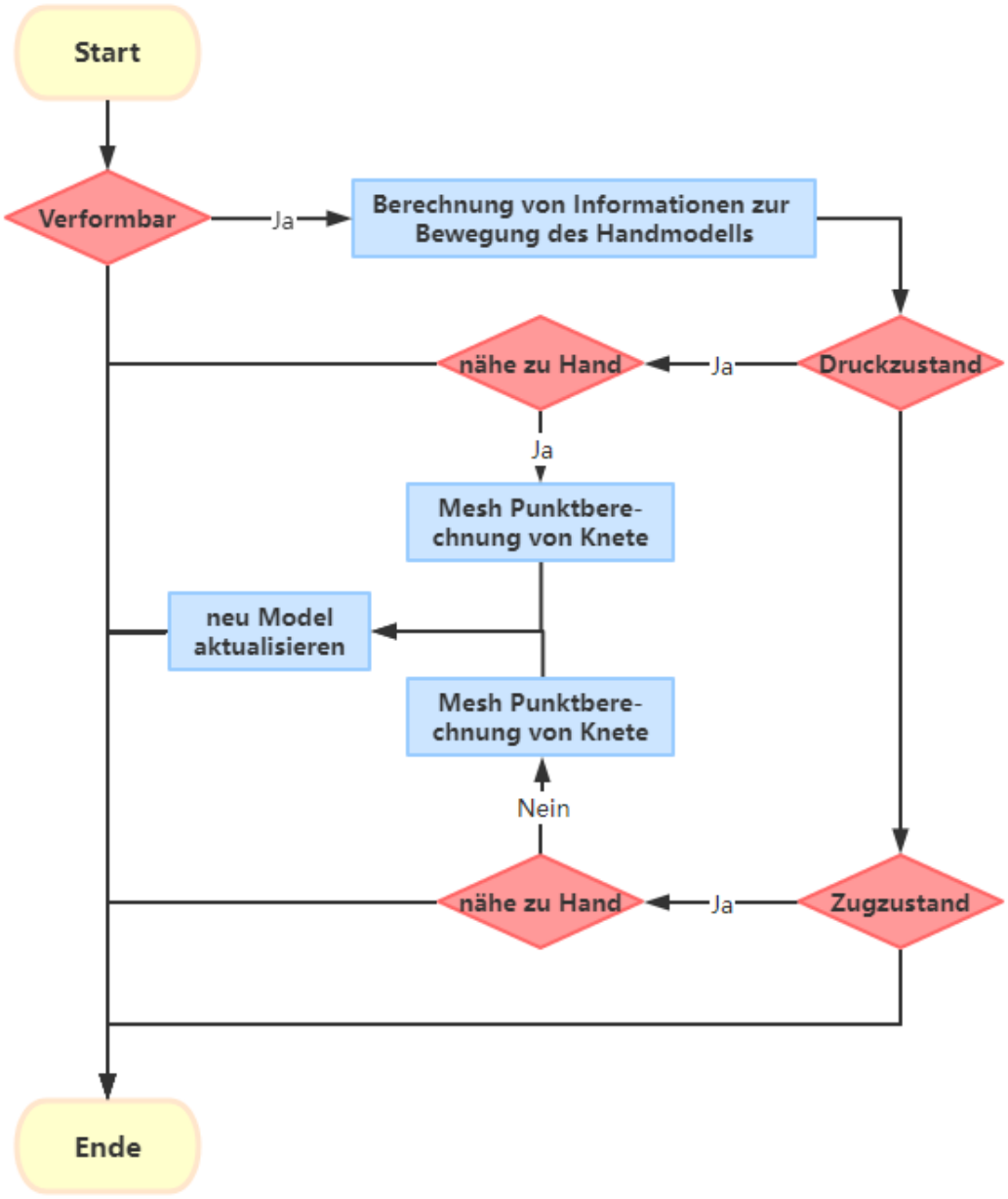

Abbildung 4 Flussdiagramm der Hauptfunktion

Dieses Flussdiagramm beschreibt den Hauptprozess zur Durchführung einer Knete Simulation.

