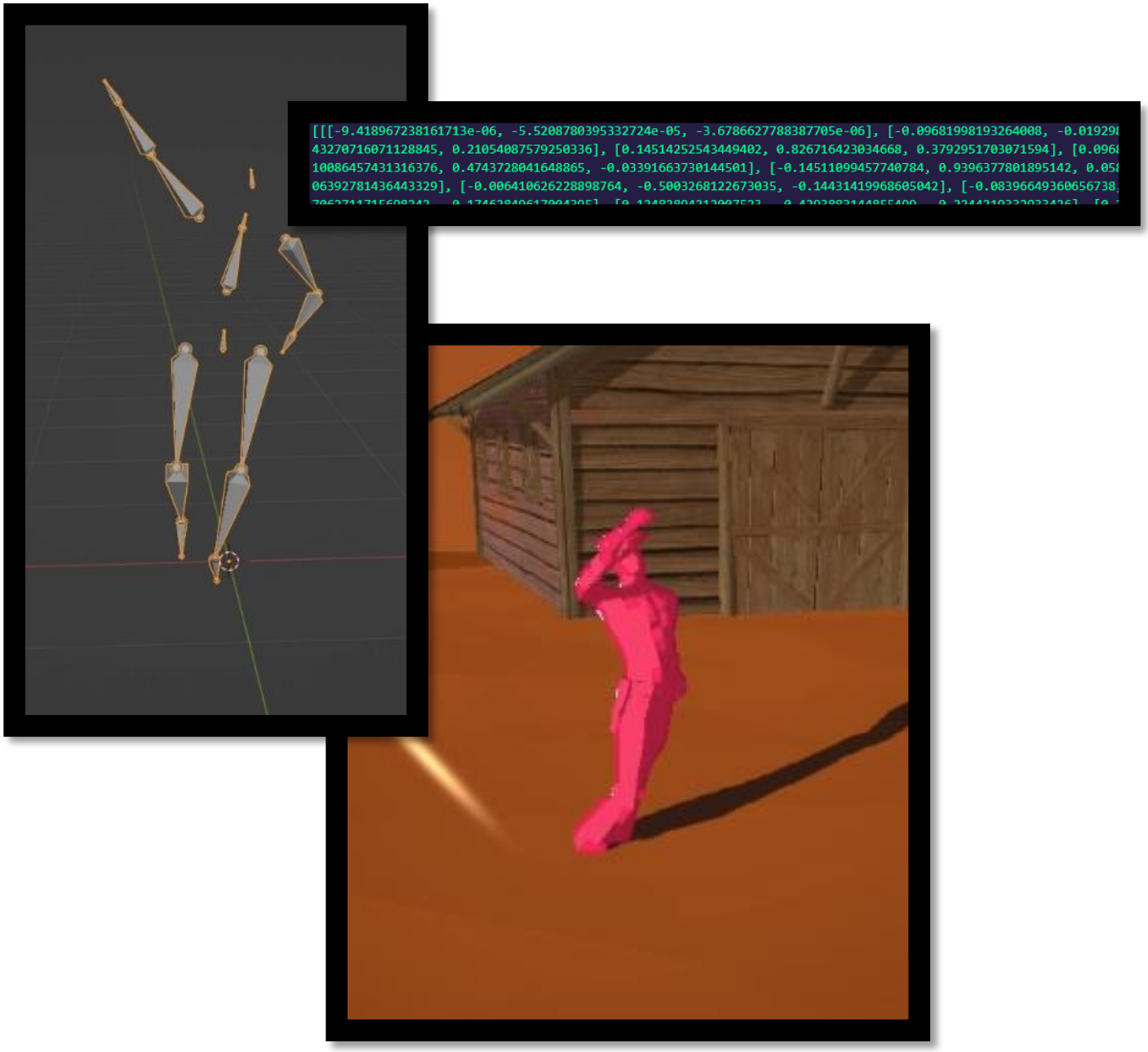
AR-Bericht 2023



< Animationsgenerierung von VidePose3D Outputs >

Hochschule Luzern - Informatik

Mike Pullen

Inhaltsverzeichnis

Einleitung

Dieses Dokument beschreibt die Arbeit, die der Student Mike Pullen für das Modul AR gemacht hat. Das Dokument ist in 5 Hauptabschnitte unterteilt: Projekteinleitung, Grundlagen, Implementierung, Auswertung/Evaluierung und Fazit.

