AURA - Technischer Bericht

Jonas Bien B.Sc., Tim Bienias B.Sc., Fabian Brenner B.Sc., Lukas Hutfleß B.Sc.,
Paul Kramp B.Sc., Kristian Krömmelbein B.Sc., Karsten Lam B.Sc., Arnold Lockstein B.Sc.,
Tobias Michalski B.Sc., Dirk Peters B.Sc., David Reinert B.Sc., Fabian Seidl B.Sc.
und Sven Steininger B.Sc.
Fachbereich Informatik, Hochschule Darmstadt
Haardtring 100, 64295 Darmstadt

1. Einleitung

Das Grafiklabor der Hochschule Darmstadt bietet Masterstudierenden die Möglichkeit am Projekt Systementwicklung teilzunehmen. Im Sommersemester 2018 partizipierte eine Gruppe von 15 Studierenden am Projekt "Bühnenbilder mit der HoloLens". Das Projekt fand unter der Schirmherrschaft von Frau Prof. Dr. Elke Hergenröther statt und wurde zusätzlich vom Unternehmen Bosch-Rexroth durch Alexander Kunkel begleitet. Diese Veranstaltung baute auf dem Projektstand auf, der im vorigen Semester erreicht wurde [1]. Der Schwerpunkt der Tätigkeiten lag darauf, zusätzliche Features zu implementieren und das Projekt zu stabilisieren.

Herausforderungen bestanden hierbei aus:

- Implementierung neuer Features
- Erweiterung der Funktionalität vorhandener Features
- Integration neuer Teammitglieder

Diese Ausarbeitung dient als Abschlussbericht des Projekts und zeigt die jeweiligen Aktivitäten der Projektteilnehmer auf.

Der Bericht ist wie folgt gegliedert: Abschnitt 2 befasst sich mit dem Projektmanagement. Folgend wird in Abschnitt 3 das User-Interface betrachtet. Kapitel 4 erörtert das *SceneReset*-Feature. Absatz 5 beschreibt die Minimap-Funktionalität mittels Übersichtskarte. In Abschnitt 6 gehen die Autoren auf die Aspekte des Texturing ein. Kapitel 7 beschäftigt sich mit der Realisierung der Tutorials und Kapitel 8 beschreibt die *PosiStageNet* Visualisierung. Abschnitt 9 widmet sich dem Multiplayer-Feature. Bevor in Kapitel 11 ein Fazit gezogen wird, geht Abschnitt 10 auf die Portierung der Software auf Android ein.

2. Projektmanagement

Dieser Abschnitt erläutert das Vorgehen beim Projektmanagement. Betrachtet werden hierbei zeitliche Rahmenbedingungen, gesetzte Ziele, Workflow, eingesetzte Tools, einige Statistiken sowie Namensgebung und die Veröffentlichung des Projekts.

2.1. Timeline

Das Projekt wurde vom 05.04.2018 bis zu seinem Ende am 28.06.2018 terminiert. Die Gesamtdauer entsprach einem zeitlichen Umfang von 12 Wochen. Unterteilt wurde diese Zeitspanne in fünf Sprints:

- Sprint 1: 05.04.2018 - 19.04.2018 (2 Wochen)
- Sprint 2: 19.04.2018 - 10.05.2018 (3 Wochen)
- Sprint 3: 10.05.2018 - 24.05.2018 (3 Wochen)
- Sprint 4: 24.05.2018 - 07.06.2018 (2 Wochen)
- Sprint 5: 07.06.2018 - 14.06.2018 (1 Woche)

Die übrigen zwei Wochen vom 14.06 bis zum 28.06 wurden als Zeitpuffer, für Aufräumarbeiten und für Dokumentationszwecke eingeplant.

2.2. Semesterziele

Definiert wurden drei Kategorien von Zielen, die jeweils nach Priorität unterteilt wurden. In der ersten Kategorie höchster Priorität wurden Must-Have-Features gelistet. Die zweite Kategorie, mittlere Priorität, beinhaltete Begeisterungsmerkmale, während die Kategorie niedriger Priorität Verbesserungen vorhandener Features oder die Implementierung experimenteller Features vorsah.

2.2.1. Hohe Priorität. Zu den Semesterzielen mit hoher Priorität gehörten die Veröffentlichung des Projekts, die Verbesserung der Nutzeroberfläche und das *SceneReset*-Feature.

Zur Veröffentlichung des Projekts war es notwendig den Code zu säubern, eine robuste Softwarearchitektur zu implementieren, eine umfassende Dokumentation zu erstellen und eine Veröffentlichungsstrategie, bezogen auf Lizenzauswahl, zu entwickeln. Dieses Ziel konnte nicht umfassend abgeschlossen werden, da bis zu diesem Zeitpunkt keine robuste Softwarearchitektur implementiert wurde.

Die Verbesserungen der Nutzeroberfläche betrafen insbesondere die Vereinheitlichung der UI-Elemente und der Nutzung bereits vorhandener Elemente aus dem HoloLens-Toolkit. Dieses Ziel wurde erfolgreich erreicht.

Ein weiteres, hoch priorisiertes Ziel war die Implementierung des *SceneReset*, um eine geladene Szene zu nullen. Auch dies wurde erfolgreich in die Software eingebaut.

2.2.2. Mittlere Priorität. Ziele dieser Kategorie waren zum einen Multiplayer-Funktionalität, sodass kooperatives Arbeiten von HoloLens zu HoloLens bzw. HoloLens zu PC ermöglicht wurde. Diese Funktionalität wurde umfassend bereitgestellt.

Zum anderen galt es einen Texture-Manager zu implementieren, um Texturen dynamisch zu wechseln. Dies wurde nicht erreicht, da es Abhängigkeiten zum neuen Asset-Server gab, welcher bis zu diesem Zeitpunkt nicht fertiggestellt wurde.

Ein weiteres Ziel mittlerer Priorität war es die Software kompatibel mit dem PosiStageNet zu machen. Dies wurde auch mit einem positivem Ergebnis abgeschlossen.

2.2.3. Niedrige Priorität. Zu den Zielen niedriger Priorität gehörten die Überarbeitung der Gizmos. Hierbei war es notwendig die Benutzerfreundlichkeit zu verbessern und kleinere Bugs zu entfernen. Dies wurde erfolgreich umgesetzt.

Die Portierung des Asset-Servers auf das NodeJS-Framework wurde begonnen, jedoch bis zum jetzigen Zeitpunkt nicht fertiggestellt.

Um neuen Nutzern eine Einführung zu geben, wurde eine Tutorial-Funktionalität vorgesehen. Diese wurde in einer ersten Version erfolgreich implementiert.

Da die Nutzung im Multiplayer auch auf mobile Endgeräte ausgeweitet werden sollte, wurde die Portierung der Software auf Android mittels eines Prototypen realisiert.

Der letzte Punkt niedrig priorisierter Features war es die Spatial-Mapping-Funktionalität vollumfänglich in die Software zu integrieren. Dies ist auch mit Erfolg gelungen.

2.2.4. Zusammenfassung Ziele. Der größte Teil der gesteckten Ziele, vor allem die für den Projekterfolg ausschlaggebenden Ziele bzw. Features, wurde erreicht. Dies machte die in den Planungen enthaltenen Abschätzungen zu einem Erfolg.

2.3. Implementierte Features

Zu den implementierten Features gehören:

- UI - Vereinheitlichung und Verbesserung
- Spatial Mapping - Integration in Software
- Multiplayer - kooperatives Arbeiten mit mehreren Geräten
- Minimap - Übersicht der aktuellen Szene
- Scene-Reset - Nullen der Szene
- PSN - PosiStageNet Kompatibilität
- Gizmos - Objekte durch Gesten manipulieren

2.4. Workflow & Tools

Als Entwicklungsmethode wurde eine Mischung aus Scrum und Kanban eingesetzt, die sich bereits im vorigen Semester bewährt hatte. Der Workflow orientierte sich am PDCA-Zyklus [2]. Zuerst wurden alle nötigen Schritte für eine Phase bzw. einen Sprint geplant. Daraufhin wurden die Planungen umgesetzt und anschließend das Ergebnis analysiert. Aus dieser Analyse gingen dann Anpassungen hervor, die bei der nächsten Planungsphase berücksichtigt wurden.

Zu den im Projekt eingesetzten Tools gehörten:

- Discord - Kommunikation
- GitLab - Source-Code-Verwaltung und Projektmanagement
- Visual Studio - IDE
- Unity - Game-Engine
- Mixed Reality Toolkit (MRTK) - Unity-Framework für HoloLens
- Vuforia - Unity-Framework für HoloLens

2.5. Statistiken & Fakten

Die von den Entwicklern geloggte Zeit für alle Aufgaben entsprach einem Umfang von ca. 860 Stunden. Dabei wurden über 360 Commits mit ca. 11700 Codezeilen hochgeladen. Der am meisten genutzte Wochentag für Projektarbeiten war Donnerstag (siehe Abbildung 1).

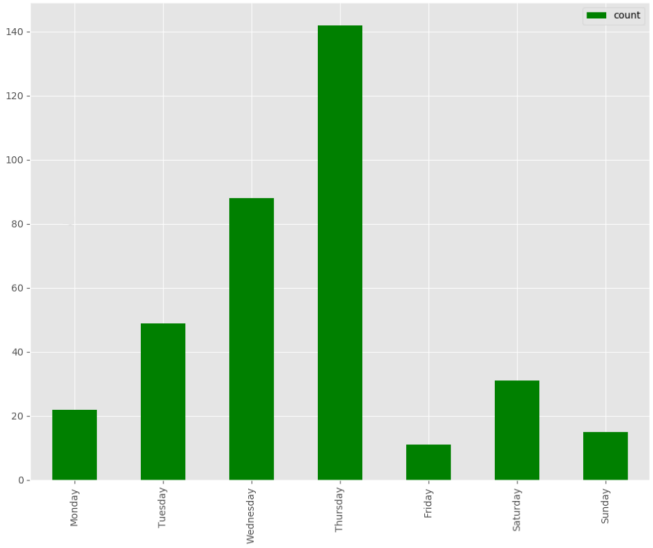


Abbildung 1. Veranschaulichung der Commits pro Wochentag

2.6. Namensgebung

Das Projekt wurde nach einer Abstimmung auf den Namen *AURA* getauft. Weitere Namen, die es in die engere Auswahl geschafft hatten waren: *OSCAR*, *ARES* und *KingLens*.

2.7. Veröffentlichung des Projekts

Um die Software unter einer Open-Source-Lizenz zu veröffentlichen, musste recherchiert werden unter welchen Bedingungen dies möglich ist. In den folgenden Abschnitten werden die dabei entstandenen Probleme, Lösungsansätze und die schlussendliche Umsetzung beschrieben.

2.7.1. Lizenzauswahl. Zur quelloffenen und unbeschränkt nutzbaren Veröffentlichung ist es notwendig eine passende Lizenz zu wählen, sodass Dritte die Software für eigene bzw. kommerzielle Projekte verwenden können.

Nach Analyse gängiger Lizenzmodelle in Form von *(L)GPL*, *BSD* und *MIT* wurde entschieden, dass die praktikabelste Lösung *LGPLv2* ist [3]. Diese erlaubt es dem Nutzer der Software diese mit dynamisch gelinkter Drittsoftware zu kombinieren, ohne dass diese Drittsoftware Bedingungen der *LGPLv2* unterliegt. Dabei wird der unter *LGPLv2* gestellte Teil des Programms auch bei Weitergabe oder Verkauf des Gesamtprodukts weiter unter *LGPL* lizenziert. Wenn eine dritte Partei Änderungen am *LGPLv2*-Teil durchführt oder ihn statisch in einem Programm einbettet, müssen diese Programmteile auch unter die *LGPLv2*-Lizenz gestellt werden.

2.7.2. Haftungsausschluss und Nutzungsrechte. In der Bundesrepublik Deutschland ist es nicht möglich einen vollständigen Haftungsausschluss per Lizenzgebung zu formulieren [4]. Dies beruht auf der Tatsache, dass Open-Source-Lizenzen rechtlich gesehen als AGB zu verstehen sind. Zumindest für grob fahrlässig oder mutwillig zugeführte Schäden ist der Lizenzgeber immer haftbar. Dies kann eingeschränkt, jedoch nie ganz ausgeschlossen werden. Auch das Einräumen von Nutzungsrechten ist nur mit der Zustimmung aller Autoren möglich. Um die Probleme, die Haftungsausschluss und Abtretung der Nutzungsrechte mit sich bringen zu lösen, wurde beschlossen, dass eine übergeordnete Organisation die Veröffentlichung des Projekts übernimmt. Hierzu war es notwendig, dass das gesamte Team seine Rechte am Code vertraglich an einen Verein abgetreten hat.

2.7.3. Verein Lab^3. Um maximalen Haftungsausschluss und einfachere Verwaltung der Nutzungsrechte zu ermöglichen, wurden Möglichkeiten evaluiert, um rechtliche Aspekte auszulagern. Vereine sind rechtlich als juristische Personen einzustufen, was sie auch in die Lage versetzt als Lizenzgeber aufzutreten. Folgende Szenarien wurden evaluiert:

- Gründung eines eigenen Vereins
- Übertragung der Rechte an einen Hochschulverein
- Übertragung der Rechte an den Verein Lab^3

Um einen Verein zu gründen, müssen rechtliche Rahmenbedingungen erfüllt sein [5]. Da diese Rahmenbedingungen nicht von den Projektteilnehmern erfüllt werden konnten, wurde die Gründung eines eigenen Vereins verworfen. Die Option für die Rechteübertragung an den Hochschulverein ZAI wurde auch nicht weiter in Betracht gezogen, da dieser Verein lediglich für hochschulrelevante Aktivitäten genutzt wird. Als beste Option wurde eine Kooperation mit dem Verein Lab^3 angestrebt. Die Rechte wurden an diesen Verein übertragen und als Gegenleistung eine Kooperation zwischen Verein und Hochschule Darmstadt vereinbart.

3. Asset-Server 2.0

Eine der größeren Baustellen des ersten Prototypen bestand in der Weiterentwicklung des bestehenden Asset-Servers in ein vollständiges Produkt, welches Wart- und Erweiterbar sein sollte. Hierbei wurden auf neue Technologien gesetzt um den bestehenden Kern zu ersetzen.

3.1. Architektur

Es wurde sich entschieden von einem Python-basierenden Server auf Node umzusteigen. Node ist eine Hochaktuelle und moderne Javascript-Umgebung die Javascript-Code innerhalb einer Shell ausführen kann. Bereichert wird dies durch eine umfangreiche Bibliothek von Drittbibliotheken, die fast jeden Funktionsumfang abdecken.

3.1.1. Express. Es wurde sich gerade aufgrund der guten Anbindung zu Node und der Zielsetzung als API-Framework für Express als Grundgerüst entschieden. Express ist ein minimales Web-Framework, welches mächtige Tools für das Erstellen einer REST API besitzt und sowohl auf Linux als auch auf Windows Problemlos funktioniert. Im früheren Asset-Server war die Kernfunktionalität fast kaum kommentiert und auch nicht mit sog. "Best Practices"vorgenommen. In diesem Kontext gibt es das Prinzip von "Routing"bei API-Design, bei der Funktionalität in selbstgeschriebene Middleware aufgeteilt wird. Dies ermöglicht das Aneinanderketten von Funktionalität wie es bei dem alten Server nicht möglich war. Der Fokus lag hierbei sehr auf zukünftiger Erweiterbarkeit und Wartbarkeit.

