Ein Hologramm für alle (Magie?)

Chris Papenfuß, data experts GmbH

Dieser Artikel soll die Leser ermutigen, sich mehr mit der Thematik Shared Augmented Reality (AR) zu beschäftigen. Er zeigt, warum Hologramme großartig sind, geteilte jedoch noch viel besser. Zusätzlich möchte ich zeigen, warum aus meiner Sicht die Gaming-App „Minecraft Earth" technisch deutlich interessanter ist als „Pokémon GO". Hinweis: Im Text ist sämtlicher Quellcode in C# geschrieben.

Folgt man der Beschreibung für Hologramme auf Wikipedia *[1]*, ist die Darstellung von 3D-Modellen in AR-Apps wohl nicht als Hologramm zu bezeichnen. Trotzdem werden in diesem Artikel all diese 3D-Modelle so bezeichnet.

AR-Anwendungen

Die Definitionen von „Augmented Reality" gehen sehr weit auseinander. Liest man beispielsweise im Cambridge Dictionary *[2]*, so geht es „nur" um das gemeinsame Anzeigen von computergenerierten Bildern mit einer Ansicht der realen Welt. Bei Wikipedia gehen

die Autoren bereits etwas weiter *[3]*. So sind die erzeugten Bilder immer im Zusammenhang beziehungsweise als Erweiterung mit den Bildern der realen Welt zu verstehen.

Für meiner Meinung nach gute AR-Anwendungen sollten diese erzeugten Bilder nicht einfach nur eine Erweiterung der Realität, sondern auch in derselben „verankert" sein. Eine einfache und klassische Variante für solch eine Verankerung ist die Verwendung von Markern.

Damit das in *Abbildung 1* gezeigte Hologramm immer relativ zu dem QR-Code dargestellt werden kann, muss die Anwendung in jedem Frame aus dem Videostream der realen Welt wissen, wo exakt sich der Code befindet und wie er ausgerichtet ist. Dieser Bereich *[4]* der Computer Vision ist sehr spannend, kann hier aber nicht weiter ausgeführt werden. Dazu sei erwähnt, dass diese Art der Verankerung nicht nur mit QR-Codes einwandfrei funktioniert, sondern auch mit (fast) beliebigen anderen Bildern, wie zum Beispiel Fotos.

Damit man Hologramme an einer „beliebigen" Position in der realen Welt anzeigen kann, muss das Gerät mit der AR-App zu jedem Zeitpunkt möglichst genau wissen, wo es sich in der realen Welt befindet. Damit ist erst einmal nicht der genaue Längen- und Breitengrad gemeint, sondern eher, wie es sich relativ zu der Position bewegt hat, an der die Anwendung gestartet wurde. Dafür wird jedes Bild aus dem Video-Stream der Kamera analysiert und Features identifiziert *[5]*. Features aus dem vorherigen Frame werden wiedererkannt und über Triangulation wird die Änderung der Position im Raum bestimmt. Dieses Verfahren wird als „SLAM" (Simultaneous Localization and Mapping) bezeichnet und ist in der Praxis deutlich komplexer. In einer Implementierung sollte man zusätzlich die Daten von anderen Sensoren wie IMUs (inertiale Messeinheiten) *[6]* verwenden. Microsoft hat für seine HoloLens sogar extra einen Chip entwickelt [6a], der die Bewegung aufgrund der gesammelten Daten voraussagt und so eine Prognose für die Bewegung „schätzt", bevor diese dann in einer Anwendung verwendet werden kann.