1. Das Programm stellt zunächst fest, ob sich der Knete in einem verformbaren Zustand befindet. Ist dies der Fall, fährt es mit dem nächsten Schritt fort. Ist dies nicht der Fall, wird der Prozess beendet.
2. Die Positionsinformationen des Handmodells und der Knete werden ermittelt und die Bewegungsrichtung des Handmodells wird berechnet.
3. Der Kraftzustand der Knete wird bestimmt.
   i. Wenn der Knete drückbar ist:
      a) Der Abstand zwischen dem Handmodell und der Mitte der Knete wird berechnet, und anhand der Differenz zwischen den beiden Abständen im vorderen und hinteren Rahmen wird bestimmt, ob das Handmodell nahe an der Knete ist.
      b) Wenn das Handmodell nahe an der Knete ist, wird die Position des zu verschiebenden Eckpunkts berechnet, ansonsten wird der Vorgang beendet.
   ii. Wenn das Handmodell streckbar ist:
      a) Der Abstand zwischen dem Handmodell und der Mitte der Knete wird berechnet, und anhand der Differenz zwischen den beiden Abständen im vorderen und hinteren Rahmen wird bestimmt, ob das Handmodell nahe an der Knete liegt.
      b) Wenn sich das Handmodell der Knete entfernt, wird die Position des zu verschiebenden Eckpunkts berechnet, andernfalls wird der Vorgang beendet.
4. Das modifizierte Knete-Netz wird aktualisiert, um den Effekt der Knete-Verformung zu erzielen.
5. Die oben genannten Schritte werden wiederholt, bis der Benutzer die Interaktion beendet.

6.4.2. Einführung in das Skript

Dieses Skript definiert einen booleschen Wert "Verformbar", um zu bestimmen, ob sich die Knete in einem verformbaren Zustand befindet. Wenn der Wert "True" ist, kann die Knete durch die Positionsinformationen des Handmodells verformt werden; wenn der Wert "False" ist, wird die Knete nicht durch das Handmodell beeinflusst und ihre Form bleibt unverändert. Der Benutzer kann die Variabilität der Knete durch Ändern dieses booleschen Wertes steuern, was eine flexiblere und vielfältigere Interaktion ermöglicht.

Wenn der Wert von "Verformbar" True ist, holt sich das Programm zunächst Informationen über die Position des Handmodells. Diese Positionsinformationen können z. B. über den Tracker oder die Sensoren des Handmodells ermittelt werden. Anschließend berechnet das Programm die Bewegungsrichtung des Handmodells, indem es die Position des Handmodells in den Bildern vor und nach dem Handmodell vergleicht, um einen 3D-Vektor zu erhalten.

Konkret zeichnet das Programm die Position des Handmodells im aktuellen Bild und im vorherigen Bild auf und berechnet die Differenz zwischen diesen beiden Positionen, d. h. den Bewegungsvektor des Handmodells. Dieser Bewegungsvektor stellt die Richtung und Entfernung des Handmodells im 3D-Raum dar. Anhand dieses Bewegungsvektors kann das Programm die Flugbahn des Handmodells bestimmen und somit festlegen, wie der Knete verformt werden sollte, um eine natürliche Interaktion mit dem Handmodell zu erreichen.

Die Positionsinformationen des Handmodells werden auch verwendet, um festzustellen, ob sich das Handmodell der Knete nähert. Wenn die Knete drückbar ist und das Handmodell sich der Knete nähert, können die Eckpunkte des Knetmusters zur Mitte der Knete hin verschoben werden, um den Effekt zu erzielen, dass die Knete gedrückt wird.