3.2. Installations-Skript

Durch Benutzung von Node gab es eine weitere Möglichkeit den Entwicklungsprozess zu vereinfachen. Der Package-Manager von Node, NPM, unterstützt die Versionierung von Bibliotheken die das Projekt benötigt. So ist durch eine Konfigurationsdatei der Prozess eine laufende Entwicklungsumgebung aufzusetzen reduziert worden auf den Aufruf

```
npm install
```

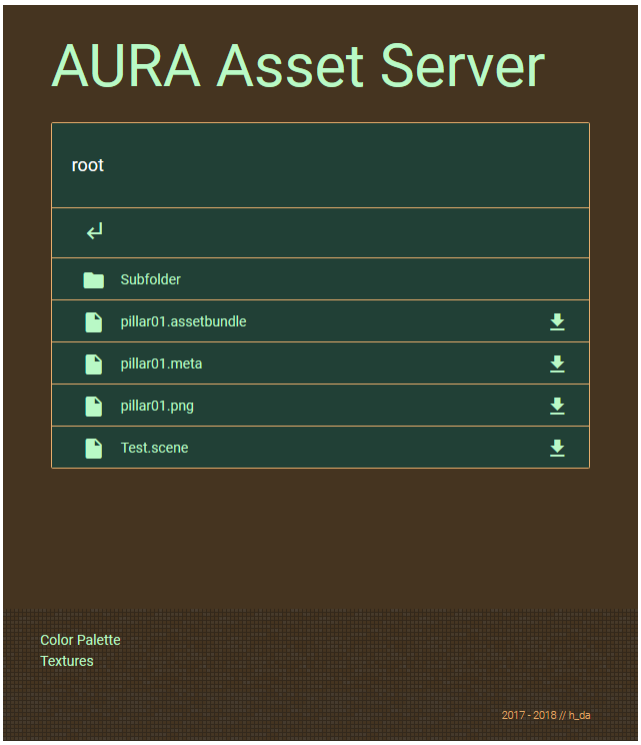
Dies funktioniert Plattformunabhängig und stellt eine deutliche Verbesserung gegenüber dem Vorgänger dar.

3.3. Lazy loading

Der alte Asset-Server hatte deutliche Performanzprobleme, da viele Operationen mehrfach, bzw. unnötig oft ausgeführt wurden. So wurden z.B. sämtliche Icons der Assets auf einmal geladen, welches eine deutliche Verzögerung beim Endnutzer verursachte.

Im Zuge der Neueentwicklungen wurde dementsprechend sogenanntes "Lazy Loadingïmplementiert. Dies sorgt durch nachladen bei bedarf dazu, dass die Stoßdatenmenge deutlich reduziert wurde. Dies funktioniert in beide Richtungen. Das Laden eines Icons sorgt automatisch schon dafür, dass die Grundgerüste für das Asset angelegt werden und nicht neu generiert werden müssen, wenn das Asset tatsächlich geladen wird. Genausosehr wird das Icon für Assets einzeln mitgeladen, denn die zusätzliche Datenmenge fällt dabei kaum ins Gewicht.

Im Sinne der zukünftigen Erweiterbarkeit wurde dabei gleich schon die Möglichkeit angelegt beliebige Metadaten auf die gleiche Art und Weise zu laden, obwohl die Applikation an sich dies noch nicht unterstützt.



3.3.1. Thumbnails. Insbesondere die Behandlung der Thumbnails wurde drastisch verbessert. Der alte Server hatte alle Thumbnails automatisch bei der Generierung der Asset-Liste mit geladen. Diesbezüglich wurde eine Möglichkeit implementiert, Thumbnails auch einzeln zu laden. Gerade beim initialen Aufruf der Anwendung bringt dies einen deutlichen Geschwindigkeitsvorteil mit sich.

3.4. Web Interface

Der neue Asset Server unterstützt ein Vollfunktionierendes Web-Interface, welches die Dateistruktur auf dem Server wiederspiegelt und Informationen über gespeicherte Assets liefern kann. Dies ist besonders für eingebettete Systeme vorteilhaft bei der ein schnelles Nachschauen oder Herunterladen von Assets nicht Problemlos möglich ist.

Dabei ist zu erwähnen, dass die Erkennung ob sich ein Web-Browser in die API einwählt automatisch geschieht durch lesen der HTTP-Header.

3.4.1. Ordnerübersicht. Standardmäßig bildet das Web-Interface die Ordnerstruktur des Servers wieder. Aus Sicherheitsgründen kann das Interface allerdings nicht den konfigurierten Projektpfad verlassen. Auch werden alle Dateipfade sowohl für die Hololens als auch für das Web-Interface von ihrer physischen Pfadstruktur bereinigt. Ein Durchsickern von Informationen des Servers ist somit unmöglich.

Zu diesem Zweck sind alle Pfade des Servers als URI, Üniform Resource Identifierßu verstehen. Ressourcen, in diesem Falle die Assets, werden darüber eindeutig identifiziert.

3.4.2. Dateiansicht. Durch das Auswählen einer Datei können Details über diese angezeigt werden. Momentan beschränkt sich dies auf die Größe der Datei, aber jede beliebige Information kann auf diese weise Dargestellt werden.

Im Zuge der Entwicklung war jedoch schnell klar, dass weitere Informationen zum jetzigen Zeitpunkt nicht benötigt werden.

3.4.3. Fehlerbehandlung. Alle gängigen HTML Statuscodes wurden implementiert. Obwohl nicht alle von der Applikation bisher verwendet werden, ist dies dennoch ein wichtiger Schritt in Richtung Zukunftsträchtigkeit. So werden z.B. korrekte 404-Codes gesendet für URI die nicht existieren. Es wurde außerdem das Code-500 Protokoll implementiert. Dies wird von der Applikation momentan nur vereinfacht behandelt, da andere Teile der Anwendungen diese Information schlichtweg nicht verwenden. Es gibt es eine Schnittstelle auf Seiten der Hololens die das Aufrufen mehrerer Events aus der Netzwerkumgebung steuert. Zukünftige Module müssen lediglich auf diese zugreifen.

3.5. Rekursive Suche

Oft erscheint es unnötig, die gesamte Dateistruktur des Servers zu durchsuchen um eine Liste von Assets zu erhalten. In diesem Zuge wurde eine Methode zur selektiven Unterordnerdurchsuchung implementiert. Dies ermöglicht das Erhalten einer Teilliste aller gespeicherten Assets. Dabei ist es sowohl möglich nur einen Ordner (und seinen Inhalt) anzufordern, oder die Unterordner zu beinhalten.

3.6. Event System

Mehrere auf Callbacks basierende Aufrufe wurden durch ein Vollständiges Event-System erweitert, durch die ein signifikanter Entwicklungsvorteil entsteht. Eine logische Schnittstelle wurde erstellt, welche sinnvolle Anbindungen an die Netzwerkschnittstelle ermöglicht.

3.7. Class Hierarchy

Bisher wurden Assets nur als Gameobjects mit einigen losen Komponenten betrachtet. Im Zuge der Weiterentwicklung wurde eine ordentliche Klassenstruktur für Assets, Icons und Metadaten entwickelt, und an verschiedenen Teilen der Anwendung implementiert. Dabei wurde darauf Wert gelegt, dass Zukünftige Felder, wie z.B. Metadaten in beliebiger Art und weise erweitert werden können.

3.8. Ausblick

Im Fazit wurden signifikante Verbesserungen gegenüber dem Asset-Server 1.0 vorgenommen. Aus einer einzigen Python-Datei ohne jedwede Dokumentation wurde ein mächtiges Tool zur Auslieferung von Assets entwickelt die Problemlos jede Weiterentwicklung der eigentlichen Applikation meistern wird.

4. UI (User Interface)

Die Aufgabe der UI-Gruppe besteht zum einen darin, einen Asset Store zur Verfügung zu stellen, der die Assets vom Server anzeigt und diese durch Auswahl in die Szene lädt. Zum anderen soll die Manipulation von Objekten in der Szene ermöglicht werden.

4.1. Konzeptphase

Im vorherigen Semester haben wir uns zunächst ein Konzept für das UI erarbeitet. Dabei entschieden wir uns letztendlich für den World-locked Ansatz. Vorteil dabei ist, dass der Nutzer das UI an geeigneter Stelle platzieren kann und anschließend an Objekten in der Szene weiterarbeitet. Möchte sich der Nutzer nun ein neues Objekt holen, braucht er sich beispielsweise nur zum UI zu drehen und kann direkt weiterarbeiten. Weiterhin erstellten wir einen ersten funktionalen Prototypen für den Asset Store. Dabei ging es uns in erster Linie um die Funktionalität. Das Design stand dabei im Hintergrund.

In diesem Semester haben wir es uns zur Aufgabe genommen, das UI einheitlich zu gestalten und allgemein nutzbarer zu machen.

4.2. UI Asset Store

Abbildung 2. Aktueller Asset Store Prototyp

Es wurden Verbesserungen am Asset Store vorgenommen. Dabei war es uns wichtig, nach den grundlegenden Designprinzipien der “Human–computer interaction”, wie sie etwa Jacobsen beschreibt, zu arbeiten. Dadurch erhoffen wir uns eine leichter und vor allem intuitiver zu bedienende Benutzeroberfläche zu erhalten.

Hierbei beziehen wir uns vor allem auf das Prinzip der “Nachbarschaft”, welche besagt, dass thematisch gleiche Oberflächeninformationen und Interaktionen nahe beieinander stehen.

Ebenfalls wichtig war uns die Ähnlichkeit. Diese hilft dem User zum Beispiel Buttons oder andere Objekte als solche zu erkennen, was die Interaktion deutlich intuitiver machen

kann. Hierfür stellten wir zum Beispiel Prefabs für Buttons bereit, damit alle Entwickler ohne Komplikationen Buttons mit Leichtigkeit hinzuzufügen können und dabei sicherstellen, dass diese designtechnisch zur restlichen UI passen, ohne dass die Entwickler das Design beachten müssen.
Für das Design der Buttons beschäftigten wir uns mit "Fitts' Gesetz". Dieser setzt die benötigte Zeit, um eine Zielfläche zu erreichen, in Abhängigkeit zu der Distanz zu dieser und der benötigten Breite des Buttons. So führten wir einige rudimentäre Selbsttests durch, bis wir eine angenehme Größe der Buttons erreichten. Weiter kümmerten wir uns um die Symmetrie der Benutzeroberfläche. Diese hilft dem User sich in der UI zu orientieren und erhöht so die Arbeitsgeschwindigkeit.

4.2.1. Assets im Store. Die Hauptaufgabe des UI Asset Stores liegt in der Anzeige der Assets, sowie der gespeicherten Szenen vom Server. Dem Nutzer soll es möglich sein, Assets durch Auswahl in die Szene zu laden. Nach dem Laden hängt das Objekt zunächst am Gaze. Unter dem Gaze versteht man die Position, an die der Nutzer schaut. Für den aktuellen Prototyp (vgl. Abb. 2) stand dabei die Funktionalität im Vordergrund. Der Nutzer kann mit den Pfeiltasten die Seiten durchblättern und so nach Assets suchen. Durch Anklicken eines Assets, wird es in die Szene geladen.

4.2.2. Asset Ladeanimation. Nachdem ein Asset im Asset Store ausgewählt wird, wird dem Nutzer eine Ladeanimation angezeigt (vgl. Abb. 3). Damit soll der Nutzer ein Feedback bekommen, falls der Download eines Assets länger dauern sollte.

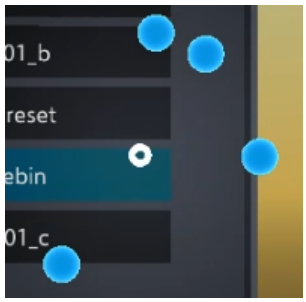
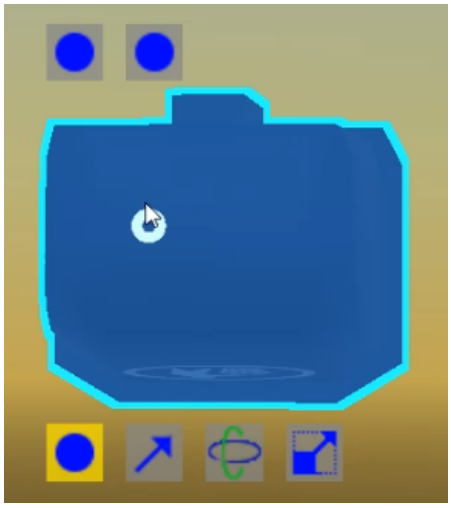


Abbildung 3. Ladeanimation

4.2.3. Auswahl eines Assets. Sobald der Nutzer ein Asset mit dem Gaze-Cursor anvisiert, wird das Asset blau umrahmt (vgl. Abb. 4a). Dadurch erhält der Nutzer als Feedback, dass er das Asset auswählen und damit arbeiten kann. Selektiert der Nutzer nun das Asset, so erhält es eine flächendeckende, durchsichtige Umrahmung und die Operationen für die Manipulation vom Asset werden dem Nutzer angezeigt (vgl. Abb. 4b).

4.2.4. Menüs. Über die beiden Buttons im oberen Bereich der Asset Store UI hat der Nutzer die Möglichkeit weitere Menüs auszuklappen. Im linken Bereich gibt es dazu ein

(a) Gaze Hover



(b) Select

Abbildung 4. Selektieren eines Assets

Menü für das Szenenmanagement (siehe Kap. 4.2.5). Über den Button im rechten Bereich wird das Hauptmenü geöffnet. Darüber kann der Nutzer verschiedene Funktionalitäten, wie das Aktualisieren der Asset-Liste, das Starten des Tutorials, Server/Multiplayer-Funktionen, Spatial-Mapping, Object-Snapping und das Scene-Reset erreichen (vgl. Abb. 5).

4.2.5. Szenenmanagement. Das Szenenmanagement kann ausgeklappt werden, indem der Pfeil in der oberen linken Ecke des Asset Stores geklickt wird. Das Szenenmanagement wir dann mit einer Unity-Animation herausgefahren. Durch Klicken auf "SaveScene" kann der Nutzer die aktuelle Szene auf dem Server speichern. Nun besteht auch die Möglichkeit, die Szene mit einem Benutzerdefiniertem Namen zu speichern. Dies passiert, indem auf "Save Scene as" geklickt wird. Dadurch öffnet sich eine Tastatur, welche durch Gazen und Klicken bedient werden kann. Nach dem Eingeben des Namens ist die Szene unter dem Namen gespeichert. Über "LoadScene" werden die Assets im Assetmenü durch die gespeicherten Szenen ersetzt. Wenn man nun auf eine der Szenen klickt, wird diese in die momentan geladenen

Abbildung 5. UI Menüs

Szene geladen. Der Szenenauswahl-Modus wird dadurch abgebrochen, indem man entweder eine Szene auswählt, oder den Szenen Manager wieder einklappt.