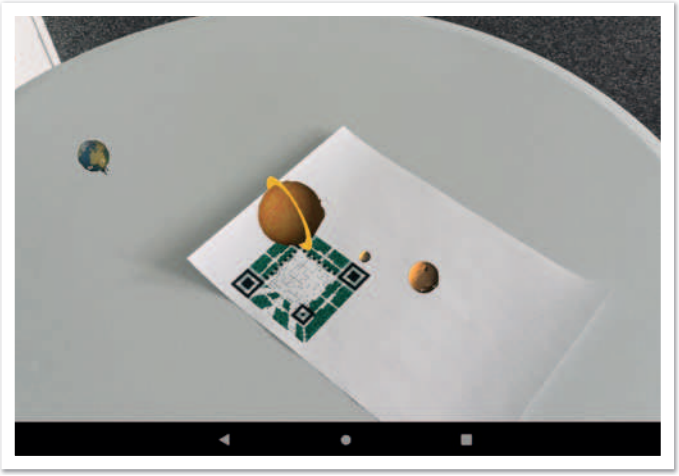

Abbildung 1: Hologramm vom Sonnensystem, verankert an QR-Code

Seit 2017 bietet Google die eigene Plattform ARCore *[7]*, die es ermöglicht, Hologramme mit entsprechendem Android-Smartphone beliebig im Raum zu platzieren. Auch Apple hat mit ARKit seit 2017 solch eine Plattform mit ähnlichem Feature-Set für entsprechende iPhones und iPads im Angebot *[8]*.

Mithilfe dieser Plattformen und der damit möglichen Verankerung verhält sich das Hologramm nun schon fast so, wie es sich für einen Anwender „natürlich" anfühlt. Das Hologramm wird an der gewünschten Position in der Realität dargestellt, ohne einen QR-Code oder anderen Bild-Marker zu verwenden *(siehe Abbildung 2)*. Das Hologramm verändert die Größe und Drehung, wenn sich der Anwender um den Anker herumbewegt *(siehe Abbildung 3 und 4)*.

Zum besseren Verständnis: Die „Verankerung" nennt sich bei Apples ARKit „ARAnchor" *[9]*, bei Googles ARCore heißt sie „Anchor" *[10]* und bei Microsofts HoloLens „World Anchor" *[11]*. In diesem Artikel wird der Begriff „Anchor" verwendet.

