

Fernunterstützung und Zusammenarbeit mit 3D Punktwolken

Masterarbeit
von

Kai Westerkamp

An der Fakultät für Informatik
Fraunhofer IOSB (IAD)

Erstgutachter:	Prof. Dr.-Ing. Rainer Stiefelhagen
Zweitgutachter:	XXXX
Betreuender Mitarbeiter:	M.Sc. Adrian Hoppe

Bearbeitungszeit: 01.06.2017 – 30.11.2017

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Zusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

1. Einleitung	1
1.1. VR	1
1.1.1. vive	1
2. Stand der Technik	3
2.1. Section	3
2.1.1. VR AR Assistance	3
2.1.2. 3D Scan	3
2.1.3. Ungenauigkeiten im Lighthouse Tracking	3
3. Punktwolke	5
3.1. Frames aufnehmen und bereinigen	6
3.1.1. Aufnahme und Glättung	6
3.2. Zusammenfügen von Frames	6
3.2.1. Kalibrierung Kinect zu Vive	7
3.3. Ergebnisse	9
4. Speichern der Punktwolke mit 3D Tiles	11
4.1. 3D Tiles	11
4.1.1. Tileset und Tiles	11
4.1.2. glTF	12
4.1.3. Struktur	13
4.1.3.1. Buffers and Accessors	14
4.2. Umsetzung	14
5. Visualisierung	15
5.1. UE4 Rendering System	15
5.2. 3D Tiles Laden und vorbereiten	15
5.3. Rendering	16
5.4. Ergebnisse	17
6. Hololens	19
6.1. Unreal Engine Un Hololens	19
6.2. Hololens Implementierung	19
6.3. Kalibrierung	19
6.3.1. Kalibrierungsfehler	19
6.4. Ergebnisse	19
6.5. Verbesserungen	19
6.5.1. Verschieben der VR Umgebung	19
7. Evaluation	21
7.1. Versuchsablauf	21
7.2. Probanden	21

7.3.	Ergebnisse	21
7.3.1.	Stein Finden	21
7.3.1.1.	Bilder	21
7.3.1.2.	VR	21
7.3.2.	Nutzung der vorhanden Hilfsmittel	21
7.3.3.	Kommunikation	21
7.3.3.1.	Videostream	21
7.3.3.2.	VR	21
7.3.4.	Fehleranzahl	24
7.3.5.	Timings	24
7.3.6.	Unabhängigkeit	24
7.3.7.	NASA TLX und UEQ	24
7.3.8.	Allgemeines	26
8.	Ausblick	27
A.	3D Tile JSON	29
B.	Evaluations Fragebogen	30
	Literaturverzeichnis	35

1. Einleitung

(TODO)

ToDo

1.1. VR

1.1.1. vive

2. Stand der Technik

related

2.1. Section

paper

2.1.1. VR AR Assistance

[OES⁺15] Viirtuelle Proxys

2.1.2. 3D Scan

<https://www.microsoft.com/de-de/store/p/3d-scan/9nblggh68pmc>

<http://www.kscan3d.com/>

(<https://steamcommunity.com/app/507090/discussions/0/144513248276793628/>)

2.1.3. Ungenauigkeiten im Lighthouse Tracking

Ein großes Problem sind Ungenauigkeiten im Lighthouse Tracking.

Noise in der Ruhelage 0.3mm [lig] RMS (Rooted mean squared, quadratisches Mittel)
1.9mm [lig]

mein Jitter bild

Im Paper [NLL17] wurden signifikante Fehler nach Tracking Abbrüchen festgestellt. bsi zu 150cm

$2m = 1,98m$

Tracker tracking (bilder

3. Punktwolke

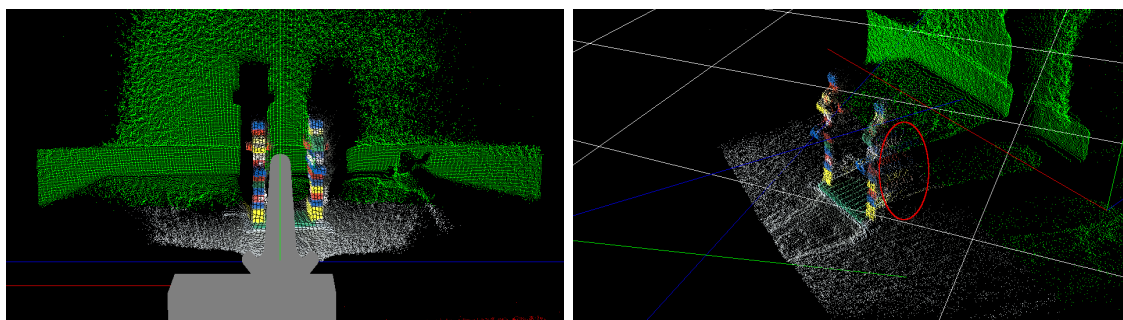
Das Erzeugen der Punktwolke soll einfach und schnell funktionieren und mit einer Kinect erfolgen. Aus einem Frame der Kinect, bestehend aus Farbbild und Tiefenbild, lässt sich einfach eine Punktwolke relativ zur Tiefenkamera der Kinect errechnen. Eine Aufnahme beinhaltet aber nur alle Informationen die aus den 2D Bildern errechnet werden können. Das heißt man erhält nur eine Seite des Objektes gut aufgelöst und einige Artefakte die sich aus dieser Berechnung ergeben. Für die Darstellung in einer VR Umgebung ist dies nicht ausreichend. Der Betrachter kann sich frei in der virtuellen Welt bewegen und erkennt schnell die nicht vorhandenen Informationen und Fehler.

Ein Fehler entsteht zum Beispiel bei Kanten und Flächen die nicht Senkrecht zur Kamera sind. Die unausreichenden Informationen in den Ausgangsdaten werden falsch angenähert und ergeben falsche Flächen. (siehe 3). **(Bild 2 ändern / vergrößern?)**

ToDo

Das zweite Problem das es zu lösen galt war das zusammenfügen von mehreren Aufnahmen aus unterschiedlichen Perspektiven zu einer großen zusammenhängenden Punktwolke. Um 2 Frames miteinander zu verbinden braucht man die relative Transformation zwischen den beiden Aufnahmen, bzw. absolute Kamerapositionen. Bei bestehenden Algorithmen wird dies zum Beispiel durch Featureerkennung oder zurückrechnen der Kamerabewegung erreicht **(quellen)**. Solche Verfahren sind meist rechenaufwändig und zeitintensiv. Für diese Arbeit war es das Ziel die Kinect mit dem Lighthouse Tracking System zu verbinden. Die Trackingdaten aus SteamVR, bzw. OpenVR geben uns eine globale Position aller

ToDo



(a) Aufnahme aus Sicht der Kinect

(b) Aufnahme von der Seite

Abbildung 3.1.: Aufnahmen aus verschiedenen Perspektiven In Bild b) sind falsche Punkte zu sehen die durch die Rekonstruktion aus einem 2D Bild entstehen.

Aufnahmen und vereinfachen das Erzeugen einer großen Punktwolke.

3.1. Frames aufnehmen und bereinigen

Als ersten Schritt wird ein Frame mit der Kinect aufgenommen und das Tiefenbild geglättet. Anschließend wird das Tiefen- und Farbbild in 3D Punkte umgewandelt und unerwünschte Punkte verworfen.

3.1.1. Aufnahme und Glättung

Das Aufnehmen einer kleinen Punktwolke wird das Kinect SDK verwendet. Sowohl Tiefenbild auch als auch Farbbild kann man aus der API erhalten. Anschließend wird das Tiefenbild geglättet. Die Rohdaten sind teilweise sehr verrauscht und so erhält man eine Punktwolke mit glatteren Flächen. Hierfür braucht man einen Filter der zwar die Flächen glättet, aber gleichzeitig Objektkanten erhält. Ein Bilateral Filter erzielt den gewünschten Effekt ist aber relativ Rechenaufwändig. Verwendet wurde der Filter der in dem Paper [MCS14] vorgestellt wird. Hierbei wird zunächst das Bild mit einem Gausfilter geglättet und anschließend mit dem Original verglichen um dabei entstehende Artefakte zu entfernen.

Nach der Glättung des Tiefenbildes wird dieses in eine Punktwolke umgewandelt. Hierfür wurde ebenfalls das Microsoft Kinect SDK verwendet das alle benötigten Methoden bereitstellt.