Abbildung 6. Szenenmanagement

4.3. Manipulation von Objekten

Eine Anforderung vom Kunden ist die Manipulation von Objekten in der Szene. Darunter fallen die gängigen geometrischen Transformationen, aber auch das dynamische Texturing von Assets. Nachdem der Nutzer ein Objekt in der Szene auswählt, erhält es zunächst eine blaue Umrandung als Feedback für den Nutzer. Unter dem Objekt erscheinen dann Icons für die geometrischen Manipulationen (vgl. Abb. 7). Im freien Modus hängt das Objekt am Gaze. Der Nutzer kann sich an eine beliebige Position bewegen und das Objekt wieder ablegen. Für eine präzisere Steuerung stehen dem Nutzer folgende Möglichkeiten zur Verfügung:

- **Translation** Änderung der Position des Objekts in x-z- bzw. x-y-Ebene.
- **Rotation** Rotation des Objekts um die x- bzw. y-Achse.
- **Skalierung** Änderung der Größe des Objekts.

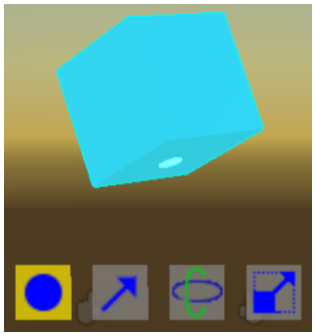


Abbildung 7. Geometrische Manipulationen eines Assets

4.4. Steuerung mit XBox Controller

Für eine komfortablere Bedienung haben wir uns in Rücksprache mit dem Kunden dazu entschieden die Steuerung über einen XBox Controller zu ermöglichen. Dadurch stehen dem Nutzer verschiedene Funktionalitäten durch Knopfdruck zur Verfügung. Außerdem können Objekte dadurch wesentlich genauer manipuliert werden. Im Folgenden wird auf die implementierten Funktionalitäten des aktuellen Prototyps und die zugehörige Tastenbelegung auf dem XBox Controller eingegangen:

- **A** Durch einfaches Drücken der A-Taste wird das Objekt ausgewählt, welches vom Gaze-Pointer getroffen wird. Durch erneutes Drücken der Taste wird das Objekt ausgewählt und es erhält als Visualisierung eine blaue Umrahmung. Als Prototyp wurde ebenfalls ein Infofenster entwickelt, welches neben einem ausgewählten Objekt erscheint. Darin kann man nützliche Informationen zum Objekt anzeigen lassen.

- **B** Durch einfaches Drücken der B-Taste wird das ausgewählte Objekt abgelegt. Durch langes Drücken der B-Taste wird das ausgewählte Objekt gelöscht. Dabei erhält der Nutzer ein visuelles Feedback in Form eines Fortschritt-Kreises, sodass kein Objekt versehentlich gelöscht wird.

- **X** Durch Drücken der X-Taste wird der Asset Store in Blickrichtung neu positioniert, bzw. ausgeblendet, falls er sich im Blickfeld befindet.

- **Y** Durch Drücken der Y-Taste wird das Debug-Fenster neu positioniert.

- **Schultertasten L/R** Sobald ein Objekt ausgewählt ist, kann durch Drücken der Schultertasten der Modus für die geometrische Manipulation des

Objekts (vgl. Abb. 7) gewechselt werden.

- **Control Stick** Sofern ein Objekt ausgewählt ist und der gewünschte Modus für die geometrische Manipulation eingestellt ist, erfolgt die eigentliche Manipulation um die Achsen mit dem Control Stick.

4.5. Minimap

In einem realistischen Anwendungsszenario besteht eine Szene (zum Beispiel die eines Bühnenbildners) schnell aus sehr vielen, unterschiedlich ausgerichteten Objekten. Dies kann dazu führen, dass der Benutzer der Anwendung schnell die Orientierung und den Überblick verliert. Um den Benutzern das Zurechtfinden in komplexen Szenen etwas zu erleichtern wurde, in Absprache mit dem Kunden, eine Übersichtskarte implementiert, die einen Ausschnitt der Szene (aus der Vogelperspektive) zeigt. Die Minimap (siehe Abbildung 8) bietet dem Benutzer so die Möglichkeit auch Objekte, die sich hinter ihm befinden zu lokalisieren und zusätzlich aus einer etwas weiter entfernten Position einen Überblick über die Szene zu erlangen.

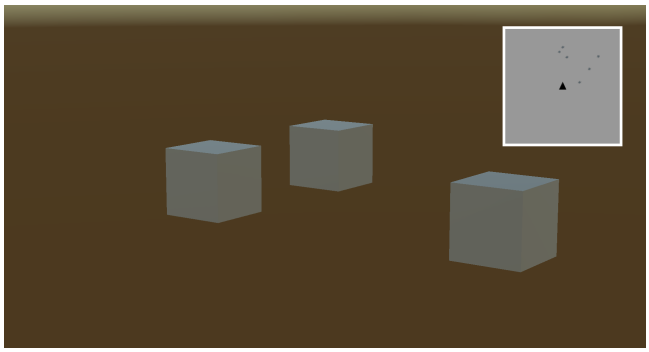


Abbildung 8. Sicht des Benutzers mit aktivierter Minimap

Bei der Umsetzung wurden folgende Punkte beachtet:

- Ansicht der Szene aus Vogelperspektive.
- Implementierung als Heads-Up Display (an das Sichtfeld des Benutzers "geheftet").
- Möglichkeit des Umschaltens.
- Position des Benutzers und Sehrichtung müssen erkennbar sein.

4.6. Gizmos

Die Gizmos sind ein Feature, welches dem Nutzer erlaubt mittels Gesten ein Hologramm zu bewegen, zu skalieren und zu rotieren. Die Handhabung sowie das Aussehen der Gizmos sollte sich dabei an ähnlichen, aus diversen 3D Editoren bekannten, Features orientieren. Da diese Editoren meist auf Desktop PCs genutzt werden, können sie auf Maus und Tastatur als Eingabemöglichkeit zurückgreifen, beides Werkzeuge, die bei entsprechender Übung, ein hohes Maß an Präzision bieten, und auf der HoloLens nicht zur Verfügung stehen.

Eine grundlegende, bereits letztes Semester geschaffene, Implementation des Features zeigte, dass die Hauptaufgabe bei seiner Umsetzung im Umgang mit den limitierten Eingabemöglichkeiten der HoloLens liegen würde.

Einige Probleme im Zusammenhang mit den Handgesten der HoloLens waren:

1) Das korrekte Ausführen einer Geste bedarf einiger Übung. Neben der korrekten Handhaltung ist besonders die Geschwindigkeit bei der Ausführung von Bedeutung. Die Geste darf weder zu schnell noch zu langsam vollführt werden.
2) Das Anvisieren von Objekten, was in herkömmlichen Anwendungen durch den Mauscursor geschieht, wird in unserer Anwendung durch die Blickrichtung des Nutzers umgesetzt. In der Praxis zeigte sich, dass präzise Kopfbewegungen äußerst schwierig sind und damit das Anvisieren, besonders von kleinen oder nahe beieinanderliegenden Objekten schwierig bis unmöglich sein kann.

Als Resultat der mangelnden Präzision wurden schon letztes Semester größere UI Elemente verwendet, um dem Nutzer das Anvisieren zu erleichtern, was sich als erfolgreiche Maßnahme erwies.
Weitere Verbesserungen, die dieses Semester unternommen wurden, war die Implementation von Feedback über die erfolgreiche Auswahl einer Achse und zweitens, die Möglichkeit Gesten frei im Raum auszuführen, anstatt die Bewegung auf der Achse des Gizmos ausführen zu müssen. Die erste der beiden Änderungen hatte zwei Gründe. Zum einen erwies sich das Auswählen der Achse, entlang der die Manipulation ausgeführt werden sollte, aus den oben genanten Gründen, als nicht einfach, was dazu führte, dass der Nutzer oft glaubte, er hätte eine Achse ausgewählt, dies jedoch nicht der Fall war und anschließend vergeblich versuchte auf dieser eine Bewegung auszuführen.
Zum anderen wünschte sich der Kunde, dass die beiden Achsen, die nicht manipuliert wurden, während der Manipulation ausgeblendet werden sollten. Es bot sich daher an, beide Anforderungen zu kombinieren.
Um dies zu erreichen, werden nun nach der erfolgreichen Auswahl einer Achse die anderen beiden ausgeblendet. Die Achsen werden wieder eingeblendet, wenn die Geste zum Manipulieren einer Achse als beendet oder abgebrochen erkannt wird. Alternativ, falls beispielsweise die falsche Achse ausgewählt wurde, kann der Nutzer einen Air Tab ins Leere ausführen.

Die Motivation hinter der zweiten Änderung war, dass wie erwähnt Gesten zur Manipulation zuvor auf dem Handle einer Achse ausgeführt werden mussten. Dies erwies sich selbst mit einiger Übung, als äußerst schwierig. Um dies zu ändern, wurde entschieden, dass Gesten frei im Raum ausführbar sein sollten.
Dies war keine kleine Änderung, da die auf der Achse ausgeführte Bewegung dazu genutzt wurde,

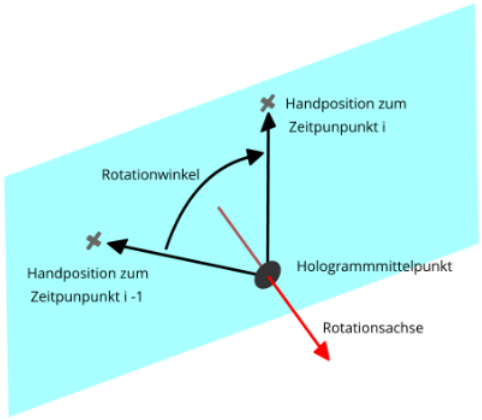


Abbildung 9. Beispiel der Berechnung des Rotationswinkels

die Bewegungsrichtung bzw. Rotationsrichtung für die Manipulation zu bestimmen. Insbesondere die Neuimplementation des Rotationsgizmos war bedeutend schwieriger als initial vermutet, da hier das Vorzeichen des Rotationswinkels je nach Lage des Objekts und seiner relativen Position zum Nutzer geändert werden musste.
Die hierbei möglichen Fälle durch eine herkömmliche Fallunterscheidung abzudecken, erwies sich als fehleranfällig und unübersichtlich, weshalb eine Methode gewählt wurde, bei der das Vorzeichen bei der Berechnung des Rotationswinkels implizit mitbestimmt wird. Hierzu werden zwei Punkte herangezogen, die jeweils die Handposition des Nutzers im Screenspace während des aktuellen und letzten Frames widerspiegeln. Weiter wird eine Ebene aufgestellt, deren Normale dem Vektor entspricht, um den das Objekt rotiert werden soll. Auf diese Ebene werden die beiden Punkte projiziert und Vektoren von ihnen zum Mittelpunkt des zu rotierenden Objekts bestimmt. Anschließend muss nur noch der Winkel zwischen diesen beiden Vektoren berechnet werden, wobei der Referenzsektor hierbei die Normale der Ebene ist. Ohne weitere Fallunterscheidungen kann dieser Winkel zum Rotieren des Objekts verwendet werden.

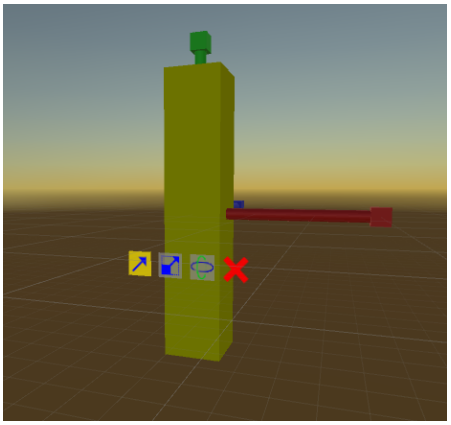
Die Berechnungen zum Bewegen und Skalieren von Hologrammen sind identisch und deutlich einfacher als die zum Rotieren. Zuerst werden wieder zwei Punkte bestimmt, die die Handposition im aktuellen und letzten Frame, angeben. Beide Punkte werden in den Screenspace konvertiert und der Vektor zwischen ihnen bestimmt. Auf die gleiche Weise werden über den End- und Startpunkt des Handles der ausgewählten Achse, die Richtung des Handles im Screenspace bestimmt. Über das Skalarprodukt der beiden Vektoren wird anschließend bestimmt, ob die in die gleiche Richtung zeigen oder nicht, wodurch sich das Vorzeichen der Manipulation ergibt. Die Distanz der Bewegung oder der Wert der Skalierung ergibt sich aus der Differenz, die die Hand des Nutzers zwischen den beiden Frames zurückgelegt hat.

Eine weitere Änderung, die wir vornahmen, war der Umgang mit unterschiedlich großen Hologrammen. Bei Hologrammen, die größer als die Gizmos selbst sind, bestand das Problem, dass das Hologramm die Gizmos überdeckte und diese daher die Interaktion mit ihnen unmöglich wurde.
Die naheliegende naive Lösung bestand darin, die Gizmos in Abhängigkeit von der Größe des ausgewählten Hologramms zu skalieren. Dies erwies sich jedoch für besonders kleine oder große Hologramme als problematisch, da dies dazu führte, dass die Gizmos entweder extrem klein oder extrem groß werden konnten.
Als Verbesserung entschieden wir uns auf das Skalieren zu verzichten und stattdessen nach der Auswahl des Objekts dieses als halb transparent zu rendern und dessen Collider zu deaktivieren. Dies ermöglicht das Anwählen der Gizmos durch das Hologramm hindurch.

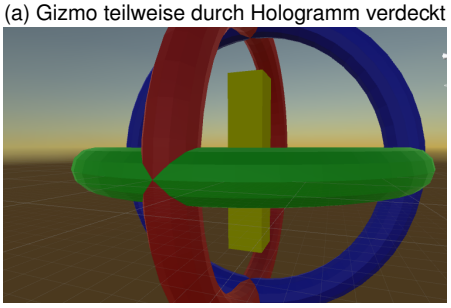
Wir möchten an dieser Stelle einige, selbstverständlich subjektive, Erfahrungen mit der Gestensteuerung der HoloLens anhängen.
Die dieses Semester gesammelten Erfahrungen haben unsere, schon im letzten Semester gefasste Vermutung, dass es sich bei der Gestensteuerung mehr um ein Gimmick als eine praxistaugliche Eingabemöglichkeit handelt, weiter gefestigt.
Die einleitend beschriebenen Probleme machen ein schnelles und präzises Arbeiten schwierig bis unmöglich, was insbesondere bei Nutzern, die eine schnell reagierende Benutzeroberfläche gewöhnt sind, die einem hohen Arbeitstempo standhält, schnell für Frustration sorgen wird. Weiterhin ist das Arbeiten mit ausgestreckten Armen schon nach kurzer Zeit anstrengend.