Abbildung 2: Hologramm einer Libelle, verankert auf einem Nachttisch

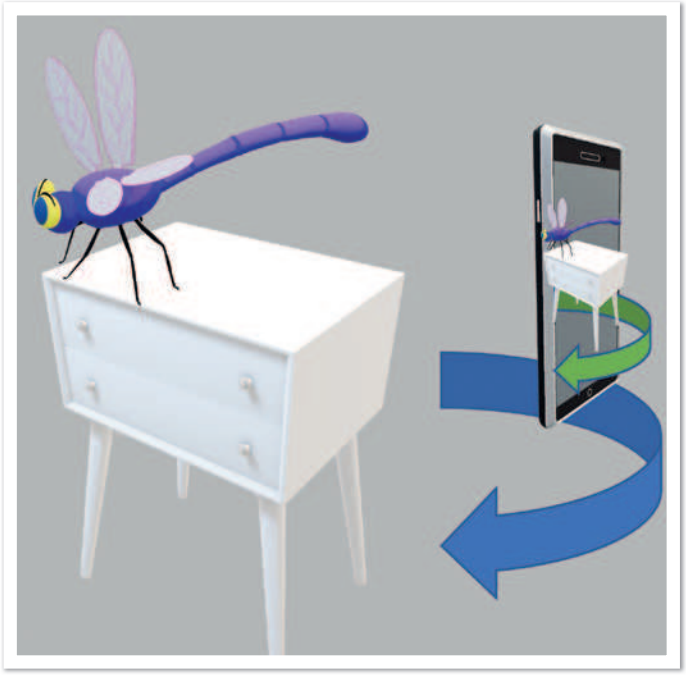

Abbildung 3: Verankertes Hologramm einer Libelle, verhalten bei Bewegung

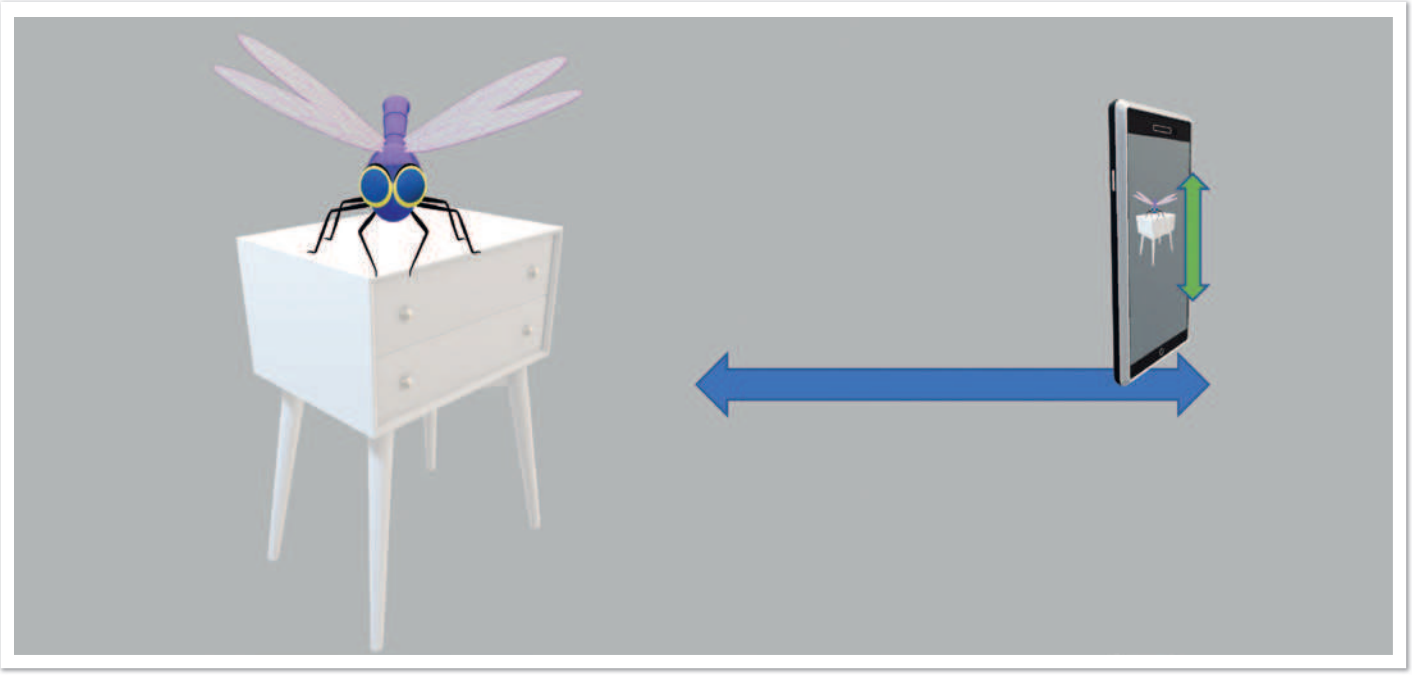

Abbildung 4: Verankertes Hologramm einer Libelle, verhalten bei Bewegung

„Multiplayer“

Damit die Interaktion mit den Hologrammen noch mehr „Spaß“ macht, sind neben der Verankerung noch ein bis zwei weitere Eigenschaften wünschenswert.

Zum einen möchte man seine Hologramme „teilen“. Dazu können entweder alle Beteiligten auf einen Smartphone-Bildschirm schauen oder noch besser: Jeder hat ein eigenes Gerät. Damit mehrere Geräte „dasselbe“ Hologramm sehen können, muss die entsprechende Verankerung gespeichert, übertragen und auf dem anderen Gerät wiedergefunden werden. Die Möglichkeit, diese Verankerung zu speichern und erneut zu laden, bieten alle erwähnten Plattformen. Will man diese allerdings übertragen und wiederfinden, gibt es aktuell die in *Tabelle 1* gezeigten Möglichkeiten/Services.

Aktuell (Beginn 2020) sind alle aufgeführten Services kostenlos, wobei Azure Spatial Anchors vermutlich in Zukunft kostenpflichtig sein wird *[15]*. Mit diesen Diensten kann man die Anchor von einem Gerät auf andere Geräte übertragen *(siehe Abbildung 5)*. Das ermöglicht mehreren Nutzern, das gleiche Hologramm an der gleichen Position in der Realität zu sehen *(siehe Abbildung 6)*.

Damit man nicht nur „statische“ Hologramme, also Statuen, sieht, benötigen die 3D-Modelle noch Interaktionsmöglichkeiten. Außerdem muss jede Interaktion möglichst synchron auf jedem Gerät ausgeführt werden. Das bedeutet, neben den geteilten Verankerungen benötigt man noch eine Verbindung zum Teilen von Interaktionen und Bewegungen. Auf der Suche nach der geeigneten IDE, um AR-Anwendungen zu entwickeln, landet man schnell bei Unity *[16]*. Diese IDE und 3D Engine hat ihre Wurzeln in der Entwicklung von Spielen. Somit ist es auch nicht weit hergeholt, das Teilen der Bewegung und Interaktionen als Multiplayer zu bezeichnen. Dementsprechend sind in diesem Bereich die Tools und Services zu finden, um unsere Anforderung zu erfüllen.