Nach der Umwandlung werden noch weiter Punkte verworfen. Zunächst werden alle Punkte zu denen keine Farbe zugeordnet werden kann verworfen. Auch alle Punkte die zu nah oder zu weit vom Sensor entfernt sind, werden nicht weiter betrachtet. Je weiter das Objekt entfernt, desto ungenauer werden die Aufnahmen. Im folgenden wurde ein Mindestabstand von 30cm und ein Maximalabstand von 90cm verwendet.

Als letztes filtern wir alle Flächen die nicht Senkrecht zur Kamera sind (siehe Abb 3 b) Diese Flächen entstehen durch die Umwandlung von einem 2D Tiefenbild in einer 3D Punktwolke. Die benötigten Informationen fehlen an dieser Stelle und Punkte werden auf die Fläche zwischen Oberflächenobjekt und Hintergrund gesetzt. Diese Ebene stimmt nicht mit der wirklichen Oberfläche überein und müssen entfernt werden. Hierfür wird die Oberflächennormale verwendet. Die Normale wird aus dem Tiefenbild geschätzt.

$$\begin{aligned} dzX Axis &= depthAt[x + 1, y] - depthAt[x - 1, y] \\ dzY Axis &= depthAt[x, y + 1] - depthAt[x, y - 1] \\ Normale &= Normalize(-dzX Axis, -dzY Axis, 1.0) \end{aligned} \quad (3.1)$$

Mit dem Skalarprodukt lässt sich der Winkel zwischen dem Kameravektor (0,0,1) und Normale ausrechnen. Ein maximaler Winkel von 65° hat in den Tests ein gutes Ergebnis geliefert. **(Quellen auf Kinect und Lighthouse)**

ToDo

3.2. Zusammenfügen von Frames

Ein wichtiger Teil beim dem Aufnehmen der Punktwolke ist das zusammenführen von mehreren Frames. Hierfür wurde die Kinect mit dem Lighthouse Tracking System verbunden und verzichtet damit auf aufwändige Berechnungen.

ToDo

(Foto Halterung) Im lokalen Koordinatensystem der Kinect, also jedes Frames liegt der Ursprung in dem Tiefensensor. die Transformation *transformControllerToKinect* zwischen dem Koordinatensystem des Controllers und der Kinect wurde bestimmt und



Abbildung 3.2.: Effekt einer falschen Kalibrierung dx auf die endgültige Punktwolke. Aufnahme 1 und 2 sind von lokalen Koordinaten in Welt Koordinaten transformiert

die globale Transformation des Controllers *transformController* ist in der OpenVR API abfragbar. Die Transformation der lokalen Punktwolke in ein globale ist mit diesen beiden Transformationen möglich.

$$globalPosition = transformController * transformControllerToKinect * localPosition \quad (3.2)$$

3.2.1. Kalibrierung Kinect zu Vive

Eine wichtige Transformation ist die zwischen dem Koordinatensystem der Kinect und dem des Vive Controllers.

Ist zum Beispiel die Transformation entlang der X Achse der Kinect verschoben so verstärkt sich der Fehler wenn man das Objekt von der Anderen Seite, also um 180° dreht aufnimmt (siehe Abb, 3.2.1). Der Fehler im Lokalen Koordinatensystem wird in das Globale transformiert und ist in dem Fall dann in genau entgegengesetzte Richtungen

Bei der Kinect ist der offiziellen Doku entnehmbar dass der Ursprung von Punktwolken in dem Tiefensensor liegt (siehe [Kina]). Aber es gibt keine offizielle Dokumentation wo dieser exakt liegt. Im Bild 3.2.1 aus dem chinesischen Microsoft Forum ist eine von Benutzern vermessene schematische Darstellung der Kinect abgebildet. Der Tiefensensor liegt hinter der kleineren runden Öffnung, aber Fertigungsungenauigkeiten lassen keine exakten Daten finden. Für die Implementation wurde mittig hinter der Öffnung angenommen. Eine digitale Kalibrierung gestaltet sich schwierig, da das Lighthouse Tracking für den Controller zusätzlich einige Ungenauigkeiten mit sich bringt. Der Ursprung des Controllers lässt sich aus den Modellen von SteamVR auslesen. Dieser liegt geschickt für VR Anwendungen ist aber für das Tracking von Objekten ungeschickt. bei der Implementation stand noch keine Vive Tracker zur Verfügung. Für die Arbeit wurde der Controller so nah wie möglich an dem Teifensensor, also direkt darüber angebracht (siehe Abb.3.2.1). **(Bilder)**

ToDo

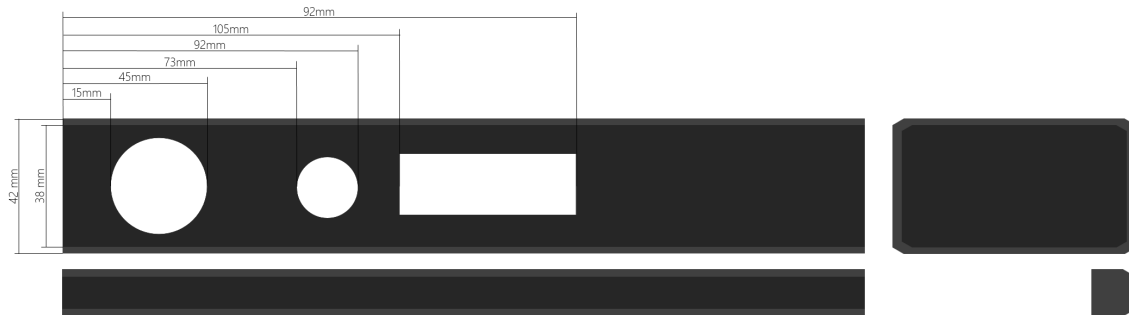


Abbildung 3.3.: Abmessungen der Kienct. Der Tiefensensor liegt in der kleinen runden Öffnung. Quelle:[Kinb]

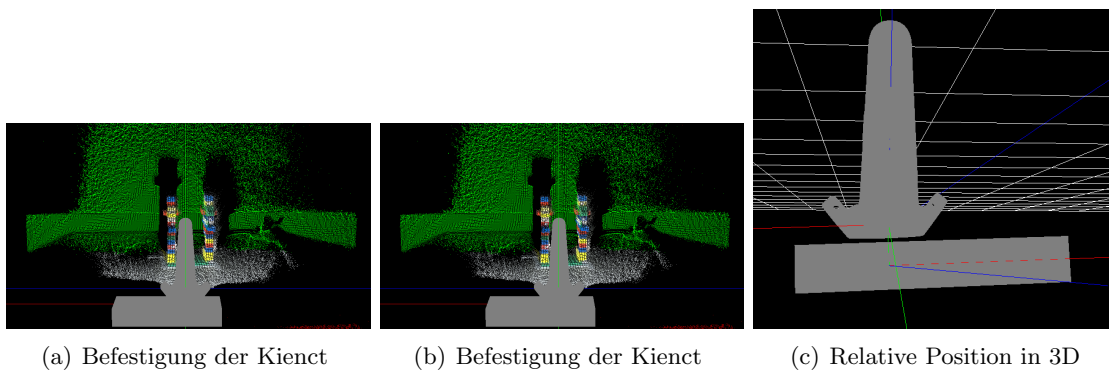


Abbildung 3.4.: Befestigung des Controllers an der Kinect. Die Mitte des Controllers ist direkt über dem Tiefensensor. Die Koordinatenkreuze zeigen den jeweiligen Ursprung des Geräts ([x,y,z] Achse=[rot,grün,blau])

3.3. Ergebnisse

Mit dem vorgestellten Verfahren lässt sich einfach und schnell eine Punktwolke erstellen. Jedoch gibt es Ungenauigkeiten in dem Vive Tracking und der Kalibrierung die die Punktwolke unbrauchbar aussehen lassen. **(Bild)** Zwischen 2 Aufnahmen und den daraus resultierenden Punktwolken ist ein Versatz bis zu 2-3 cm sichtbar. In einer 3D Umgebung insbesondere in VR ist das eine zu große Ungenauigkeit. Der Versatz zwischen den Punktwolken ist leider nicht konstant und ändert sich teilweise zwischen Durchläufen. Das Vive Tracking ist hierfür ein Grund. Vergleicht man mit einem 2m Zollstock die reale Distanz zu der in VR gemessenen dann wird daraus 1,98-2m virtuelle Distanz. Die Distanz ist hierbei abhängig von der Orientierung zu den Basistationen und der aktuellen Kalibrierung des Lighthous Tracking Systems. Dieses Problem erschwert es die Kalibrierung zwischen Vive und Kinect zu überprüfen die zusätzlich für einen Versatz der Punktwolke verstärken kann.