4.7. Objektsnapping

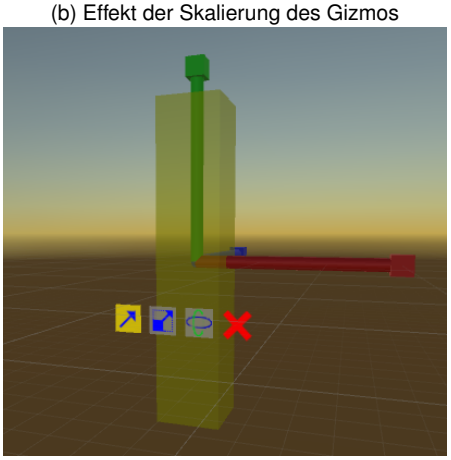
Der Mehrwert, den Objektsnapping oder die Möglichkeit Hologramme in einem Grid anzuordnen für die von uns entwickelte Anwendung mit sich bringt, wurde schon sehr früh offensichtlich, als wir in einem der ersten Tests versuchten individuelle Hologramme zu einem größeren Gebilde zusammenzufügen. Grundsätzlich war dies zwar möglich, doch war dies nicht so benutzerfreundlich und Präzise, wie man sich dies wünschen würde.
Objektsnapping sollte dies ändern. Es sollte ein vom Nutzer bewegtes Hologramm automatisch, sobald es nahe genug an ein anderes heranbewegt wurde, auf dessen Oberfläche platzieren. Alle weiteren Bewegungen sollten anschließend so ausgeführt werden, dass das Hologramm auf der Oberfläche des Hologramms entlanggleitet, ohne dass der Nutzer hierzu besondere Anstrengungen unternehmen müsste. Weiterhin sollte es möglich sein, Objekte in einem Grid anzuordnen, sodass es möglich ist, aus individuellen Objekten größere Strukturen zusammenzusetzen.



(a) Gizmo teilweise durch Hologramm verdeckt



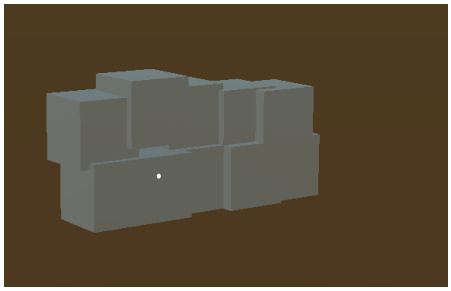
(b) Effekt der Skalierung des Gizmos



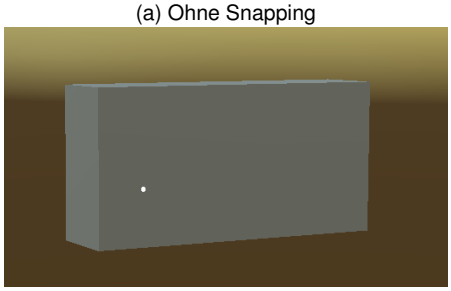
(c) Lösung des Skalierungsproblems

Abbildung 10. Gizmos der zweiten Version

Zwei grundlegende Designentscheidungen, die getroffen wurden, waren, dass erstens nicht alle Objekte in der Szene vom Snapping betroffen sein sollten. Beispielsweise wäre es, in den meisten Fällen, nicht wünschenswert ein Hologramm an ein GUI Element zu snappen. Um dies zu erreichen, wurde entschieden, dass Objekte, an die andere herangesnappt werden können, mit einem speziellen Tag versehen werden müssen.
Die zweite Entscheidung war, eine möglichst einfache Schnittstelle zum Umsetzen des Snappings anzubieten. Dazu wurde die Funktionalität in möglichst einfache Funktionen gekapselt. Die Funktion zum Bewegen eines snappbaren Objekts nimmt dementsprechend lediglich das Gameobject und einen Bewegungsvektor entgegen. Alle



(a) Ohne Snapping



(b) Mit Snapping

Abbildung 11. Zusammengesetzte Strukturen aus kleineren Hologrammen

weiteren Berechnungen, ob und wie der Bewegungsvektor abgeändert werden muss, um das gewünschte Verhalten zu erreichen, geschehen intern. Diese Kapselung, welche allein aus Gründen der Wartbarkeit und softwaretechnischer Designprinzipien wünschenswert ist, ist für unser Projekt zusätzlich interessant, da wir verschiedene Möglichkeiten anbieten ein Objekt zu bewegen. So können Objekte mit dem Controller, der Blickrichtung oder den Gizmos bewegt werden. Die Kapselung in einen einzigen Funktionsaufruf ermöglicht es, das Snapping für alle drei Eingabevarianten mit minimalem Aufwand zu implementieren.

Das Verhalten des Snappings lässt sich wie folgt beschreiben:
Der Nutzer selektiert ein Objekt, bewegt dies umher und solange das bewegte Objekt nicht in die Nähe anderer snappbarer Objekte kommt, werden die Bewegungen eins zu eins umgesetzt. Wird dagegen ein bestimmter Threshold unterschritten, greift das eigentliche Snapping. Bei der Berechnung des Thresholds ist zu beachten, dass dies dynamisch auf Basis der beiden Objekte geschehen muss. Der naive Ansatz, einen festen Threshold zu wählen und die Entfernung der Objekte aus der Differenz ihrer Positionen zu berechnen, führt dazu, dass sehr große Objekte schon ineinander liegen, bevor die Entfernung ihrer Mittelpunkte den Threshold unterschreitet.
Wurde der Threshold unterschritten, wird zunächst durch einen Raycast auf das nächste Objekt, die Ebene bestimmt, an die das Hologramm herangesnapped werden soll. Die Ebene setzt sich dabei aus der Normalen der Facette des getroffenen Colliders und der Ausdehnung des getroffenen Objekts in dieser Richtung zusammen. In Verbindung mit der Dimensionierung des bewegten Objekts kann so ein Bewegungsvektor errechnet und angewandt werden, der

das Hologramm an die Oberfläche des anderen Objekts heranbewegt. Das Hologramm befindet sich nun im Follow Mode und klebt an der Oberfläche des Objekts. Alle Bewegungen, die nun auf das Hologramm angewandt werden sollen, werden auf die zuvor berechnete Ebene projiziert, wodurch sich das Hologramm auf der Ebene bewegt. Nähert sich das Hologramm einem anderen Objekt, wird eine neue Ebene zum Folgen ausgewählt. Das Gleiche geschieht, wenn das Hologramm die Kante eines Objekts erreicht und einer anderen Ebene folgen muss, um weiter der Oberfläche des Objekts zu folgen. Um das Hologramm von der Oberfläche des Objekts zu lösen, muss das Snapping über einen Button im Menü ausgeschaltet werden.

Der momentane Stand des Snappings unterstützt nur Objekte, die über einen Boxcollider verfügen und die Objekte selbst sollten ebenfalls Rechtecke sein. Zwar ist es möglich beliebige Formen zu verwenden, das Verhalten des Snappings ist dann jedoch nicht definiert. Weiterhin ist die Auswahl der zur folgenden Ebene, zwar für einzelne Rechtecke und einfache Formationen von Rechtecken ausreichend, jedoch bei komplexeren Anordnungen nicht immer korrekt. Da lediglich die Entfernung zu einem anderen Objekt darüber bestimmt, welcher Ebene gefolgt wird, kann es dazu kommen, dass das Hologramm der falschen folgt. Dies kann dazu führen, dass ein Objekt zum Folgen ausgewählt wird, es beim Ausführen der Bewegung jedoch in ein anderes hineinläuft. Eine mögliche Verbesserung könnte darin liegen, die Bewegungsrichtung des Hologramms heranzuziehen, um zu bestimmen, welcher Ebene gefolgt werden soll.
Eine weitere noch offene Fragestellung besteht darin, wie das Objektsnapping mit den Daten des Spatial-Mappings verbunden werden kann, da im Grunde beide Systeme eine ähnliche Funktion besitzen.
Ein weiteres noch ungelöstes Problem, ist der Umgang mit rotierten Objekten, da die Facette mit der das Hologramm an der Oberfläche eines Objekts klebt, die gleiche Ausrichtung, wie diese Oberfläche haben sollte. Momentan ist dies so gelöst, dass das Hologramm die Rotation des Objekts, dem es folgt, übernimmt. Ein weitaus wünschenswerters Verhalten wäre jedoch, wenn das Hologramm seine Ausrichtung im Raum so weit wie möglich beibehält und die angewandte Korrektur so klein wie möglich ist.
Um das Anordnen von Objekten in einem Grid zu ermöglichen, entschieden wir uns dagegen das Raster des Grids vordefiniert und auf Basis der Weltkoordinaten festzulegen. Stattdessen sollte das Raster relativ zu einem Referenzobjekt aufgebaut werden. Das Referenzobjekt ist dabei stets das Objekt, dessen Oberfläche das Hologramm folgt. Weiterhin bestimmt die Ebene, an welcher Facette das Objekt sich ausrichten sollte.
Der Grid besitzt daher kein Raster in dem Sinn, sondern Objekte werden auf Knopfdruck so ausgerichtet, dass zwei ihrer Facetten aufeinanderliegen und beide zentriert aufeinanderliegen. Bewegt wird dabei nur das Hologramm, welches der Nutzer bewegt.

4.8. Spatial Mapping

Spatial Mapping ermöglicht es eine 3D-Map der Umgebung zu erstellen. Mit dieser 3D-Map wird der HoloLens die Interaktion zwischen der virtuellen und realen Welt ermöglicht. Hierbei scannt die HoloLens konstant die Umgebung ab und erstellt ein Abbild der gescannten Werte. Dieses Abbild wird bei jeder Bewegung der HoloLens aktualisiert um neue Elemente hinzuzufügen. Dieses 3D-Abbild wird dem Nutzer durch ein Gitter, bestehend aus Dreiecken, angezeigt.

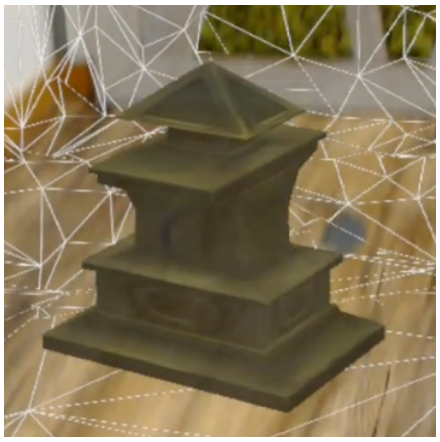


Abbildung 12. Spatial Mapping - Gitter

4.8.1. Verwendung im Projekt. Das Spatial Mapping wird in unserem Projekt für die Platzierungen der Assets in der realen Welt verwendet. Somit ist dem Nutzer erlaubt Assets z.B. auf einem realen Tisch zu platzieren. Für die Realisierung wurde das TapToPlace-Skript vom MixedRealityToolkit verwendet.

4.8.2. Realisierung und Probleme. Bei der Realisierung gab es zwei wesentliche Probleme. Das erste Problem war es, dass es insgesamt zwei Möglichkeiten der Platzierung bzw. Bewegung von Assets gibt. Zum einen das Standard-Verfahren, bei dem die reale Welt (ohne Spatial Mapping) nicht berücksichtigt wird und zum anderen das Spatial Mapping-Verfahren. In der Praxis musste der HoloLens bekannt gegeben werden, in welchem Modus man sich nun befinden wollte. Dazu wurde ein Toggle-Button für das Spatial Mapping erstellt, mit dem man diese de-/aktivieren kann. Wird nun ein Asset anvisiert, so wird immer geprüft, in welchem Modus sich der Nutzer befindet. Dementsprechend wird das jeweilige Skript für die Bewegung der Assets aktiviert.

Das zweite Problem war es, dass das TapToPlace-Skript nur für die AirTap-Geste ausgelegt ist und eine Anforderung war, dass die Applikation auch mit einem XBOX-Controller zu steuern sein soll. Deswegen musste das Skript verändert werden, damit auch diese Anforderung erfüllt werden konnte. Es wurden somit Funktionen implementiert, welche den SpatialMapping-Modus de-/aktiviert haben, wenn der

Nutzer ein Asset mit dem XBOX-Controller bewegen oder ablegen will.

5. Nullpunkt der Szene setzen - SceneReset

Es musste eine Möglichkeit geschaffen werden, um eine Szene in der realen Welt anhand eines im virtuellen Raum festgelegten Nullpunkts ausrichten zu können. Die in diesem Abschnitt vorgestellte **SceneReset**-Funktionalität löst alle Probleme bei der Positionierung einer virtuell aufgebauten Szene in der Realität im Einzel- sowie im Multiuser-Modus.

5.1. Grundlegende Idee des SceneReset

Für die Anwendungsfälle der Applikation wurde eine Funktionalität benötigt, mit der eine virtuelle Szene beispielsweise in einem Raum oder auf einer Bühne mit genauer realer Positionsreferenz ausgerichtet werden kann. Das Konzept, das hierfür entwickelt wurde sieht vor, dass in jeder virtuellen Szene ein Asset (*SceneReset*) hinzugefügt wird, das den vorher festgelegten Referenzpunkt im realen Raum darstellt. Realisiert wurde dieses Konzept mit Hilfe der Bildziel-Erkennung des *Vuforia*-Frameworks. Damit können Assets zentimetergenau auf vorher festgelegte Bilder mit Hilfe der Webcam der HoloLens positioniert werden.

5.2. Umsetzung

In der Hauptszene der Applikation (*AURA-Showcase*) wird ein im Framework enthaltenes *ImageTarget*, also das Bild, das vom Vuforia-Framework erkannt wird, eingesetzt. Dieses in der kostenlosen Version des Frameworks vorhandene Bild eines Astronauten stellt später den Mittelpunkt der erstellten Szene dar. Für die Implementierung wurde im Bild-Objekt ein Script erstellt, das den Eventhandler des Vuforia-Frameworks verwendet.

Vuforia erzeugt einen hohen Rechenaufwand beim Suchen von ImageTargets, was dazu führt, dass die Applikation nur noch mit durchschnittlich 20 Bildern pro Sekunde abläuft. Aufgrund dieser Performanzprobleme wurde gleich zu Beginn implementiert, dass das Framework nur eingesetzt wird, wenn es wirklich benötigt wird, also eine virtuelle Szene im reelen Raum ausgerichtet wird. Über die Standard-Funktion *Start()* wird beim Start der Applikation auf der HoloLens der Vuforia-Service aktiviert, um den Eventhandler auf dem ImageTarget registrieren zu können. Danach wird der Vuforia Service direkt wieder deaktiviert, wodurch erreicht wird, dass das Tracking nur für eine sehr kurze Zeit aktiv ist und die Applikation störungsfrei abläuft.

Der Nutzer hat nun die Möglichkeit, im Hauptmenü der Applikation durch den Button mit der Aufschrift *Reset the Scene* den Vuforia-Service zu aktivieren. Jetzt wird die Kamera der HoloLens aktiviert und nach dem ImageTarget gesucht. Wird ein ImageTarget gefunden wird durch einen Callback des zum Applikationsstart registrierten EventHandlers die Funktion *OnTrackableStateChanged()* aufgerufen, die wiederum *OnTrackingFound()* aufruft, wenn vom Framework ein neues Bild gefunden wurde.