Aktuell verwende ich bei meinen Anwendungen für den Multiplayer-Anteil das SDK und den Service von Photon *[17]*. Die Integration in Unity ist gut gelungen und der Server kann je nach Anforderung

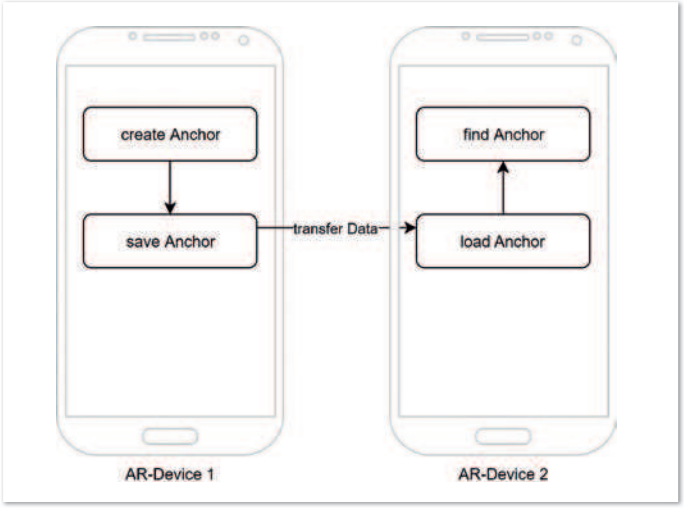

Abbildung 5: Ablauf Anchor-Sharing

	WorldAnchor [12]	Cloud Anchor [13]	Azure Spatial Anchor (ASA) [14]
iOS		x	x
Android		x	x
HoloLens	x		x

Tabelle 1

Abbildung 6: Anchor-Sharing in einer Anwendung mit der HoloLens

auch als Self-Hosted-Lösung verwendet werden. Die zusammengefasste Minimal-Architektur für eine Multiplayer-Anwendung ist in *Abbildung 7* dargestellt. Für die Verteilung des Anchors verwende ich Azure-Spatial-Anchors, da dort die meisten Plattformen bedient und zusätzliche Features, wie zum Beispiel Wayfinding, angeboten werden *[18]*.

„Simple" Anwendung

In diesem Kapitel erfahren Sie, wie alles zu einer funktionierenden Anwendung zusammengebaut wird.

Wie bereits erwähnt, setzte ich bei der Entwicklung auf Unity (Version 19.2.x) *[16]*. Nach der Erstellung eines neuen 3D-Projektes müssen das Azure-Spatial-Anchors SDK *[19]* und das Photon SDK *[20]* importiert werden. Ich verwende außerdem gerne das Open Source MixedRealityToolkit *[21]*, da viele Skripte, Designs und Tools für die Arbeit mit VR/AR-Anwendungen bereits enthalten sind. Um die Arbeit mit verschiedenen Zielplattformen (Android, iOS, HoloLens) weiter zu vereinfachen, ist es hilfreich, eine Abstraktionsebene auf AR-Core und AR-Kit aufzusetzen. Mit AR-Foundation *[22]* bietet Unity inzwischen genau solch eine Abstraktion an.

Zuerst sollte in der Anwendung die Verbindung zu Photon aufgebaut werden *(siehe Listing 1)*.