ToDo

Die Rotation der einzelnen Aufnahmen war kein Problem und hat keine sichtbaren Probleme produziert.

4. Speichern der Punktwolke mit 3D Tiles

In diesem Kapitel wird ein grober Überblick über die Struktur und die Komponenten des GL Transmission Formats und der 3D Tiles gegeben. Diese wurden verwendet um die Punktwolken zu speichern.

3D Tiles und gltf 3D Tiles are an open specification for streaming massive heterogeneous 3D geospatial datasets Tile Struktur (Tielset) different Tiles (batched, instanced, points ...) Bounding Volumes und LOD Dynamic loading GLTF Struktur Optimized for OpenGL and streaming scenes, nodes, meshes materials animations ignored

4.1. 3D Tiles

3D Tiles [3DT] ist eine neue offene Spezifikation für das streamen von massiven, heterogenen, geospatialen 3D Datensätzen. Die 3D Tiles können genutzt werden um Gelände, Gebäude, Bäume und Punktwolken zu streamen und beiden Features wie Level of Detail (LOD). Für die Arbeit wurde erwartet das insbesondere LOD notwendig werden könnte, es wurde aber nicht verwendet.

4.1.1. Tileset und Tiles

Als Basis der 3D Tiles wird JSON formatiertes Tileset verwendet das auf die eigentlichen Daten in Tiles verweist. Das Tileset hat eine baumartige Struktur aus Tiles und deren Metadaten. Jedes Tile hat hierbei ein 3D Volumen der den geografischen Bereich beschreibt, einen geometrischen Fehler zur Echtwelt. Außerdem können Kinder und deren Transformationen zu dem Elternteil angegeben werden. Alle Kinder liegen hierbei in dem Volumen des Elternknotens und können mit verschiedenen Datenstrukturen, wie K-D Bäumen Quadrees oder ähnlichem die Region genauer spezifizieren (siehe Bild 4.1.1. Hierbei können die Kinder das Elterntile ersetzen (replace, z.B. ein genaueres Mesh) oder das bestehende Tile ergänzen (refine, zusätzliche Gebäude oder Details). Die eigentlichen Daten der Tiles sind durch eine URL verlinkt und können dynamisch nachgeladen werden.

Tiles können in unterschiedlichen Formaten sein zum Beispiel:

Batched3D Model 3D Daten die als glTF übertragen werden. Zusätzlich können pro Modell Metadaten für das Visualisieren enthalten sein.

Instanced3D Model Tileformat für Instancing. Die Geometrie wird als glTF übertragen und zusätzlich eine Liste aus Positionen an denen die Objekte Instanziiert werden sollen. Kann zum Beispiel für Bäume genutzt werden.

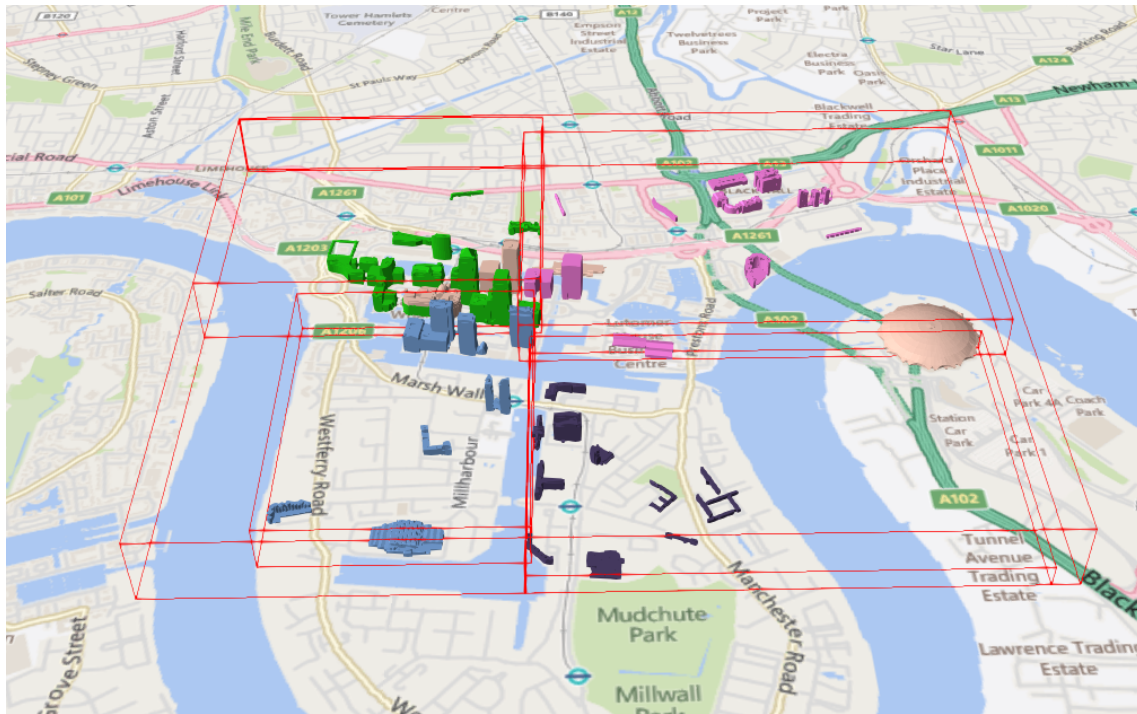


Abbildung 4.1.: Ein Tile mit 4 Kindern. Die 4 Kinder fügen die Gebäude hinzu und liegen im Volumen des Elterntiles. Als Datenstruktur liegt ein nicht uniformer Quadtree vor.

Point Cloud Format um Punktwolken zu übertragen. Das Teileformat enthält einen kleinen Header mit allgemeinen Metadaten. Danach folgt ein JSON String in dem steht welche Daten wie in dem Binärteil vorliegen. Außerdem ist enthalten welche und wie die Daten wie Position und Farbe dabei sind. Die eigentlichen Daten werden als Binärdaten übertragen und können so ohne Parsen direkt in den Speicher und Grafikspeicher geladen werden.

Composite Tileformat zum gleichzeitigen Übertragen mehrerer einzelner Tileformate in einem. Es lässt sich zum Beispiel ein Batched3D Modell für Gebäude mit Instanced3D Modell für Bäume verbinden und als ein Tile übertragen.

4.1.2. glTF

Das GL Transmission Format (glTF [GLT]) ist ein Format zum effizienten Übertragen von 3D Szenen für GL Api's wie WebGL, OpenGL ES und OpenGL wird hier nur Kurz erwähnt da es für die Implementierung nicht verwendet wurde. glTF dient als effizientes, einheitliches und erweiterbares Format zur Übertragung und Laden von 3D Daten. Im Vergleich zu aktuellen Standards wie COLADA ist glTF optimiert, schnell übertragen und kann schnell in eine Applikation geladen werden. In einer JSON formatierten Datei (.gltf) wird eine komplette Szene samt Szenegraf, Materialien und deren zugehörigen Shadern, Kamerapositionen, Animationen und Skinning Informationen übertragen. Dabei kann auf externe Dateien verwiesen werden. Diese sind zum Beispiel Binärdaten oder Bildern die für das einfache und effiziente Übertragen von Geometrie, Texturen oder den nötigen GLSL Shadern genutzt werden