Das Programm zeichnet die Position des Handmodells und des Mittelpunkts der Knete auf und berechnet den Abstand zwischen ihnen. Ist die Abstandsdifferenz negativ, d. h. der Abstand des aktuellen Bildes ist geringer als der Abstand des vorherigen Bildes, geht das Programm davon aus, dass sich das Handmodell der Knete nähert. An diesem Punkt berechnet das Programm die Position der zu verschiebenden Eckpunkte auf dem Knete Modell auf der Grundlage des Abstands zwischen dem Handmodell und der Knete sowie des Bewegungsvektors. Die Simulation der Interaktion mit dem Handmodell wird durch die Aktualisierung des Modells der Knete erreicht.

6.4.3. Wichtige Parameter

Während der Entwicklung einer Knete-Simulation werden verschiedene wichtige Parameter festgelegt, die bei der Umsetzung der verschiedenen Funktionen der Simulation helfen.

Variablen	Typ	Einführung
„verformbar“	Boolean	Ein boolescher Wert, der verwendet wird, um zu bestimmen, ob der Knete verformt werden kann. Wenn dieser Wert auf "True" gesetzt ist, kann die Knete durch die Positionsinformationen des Handmodells verformt werden. Wenn er auf "False" gesetzt ist, bleibt die Form der Knete unverändert.
„isDruck“	Boolean	Ein boolescher Wert, der verwendet wird, um zu bestimmen, ob sich der Knete in einem drückbaren Zustand befindet. Dieser Wert wird verwendet, um festzustellen, ob der Knete gedrückt werden kann.
„isZug“	Boolean	Ein boolescher Wert, der verwendet wird, um zu bestimmen, ob sich der Knete in einem streckbaren Zustand befindet. Dieser Wert wird verwendet, um festzustellen, ob der Knete gestreckt werden kann.
„isnah“	Boolean	Ein boolescher Wert, der angibt, ob sich das Handmodell der Knete nähert.
„kontaktPunkt“	Vector3	Ein Parameter, der die Koordinaten der Punkte auf dem Modellnetz der Knete bestimmt, die in Kontakt mit dem

		Handmodell stehen, und der zur Berechnung und Durchführung der Simulation der Knete-Verformung verwendet wird.
„einflussRadius“	Fließkommazahl	Ein Parameter, der verwendet wird, um den Einflussbereich des Handmodells auf die Punkte der Knete zu bestimmen. Durch Anpassung dieses Parameters kann der Benutzer den Bereich der Verformung der Knete durch das Handmodell erhöhen oder verringern. Ein größerer Wert führt zu einem größeren Einflussbereich.

Table 6 Wichtige Parameter in Skript

Durch flexible Anpassung dieser Parameter während der Entwicklung können verschiedene Verformungseffekte der Knete erzielt werden, um dem Benutzer ein abwechslungsreiches Interaktionserlebnis zu bieten. Gleichzeitig können diese Parameter erweitert und optimiert werden, um den Bedürfnissen der verschiedenen Nutzer gerecht zu werden.

7. Diskussion

Im Rahmen dieses Praktikums wurde die anfängliche Simulation von VR-Knete erfolgreich abgeschlossen, es gibt jedoch noch einige bekannte Probleme, die noch nicht gelöst wurden. In diesem Kapitel werden diese Probleme und mögliche Lösungen für zukünftige Entwicklungen des Projekts dargestellt.