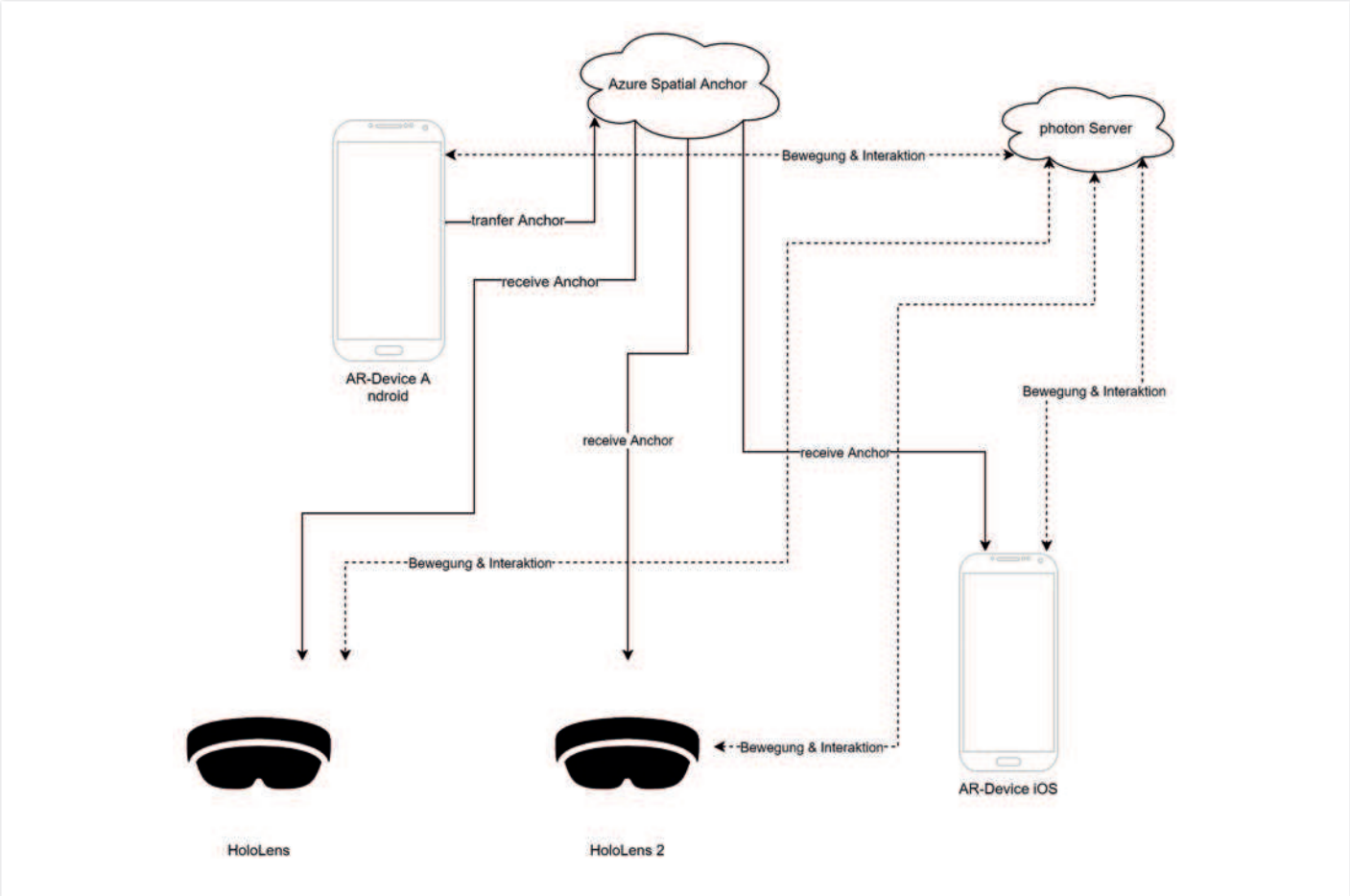

Abbildung 7: Minimalarchitektur „Multiplayer"-AR-Anwendung

```
// connect to photon server using settings from
// Assets\Photon\PhotonUnityNetworking\Resources\PhotonServerSettings.asset
PhotonNetwork.ConnectUsingSettings();
```

Listing 1: Verbindung zu Photon-Server

```
public override void OnConnectedToMaster()
{
    base.OnConnectedToMaster();
    var roomOptions = new RoomOptions
    {//60 seconds
        EmptyRoomTtl = 60000
    };
    PhotonNetwork.JoinOrCreateRoom("ROOM_NAME", roomOptions, null);
}
```

Listing 2: Verbindung zu einer Session „ROOM_NAME" auf einem Photon-Server

```
public SpatialAnchorManager ASAManager;
private async Task StartASAsync()
{
    try
    {
        //create session and wait for it
     //settings can be modified in file Assets\AzureSpatialAnchors.SDK\Resources\SpatialAnchorConfig.asset
        await ASAManager.CreateSessionAsync();
        //session creation was successfull now start session and wait for it
        await ASAManager.StartSessionAsync();
        //session is now successfully running
    }
    catch (Exception e)
    {
        Debug.LogError("Sessions SetUp failed");
        Debug.LogException(e);
    }
}
```