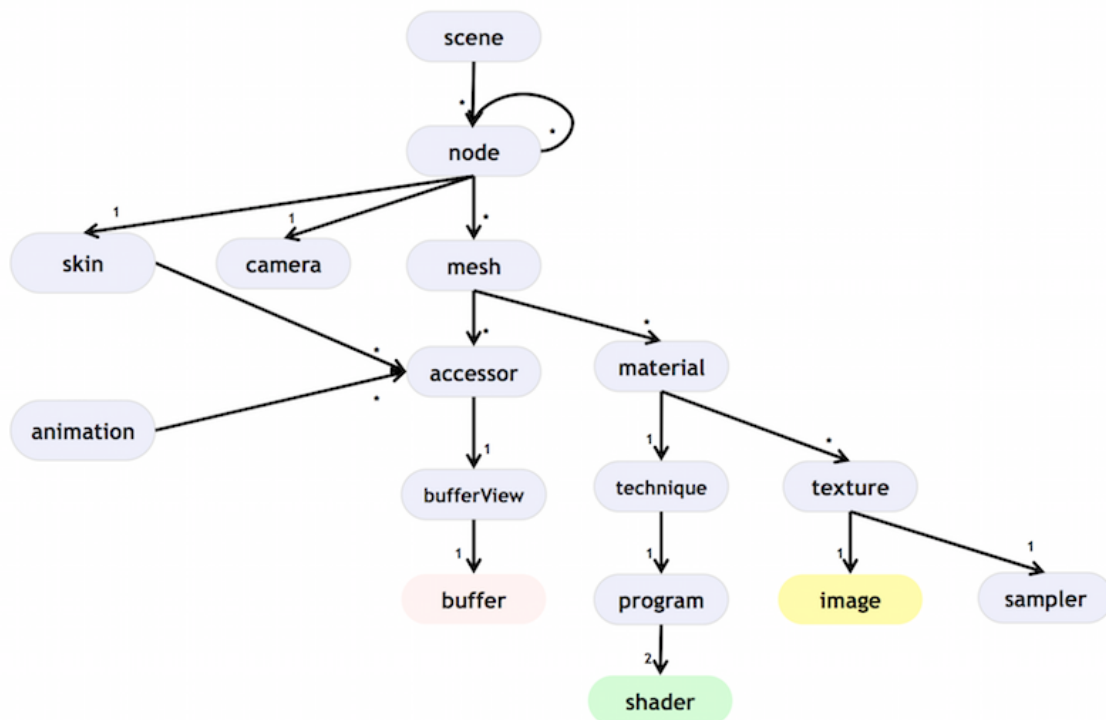


Abbildung 4.2.: Struktur einer glTF Szene.

4.1.3. Struktur

Die .glTF Datei ist eine JSON formatiert und bildet den Kern jedes Modells. In Ihr werden alle grundlegenden Informationen wie zum Beispiel die Baumstruktur des Szenengrafen und die Materialien gespeichert (siehe Abb. 4.1.3). Eine Szene bildet hierbei den Startpunkt für die zu rendernde Geometrie. Szenen bestehen aus Knoten (Nodes), die beliebig viele Knoten als Kinder haben können. Jeder Knoten kann eine Transformation im lokalen Raum definieren, bestehend aus eine Translation, einer Rotation und eine Skalierung. Jeder Knoten kann eine Mesh und damit die eigentliche Geometrie referenzieren. Diese Geometrie wird in Buffern als Binärdaten gespeichert. Diese sind entweder als Base64 String direkt im JSON oder als zusätzliche Binärdatei gespeichert. Auf einen Buffer wird mit einem Asccessor und einer Bufferview zugegriffen. In diesen ist spezifiziert in welchem Format die Daten vorliegen (z.B. ein Array aus 2D Vektoren (VEC2) aus UNSIGNED SHORT). Alle Datenformate entsprechen Formaten die in OpenGL vorliegen, sodass die Daten ohne Konvertierung in OpenGL Vertex Array Objekts (VAO), bzw. Vertex Buffer Objekts (VBO) umgewandelt werden können.

Jedes Mesh kann auf ein Material referenzieren. Materialien bestehen aus einem Materialparametern, Texturen und einer Techniken. Techniken bestehen hauptsächlich aus einem GLSL Shader Programm das ebenfalls im glTF mitgeliefert wird. Außerdem wird spezifiziert wie die VAO und VBO aus dem Mesh bei dem Rendervorgang an den Shader gebunden werden müssen.

Ein weiteres Feature von glTF Dateien ist die Möglichkeit Animationen und Skinning Informationen zu übertragen.

4.1.3.1. Buffers and Accessors

Buffer sind die eigentlichen Daten in einem Binären Block. Diese können entweder als externe Datei (.bin) oder als BASE64 encodierter String in der JSON Datei angefügt werden. Die Hauptaufgabe der Buffer ist es große mengen an Daten wie die Geometrie effizient zu übertragen.

4.2. Umsetzung

Für das Speichern der Punktwolke wurden keine LOD verfahren angewendet. In der Praxis hat sich gezeigt das die Wolken klein genug sind, sodass sie als ganzes effizient gerendert werden konnten. Sollte man größere Punktwolken, z.B. von einem ganzen Raum erstellen könnte das Performancevorteile beim visualisieren bringen. Das verwendete Tileset ist statisch und sehr einfach gehalten (siehe Anhang A) Es beinhaltet ein Tile das auf die Punktwolke referenziert. Es ist nicht transformiert und hat ein statisches Boundigvolume eine 5m große Kugel.

Die eigentlichen Daten werden in einem Point Cloud Tile abgespeichert. Die Positionsdaten der einzelnen Punkte wird als Array aus float abgespeichert. Dabei bilden 3 floats immer die x,y,und z Koordinaten eines Punktes. Zusätzlich speichern wir einen Array an Farbdaten. Pro Punkt wird jeweils ein Byte pro RGB gespeichert.

Um das Kalibriern zwischen der Echtwelt zu vereinfachen wurde beim aufnehmen ein Vive Tracker in der Welt platziert und als Ursprung verwendet. Alle Punkte wurden vor dem schreiben der Datei in das lokale Koordinatensystem des Trackers transformiert und können beim visualisieren erneut an dem Tracker orientiert werden.

$$exportPosition = TrackerPosition^{-1} * globalPosition \quad (4.1)$$

5. Visualisierung

In diesem Kapitel wird die Visualisierung der Punktwolke in der Unreal Engine 4 [UE4] erklärt. Die Unreal Engine ist eine mächtige Game Engine, die unter anderem Support für verschiedene Virtual Reality Systeme bietet. Durch die Verwendung einer Game Engine muss kein eigener performanter Renderer geschrieben werden, der den Anforderungen für Virtual Reality entspricht.

5.1. UE4 Rendering System

Die Unreal Engine verwendet ein Rendering System, das auf DirectX aufgebaut ist. Die gesamte Rendering ist abstrahiert und aus der Engine hat man keinen direkten Zugriff auf die Grafikkarte und die Shader. Das Unreal Engine Materialsystem ist der vorgesehene Weg, um Shader zu implementieren. Im Stil der UE4 Blueprints lassen sich damit grafische Materialien definieren, die von der Engine in zugehörige Shader umgewandelt werden.

5.2. 3D Tiles Laden und vorbereiten

Die Punktwolken können als Array direkt geladen werden, jedoch lässt die Unreal Engine nicht zu, diese auch direkt auf die Grafikkarte zu laden und zu verwenden. Als Alternative wurde das eindimensionale Buffer in eine quadratische 2D Textur umgewandelt. Jeder Pixel der Textur besteht aus 3 Werten, jeweils einen für die 3 Farbkanäle Rot, Grün und Blau. Ein Punkt im 3D Raum besteht ebenfalls aus 3 Werten für die Achsen X, Y und Z. Um die Daten auf die Grafikkarte zu laden, codieren wir die Positionen in den Farbkanälen. Farbwerte sind im Wertebereich $[0 - 1]$ und werden in unterschiedlichen Datenformaten gespeichert. Für die Implementierung wurde das HDR Datenformat der Unreal Engine verwendet. Dabei wird jeder Farbkanal in einer 16 Bit Integer codiert und ergibt damit pro Farbkanal 2^{16} diskrete Farbwerte. Das bedeutet auch für eine Punktwolke, die wir als Farbtexture codieren, haben wir eine maximale Auflösung pro Achse.