5.3. Implementierung

Nach der Erkennung eines ImageTargets wird überprüft, ob es sich dabei um das Bild “Astronaut” handelt, da nur in der kostenpflichtigen Version von Vuforia eigene Bilder als ImageTargets hinzugefügt werden können. Danach werden Translations- und Rotationsvektoren initialisiert, mit denen die Verschiebung der Szene auf das getrackte Bild ausgeführt wird. Die in der Einführung vorgestellte *SceneReset*-Platte wird über den Namen in der Szene gesucht - wird die Platte nicht gefunden, wird der Vorgang an dieser Stelle abgebrochen. Als nächstes wird abgefragt, ob die Anwendung im Single- oder im Multiplayermodus läuft, da im Multiplayermodus nicht die Originalobjekte, sondern Kopien davon in der Szene auf den Nullpunkt verschoben werden müssen. Damit wird gewährleistet, dass für alle angemeldeten Nutzer die Objekte auf den Szenennullpunkt verschoben werden.

Nachdem entschieden wurde, ob die Netzwerk- oder die eigentlichen Objekte der Szene verschoben werden sollen, wird das *SceneReset*-Objekt als Parent aller zu verschiebenden Objekte gesetzt. Das hat zur Folge, dass alle Translationen und Rotationen, die auf das SceneReset Objekt angewendet werden, sich auch auf die Kinder (also die Objekte der Szene) relativ zum SceneReset auswirken. Der benötigte Translationsvektor wird errechnet, indem die Position des SceneReset Objekts subtrahiert wird von der Position des getrackten Bildes. Der Rotationsvektor wird errechnet, indem die Eulerwinkel des SceneReset Objekts von den Eulerwinkeln des getrackten Bildes subtrahiert werden. Um eine sehr genaue Positionierung zu erreichen, wird die Translation und Rotation des SceneReset Objekts in einer while-Schleife ausgeführt, in der nach dem Verschieben die Vektoren mit der neuen Position des Objekts neu berechnet werden. Diese while-Schleife aus Translieren, Rotieren und neu berechnen wird so oft ausgeführt, bis die errechneten Translations- und Rotationsvektoren 0 betragen, das SceneReset Objekt sich also exakt auf dem getrackten Bild befindet. Die anfängliche Angst, dass diese while-Schleife Performance-Einbußen bedeuten könnte war unbegründet, da in allen getesteten Fällen maximal 3 Durchläufe durchgeführt wurden, bis die Vektoren exakt übereinstimmen, was sich immer in einer Zeitspanne von unter 0.1 Sekunden abspielte. Nach dem Verschieben der Szene müssen alle Objekte “entparentisiert” (also der Parent auf null gesetzt) werden, dass danach wieder alle Objekte einzeln durch den Anwender verwendbar sind. Nachdem diese Vorgänge abgeschlossen sind, wird der Vuforia-Dienst deaktiviert, damit die Anwendung wieder relativ flüssig läuft.

5.4. Problemstellungen im Projekt

Die Integration von Vuforia in der Applikation wurde bereits im letzten Semester als zur damaligen Zeit unlösbar eingestuft. Nach vielen Tests konnte eine Lösung gefunden werden, bei der eine Konstellation der für Vuforia benötigten Kamera als Kind Element der MixedRealityCamera des

HoloToolkits eingerichtet wird. Diese Lösung läuft allerdings nur mit einem Skript zuverlässig, das die Kamera von Vuforia im Unity Editor deaktiviert. Durch die Integration der Vuforia Kamera in der Applikation sind im Unity Editor enorme Probleme mit der Steuerung und der Hauptkamera aufgetreten. Das implementierte Skript erkennt beim Start, ob die Anwendung im Unity Editor oder auf der HoloLens läuft und aktiviert respektive deaktiviert die Vuforia Kamera in der Szene. Im vorigen Abschnitt wurde bereits die Performanz von Vuforia in der Anwendung erwähnt, sobald Vuforia die Kamera der HoloLens und das Tracking aktiviert, steigt die Auslastung der Brille auf 100% und die Bildrate fällt um 20-30 FPS. Diese Probleme konnten weitestgehend reduziert werden, indem der Dienst nur aktiviert ist, wenn damit Aufgaben erledigt werden sollen, bei den restlichen Aufgaben der Anwendung wartet der Dienst auf die manuelle Aktivierung durch Button oder einen Sprachbefehl. Was ebenfalls noch erwähnt werden muss ist, dass beim SceneReset im Multiuser Teil der Anwendung das Problem aufgekommen ist, dass die Positionierung mehrerer Hololenses nicht exakt übereinstimmt, obwohl die miteinander geteilten Netzwerkobjekte im Koordinatensystem exakt übereinstimmen. Dieses Problem könnte umgangen werden, indem sich die Hololenses beim Programmstart an der exakt gleichen Position befinden. Die Positionierungsgenauigkeit der Hololens wird von Microsoft selbst nur auf ungefähr 3m im Radius der Startposition angegeben. Bei einigen Tests konnte festgestellt werden, dass während auf der einen Seite eines Raumes beide Hololens-Nutzer alle Objekte an der exakt gleichen Position sehen konnten, die Objekte bei erneutem SceneReset auf einem 6 Meter entfernten Punkt auf der anderen Seite des Raumes bereits mehrere cm auseinander lagen.

6. Minimap-Implementierung für den Überblick über die Szene

Um den Nutzer eine Möglichkeit zu bieten einen Überblick über die Szene zu bekommen, in der er sich befindet, wurde bei der Entwicklung der Applikation eine Art Übersichtskarte mit Hilfe des im vorigen Abschnitts bereits vorgestellten Vuforia-Framework erstellt.

6.1. Funktionsweise

Wenn der Nutzer nach dem Aktivieren des Vuforia-Frameworks auf die Karte (in Abbildung 13) schaut, wird eine Kopie aller relevanten Objekte der Szene erstellt, in einer Liste gespeichert und klein skaliert auf die Karte projiziert. Die Verschiebung der Objekte auf die getrackte Karte funktioniert nach dem selben Prinzip wie beim SceneReset. Unterschiedlich zum SceneReset ist hier, dass alle Objekte relativ zur kopierten SceneReset-Platte auf einen kleinen Wert skaliert werden und dass das Tracking nicht nach Positionierung deaktiviert wird, sondern sich die skalierten Objekte mit der Karte bewegen, die der Nutzer frei bewegen kann. Deaktiviert wird das Vuforia Framework automatisch,

Abbildung 13. Miniatur der Szene auf ein reales Bild projiziert

sobald die Karte nicht mehr im Blickfeld der Webcam der HoloLens ist. Die kopierten Miniaturobjekte werden gelöscht, die Liste in der die Objekte gespeichert waren wird aufgeräumt. Aktiviert werden kann die Minimap durch den Sprachbefehl “map”. Hierbei muss erwähnt werden, dass das Vuforia Framework immer nach allen Bildern sucht, die dem Framework bekannt sind, wenn also der Button des SceneReset gedrückt wird und auf die Minimap geschaut wird dann wird trotzdem die Funktion der Übersichtskarte aufgerufen. Das funktioniert auch in die andere Richtung, man könnte mit dem Sprachbefehl “map” auch den SceneReset aufrufen, wenn man auf den Astronaut schaut.

7. Texturing

Bereits zu Beginn der Entwicklung wurde schnell ersichtlich, dass die HoloLens gewisse Einschränkungen (hauptsächlich im Bezug auf leistungsschwache Hardware, drahtlose Verbindungen) mitbringt. Dadurch wurde es notwendig, dass das Team immer eine Optimierung des Laufzeitverhaltens bei der Umsetzung der Anforderungen im Hinterkopf behielt. Schnell stellte sich dabei das Laden der Assets und Materials vom AssetServer auf die HoloLens als sehr teure Operation heraus. Deshalb sollte durch das UI-Team eine Lösung gefunden werden, die es ermöglicht auf redundante Informationen bei der Übertragung der Assets zu verzichten.

7.1. Technischer Hintergrund

Um die Umsetzung im Detail nachvollziehen zu können zunächst ein kurzer Überblick zu Materials, Shaders und Textures in Unity. Alle Drei hängen eng zusammen.

- **Materials** definieren wie ein Oberfläche gerendert werden soll einschließlich der Texturen, die verwendet werden, Farbtönen, Rasterinformationen etc. Die verfügbaren Optionen für ein Material hängen von der Wahl des Shaders ab.

- **Shaders** sind kleine Skripte, die mathematischen Berechnungen und Algorithmen für die Farbberechnung der einzelnen Pixel, in Abhängigkeit vom Lichteinfall und der Materialkonfiguration, beinhalten.
- **Textures** sind bitmap Bilder. Ein Material kann Referenzen zu Texturen enthalten, die der Shader für diese Textur während der Berechnung der Oberfläche verwenden kann.

Ein Material spezifiziert einen Shader, der verwendet wird, dieser wiederum bestimmt welche Optionen dem Material zur Verfügung stehen. Ein Shader spezifiziert einen oder mehrere Texturen, die verwendet werden. [6]

7.2. MaterialManager

Der MaterialManager bietet dem Benutzer der Anwendung die Möglichkeit das Material eines sich in einer Szene befindlichen Objekts dynamisch auszutauschen und somit eine hohe Wiederverwendung der auf die HoloLens geladenen (und damit für die Anwendung verfügbaren) Assets zu garantieren. Diese Trennung bietet nicht nur den Vorteil die Laufzeit der Applikation und die Ladezeiten zu verbessern, sondern sie bietet auch die Möglichkeit für den Benutzer schnell das Aussehen eines Objekts anzupassen. Statt ein vorhandenes Objekt (z.B. einen Holztisch) zu löschen und dann ein neues Objekt (z.B. einen Plastiktisch) zu platzieren, kann einfach das Material geändert werden. Somit bleiben zudem Position, Ausrichtung und andere laufzeitbezogenenen Informationen erhalten, was die Softwareergonomie deutlich erhöht.

Abbildung 14. Beispiel einer Materialpalette

Das UI ist idealisiert in Abbildung 14 dargestellt. Ein einheitliches Bedienkonzept ist in Planung, aber derzeit noch nicht umgesetzt. Zusätzlich zu diesem Bedienkonzept wurde als weitere Verbesserung die individuelle Texturierung von Teilobjekten, statt ausschließlich gesamter Objekte, vorgeschlagen.

Die dynamische Texturierungsart ist in Abbildung 15 zu sehen.

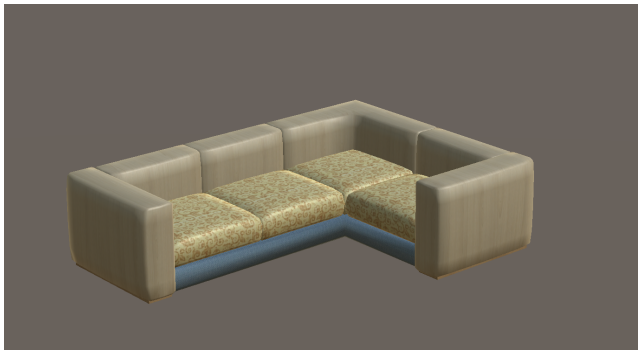


Abbildung 15. Texturiertes Sofa mit verschiedenen Untertexturen

8. Tutorials

Ein Tutorial bietet einem neuen Benutzer einen erleichterten Einstieg bei der Nutzung von Software. Für dieses Projekt ist ein Tutorial für die Manipulationsmöglichkeiten mittels XBox Controller von zur Laufzeit erzeugten Objekten vorhanden. Dies beinhaltet die Bewegung, Skalierung und Rotation solcher Objekte. Das Tutorial kann jederzeit durch klicken eines Buttons vom Benutzer gestartet oder beendet werden. Die Struktur des Tutorials ist so aufgebaut, dass weitere Tutoriallevel und zugehörige Schritte zu einem Level ohne größeren Aufwand hinzugefügt werden können.

8.1. Beteiligte Komponenten

Das Tutorial ist weitestgehend von den restlichen Funktionen von AURA getrennt. Durch Starten des Tutorials werden verschiedene Komponenten aktiv, um für einen ordnungsgemäßen Ablauf des Tutorials zu sorgen. Die folgenden Komponenten sind an der Durchführung des Tutorials beteiligt und werden im weiteren Verlauf dieses Abschnitts genauer erläutert.

- TutorialInitializer
- TutorialEventManager mit Events
- TutorialManager mit Tutorial und TutorialStep

8.1.1. TutorialInitializer. Nach Starten des Tutorials mittels Drücken des Buttons im Menü, wird der *TutorialInitializer* aktiv und erzeugt bzw. aktiviert ein *GameObjekt* bestehend aus dem *TutorialManager* und dem *TutorialReader*. Diese sind am planmäßigen Ablauf des Tutorials beteiligt und werden in Abschnitt 8.1.3 genauer erläutert. Nach erfolgreicher Initialisierung des *Tutorial-GameObjekt* wird der Button im Menü angepasst. Dessen Handler für Klicks wird zu diesem Zeitpunkt durch eine andere Methode ersetzt, damit das laufende Tutorial stets beendet werden kann.
Bei vorzeitigem Abschluss des Tutorials durch Drücken des Buttons oder nach vollständigem Durchspielen aller Level werden noch einige Schritte durchlaufen, um das Tutorial zu beenden und wieder in den anderen Modus

zurück zu gelangen. Zunächst wird die Funktion zum Starten des Tutorials mittels Button wiederhergestellt. Weiterhin werden verbleibende *EventListener*, die genauer in Abschnitt 8.1.2 beschrieben werden, entfernt, um ein mögliches Speicherleck zu verhindern. Abschließend wird das eingangs erwähnte *Tutorial-GameObjekt* deaktiviert und verbleibt in diesem Zustand bis das Tutorial erneut gestartet wird. Dies ist nötig, da bei der Aktivierung dieses Objekts gleichzeitig mehrere Vorgänge zur Initialisierung des Tutorials mittels von Unity bereitgestellten Mechanismen vorgenommen werden.

8.1.2. TutorialEventManager mit Events. Das gesamte Tutorial besteht aus einzelnen Levels unterteilt in einzelne Schritte. Damit erkannt werden kann, wann ein Schritt bzw. Level beendet wurde, gibt es ein Event System. Dessen Ablauf wird von dem *TutorialEventManager* gesteuert. Für jedes Level bzw. jeden Schritt wird ein Listener mit einem bestimmten Schlüsselwort als Event auf das gehört werden soll beim *TutorialEventManager* angemeldet. Zudem wird jedem Listener eine Aktion, in diesem Fall den nächsten Schritt bzw. das nächste Level einzuleiten, mitgegeben. Die entsprechende Aktion wird dann in Form eines Methodenaufrufs ausgeführt, wenn das zugehörige Event getriggert also beispielsweise der Abschluss eines Levels erkannt wird.
Für das Auslösen solcher Events ist ebenfalls der *TutorialEventManager* zuständig. Er verwaltet alle angemeldeten Listener. Damit diese nicht zu einem Speicherleck führen, gibt es eine Möglichkeit alle Listener zu löschen. Für einzelne Schritte geschieht dies nach Abschluss eines Levels. Beim beenden des Tutorials werden alle Listener für die Level und zudem eventuell verbliebene Listener für die einzelnen Schritte entfernt.
Die Events sind alle ähnlich aufgebaut. Ein Bestandteil ist ein boolean, welcher zu Beginn des jeweiligen *TutorialStep* auf *false* gesetzt ist. Wird die Kondition eines *TutorialStep* erfüllt, beispielsweise nachdem der im Tutorial genutzte Würfel auf bestimmte Weise rotiert wurde, wird das Event getriggert und der zuvor erwähnte boolean auf *true* gesetzt. Dies soll verhindern, dass ein Event mehrmals aufgrund von langsamen Berechnungen getriggert wird. Ein weiterer Bestandteil ist die von Unity bereitgestellte *Update-Methode*, die hier dazu genutzt wird, um zuvor erwähntes Erfüllen einer Kondition überhaupt zu ermöglichen.