7.1. Bekannte Probleme und Lösungen

1. Bedienbarkeit: Aufgrund von Zeitbeschränkungen bei der Entwicklung unterstützt das Projekt derzeit nur die einhändige Manipulation des Knetmodells, d.h. nur die Veränderung der Form mit der rechten Hand ist möglich. Eine Steuerung mit der linken Hand kann durch Ändern von Variablen im Programm ermöglicht werden. Um die Form der Knete mit beiden Händen gleichzeitig zu ändern, müssten weitere Variablen hinzugefügt und ein Bewertungsprozess implementiert werden.
2. Interaktivität: Das Knete-modell kann seine Form nicht mehr ändern, nachdem es gegriffen und bewegt wurde. Dieses Problem könnte durch Entfernen und erneutes Hinzufügen der Starrkörperkomponente des Knetobjekts behoben werden. Die Ursache für dieses Problem könnte ein Konflikt zwischen der verformten MESH-Komponente und der Starrkörperkomponente sein.
3. Die Verformung von Knete: Wenn der Controller eine Formänderung des Knetmodells durchführt, nachdem das Modell gedreht wurde, kann es zu einem Richtungsfehler bei der Verformung kommen. Dies liegt daran, dass die Punkte auf dem Modellnetz durch die TRANSFORM-Komponente des Modells positioniert werden und nach der Drehung des Modells die Berechnung der Punkte auf dem Netz falsch ist. Es gibt zwei Lösungsmöglichkeiten: Die erste besteht darin, die Drehung des Modells durch Hinzufügen einer Variablen zu erfassen und diese zur Berechnung der Eckpunkte hinzuzufügen. Die zweite Möglichkeit besteht darin, die Koordinaten der Punkte auf dem Modellnetz nach der Drehung neu zu definieren.
4. Verformungsgenauigkeit: Da sich die Form des Knetmodells durch Änderung der Punkte auf dem Gitter ändert, hängt die Genauigkeit der Modelländerung von der Dichte der Eckpunkte auf dem Knete Gitter ab. Wenn der Abstand zwischen zwei Eckpunkten zu groß ist, kann der Benutzer die Form des Modells zwischen den beiden Eckpunkten nicht ändern. Eine mögliche Lösung des Problems besteht darin, den Abstand zwischen den beiden nächstgelegenen Eckpunkten zu berechnen und einen Schwellenwert festzulegen; wenn der Abstand diesen Schwellenwert überschreitet, wird ein dazwischen liegender Eckpunkt hinzugefügt und das Modell aktualisiert.

5. Gesamtvolumen der Knete: Im Programm wird das Modell der Knete durch Vermaschung berechnet und das Gesamtvolumen der Knete ändert sich nach der Verformung. Das bedeutet, dass das Knetmodell unbegrenzt gedehnt oder gestaucht werden kann, was nicht den Gesetzen der Physik entspricht. Eine mögliche Lösung besteht darin, mehrere Kollisionskörper in das Modell einzufügen, um das Gesamtvolumen der Knete zu kontrollieren.

7.2. Möglichkeiten der Programmoptimierung

1. Realismus: Obwohl die VR-Simulation von Knete Veränderungen in der Form der Knete simulieren kann, ist es schwierig für den Benutzer, den Realismus und die physikalischen Eigenschaften der Knete wirklich zu spüren, aufgrund des Fehlens eines realistischen Gefühls für Berührung und Gewicht. Dies beeinträchtigt das Eintauchen des Benutzers und die Qualität der Erfahrung. Techniken wie Vibrations- und Kraftrückkopplung können hinzugefügt werden, um die Interaktion des Benutzers mit der virtuellen Knete realistischer zu gestalten und die Qualität der Benutzererfahrung zu verbessern.
2. Mehr interaktive Funktionen: Aktuelle VR-Simulationsanwendungen für Knete bieten in der Regel nur grundlegende Funktionen zur Form- und Farbanpassung und lassen komplexere interaktive Funktionen vermissen. Benutzer können beispielsweise keine fortgeschrittenen Techniken wie Schneiden und Bildhauerei anwenden, um vielfältigere Formen und Strukturen zu schaffen. Zusätzliche Technologien wie Gestenerkennung und Eye-Tracking könnten eingeführt werden, um komplexere Interaktionsfunktionen zu ermöglichen. Dies würde den Benutzern mehr Freiheit beim Erschaffen und Erforschen der Formen und Strukturen der Knete geben.
3. Pädagogische Elemente verstärken: VR-Anwendungen für simuliertes Kneten sollten nicht nur ein unterhaltsames Erlebnis bieten, sondern auch pädagogische Elemente enthalten. Während der grundlegende Prozess des Knetens durch Beobachtung und Übung verstanden werden kann, mangelt es oft an vertieftem Wissen und der Erforschung der Eigenschaften, Merkmale und Verwendungsmöglichkeiten von Knete. Durch das Hinzufügen von mehr Hintergrundwissen und Anleitungen können VR-Simulationsanwendungen für Knete in pädagogische Anwendungen umgewandelt werden. Dies wird den Benutzern helfen, das Wissen über die Eigenschaften, Merkmale und Verwendungsmöglichkeiten von Knete besser zu verstehen.
4. Verbesserung der sozialen Interaktion: VR-Simulationsanwendungen für Knete werden in der Regel von einer einzelnen Person genutzt, und es fehlt das Element der sozialen Interaktion. Dies bedeutet, dass die Nutzer nicht in der Lage sind, sich mit anderen Nutzern auszutauschen