Um die Daten in eine Textur zu überführen, müssen wir diese vorbereiten. Zunächst wird das Minimum und die Größe entlang jeder Achse bestimmt. Mit diesen Daten lässt sich

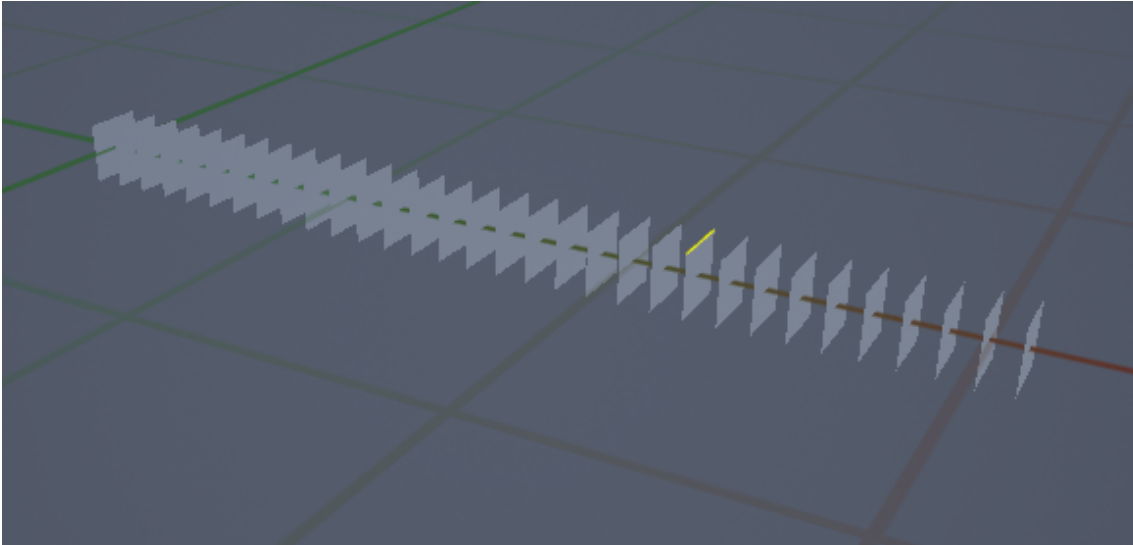


Abbildung 5.1.: Ein Quadchain mit 32 Quads

die gesamte Punktwolke in den Einheitswürfel transformieren.

$$\begin{aligned}
 \forall p \in \text{Punktwolke} \\
 \min &= \text{Minimum}(p) \\
 \max &= \text{Maximum}(p) \\
 \text{size} &= \max - \min \\
 p_{\text{Einheitswürfel}} &= \frac{p - \min}{\text{size}}
 \end{aligned} \tag{5.1}$$

Die so codierten Punkte lassen sich in einer Textur abspeichern und im Shader wird diese Transformation umgekehrt.

Das Aufbereiten der Farbtextur ist einfach. Die Farben sind im 3D Tile als Byte große Integer gespeichert und können direkt in eine Unreal Textur konvertiert werden.

Die erzeugten Texturen sind immer quadratisch aber die Anzahl Punkte ist nicht vorgegeben. Für die Implementierung wurde immer die Quadratische Textur bestimmt die genug Platz für alle Punkte bietet und die nicht verwendeten Punkte haben in der Farbtextur einen α Wert (4. Farbkanal, Transparenz) von 0 sonst 1.

5.3. Rendering

In der Unreal Engine ist das Rendern von Punktwolken nicht vorgesehen und deshalb gibt es keinen Punkt Modus wie in OpenGL. Als Basis für den Rederingprozess wurde deshalb eine Quadchain verwendet (Siehe Abbildung ??) Diese Quadchain besteht aus 2^{20} einzelnen Quadraten entlang der Z Achse. Jedes Quad ist dabei an einer Ganzzahligen Z Position und die Ecken sind bei 1 und -1.

Beim eigentlichen Rendervorgang kann im Shader, also in UE4 Material jedes Quad an eine Position eines Punktes geschoben werden. Hierbei wird die X-Position des Quads als Index in die Punktwolke verwendet. Aus der Größe der quadratischen Textur und dem Index können die Texturkoordinaten (uv) in der 2D Textur berechnet werden. **(Formel?)**

ToDo

Mit den Texturkoordinaten kann die Position und die Farbe des Punktes ausgelesen werden. bevor die Position verwendet werden kann muss diese aus dem Einheitswürfel in die wirklichen Koordinaten zurück transformiert werden. Das Minimum und die Größe aus Gleichung [?] der Vorbereitung lässt sich das einfach zurückrechnen. Anschließend fehlt nur noch die Translation und Rotation der Punktwolke in der Welt. Auch hier lässt die Unreal Engine nicht zu das direkt eine Matrix als Parameter für das Material übergeben werden kann, aber durch eine Position und eine Rotation lässt sich die Transformation im Material ausrechnen. Die berechnete Endposition wird als WorldPositionOffset an die Grafikkarte weitergegeben, die damit den Punkt an die richtige Weltposition setzt.

Für die Sichtbarkeit wird der Quad zu dem Betrachter gedreht und in der Größe skaliert. Zusätzlich wird das Quad noch durch eine runde Maske zu einem kleinen Kreis umgewandelt und mit der entsprechenden Farbe texturiert.

Die Quadchain hat eine feste Länge und die Punktwolke hat eine veränderbare Größe. Deshalb werden größere Punktwolken in 2 aufgeteilt. Bei kleineren Punktwolken werden die nicht verwendeten Quadrate verworfen.

5.4. Ergebnisse

Die resultierenden Punktwolken sehen gut aus und die in dieser Arbeit verwendeten relativ kleinen Punktwolken sind in VR visualisierbar. Außer das Deaktivieren von Schattenberechnungen wurden keine Performance Optimierungen vorgenommen. Die Beispielwerte wurden mit einem i7 6700 und eine GTX 1070 aufgenommen, eine genaue Analyse ist nicht erfolgt. Eine Punktwolke mit 1596685 einzelnen Punkten erreicht ca 60fps (siehe Bild 5.4. Hierbei ist zu bedenken das die ganze Quadchain von 2^{20} Punkten gerendert wird und überflüssige Geometrie erst im Shader/Material verworfen wird. Bei 4 einzelnen Punktwolken mit 443175 Punkten (110369, 115991, 110377, 106438; 4 Instanzen der Quadchain) erreicht dieses vorgehen ca 44 fps. Für eine Große Punktwolke mit 1797690 Punkten gesplittet in 2 Instanzen sind es ebenfalls 44 fps.



(a) Gesamte Punktwolke



(b) Nahaufnahme

Abbildung 5.2.: Punktwolke mit 1596685 Punkten. Die Textur im Hintergrund ist die zugehörige Positionstextur, Rechts oben ist FPS und die Renderzeit in ms zu sehen

6. Hololens

Ziel Daten aus der VR Umgebung Visualisieren Zeitgründen nur einen MA controller befestigten laserstrahl

6.1. Unreal Engine Un Hololens

Microsoft UWP Repository Development Branch HAt nicht Compiled Videos zeigen lag rotation

6.2. Hololens Implementierung

Welt Anker verschiebbar im Debug Modus Eine Achse Senkrecht zum Boden empfängt JSON per UDP zeichnet Line relativ zu Welt anker

6.3. Kalibrierung

Vive Tracker an Position und rotation des Welt anker speichern auf Knopfdruck der Transformation des Tracker Origins

Senden über UDP inverse Transformation origin

6.3.1. Kalibrierungsfehler

Ungenauigkeiten Vive Ungenauigkeiten HoloLens Unterscheide in Längen Faktor 1,025 vlt Bilder / video mehrfaches knopf drücken ungenauigkeiten

6.4. Ergebnisse

Bilder aus HoloLens perspektive Funktioniert gut wenn nach am gemeinsamen Punkt

6.5. Verbesserungen

hier oder Ausblick Marker auf tracker kleben (kein menschlicher Fehler) Mehr Tracker und damit gesyncte Längen

6.5.1. Verschieben der VR Umgebung

vlt besser im Szenario der Evaluation

7. Evaluation

7.1. Versuchsablauf

7.2. Probanden

Statistik Etwas erfahrung mIt VR AR erfahrung quasi ncih vorhanden Pokemaon Go Rot grün schwäche (genauen Namen erfragen) ha keine Probleme gemacht, Farbbend der Steine war kräftig genug

(allgemin Boxplot beschreiben)

ToDo

7.3. Ergebnisse

7.3.1. Stein Finden

7.3.1.1. Bilder

7.3.1.2. VR

Punktwolken teilweise usncharf (feedback merherer Probanden nciht ersichtlicch aus zeiten siehe Grafik

7.3.2. Nutzung der vorhandenen Hilfsmittel

7.3.3. Kommunikation

GRafik

7.3.3.1. Videostream

teilweise IGnoriert weil verwirrend nur am anfang um sich auf Bezeichnen zu einigen links rechts merkmale wie der schwarze turm

teilweise asl feedback genutz auf Turm Zeigen “derda?”