8.1.3. TutorialManager mit Tutorial und TutorialStep. Der ordnungsgemäße Ablauf des Tutorials wird vom *TutorialManager* gesteuert. Dieser verwaltet alle verfügbaren Tutoriallevels in einer Liste und startet diese Levels beim Beginn des Tutorials oder nachdem ein Level abgeschlossen wurde. Dies geschieht entsprechend der Reihenfolge der Level innerhalb der Liste. Für den Fall, dass ein Level beendet wurde, besitzt der *TutorialManager* einen Listener. Wird das zugehörige Event, Abschluss eines Levels, getriggert, werden vom Tutorial erzeugte Objekte gelöscht und das nächste Level der Liste wird gestartet. Befindet man sich zu diesem Zeitpunkt am Ende der Liste, genauer nachdem alle Levels durchgespielt wurden, wird stattdessen die in Abschnitt 8.1.1 beschriebene Abschlussroutine durchlaufen.
Die Tutoriallevel sind Implementierungen der abstrakten Klasse *Tutorial*. Diese Klasse besteht aus einer Liste für die einzelnen Schritte, Listener für diese und Methoden für den Start und Ablauf des aktuellen Levels. Die einzelnen Schritte sind dabei selbst von einer abstrakten Klasse abgeleitet und werden später in diesem Abschnitt noch genauer erläutert. In Implementierungen für *Tutorial* müssen alle nötigen Schritte des jeweiligen Levels zur Liste mit den Schritten hinzugefügt werden. Zum Beispiel besteht das Level für Rotation aus drei Schritten - Anwählen des erzeugten Würfels, Auswählen des Rotations-Modus und abschließend Rotieren des Würfels. Die einzelnen Schritte werden während der Laufzeit bei der Initialisierung des jeweiligen Levels hinzugefügt.
Alle weiteren, notwendigen Methoden für den Ablauf des Levels sind bereits in der abstrakten Klasse implementiert. Diese beinhaltet unter anderem das Hinzufügen und Entfernen eines Listeners für die Erkennung abgeschlossener Schritte. Des Weiteren gibt es Methoden, um den Ablauf eines Levels zu regeln. Ähnlich zu dem Vorgehen, wie die Level durchlaufen werden (siehe Erklärung zum *TutorialManager*), werden hier die einzelnen Schritte der Reihe nach durchlaufen.
Für jeden Schritt wird eine Implementierung der abstrakten Klasse *TutorialStep* benötigt. Die Aufgabe jedes Schrittes wird dem Spieler mittels Text-To-Speech erklärt. Hierfür muss für jede Implementierung eines Schritts eine spezifische Anleitung im Code in Form eines Strings geschrieben werden. Die nötigen Prozesse, um diese Anleitung während des Tutorials vorzulesen sind in der abstrakten Klasse vorhanden. Hier wird lediglich die entsprechende Methode der Basisklasse zum Vorlesen aufgerufen.
Neben der Anleitung wird in jedem Schritt dessen spezifischer Ablauf mit beispielsweise nötigen Objekten und Methoden zur Erkennung, ob der Schritt erfolgreich abschlossen wurde, implementiert. Als konkretes Beispiel soll hier der generische und momentan in jedem Tutoriallevel enthaltene Schritt zum Auswählen eines Objekts dienen. Dem Spieler werden Instruktionen zum Auswählen eines neu erzeugten Würfels vorgelesen. Ungefähr zeitgleich wird ein blauer Würfel in der nähe des Spielers im dreidimensionalen Raum erzeugt. An diesen Würfel wird ein Skript angehängt, um den Abschluss dieses Schrittes erkennen zu können.
Gemäß der Beschreibung in Abschnitt 8.1.2, wird in diesem spezifischen Schritt ein Event getriggert, wenn der neu erzeugte Würfel mittels der Tasten auf dem XBox Controller ausgewählt wurde. Damit ist der aktuelle Schritt abgeschlossen und falls vorhanden, wird nun der nächste dieses Levels eingeleitet. In solchen weiteren Schritten können zum einen ebenfalls neue Objekte mit bestimmten Eigenschaften oder angehängten

Skripten erzeugt werden. Zum anderen kann das Verhalten vorhandener Objekte beispielsweise durch Entfernen angehängter oder Hinzufügen neuer Skripte geändert werden. Im konkreten Beispiel wird nach der Auswahl des Würfels, zu diesem ein Skript zur Erkennung der Auswahl des Rotationsmodus angehängt.

8.2. Erwägungen über die Umsetzung

Mit der aktuellen Implementierung ist eine gute Basisfunktionalität für Tutorials gegeben. Die spezifischen Implementierungen der Tutoriallevel für Translation, Rotation und Skalierung von Objekten mittels XBox Controller sind nicht besonders umfangreich, aber für einen neuen Benutzer leicht verständlich. Aufgrund der flexiblen Struktur können zukünftig problemlos neue Level bzw. auch weitere Schritte zu vorhandenen Levels hinzugefügt werden.
Neuen Benutzern kann in der Zukunft das Durchspielen des Tutorials und somit auch generell der Umgang mit der Software aufgrund einer guten Anleitung noch mehr erleichtert werden. Dafür kann die vorhandene Basisfunktion durch neue Funktionen erweitert werden. Zum Beispiel kann man zusätzlich zum Vorlesen, die Texte der Anleitungen auch mittels *Canvas* zum Lesen bereitstellen. Weiterhin wäre ein Mechanismus zur Auswahl bestimmter Level nützlich, damit sich ein Benutzer mit nur für ihn wichtigen Tutorials befassen kann.

9. PosiStageNet Visualisierung

9.1. Einleitung

Eine zu implementierende Teilfunktion des Frameworks war die Visualisierung von beweglichen Objekten der realen Welt in der Augemented Reality Anwendung. Diese beweglichen Objekte werden dabei von einem Fremdsystem getrackt oder gesteuert. Das Fremdsystem sendet dabei durchgehend die aktuellen Objektdaten im Format des PosiStageNet Protokolls in das Netzwerk. Diese Daten zu empfangen und auf der HoloLens zu visualisieren ist Gegenstand dieser Teilfunktion.

9.2. PosiStageNet

PosiStageNet ist ein von der VYV Corporation entwickeltes Open Source Protokoll zum Übertragen vom Positionsdaten über das Netzwerk. Hauptanwendungsgebiet ist dabei die Übertragung von Daten in Lichtsystemen, um unter anderem Position, Geschwindigkeit und Orientierung einzelner Beleuchtungselemente zu übertragen. Jedoch lässt sich das Protokoll auch auf andere Anwendungsfälle anwenden. Die Übertragung ist Ethernet basiert und sendet Datenpakete als UDP-Multicast in das Netzwerk, jedes Teilsystem im selben Netzwerk kann so die Daten empfangen und entsprechend verarbeiten. Die digitale Referenz auf ein einzelnes Objekt wird dabei als Tracker bezeichnet und ist über die Tracker ID eindeutig identifizierbar [7].

PosiStageNet unterscheidet zwischen Info und Daten Paketen. Die Info Pakete enthalten den Namen des senden Systems sowie die Namen der einzelnen Tracker. Die Datenpakete enthalten die eigentlichen Tracking Daten. Die Datenübertragung der Pakete ist standardmäßig auf 1Hz für Info Pakete und 60Hz für Daten Pakete definiert und die Größe der einzelnen Pakete ist auf 1500 Bytes begrenzt. Dies kann dazu führen, dass große Pakete geteilt werden müssen, PosiStageNet definiert deshalb die Unterteilung von Paketen in einzelne Chunks, welche rekursiv aufeinander aufbauen. Jeder Chunk besitzt einen 32-Bit Header der Art des Chunks sowie Länge der enthaltenen Daten bestimmt und angibt ob weitere Subchunks enthalten sind. Der Aufbau sowie Inhalt der zwei Pakettypen ist in Abbildung 16 dargestellt [7].

Chunk Header	Chunk Data	Type
{0x6756} PSN_INFO_PACKET		pch32
{0x0000} PSN_INFO_PACKET_HEADER		pch32
	packet_timestamp	uint64
	version_high	uint8
	version_low	uint8
	frame_id	uint8
	frame_packet_count	uint8
{0x0001} PSN_INFO_SYSTEM_NAME		pch32
	system_name[chunk data len]	char[]
{0x0002} PSN_INFO_TRACKER_LIST		pch32
{tracker_id} PSN_INFO_TRACKER		pch32
{0x0000} PSN_INFO_TRACKER_NAME		pch32
	tracker_name[chunk data len]	char[]

(a) PosiStageNet Info Packet

Chunk Header	Chunk Data	Type
{0x6755} PSN_DATA_PACKET		pch32
{0x0000} PSN_DATA_PACKET_HEADER		pch32
	packet_timestamp	uint64
	version_high	uint8
	version_low	uint8
	frame_id	uint8
	frame_packet_count	uint8
{0x0001} PSN_DATA_TRACKER_LIST		pch32
{tracker_id} PSN_DATA_TRACKER		pch32
{0x0000} PSN_ DATA_TRACKER_POS		pch32
	pos_x	float
	pos_y	float
	pos_z	float
{0x0001} PSN_ DATA_TRACKER_SPEED		pch32
	speed_x	float
	speed_y	float
	speed_z	float
{0x0002} PSN_ DATA_TRACKER_ORI		pch_32
	ori_x	float
	ori_y	float
	ori_z	float
{0x0003} PSN_ DATA_TRACKER_STATUS		pch_32
	validity	float
{0x0004} PSN_ DATA_TRACKER_ACCEL		pch32
	accel_x	float
	accel_y	float
	accel_z	float
{0x0005} PSN_ DATA_TRACKER_TRGTPOS		pch32
	trgtpos_x	float
	trgtpos_y	float
	trgtpos_z	float

(b) PosiStageNet Data Packet

Abbildung 16. Aufbau der PosiStageNet Pakete [7]

Es existieren zwei offizielle PosiStageNet Implementierungen. Die Implementierungen umfassen das komplette Protokoll für den Datentransfer sowie client- und serverseitige Netzwerkendpunkte. Die Implementierungen übernehmen die Synchronisierung der Tracker zwischen Client und

Server. Die erste Implementierung ist für C++ und wurde von VYV selbst entwickelt [8]. Die zweite ist für DotNet und wurde von David Butler entwickelt [9].

9.3. Implementierung

Die PosiStageNet Visualisierung innerhalb des Frameworks besteht aus drei Hauptkomponenten, der PosiStageNet Implementierung, dem PsnVisualizationManager sowie dem PsnVisualizationObject. Die Psn Implementierung ist die im vorherigen Kapitel vorgestellte DotNet Implementierung von David Butler. Diese wurde für den Einsatz auf der HoloLens leicht modifiziert. Genutzt wird hiervon der Client, um Tracker Daten von einem Fremdsystem zu empfangen und diese innerhalb des Frameworks verfügbar zu machen. Der PsnVisualizationManager ist die Hauptklasse der PosiStageNet Visualisierung. Sie erzeugt den Client und verwaltet die PsnVisualizationObjects. PsnVisualizationObject übernimmt die Visualisierung für einen Tracker, es enthält die für Visualisierung notwendigen Daten, welche vom PsnVisualizationManager mit den Daten des Clients synchronisiert werden.

9.3.1. Modifizierung der PosiStageNet Implementierung. Ein generelles Problem bei der Entwicklung von Unity Apps für die HoloLens sind die unterschiedlichen DotNet Versionen von Unity und UWP. Dies führt besonders bei neuer DotNet Funktionalität zu Problemen, wenn diese Funktionalität in der einen Versionen vorhanden ist und der anderen fehlt.

Dieses Problem trifft auch auf die PosiStageNet DotNet Implementierung zu, so ist diese zwar mit UWP kompatibel, kompiliert jedoch im Unity Editor nicht. Um dieses Problem zu lösen wurden zunächst allen Dateien der PosiStageNet Implementierung mittels Präprozessoranweisungen vom Kompilieren außerhalb des UWP Targets ausgeschlossen. Ein Beispiel dazu findet sich in Codeausschnitt 1.

```
#if WINDOWS_UWP
/* Code here builds only in UWP */
#endif
```

Listing 1. Präprozessoranweisung für UWP Target bedingten Code

Das zweite Problem bei der Portierung der der PosiStageNet DotNet Implementierung war der Code zur Netzwerkkommunikation innerhalb des Clients. Selbiges gilt für den Server, welcher jedoch nicht benötigt wird und komplett entfernt wurde. Das Problem hier war, dass die verwendete Socket Implementierung nicht auf dem UWP Target unterstützt wird, da dort eine eigene Socket Implementierung von Windows genutzt werden muss. Um dies zu lösen wurden die betroffenen Codestellen auf Basis der UWP Socket Implementierung neu implementiert.

Bei der neu Implementierung des Netzwerkcodes gab es zunächst auch Probleme eine Verbindung zu einigen WLAN Netzwerken herzustellen, beziehungsweise die Daten anschließend auf der HoloLens zu empfangen. Dieses Problem wurde durch die explizite Auswahl des IPv4 Netzwerkadapters der HoloLens gelöst, sodass nun die Übertragung von PosiStageNet Paketen auch in den vorher erfolglos getesteten Netzwerken funktioniert.

Mit dem nun funktionierenden Netzwerkcode lassen sich über den Client PosiStageNet Pakete empfangen. Nach Instanziierung des Clients dekodiert dieser alle auf der IP-Adresse empfangenen Pakete und stellt die synchronisierten Tracker in einem Dictionary unter deren Tracker ID zur Verfügung.

9.3.2. PsnVisualizationObject. Das PsnVisualizationObject ist für die Visualisierung eines Trackers verantwortlich. Dazu hält es zum einen eine Kopie der aktuellen Trackerdaten, sowie einige Unity Objekte für die Visualisierung. Die Visualisierungsobjekte werden bei jedem Frame entsprechend der lokalen Trackerdaten modifiziert. Zu den Objekten für die Visualisierung zählen zwei Kugeln für den Start- und Endpunkt einer Objektbewegung und ein LineRenderer für die Visualisierung des Weges. Zusätzlich lassen sich Daten wie ID, Name, Position und Geschwindigkeit in Textform auf einem Billboard Canvas anzeigen. Letzteres lässt sich über das zweckentfremdete Validity Flag eines Trackers an und ausschalten.