und zusammenzuarbeiten, wodurch das soziale und pädagogische Potenzial der App eingeschränkt wird. Der soziale und pädagogische Charakter von VR-Simulationsknete-Apps kann durch das Hinzufügen von Multiplayer- und Kollaborationsfunktionen verbessert werden. Dadurch würde es den Nutzern ermöglicht, sich auszutauschen und zusammenzuarbeiten, um gemeinsam ein breiteres Spektrum an Knetprozessen zu erstellen und zu erforschen.

5. Spannungssimulation: Eine weitere mögliche Entwicklung ist die Verbesserung der Spannungssimulationsfunktion. Die derzeitigen VR-Simulationsanwendungen für Knete können zwar Formveränderungen der Knete simulieren, nicht jedoch die Verformung und den Bruch von Knetmaterialien unter Einwirkung äußerer Kräfte. Verbesserte Spannungssimulationsfunktionen könnten die physikalischen Eigenschaften von Knete genauer simulieren und so den Realitätssinn und das Eintauchen des Benutzers erhöhen. Darüber hinaus können Spannungssimulationen in Bildung und Forschung eingesetzt werden, z. B. zur Simulation von Kräften auf verschiedene Materialien und Formen, um den Benutzern ein besseres Verständnis von Aspekten wie Materialwissenschaft und technischer Mechanik zu vermitteln.
6. Kombination von virtueller Realität und erweiterter Realität: VR-Simulationsanwendungen für Knete können auch mit der Technologie der erweiterten Realität kombiniert werden, um ein reichhaltigeres und realistischeres Erlebnis zu schaffen. So können die Nutzer beispielsweise mit Hilfe der erweiterten Realität Knetformen in der realen Welt simulieren und dann die VR-Technologie nutzen, um sie in einer virtuellen Umgebung weiter zu erforschen und anzupassen. Diese Kombination kann die Grenzen zwischen virtueller und realer Welt aufheben und so eine freiere und innovativere kreative Erfahrung ermöglichen.

8. Ausblick

Mögliche Erweiterungen für die Zukunft der VR-Knet-Simulationsanwendungen könnten auch die Integration von haptischem Feedback, Augmented Reality (AR) und künstlicher Intelligenz (KI) umfassen.

Durch die Integration von haptischem Feedback können die Benutzer nicht nur die visuellen, sondern auch die taktilen Aspekte der Knete-Simulation in VR erleben. Dadurch wird das Eintauchen in die virtuelle Umgebung noch realistischer und immersiver.

Die Integration von AR in die Knete-Simulationsanwendungen könnte es den Benutzern ermöglichen, ihre Kreationen in der realen Welt zu betrachten und zu interagieren. Zum Beispiel könnten die Benutzer ihre Kreationen auf eine reale Oberfläche projizieren und dann weiter daran arbeiten.