7.3.3.2. VR

Viel genutzt teilweise kontne der Versatz im Kopf ausgeglichen werden (Neben dem Turm) guter orientierungsanfang wenn trackign ideal reicht der rote stein Wackelige hand +tracking Feedback wäre interessssanter ohne erlaubte kommunikation

Abbildung 7.1.: Erfahrung mit VR und AR. Median sind 1 und -1

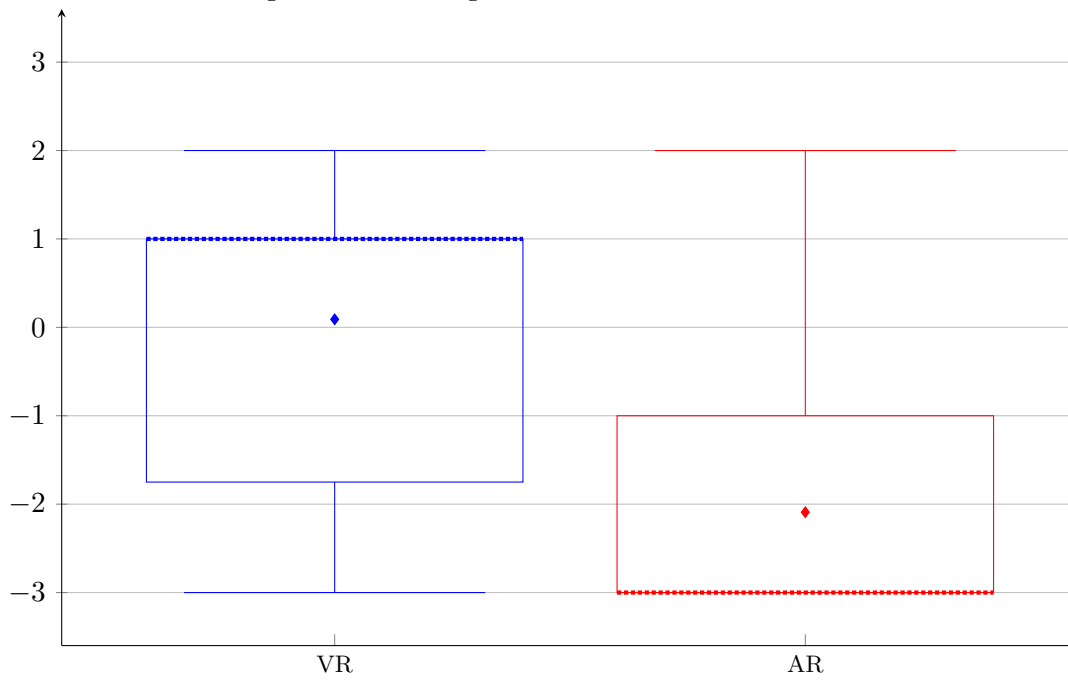


Abbildung 7.2.: Wie schwierig war die Zusammenarbeit.

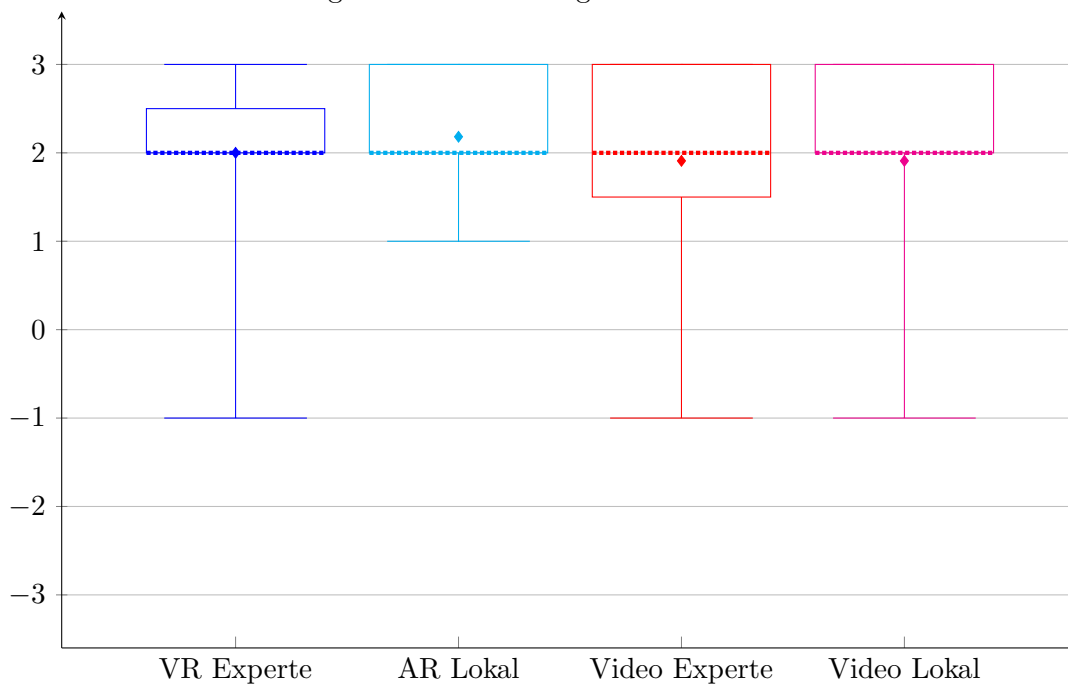


Abbildung 7.3.: Wie einfach war es den beschriebenen Stein zu finden?

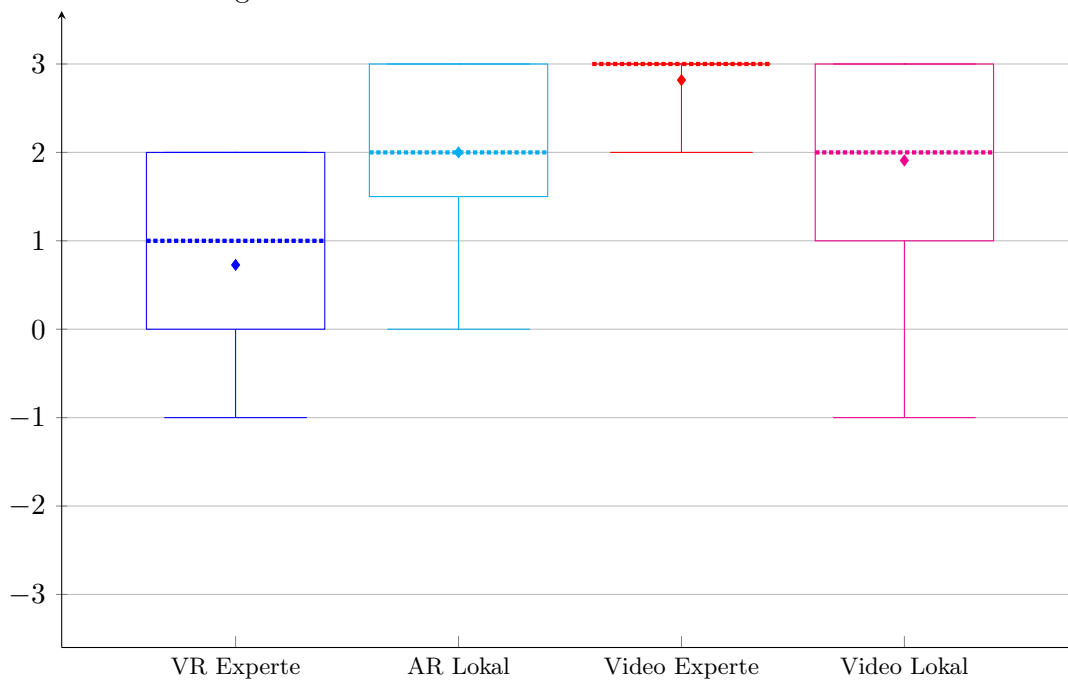


Abbildung 7.4.: Wie wichtig war die sprachliche Kommunikation?