9.3.3. PsnVisualizationManager. Der PsnVisualizationManager ist das Bindeglied zwischen dem Client, welcher asynchron Tracker-Daten aus dem Netzwerk empfängt und den PsnVisualizationObjects welche diese innerhalb des Unity Haupt-Threads visualisiert. Der PsnVisualizationManager besitzt ähnlich wie der Client ein Dictionary, jedoch werden hier unter der ID des Trackers nicht die Tracker selbst verwaltet, sondern die PsnVisualizationObjects. Die Verbindung zwischen Tracker und PsnVisualizationObjekt wird also durch die Tracker ID als eindeutiger Schlüssel hergestellt.

Bei jedem Frame wird nun über alle Tracker des Clients iteriert und entsprechend der Tracker ID das passende PsnVisualizationObject aus dem Dictionary mit den aktuellen Daten aktualisiert. Ist noch kein PsnVisualizationObject unter der Tracker ID hinterlegt wird es erzeugt.

Um die implementierte Funktionalität zu Nutzen muss nun im Unity Editor der Szene ein neues Objekt hinzugefügt werden. Diesem Objekt wird anschließend das PsnVisualizationManager Skript als Komponente angehängt. Dort lassen sich die IP-Adresse und der Port auf dem die PosiStageNet Pakete empfangen werden sollen konfigurieren. Eine Beispielkonfiguration des VisualizationsManagers ist in Abbildung 17 dargestellt.

Kompiliert man nun die Anwendung und startet diese auf der HoloLens werden PosiStageNet Daten unter der konfigurierten IP-Adresse empfangen und visualisiert. Dies setzt voraus, dass ein PosiStageNet Server im Netzwerk Daten sendet. Ein Beispiel für die Visualisierung ist in Abbildung 18 dargestellt. Dort werden drei Tracker visualisiert, die beiden Tracker mit der vertikalen Bewegung besitzen zusätzlich ein Label mit zusätzlichen Informationen.

9.3.4. Testdaten-Server. Um sowohl das Empfangen als auch die Visualisierung zu testen wurde ein Server zum

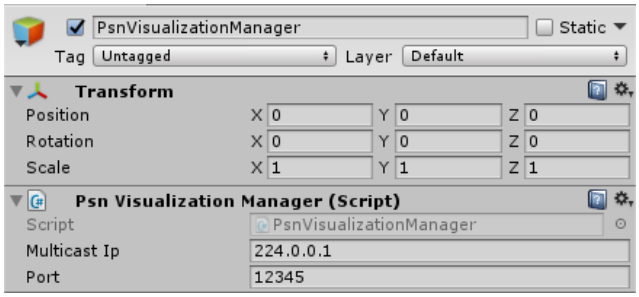


Abbildung 17. Konfiguration des PsnVisualizationManagers in Unity

Abbildung 18. Beispiel zur Visualisierung von drei Trackern

Senden von Testdaten entwickelt. Der Server basiert auf einer Beispielimplementierung, die Teil der PosiStageNet DotNet Implementierung ist. Dieser Server wurde leicht modifiziert, sodass er eine flüssige Bewegung von einer bestimmten Anzahl von Objekten simuliert. Der Server schickt Infopakete mit 1Hz und Datenpakete mit 5Hz.

9.3.5. Fazit und Ausblick. Das implementierte Teilmodul läuft stabil und lässt sich unabhängig vom Rest der Module benutzen. Jedoch gibt es noch zwei kleinere Probleme. Zum einen ist das Dekodieren der PosiStageNet Pakete und deren Visualisierung auf Grund der großen Datenmengen sehr rechenintensiv, was sich auf der HoloLens sehr schnell bemerkbar macht. Die derzeitige Lösung dazu ist die Senderate des Server gering zu halten. Ein zweites Problem ist der PosiStageNet DotNet Implementierung zu schulden, welche konstant Tracker in bestimmten Abständen nicht korrekt synchronisiert. Die Vermutung dahinter ist ,dass in mehrere Pakete gesplittete Datensätze beim Empfangen nicht korrekt rekonstruiert werden. eine mögliche Lösung ist das Upgrade auf eine neue Version der Implementiernug. Insgesamt wurden die zu Beginn des Semesters definierten Features umgesetzt und das implementierte Teilmodul bietet einen guten Grundstein für Objekt-Tracking und Visualisierung mit dem PosiStageNet Protokoll.

10. Multiplayer

10.1. Einleitung

Ein sinnvoller Anwendungsfall für die AURA Applikation ist es, Szenen zusammen, auf mehreren, sowohl HoloLens- als auch Desktop- und Tablet-Geräten in Echtzeit bearbeiten und betrachten zu können. Dieser Bericht beschäftigt sich mit der Realisierung der AURA Multiplayer Schnittstelle. Hierzu wird zunächst die Problemstellung und Ausgangssituation der Anforderung erläutert um nachfolgend die allgemeine Funktionalität der *Unity Netzwerkschnittstelle* zu erörtern und das daraus abgeleitete, dieser Implementierung zu Grunde liegende, Konzept vorzustellen. Darüber hinaus wird die Eingliederung der Implementierung in das Gesamtprojekt dargestellt und das Ergebnis, eine Evaluierung des Entwicklungsprozess und ein Fazit gegeben.
Abschließend werden in diesem Entwicklungszyklus noch nicht implementierte aber sinnvolle Funktionalitäten in einem Ausblick festgehalten.

10.2. Problemstellung

Als eine weitere Funktionalität des AURA Projektes, sollte in diesem Entwicklungszyklus die Möglichkeit implementiert werden, die Applikation netzwerkfähig zu machen. Im Einzelnen bedeutet dies, dass Objekte (Assets), die in die Szene geladen oder auch gelöscht werden, auf allen zu diesem Zeitpunkt zur Netzwerk-Sitzung verbundenen Clients synchronisiert werden. Weiterhin soll die Position und Bewegung dieser Objekte über das Netzwerk auf allen verbundenen Clients synchronisiert werden. Hierbei soll es egal sein, ob sich ein Client zu Beginn bereits in der Netzwerk-Sitzung befand oder erst nach einiger Zeit beitritt. Die bereits in der Szene vorhandenen Objekte sollen auch für ihn instanziiert und synchronisiert werden. Außerdem soll eine Art Autorität-Management für Objekte in der Szene bereitgestellt werden, um zu gewährleisten, dass Objekte nur von einem Client zur selben Zeit manipuliert werden können. Darüber hinaus soll es die Möglichkeit geben über das Hauptmenü eine Netzwerk-Sitzung sowohl zu eröffnen als auch zu betreten.
Die Multiplayer- oder auch Netzwerk-Funktionalität soll sowohl für die HoloLens- als auch für eine Desktop- (Windows) und Tablet-Version (Android) der Software zur Verfügung stehen.
Unity selbst bietet für diese Art von Funktionalität bereits eine Netzwerk Schnittstelle und viele der benötigten Methodiken. Zum Beispiel ist der *Netzwerk-Manager*, der die Funktionalität des Servers übernimmt, bereits vorimplementiert und wird im nächsten Kapitel vorgestellt.

10.2.1. Ausgangssituation. Die Ausgangssituation für die Umsetzung der Multiplayer Funktionalität war die Version der AURA Software des letzten Entwicklungszyklus. Schon damals wurde über die Realisierung einer Netzwerk Schnittstelle nachgedacht, jedoch wurde in dieser Richtung bisher

nichts implementiert. Es galt die Funktionalität ohne Vorkenntnisse zum AURA Projekt, der Unity Engine oder der Unity-Multiplayer Schnittstelle zu konzeptionieren und in die vorhandene Version des AURA Projekts zu integrieren, sodass alle bisher entwickelten und zukünftig entwickelten Funktionalitäten netzwerkfähig sind.

10.3. Konzept

In diesem Kapitel wird die der Unity Netzwerkschnittstelle zu Grunde liegende Logik erläutert und das daraus für uns resultierende Umsetzungskonzept vorgestellt.

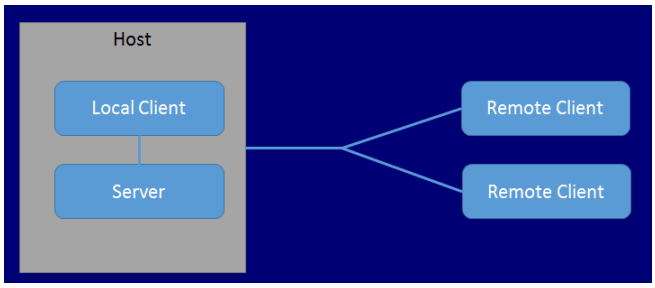


Abbildung 19. Die Unity Client-Server Architektur. [10]

10.3.1. Unity Networking. Abbildung 18 zeigt die grundsätzliche Unity Netzwerk Client-Server Architektur. Hierbei ist die Netzwerkinstanz, die eine Sitzung eröffnet, auch gleichzeitig ein lokaler Client, dementsprechend Server und Client in einem. Alle anderen Clients verbinden sich zu diesem Server und werden als ein *Player Objekt* dort repräsentiert. Dieses Objekt enthält die Verbindungsdaten des jeweiligen Clients. Abbildung 19 zeigt eine beispielhafte Kommunikation zwischen Client und Server. Hierbei ist zu beachten, dass instanziierte Netzwerk-Objekte auf allen Netzwerk Instanzen existieren. Der Server spielt hier die Rolle eines Vermittlers, der Manipulationen der Objekte registriert und sie an alle Clients weitergibt. Diese Art der Kommunikation nennt man *ClientRPCs*. Funktionen die vom Server auf allen verbundenen Clients aufgerufen werden.
In der anderen Richtung, vom Client zum Server gibt es sogenannte *Commands*, die vom Client auf dem Server aufgerufen werden. So kann zum Beispiel vom Client ein Objekt auf dem Server instanziiert werden, das wiederum vom Server allen verbundenen Clients bekanntgemacht wird und per ClientRPC synchronisiert wird. Wichtig ist hierbei die Tatsache, dass alle Objekte auf allen Clients existieren und auch alle Methodenaufrufe auf allen Clients ausgeführt werden. Jeder Client kann dann selbst, anhand von Steuervariablen, wie zum Beispiel *isLocalPlayer* differenzieren, ob der Aufruf an ihn gerichtet ist und ausgeführt wird, oder nicht.

10.3.2. Ableitung des ersten Prototyps. Aus dem zuvor erläuterten Kommunikationskonzept wurde zunächst ein einfacher, vom AURA Projekt losgelöster Prototyp entwickelt um

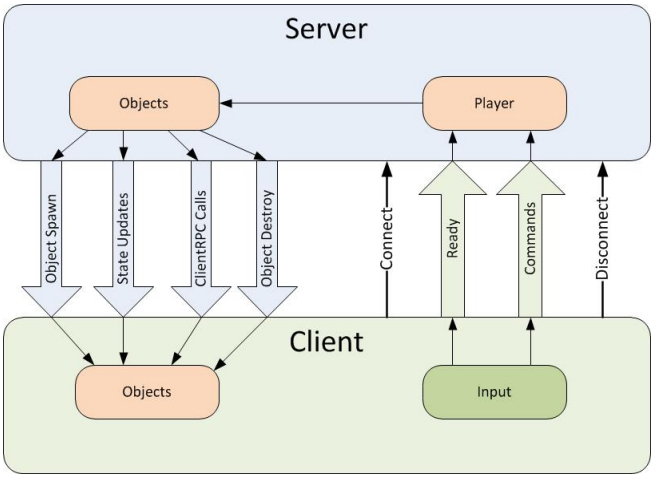


Abbildung 20. Möglichkeiten der Kommunikation zwischen Server und Client in Unity. [10]

die Netzwerkfunktionalität von Unity zu testen. Hierbei wurde zunächst eine Serverinstanz in Form der Unity *Network Manager* Komponente mit zugehörigem Netzwerk-Sitzungs-Management erstellt. Weiter wurde ein Script geschrieben, das simple Objekte auf Knopfdruck per Command auf dem Server und somit im Netzwerk in eine leere Szene instanziiert. Abbildung 20 zeigt die ersten Kommunikationsaufrufe dieses Prototyps. Hierbei waren alle Objekte die instanziiert werden konnten, als Unity Prefabs vorher erstellt worden und sowohl dem Server als auch jedem Client bekannt. Im Klartext bedeutet dies, dass jede Instanz im Netzwerk alle Objekte kennt und in dem Moment wo neue Objekte instanziiert werden, weiß um welche Art von Objekt es sich handelt. Für die spätere Implementierung in Verbindung mit einem Server von dem Assets dynamisch, bei Bedarf, geladen werden, würde dieser Prototyp nicht funktionieren, da nicht jeder Client das Objekt kennt das ein anderer Client vom Server lädt und instanziieren möchte. Dass dies Probleme bereiten würde, war uns allerdings zu diesem Zeitpunkt noch nicht klar.
Grundlegend ist festzuhalten, dass in diesem Prototyp Objekte instanziiert werden, und diese beweglich und greifbar sind. Eine simple Variante eines Autoritätsmanagements war ebenfalls implementiert. Wobei sobald ein Objekt aufgehoben wird, die Autorität beim Server angefragt, alle Autoritäten vom Objekt entfernt und anschließend die Autorität an den anfragenden Client gegeben wird. Es war also möglich, dass mehrere Clients Objekte instanziieren, durch den Raum bewegen und die Position und Bewegung auf allen Clients zu sehen war.