Listing 3: ASA (Azure Spatial Anchors)-Session initialisieren

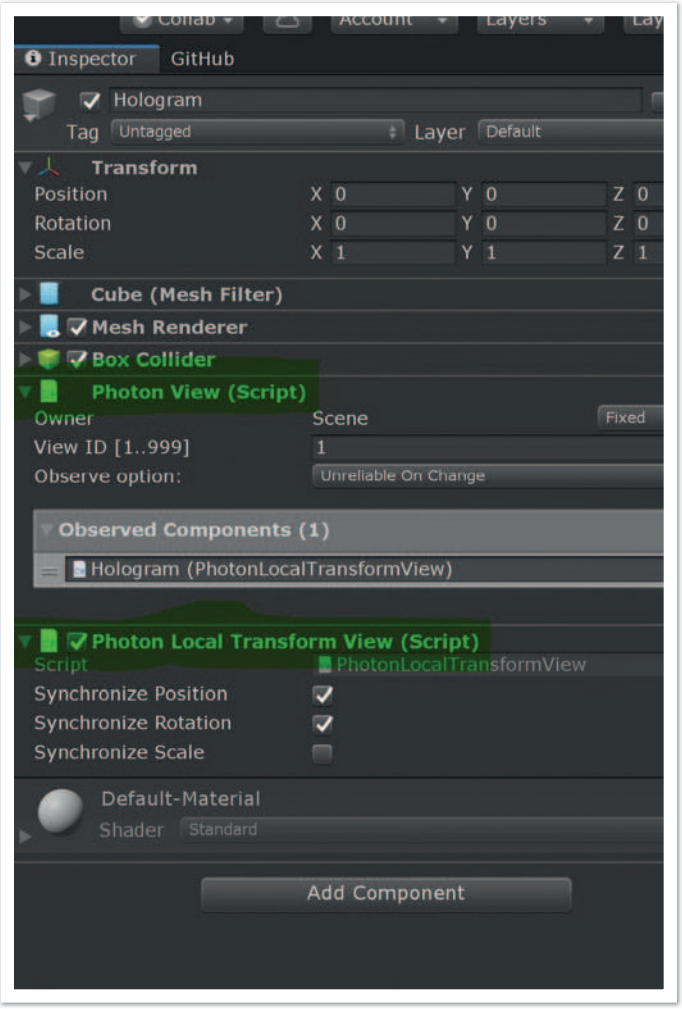

Abbildung 8: Unity-Editor: Gameobjekt mit Photon-Skripten

Da auf einem Server mehrere Sessions parallel laufen können, muss nach der Verbindung zum Server einem „Raum" beigetreten werden. In *Listing 2* wird gezeigt, wie man mit dem Callback der Photon-Klasse „MonoBehaviourPunCallbacks" einem definierten Raum beitritt.

Alle Teilnehmer eines Raumes können nun Daten, zum Beispiel Bewegung und Rotation von 3D-Objekten, miteinander teilen und synchronisieren. Das Photon-SDK bietet für verschiedene Aufgaben bereits vorbereitete Scripts. So gibt es PhotonView.cs *[23]* zur Markierung und Identifizierung eines Objektes, das Eigenschaften über das Netzwerk teilen soll. PhotonTransformView.cs *[24]* muss zusätzlich an Unity-Game-Objekte gehängt werden, um die Position und Rotation zu synchronisieren *(siehe Abbildung 8)*.

Über die Photon-Verbindung kann der Spatial-Anchor zwischen den Geräten geteilt werden. Bevor allerdings ein Anchor gefunden oder erstellt werden kann, muss eine Session aus dem Azure Spatial Anchors SDK initialisiert und gestartet werden *(siehe Listing 3)*.

Mit der gestarteten Session ist alles zum Erstellen, Speichern, Laden und Finden eines Anchor vorbereitet. In *Listing 4* werden genau diese beiden Optionen dargestellt. Dabei wird die ID des erstellten Anchor mit Photon an allen verbundenen Geräten übertragen.

Zu beachten ist noch, dass sämtlicher Quellcode hier ein Minimal-Beispiel darstellt. In einer „echten" Anwendung sollten an den nötigen Stellen Ausfall-Szenarien implementiert werden.