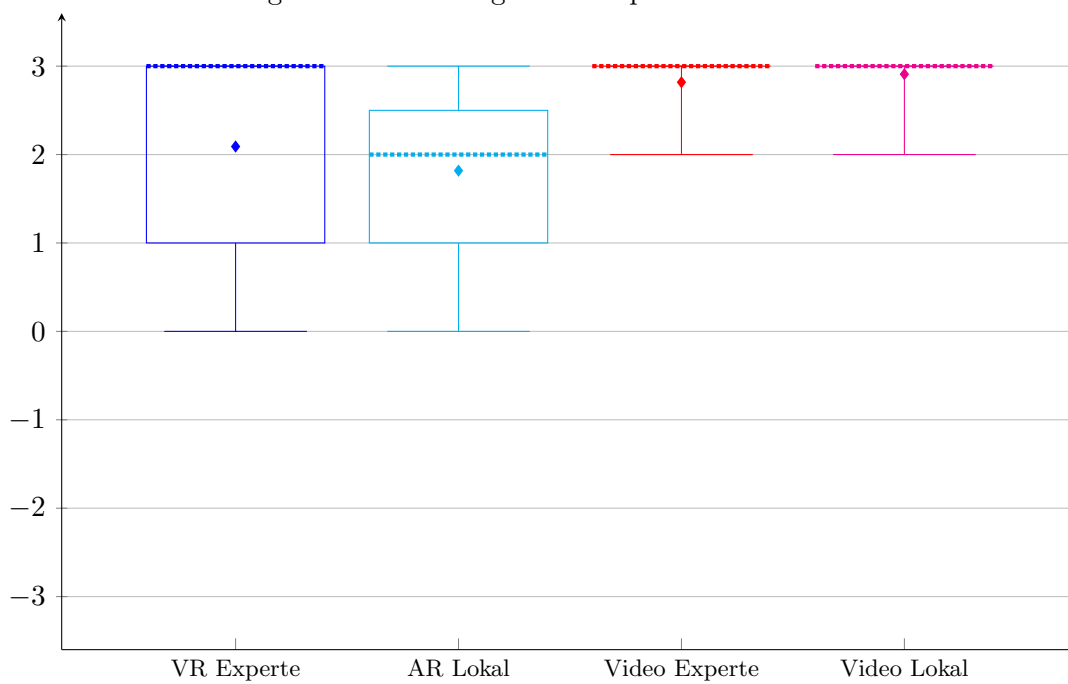
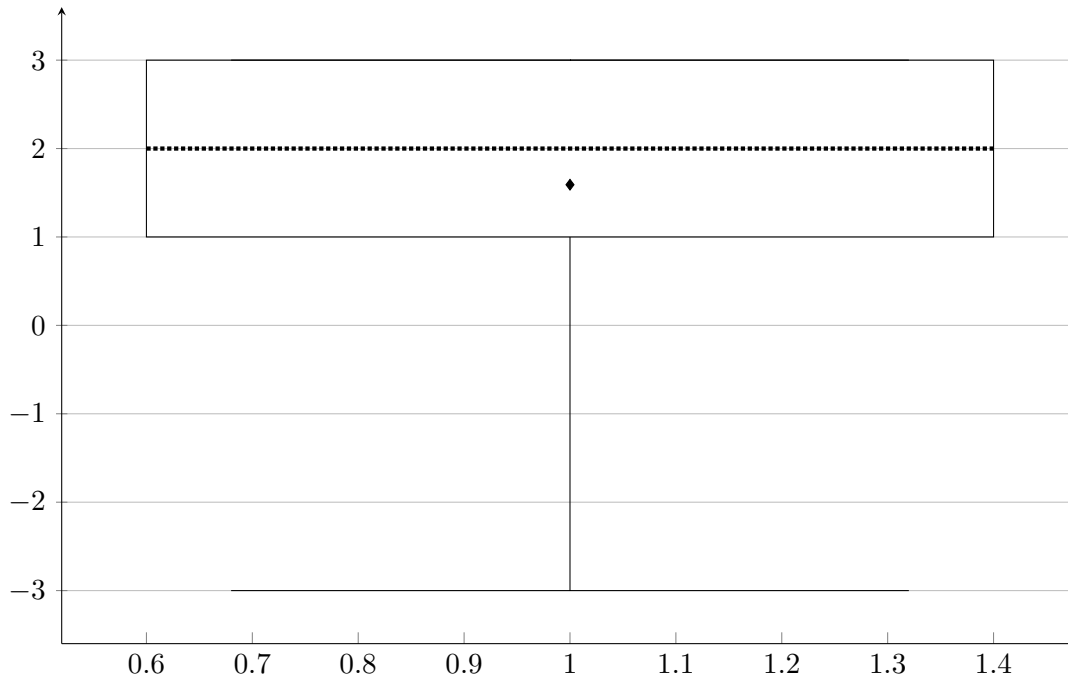


Abbildung 7.5.: War es im VR/AR Szenario von Vorteil von der Perspektive und Bewegung der anderen Person unabhängiger zu sein und nicht an die Ansicht aus des Videos gebunden zu sein?



7.3.4. Fehleranzahl

Satistik überwiegend fehlerfrei Allgemeine Fehler im Stein verutsch letzter stein der sequenzn, nicht der davor Sprachliche ungenauigkeitne der 2. schwarze stein vs der 2. stein der schwarze

FehlerquellenVR falsche kalibrierung ca 2 steine drüber-> stein mit gelicher Farbe Fehlerquelle Video links rechts schwäche verstärkt durch gegebenfalls ander perspektive