Die Integration von KI in die Knete-Simulationsanwendungen könnte es den Benutzern ermöglichen, vorgefertigte Formen und Vorlagen automatisch zu generieren und so den kreativen Prozess zu beschleunigen. KI-Unterstützung könnte auch dazu beitragen, komplexe Knete-Simulationsprozesse automatisch zu optimieren und zu verbessern.

Zusammenfassend gibt es viele Möglichkeiten, die Zukunft der VR-Knete-Simulationsanwendungen zu erweitern und zu verbessern, und es wird spannend sein zu sehen, welche neuen Technologien und Innovationen in Zukunft entwickelt werden, um die Benutzererfahrung weiter zu verbessern.

Literaturverzeichnis

[Al15] Alaraj, A. et al.: Virtual reality cerebral aneurysm clipping simulation with real-time haptic feedback. Neurosurgery 0 2/11, S. 52, 2015.

[Ch16] Christoph Anthes et al. Hrsg.: State of the art of virtual reality technology. IEEE, 2016. ISBN: 1467376760.

[De22] Dejan Gajsek: Unity vs Unreal Engine for XR Development: Which One Is Better?

[HCS06] Hambli, R.; Chamekh, A.; Salah, H. B. H.: Real-time deformation of structure using finite element and neural networks in virtual reality applications. Finite elements in analysis and design 11/42, S. 985–991, 2006.

[He22] Henry_Lau617: Unity XR Interaction Toolkit 中 Action 与 Device 的差异探究.

[Ho12] Ho, A. K. et al.: Virtual reality myringotomy simulation with real-time deformation: development and validity testing. The Laryngoscope 8/122, S. 1844–1851, 2012.

[Hu21] Huang, R.: Konzept eines Biotech-Hauses in Virtual Reality am Beispiel der Joghurt-Herstellung, Köthen, 2021.

[Mi21] Mingjian Liu: xR-Szenensynchronisation zwischen unterschiedlichen Plattformen, 2021.

[Pa18] Panggung Sutapa et al. Hrsg.: Differences of Influence of Playing Playdough and Puzzles on Fine Motor Skills and Logical-Mathematical Intelligence in Early Childhood. Atlantis Press, 2018. ISBN: 9462526346.

[Pr20] Programmerwiki: Unity vs ue4, Wie wählen die Virtual Reality-Entwicklungsmotoren aus?

[Sc17] Schulze, G.: Entwurf und Implementierung einer dynamischen Softwareplattform für konfigurier-und erweiterbare Simulationen auf Basis der Unity Engine, 2017.

[Un21] Unity Manual: The GameObject in the Unity Editor. https://docs.unity3d.com/cn/current/Manual/GameObjects.html.

Anhang

Code von KneteMeshkontroll:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;
[RequireComponent(typeof(MeshFilter))]

public class KnetenMeshKontroll : MonoBehaviour
{
    public Transform leftHand;
    public Transform rightHand;
    public Transform hand;
    public bool isVerformbar = false;
    public bool isKontakt = false;
    public bool isDurck = false;
    public bool isZug = false;

    Vector3 positionAbweichung1BildVector;
    Vector3 positionAbweichung2BildVector;
    Vector3 positionAbweichung;

    float Abstand1Bild;
    float Abstand2Bild;

    private bool isnah = false;
    private Mesh mesh;
    private Vector3[] vertices;
    private float influssRadius;
    private Vector3 contactHandPoint;
    private Vector3 position;
    void Start()
    {
        hand = kleinKugel;
        mesh = GetComponent<MeshFilter>().mesh;
        influssRadius = kleinKugel.gameObject.GetComponent<SphereCollider>().radius*0.1f;

    }
    // Update is called once per frame
    void Update()
    {
        position = GetComponent<Transform>().position;
        Verformbar();