Bewegung von Hologrammen und Anchor

Anchor sind darauf ausgelegt, möglichst stabil an einem Ort in der Realität zu verharren, daher sollte in einer Anwendung davon ab-

```
async Task CreateAndSaveAnchor()
{
    // get or create cloud-native anchor
    var cna = GetComponent<CloudNativeAnchor>();
    if (!cna)
    {
        Debug.Log("Adding CloudNativeAnchor");
        cna = gameObject.AddComponent<CloudNativeAnchor>();
    }

    // If the cloud portion of the anchor hasn't been created yet, create it
    if (cna.CloudAnchor == null) { cna.NativeToCloud(); }

    // Get the cloud portion of the anchor
    var cloudAnchor = cna.CloudAnchor;

    // make anchor expire in 3 days
    cloudAnchor.Expiration = DateTimeOffset.Now.AddDays(3);

  //wait while device is capturing enough visual data to create anchor
    while (!ASAManager.IsReadyForCreate && ASAManager.SessionStatus.RecommendedForCreateProgress < 1F)
    {
        await Task.Delay(330);
        var createProgress = ASAManager.SessionStatus.RecommendedForCreateProgress;
        _debugText = $"Move your device to capture more environment data: {createProgress:0%}";
    }
    try
    {
        // save anchor to cloud
        await ASAManager.CreateAnchorAsync(cloudAnchor);

        // successfull?
        if (cloudAnchor != null)
        {
            _debugText = "Anchor saved";
            // save anchor-ID as property to photon room
            PhotonNetwork.CurrentRoom.SetCustomProperties(
                new Hashtable()
                {
                    { ANCHOR_ID_CUSTOM_PROPERTY, cloudAnchor.Identifier }
                }
            );
        }
        else
        {
            Debug.LogError(new Exception("Failed to save, but no exception was thrown."));
            _debugText = "Anchor saving failed";
        }
    }
    catch (Exception ex)
    {
        Debug.LogError(ex);
    }
}

async Task SearchAnchorAsync()
{
    // is there an anchor-ID in properties of photon room?
    if (PhotonNetwork.CurrentRoom.CustomProperties.TryGetValue(ANCHOR_ID_CUSTOM_PROPERTY, out var anchorID))
    {
        _debugText = "loading Anchor";
        Debug.Log("Achnor loading");
     // tell the watcher which anchor to look for
        var anchorLocateCriteria = new AnchorLocateCriteria
        {
            Identifiers = new[] { (string)anchorID }
        };
        _watcher = ASAManager.Session.CreateWatcher(anchorLocateCriteria);
        //don't do it this way in production code!
     //callback will be called, when anchor is visually found
        ASAManager.AnchorLocated += CloudManager_AnchorLocated;
    }
}
private void CloudManager_AnchorLocated(object sender, AnchorLocatedEventArgs args)
{
    if (args.Status == LocateAnchorStatus.Located)
    {
        var anchor = args.Anchor;
        UnityDispatcher.InvokeOnAppThread(() =>
        {//operation need to run in app thread
            var cna = GetComponent<CloudNativeAnchor>();
            if (!cna)
            {
                Debug.Log("Adding CloudNativeAnchor");
                cna = gameObject.AddComponent<CloudNativeAnchor>();
            }
        //adding found cloud anchor to cloud native anchor so the object will place itself at real world location
            cna.CloudToNative(anchor);
            _debugText = "Anchor Located";
        });
    }
    else if (args.Status == LocateAnchorStatus.NotLocatedAnchorDoesNotExist || args.Status == LocateAnchorStatus.NotLocated)
    {
        _debugText = "Anchor not Located";
    }
}
```

Listing 4: ASA Anchor Erstellungs- und Lokalisierungsprozess

gesehen werden, diese ständig zu bewegen. Da wir aber bewegte Hologramme wollen, bietet es sich an, die Anchor immer als eine Art „Basis" zu behandeln. Diese Basis wird in der Realität möglichst nah an realen Objekten verankert. Die Hologramme, die man bewegen möchte, werden dann relativ zu solch einem Anchor bewegt. Das bedeutet natürlich, dass man hier entweder wieder sein Mathebuch zum Thema Vektor- und Matrizenrechnung hervorholt oder sich auf die entsprechenden Methoden in der Unity-Engine verlässt. Meiner Meinung nach ist es durchaus hilfreich, wenn man versteht, was passiert. Dieser Anchor muss keine grafische Repräsentation in einer Anwendung besitzen. Außerdem sollte man drauf achten, dass die Hologramme sich nicht zu weit von einem Anchor wegbewegen, da bereits kleine Fehler in der Positionierung sich über größere Entfernung immer mehr bemerkbar machen. Daher sollte man bei größeren Entfernungen einen neuen Anchor erstellen und das Hologramm nun relativ zu diesem bewegen oder aber eine entsprechende Boundary in der Anwendung implementieren.