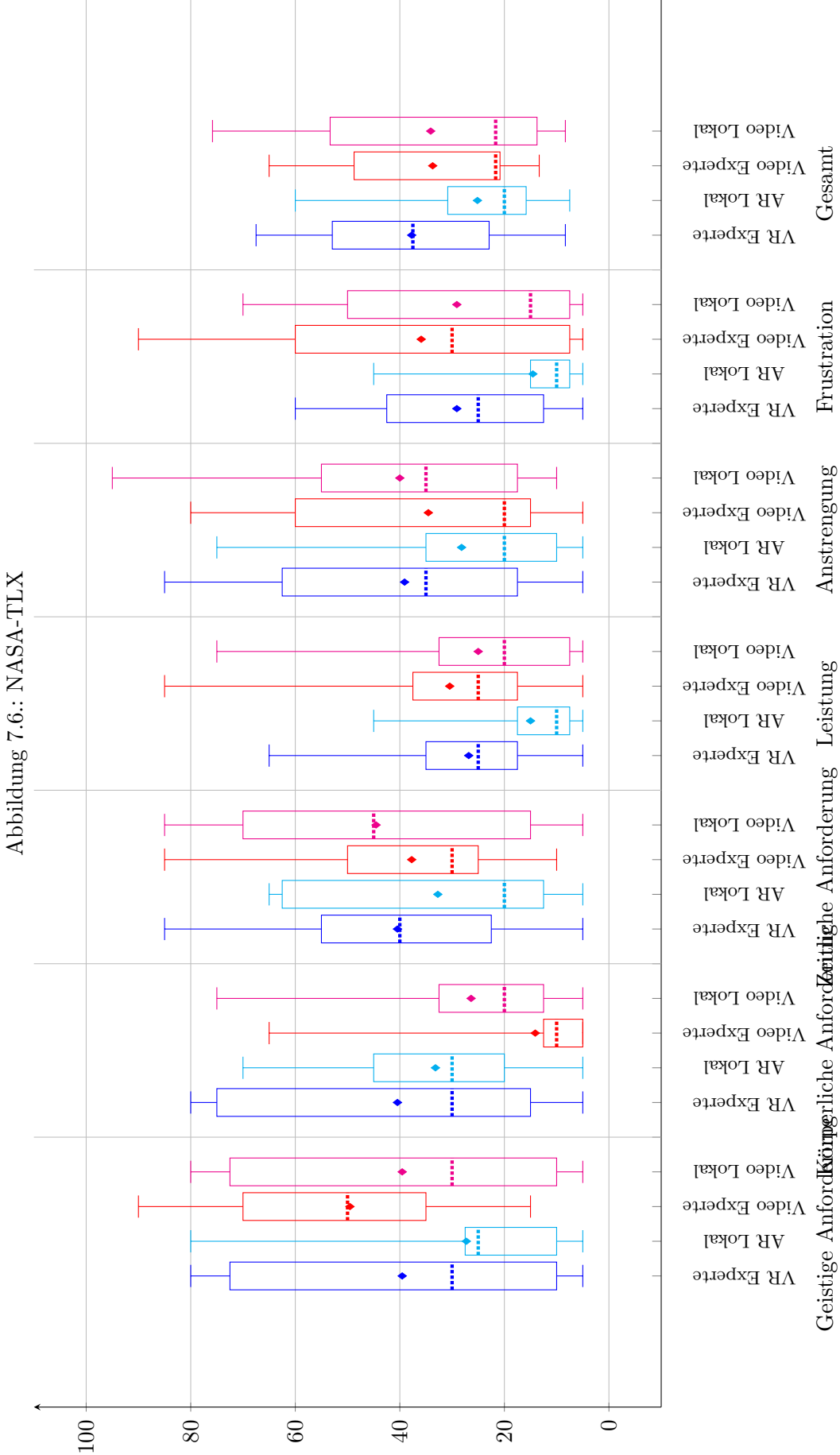
7.3.5. Timings

Gruppen signifikaten schnelle in vr: 2 video:2 Durschnitt overall gleich Person Selber eien zu große beeinflussung erkenne der Farbsequenz schwankt stark zwischen den personen Beim Ablesne positionierung und finden des Labels

7.3.6. Unabhängigkeit

Mießt Positiv die kamera zeigt nicht dahin wo es gebraucht wird, wacklet, perspektive anders als in den Bidlern Aber einzelne sehr schlechte bweertungen da es zu fehlern führt kein gewohntes links rechts mehr zeigen quer durch den Turm -> erste aufzeigepunkt gemeint, steht auf der andern Seite und sieht den austritpunkt todo nachscuane ob hololens da mit raum mesh schneidet

7.3.7. NASA TLX und UEQ



TLX: Experte median schwieriger Video experte geringste körperliche anstrengung UEQ
Atraktivität Stimulation Originalität in VR besser durchschaubarkeit, effizienz, steuerbarkeit
kein Statistisch signifikanter unterschied trends???

7.3.8. Allgemeines

Hololens kliense FOV schwer schlechte Auflösung

8. Ausblick

Zukunftstechnologie aber aktuelle Consumer Hardware noch zu schlecht Verbesserung des Lighthouse Trackings Kienct mit integrierten Vive tracking

Hololens ist eigtlch awesome, aber zu schwer zu kleines FOV

generiertes Mehs besser als Punktwolke+

A. 3D Tile JSON

In diesem Anhang finden Sie das verwendete Tileset der 3D Tiles

```
1 {
2   "asset": {
3     "version": "0.0"
4   },
5   "geometricError": 0,
6   "root": {
7     "boundingVolume": {
8       "sphere": [
9         0,
10        0,
11        0,
12        5
13      ]
14    },
15    "geometricError": 0,
16    "refine": "add",
17    "children": [
18      {
19        "transform": [
20          1.0, 0.0, 0.0, 0.0,
21          0.0, 1.0, 0.0, 0.0,
22          0.0, 0.0, 1.0, 0.0,
23          0.0, 0.0, 0.0, 1.0
24        ],
25        "boundingVolume": {
26          "sphere": [
27            0,
28            0,
29            0,
30            5
31          ]
32        },
33        "geometricError": 0,
34        "content": {
35          "url": "example.pnts"
36        }
37      }
38    ]
39  }
40 }
```

B. Evaluations Fragebogen

Anfangsfragebogen

ID:

Datum/Uhrzeit:

Allgemeine Angaben:

Geschlecht	<input type="radio"/> männlich	<input type="radio"/> weiblich		
Alter	<input type="radio"/> <20	<input type="radio"/> 20-30	<input type="radio"/> 30-40	<input type="radio"/> >40
Brillenträger	<input type="radio"/> Ja	<input type="radio"/> Nein		
Rot-Grün-Sehschwäche	<input type="radio"/> Ja	<input type="radio"/> Nein		
Farbenblind	<input type="radio"/> Ja	<input type="radio"/> Nein		

Erfahrung mit Virtual Reality (VR)

Haben Sie bereits Erfahrungen mit Virtual Reality gesammelt?

Ich habe ...	--- -- - -/+ + ++ +++	Ich habe ...
noch keine Erfahrungen gesammelt	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	sehr viele Erfahrungen gesammelt

Falls Sie schon Erfahrungen mit VR gesammelt haben:

Welche Hardware wurde von Ihnen genutzt?

Erfahrung mit Augmented Reality (VR)

Haben Sie bereits Erfahrungen mit Augmented Reality gesammelt?

Ich habe ...	--- -- - -/+ + ++ +++	Ich habe ...
noch keine Erfahrungen gesammelt	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	sehr viele Erfahrungen gesammelt

Falls Sie schon Erfahrungen mit AR gesammelt haben:

Welche Hardware wurde von Ihnen genutzt?

Abbildung B.1.: Der Anfangsfragebogen

ID:

Datum/Uhrzeit:

Scenario	<input type="radio"/> VR/AR	<input type="radio"/> Video Stream
Rolle	<input type="radio"/> Experte	<input type="radio"/> Techniker

Wie einfach hat die Zusammenarbeit in dem Scenario geklappt

Die Zusammenarbeit war ...	---	--	-	-/+	+	++	+++	Die Zusammenarbeit war ...
sehr schwierig	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	sehr einfach

Wie einfach war es den beschriebenen Stein zu finden

Das Finden war ...	---	--	-	-/+	+	++	+++	Das Finden war ...
sehr schwierig	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	sehr einfach

Wie sehr waren sie von der sprachlichen Kommunikation abhängig

Die Kommunikation war ...	---	--	-	-/+	+	++	+++	Die Kommunikation war ...
unwichtig	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	sehr wichtig

Freie Kommentare

BITTE WENDEN

Abbildung B.2.: Der Fragebogen nach jedem Versuchsdurchlauf

Abschlussfragebogen

War es im VR/AR Szenario von Vorteil von der Perspektive und Bewegung der anderen Person unabhängiger zu sein?

Die Unabhängigkeit war ...	- - - - -	-	-/+	+	++	+++	Die Unabhängigkeit war ...
Nicht von Vorteil	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Sehr vorteilhaft

Freie Kommentare

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/>

Abbildung B.3.: Der AbschlussFragebogen

Literaturverzeichnis

- [3DT] *3d tiles spezifikation.* <https://github.com/AnalyticalGraphicsInc/3d-tiles>. Accessed: 2017-09-13.
- [GLT] *Gltf spezifikation.* <https://github.com/KhronosGroup/glTF>. Accessed: 2017-09-13.
- [Kina] *Kinect dokumentation koordinatensysteme.* <https://msdn.microsoft.com/de-de/library/dn785530.aspx>. Accessed: 2017-11-02.
- [Kinb] *Kinect tiefensensor position.* <https://social.msdn.microsoft.com/Forums/sqlserver/ja-JP/05a6d2b3-9096-4236-b77a-691c5f047066/kinect-for-windows-v2-?forum=windowsgeneraldevelopmentissuesja>. Accessed: 2017-11-02.
- [lig] *Lighthouse tracking examined.* <http://doc-ok.org/?p=1478>. Accessed: 2017-11-02.
- [MCS14] Manuel Martin, Florian van de Camp, and Rainer Stiefelhagen: *Real time head model creation and head pose estimation on consumer depth cameras.* In *Proceedings of the 2014 2Nd International Conference on 3D Vision - Volume 01*, 3DV '14, pages 641–648, Washington, DC, USA, 2014. IEEE Computer Society, ISBN 978-1-4799-7000-1. <http://dx.doi.org/10.1109/3DV.2014.54>.
- [NLL17] Diederick Christian Niehorster, Li Li, and Markus Lappe: *The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research.* In *i-Perception*, 2017.
- [OES⁺15] Ohan Oda, Carmine Elvezio, Mengü Sukan, Steven Feiner, and Barbara Tversky: *Virtual replicas for remote assistance in virtual and augmented reality.* In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 405–415, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-3779-3. <http://doi.acm.org/10.1145/2807442.2807497>.
- [UE4] *Unreal engine 4.* <https://www.unrealengine.com/en-US/blog>. Accessed: 2017-11-02.

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, xx.xx.xx

.....
(Max Mustermann)