    }
    IEnumerator Abstand(Vector3 positionHand)
    {
        Vector3 position2Bild = GetComponent<Transform>().position;
        Vector3 position2BildHand = positionHand;
        positionAbweichung2BildVector = position2Bild - position2BildHand;
        float x2 = positionAbweichung2BildVector.x;
        float y2 = positionAbweichung2BildVector.y;
        float z2 = positionAbweichung2BildVector.z;
        if (Abstand1Bild == 0)
            Abstand2Bild = 0;
        else
            Abstand2Bild = Mathf.Sqrt(x2 * x2 + y2 * y2 + z2 * z2);
        if ((Abstand2Bild - Abstand1Bild) > 0)
        {
            isnah = false;
            Abstand1Bild = 0;
            Abstand2Bild = 0;
        }
        if ((Abstand2Bild - Abstand1Bild) < 0)
        {
            isnah = true;
            Abstand1Bild = 0;
```

```
66.             Abstand2Bild = 0;
67.         }
68.         positionAbweichung = positionAbweichung2BildVector - positionAbweichung1BildVector;
69.         yield return null;
70.         Vector3 position1Bild = position2Bild;
71.         Vector3 position1BildHand = position2BildHand;
72.         positionAbweichung1BildVector = position1Bild - position1BildHand;
73.         float x1 = positionAbweichung1BildVector.x;
74.         float y1 = positionAbweichung1BildVector.y;
75.         float z1 = positionAbweichung1BildVector.z;
76.         Abstand1Bild = Mathf.Sqrt(x1 * x1 + y1 * y1 + z1 * z1);
77.     }
78.     void Verformbar()
79.     {
80.         contactHandPoint = hand.transform.position;
81.
82.         StartCoroutine(Abstand(hand.position));
83.         if (isVerformbar == true)
84.         {
85.             if (GetComponent<MeshCollider>().convex == true)
86.                 GetComponent<MeshCollider>().convex = false;
87.             if (isKontakt)
88.             {
89.                 StartCoroutine(MeshControll());
90.             }
91.         }
92.     }
93.
94.     IEnumerator MeshControll()
95.     {
96.         vertices = mesh.vertices;
97.
98.         if (isDurck)
99.         {
100.            if(isnah)
101.            {
102.                for (int i = 0; i < vertices.Length; i++)
103.                {
104.                    if (vertices[i].x + position.x < contactHandPoint.x + influssRadius &&
vertices[i].x + position.x > contactHandPoint.x - influssRadius)
105.                    {
106.                        if (vertices[i].y + position.y < contactHandPoint.y + influssRadius
&& vertices[i].y + position.y > contactHandPoint.y - influssRadius)
107.                        {
108.                            if (vertices[i].z + position.z < contactHandPoint.z +
influssRadius && vertices[i].z + position.z > contactHandPoint.z - influssRadius)
109.                            {
110.                                vertices[i] -= positionAbweichung * 0.5f;
111.                            }
112.                        }
113.                    }
114.                }
115.            }
116.        }
117.        if (isZug)
118.        {
119.            if (!isnah)
120.            {
121.                for (int i = 0; i < vertices.Length; i++)
122.                {
123.                    if (vertices[i].x + position.x < contactHandPoint.x + influssRadius*2 &&
vertices[i].x + position.x > contactHandPoint.x - influssRadius*2)
124.                    {
125.                        if (vertices[i].y + position.y < contactHandPoint.y +
influssRadius*2 && vertices[i].y + position.y > contactHandPoint.y - influssRadius*2)
126.                        {
127.                            if (vertices[i].z + position.z < contactHandPoint.z +
influssRadius*2 && vertices[i].z + position.z > contactHandPoint.z - influssRadius*2)
128.                            {
129.                                vertices[i] -= positionAbweichung * 0.5f;
```

```
130.                             }
131.                         }
132.                     }
133.                 }
134.             }
135.         }
136.         mesh.vertices = vertices;
137.         mesh.RecalculateNormals();
138.         yield return 1;
139.         Destroy(GetComponent<MeshCollider>());
140.         gameObject.AddComponent<MeshCollider>();
141.
142.     }
143. }
```