10.4. Implementierung

In diesem Kapitel wird die Implementierungsphase, dabei aufgetretene Schwierigkeiten und deren Lösung vorgestellt. Es soll ein Überblick über den Multiplayer-Entwicklungsprozess mit dem Unity-Editor auf Desktop-Geräten und der HoloLens entstehen. Vorab ist festzuhalten,

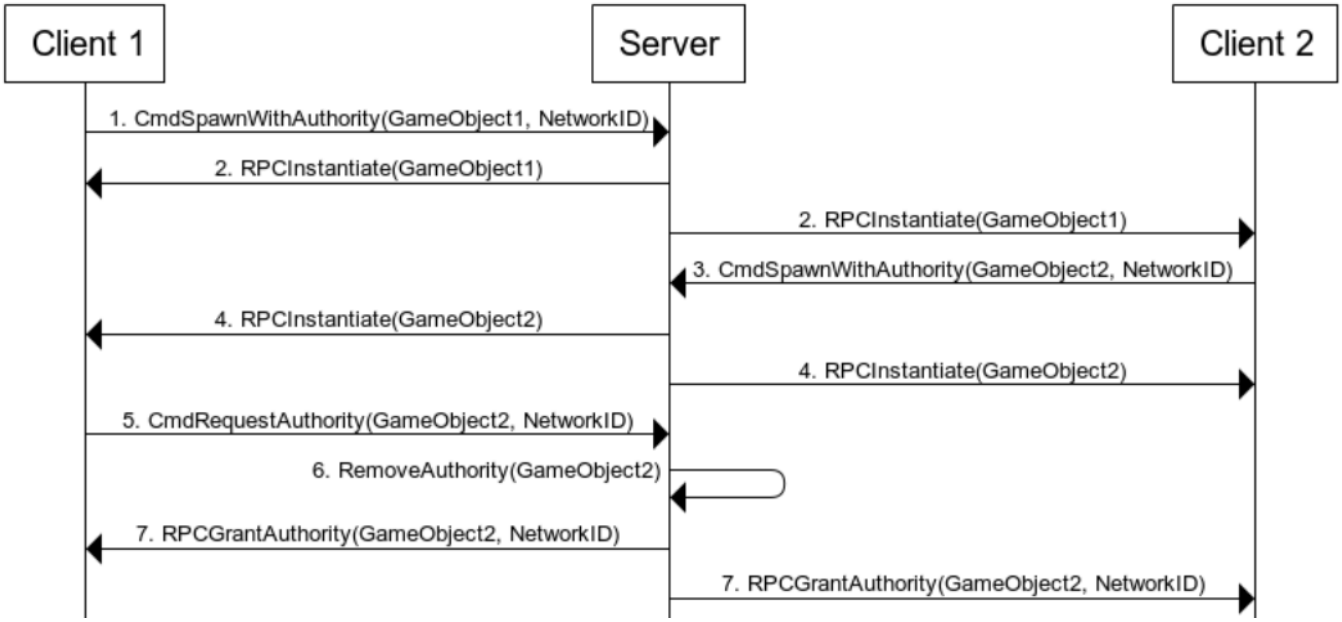


Abbildung 21. Kommunikations-Sequenzdiagramm des ersten Prototypen. Das in den Methoden 1. und 3. übergebene GameObject ist das zu instanziierende allen Netzwerk-Instanzen bereits bekannte Prefab. NetworkID ist die eindeutige Identifizierung des Clients der den Command aufgerufen hat. Diesem Client wird nach der Instanziierung die Autorität über das Objekt zugewiesen. Wenn wie in 5. ein Client die Autorität über ein Objekt anfordert, werden zunächst alle Autoritäten vom Objekt entfernt (6.) und dann per RPC allen Clients der neue Autoritätsinhaber mitgeteilt (7.).

dass während der Entwicklungsarbeit die Unity-Version von 2017.4.1 auf 2017.4.3 geändert wurde, da Version 2017.4.1 auffällige Probleme (Abstürze) mit dem Code der Unity Netzwerkschnittstelle hatte. Es war unmöglich unter diesen Umständen effektiv zu entwickeln. Außerdem war es nötig die Funktionalitäten mit zwei Geräten, zu Beginn Windows-Desktop-PCs, später dann sowohl Desktop-PC als auch HoloLens, zu testen. Um nun den Code auf beiden Geräten gleich zu halten, wurden einige "Synchronisierungs"-Commits auf dem Multiplayer-Entwicklung-Branch erstellt. Diese Art der Entwicklung ist sehr fehleranfällig, dauert lang und ist sehr mühsam. Alles in allem wurde etwa 40% der Zeit auf Testen und technische Probleme verwendet.

10.4.1. Problematik: Dynamisch geladene Objekte. Wie bereits erwähnt, stießen wir beim Übertragen der Prototyp-Funktionalität in die AURA Applikation bereits nach wenigen Schritten an eine Grenze. Die vom Asset-Server geladenen Objekte, konnten nicht auf dem Server und somit auch nicht auf Clients instanziiert werden, da sie das Objekt nicht kennen. Demnach musste zunächst dem Server und allen Clients ein Objekt, das vom Server geladen wurde, bekannt gemacht werden. Es war jedoch nicht möglich ein Objekt als GameObject per ClientRPC an einen Client zu übergeben.

Lösung. Dieses Problem wurde umgangen indem ein vorher jedem bekanntes Netzwerk-Objekt als Platzhalter Prefab in die Applikation integriert wurde. Dieses Objekt fungierte dann als Netzwerk-Objekt über das Manipulationen am eigentlichen Asset auf Netzwerkebene synchronisiert werden. Das eigentliche Asset wird über seinen Namen bei jedem Client lokal instanziiert und als Child-Objekt an das Netzwerk-Objekt angehängt. Der Benutzer greift das Asset-Child-Objekt, dessen Bewegung auf sein Parent-Objekt übertragen wird. Dieses Parent-Netzwerk-Objekt wird vom Server auf allen Clients synchronisiert, und synchronisiert wiederum sein Child-Objekt (das eigentliche Asset). Die Synchronisation der Parent-Netzwerk-Objekte übernimmt die Unity *Network-Transform* Komponente die Teil des Netzwerk-Servers ist.

10.4.2. Problematik: Nachladen von Objekten für später beitretende Clients & Laden von gespeicherten Szenen. Wenn ein Client der Netzwerk-Sitzung erst beitritt, wenn bereits Objekte instanziiert wurden, müssen diese Objekte für ihn nachgeladen werden. Auch wenn abgespeicherte Szenen geladen und ihre Objekte auf allen verbundenen Clients instanziiert werden sollen kommt es zu dieser Problematik. Das Problem dabei ist, dass gleichzeitige Anfragen vom Asset-Server nicht bearbeitet werden können. Es muss eine Art Wartefunktion implementiert werden, die abwartet bis die vorausgehende Anfrage an den Asset-Server abgearbeitet wurde.

Lösung. Für die Lösung dieses Problems bietet Unity in Verbindung mit C# die Möglichkeit sogenannte Co-Routinen auszuführen, die von Unity intern zeitlich eingeplant und nacheinander abgearbeitet werden. Für die Abarbeitung der Anfragen an den Asset-Server ist dies eine gute Lösung da die Funktion aus der die Anfrage besteht kein komplexer Netzwerk-Funktionstyp, wie zum Beispiel ein Command oder RPC ist. Commands und RPCs können nicht als Co-Routine ausgeführt und abgearbeitet werden.

10.4.3. Problematik: Konnektivität. Während der Tests der Multiplayer Funktionalität an verschiedenen Locations in verschiedenen Netzwerken kam es immer wieder zu Problemen mit der Konnektivität im Allgemeinen. So war es teilweise nicht möglich sich im selben Netzwerk befindliche Clients miteinander zu verbinden. Auch war häufig keine Konnektivität zum Asset-Server gegeben.
Eine Lösung für dieses Problem ist nicht in der Multiplayer Implementierung zu suchen sondern es handelt sich hierbei um ein generelles Problem, das auf der darunter liegenden Netzwerk-Schicht zu beheben ist. Viele Netzwerke verfügen über Firewalls und Beschränkungen was Datentransfer im Netzwerk betrifft. So müssen zum Beispiel bereits auf den HoloLens und Windows Clients und Servern Freigaben in der Windows-Firewall gemacht werden, um eine Konnektivität überhaupt erst zu erlauben.

10.5. Ergebnisse

Folgende Funktionalitäten wurden umgesetzt und funktionieren reibungslos auch im Zusammenspiel mit anderen Komponenten:

10.5.1. Netzwerkmanagement. Es wurde eine Client-Server Verbindungsarchitektur umgesetzt bei der der Server auch gleichzeitig einer der Benutzer ist. Der Server eröffnet eine Multiplayer-Sitzung. Alle Spieler werden durch sogenannte Player-Objekte beim Server registriert und repräsentiert. Die Umsetzung dieser Funktionalität benutzt die Unity Network-Manager Komponente.

10.5.2. On-Demand Laden. Diese Funktionalität verwendet die AssetManager Klasse um Objekte dynamisch zu laden. Sei es beim Laden einer Szene oder beim Spawnen eines Assets. Außerdem übernimmt sie das Nachladen von Objekten für Benutzer die der Sitzung später beitreten. Objekte bestehen grundsätzlich aus zwei Komponenten; dem Parent-Netzwerk-Objekt, das über das Netzwerk synchronisiert wird und dem an dieses als Child-Objekt angehängte eigentliche Asset.

10.5.3. Objekt-Synchronisation. Zur Synchronisation wird die Unity Network Transform Komponente verwendet. Sie synchronisiert Manipulationen für alle Netzwerk-Objekte, die diese Bewegungen an ihre angehängten Child-Objekte weitergeben. Dies inkludiert Bewegungen, Rotationen und Skalierungen.

10.5.4. Objekt Autoritäts-Management. Zur Vergabe und Verwaltung der Autorität über Objekte in einer Szene wurde ein Autoritäts-Management realisiert. Darüber kann ein Benutzer eine Autorität beim Server anfragen. Der Server entfernt dann die aktuelle Autorität für dieses Objekt und gibt sie dem anfragenden Benutzer. Hierbei besteht eine Kopplung der Autorität und dem UserInput, sodass auch keine lokale Manipulation der Objekte möglich ist wenn keine Autorität vorhanden ist.

Darüber hinaus ist es möglich aus einer bestehenden Singleplayer-Szene in eine Multiplayer-Sitzung zu wechseln.

10.6. Fazit

Abschließend bleibt festzuhalten, dass die wichtigsten Funktionen für einen funktionierenden Multiplayer implementiert wurden. Es existiert eine gesunde, funktionierende Basis auf der nun aufgebaut, weiterentwickelt und optimiert werden kann.

10.7. Ausblick

Weiterführende Entwicklungsarbeit an der Multiplayer-Funktionalität könnte sich zum Beispiel mit der Integration und Anbindung weiterer AURA Features, wie zum Beispiel der Android Version der Software oder dem weiterentwickelten Asset-Server, beschäftigen. Aber auch das dynamische Laden von Assets im Allgemeinen kann noch optimiert werden. Genauso muss noch eine Host-Wechsel oder auch Host-Drop Kompensation eingebaut werden, sodass es möglich ist zur Laufzeit den Host der Multiplayer-Sitzung zu wechseln.
Darüber hinaus muss an der starken Kopplung zu anderen Komponenten gearbeitet werden. Es sollte eine tatsächliche reine Multiplayer-Schnittstelle erarbeitet und zur Verfügung gestellt werden.

11. Android Kompatibilität

Um beim Szenen-Aufbau eines HoloLens-Nutzers teilhaben zu können wurde an einer Android Version der HoloLens-Applikation gearbeitet. Ziel ist es auf einem Android Smartphone oder Tablet beobachten zu können, wie die Szene von HoloLens-Nutzern aufgebaut wird.

11.1. Umsetzung

Bei der Android Version wurde darauf geachtet, dass kein Android spezifischer Code auf der HoloLens ausgeführt wird. Dazu wurde der Präprozessor Befehl #if UNITY_ANDROID verwendet. Es wird damit sichergestellt, dass Code innerhalb eines solchen Blocks ausschließlich auf Android-Endgeräten ausgeführt wird.

Systemvoraussetzung ist mindestens Android 4.1 (API Level 16).

11.1.1. Touchscreen Steuerung. Die Veränderung der Sicht auf die Szene, die bei der HoloLens durch einfaches Drehen und Bewegen der HoloLens geschieht, funktioniert auf einem Android Endgerät durch Touchgesten. Das Bewegen in der Szene wird durch Drehung und Rotation der Kamera realisiert. Zur Änderung der Blickrichtung und der Entfernung der Kamera innerhalb einer Szene stehen folgende zwei Gesten zur Verfügung:

- **Swipe-Geste** Durch ein einfaches Swipen kann die Kamera in Swipe-Richtung rotiert werden. Der Grad der Kamerarotation wird durch die Länge der Swipe-Geste bestimmt.
- **Pinch-Geste** Bei zueinander gehender Fingergeste erfolgt ein Herauszoomen aus der Szene. Bewegen sich die Finger auseinander, erfolgt ein Zoomen in Blickrichtung.

Die Auswahl eines Assets sowie das Aufnehmen und Ablegen eines Assets erfolgt durch eine Double-Tap-Geste. Realisiert wurde dies folgendermaßen:

- **Laden von Assets** Zunächst muss der Gaze-Pointer auf ein Asset im Assetstore gerichtet sein. Durch anschließenden Double Tap wird das Asset geladen und angezeigt.
- **Bewegen von Assets** Assets können nach initialer Positionierung erneut durch einen Double Tap ausgewählt werden. Mittels Swipe-Gesten kann das Objekt an einer anderen Stelle positioniert werden. Nach erneuten Double Tap wird das Objekt deselektiert und abgelegt.

11.2. Ausblick

Die Anwendung wurde so angepasst, dass sie auch auf einem Android-Endgerät lauffähig ist. Es ist möglich sich in der Szene mittels Touchgesten zu orientieren und bewegen. Außerdem können Assets geladen als auch verschoben werden. Um HoloLens-Nutzern beim Aufbau der Szene beobachten zu können, fehlen jedoch noch die folgenden zwei Funktionalitäten:

- Verbindungsmöglichkeit von Android zu HoloLens (Multiplayerfunktionalität für Android).
- der Assetserver muss Assets bzw. Assetbundles für Android bereitstellen.

12. Abschlussbetrachtung

Durch eine starke Teamleistung, die sich durch kontinuierliches und fokussiertes Vorgehen äußerte, war es möglich die geplanten Ziele und Erwartungen der Stakeholder mit großem Erfolg zu erfüllen. Da der Großteil der Entwickler keine weitere Teilnahme an diesem Projekt anstrebt, wird abzuwarten sein wie sich das AURA-Projekt zukünftig entwickelt. Hierbei wurden allerdings Wege aufgezeigt, die ein weiteres Fortbestehen des Projekts sichern werden. Letztendlich konnten alle Projektteilnehmer (Stakeholder, Product Owner und Entwicklerteam) das Projekt, aufgrund der Durchführung und Erreichung aller Ziele, als großartigen Erfolg verbuchen.

Literatur

[1] Jonas Bien, Tim Bienias, Fabian Brenner, Ibrahim Cinar, Ahmed Danyal, Hakan Durgel, Lukas Hutfleß, Kristian Krömmelbein, Arnold Lockstein, Tobias Michalski, Dirk Peters, Fabian Seidl, Sven Steininger *MPSE HoloLens - Technischer Bericht*. Hochschule Darmstadt, Darmstadt, Deutschland, 2018.

[2] Tague, Nancy R. (2005) [1995]. "Plan–Do–Study–Act cycle". The quality toolbox (2nd ed.). Milwaukee: ASQ Quality Press. pp. 390–392. ISBN 0873896394.

[3] Was beim Linken auf eine LGPL-Programmbibliothek zu beachten ist - IT Fachanwalt Steinle http://www.it-rechtsanwalt.com/open-source-software/was-beim-linken-auf-eine-lgpl-programmbibliothek-zu-beachten-ist-4382.php

[4] HAFTUNGSBESCHRÄNKUNGEN IN IT-VERTRÄGEN - Dr. Thomas Helbing https://www.thomashelbing.com/de/haftung-gewaehrleistung-fuer-software-diesen-tipps-reduzieren-risiko-it-anbieter

[5] Vereinsrecht - BGB http://www.gesetze-im-internet.de/bgb/BJNR001950896.html#BJNR001950896BJNG000502377

[6] https://docs.unity3d.com/Manual/Shaders.html

[7] PosiStageNet Protocol description v2.0, http://www.posistage.net/wp-content/uploads/2016/09/PosiStageNetprotocol_v2.02_2016_09_15.pdf

[8] PosiStageNet Downloads, http://www.posistage.net/posistagenet-specification/

[9] PosiStageDotNet, https://github.com/impsnldavid/posistagedotnet

[10] Unity Online Multiplayer Dokumentation https://docs.unity3d.com/Manual/UNet.html