Pokémon Go versus Minecraft Earth

Eingangs hatte ich erwähnt, dass ich Minecraft Earth technisch interessanter im Vergleich zu Pokémon Go finde. In Minecraft Earth können nämlich mehrere Personen gemeinsam an ihren Bauwerken bauen und somit ihre Hologramme teilen. Auch wenn Pokémon Go einen großen Beitrag dazu geleistet hat, AR Mainstream zu machen, so waren zu Beginn die Pokémon-Hologramme nicht einmal in der Realität verankert.

Dieser Artikel erscheint begleitend zu meinem Vortrag in der JavaLand im März 2020 *[25]*.

Quellen

[1] *https://de.wikipedia.org/wiki/Holografie*
[2] *https://dictionary.cambridge.org/dictionary/english/augmented-reality*
[3] *https://en.wikipedia.org/wiki/Augmented_reality* (Englisch)
https://de.wikipedia.org/wiki/Erweiterte_Realit%C3%A4t (Deutsch)
[4] *https://docs.opencv.org/master/d9/d97/tutorial_table_of_content_features2d.html*
[5] *https://docs.opencv.org/master/d7/d66/tutorial_feature_detection.html*
[6] *https://de.wikipedia.org/wiki/Inertiale_Messeinheit*
[6a] *https://www.pcwelt.de/news/Hololens-Microsoft-lueftet-HPU-Geheimnis-10028103.html*
[7] *https://android-developers.googleblog.com/2017/08/arcore-augmented-reality-at-android.html*
[8] *https://9to5mac.com/2017/06/05/apple-announces-arkit-for-ios-11/*
[9] *https://developer.apple.com/documentation/arkit/aranchor*
[10] *https://developers.google.com/ar/develop/developer-guides/anchors*
[11] *https://docs.unity3d.com/ScriptReference/XR.WSA.WorldAnchor.html*
[12] *https://docs.microsoft.com/en-us/windows/mixed-reality/persistence-in-unity*
[13] *https://developers.google.com/ar/develop/java/cloud-anchors/overview-android*
[14] *https://docs.microsoft.com/en-gb/azure/spatial-anchors/overview*
[15] *https://azure.microsoft.com/en-us/pricing/details/spatial-anchors/*
[16] *https://unity.com/*
[17] *https://www.photonengine.com/*
[18] *https://docs.microsoft.com/en-us/azure/spatial-anchors/concepts/anchor-relationships-way-finding*
[19] *https://github.com/Azure/azure-spatial-anchors-samples/releases*
[20] *https://www.photonengine.com/en-US/sdks#realtime-unity-sdkrealtimeunity*
[21] *https://github.com/Microsoft/MixedRealityToolkit-Unity*
[22] *https://unity.com/unity/features/arfoundation*
[23] *https://doc-api.photonengine.com/en/pun/v2/class_photon_1_1_pun_1_1_photon_view.html*
[24] *https://doc-api.photonengine.com/en/pun/v2/class_photon_1_1_pun_1_1_photon_transform_view.html*
[25] *https://programm.javaland.eu/2020/#/scheduledEvent/590747*

Chris Papenfuß
data experts GmbH
Chris.Papenfuss@data-experts.de

Chris Papenfuß leitet seit 2017 das Team Holographic bei der Firma data experts GmbH. In den letzten drei Jahren hat sein Team diverse AR/VR-Projekte für Kunden wie zum Beispiel Rolls-Royce Deutschland, Deutsche Bahn und Homag realisieren und regelmäßig erfolgreich an Hackathons teilnehmen können. data experts gehört zu den ersten Microsoft-Mixed-Reality-Partnern.

Java aktuell

Mehr Informationen zum Magazin und Abo unter:

https://www.ijug.eu/de/java-aktuell

iJUG Verbund
www.ijug.eu

Eventpartner: AOUG SOUG swiss oracle user group iJUG Verbund ORACLE