Das Dokument ist in einem persönlichen, anekdotischen Stil verfasst, wobei die Sätze in der Ich-Perspektive geschrieben sind (mit "ich" oder "mich"). Der Autor, Mike Pullen, ist der Einzige, der die Perspektive einnimmt.

Projekteinleitung

In diesem Projekt soll die Verwendung von VidePose3D für die Erstellung von Animationen in Unity ermittelt werden. Ziel ist es, einen Workflow zu etablieren, der FBX-Dateien unter Verwendung der Gelenkdaten von VidePose3D generiert und den Prozess sorgfältig dokumentiert. Zusätzlich wird das Automatisierungspotenzial durch die Erweiterung von Features bestehender Tools wie Poseify oder die Implementierung eines Unity-Plugin untersucht.

Ich persönlich bin sehr motiviert, an diesem Projekt zu arbeiten, weil es für die Open-Source-Software Poseify von Nutzen sein kann. Die Möglichkeit, Animationen mit Hilfe von Poseify zu erstellen, würde bedeuten, dass die Software in Bezug auf ihre Nützlichkeit sehr viel gewinnen würde. Die Möglichkeit, Animationen aus einem einzigen Video zu erstellen, wäre ein grosser Vorteil, nicht nur für mich als Indie-Spieleentwickler, sondern auch für viele andere Menschen in verschiedenen Branchen.

Implementierung

Die Umsetzung erfolgte in 3 klar definierten Phasen:

- **Forschung**
  Da es sich hier um ein Nischenthema handelt, musste viel recherchiert werden, bevor eine Implementierung möglich war.
- **Python-Skript**
  Schreiben eines Python-Skripts, das die VideoPose3D-Ausgabe parsen kann.
- **Poseify-Integration**
  Wie die evaluierte Lösung in Poseify integriert wurde.

Forschung

Da ich keine Erfahrung mit der Arbeit an einer solchen mathematikbezogenen Softwareentwicklung hatte, war es klar, dass ich viel recherchieren musste, um das Thema besser zu verstehen und zu wissen, was das eigentliche Ziel sein sollte. Nach stundenlangem Stöbern in verschiedenen GitHub Repos/Issues, Stack Overflow und vielen anderen Seiten wurde folgendes gefunden, was für ein erfolgreiches Projekt relevant war.

Skelettformat

Es war sehr wichtig, die Ausgabe von VideoPose3d zu verstehen, die an sich keine komplexe Struktur ist, aber es fehlte eine Dokumentation, die klar definiert, was sie ist.

Eine Videopose3d-Predicition enthält, vereinfacht ausgedrückt, eine Liste von Frames, in der für jedes Frame eine Liste von Gelenkpositionen und -drehungen gespeichert ist. Genauer definiert ist es eine Menge von Keypoints, wobei jeder Keypoint durch seine Koordinaten und einen Keypoint-Index definiert ist. Zusätzlich spezifiziert das Format die Konnektivität zwischen den Keypoints und bildet damit eine «Skelettstruktur».

Für den Output kann das folgende index binding verwendet werden:

```
self.keypoint2index = {
    'Hip': 0,
    'RightHip': 1,
    'RightKnee': 2,
    'RightAnkle': 3,
    'LeftHip': 4,
    'LeftKnee': 5,
    'LeftAnkle': 6,
    'Spine': 7,
    'Thorax': 8,
    'Neck': 9,
    'HeadEndSite': 10,
    'LeftShoulder': 11,
    'LeftElbow': 12,
    'LeftWrist': 13,
    'RightShoulder': 14,
    'RightElbow': 15,
    'RightWrist': 16,
    'RightAnkleEndSite': -1,
    'LeftAnkleEndSite': -1,
    'LeftWristEndSite': -1,
    'RightWristEndSite': -1
}
```

Abbildung 1 Keypoint zu Index Binding

Erklärt an einem Beispiel, der Joint RightAnkle im h36m Skeleton kann bei der VideoPose3D-Ausgabe an der Position 3 gefunden werden. -1 definiert das dieses h36m Joint keinen passenden Keypoint hat. Auf der Grundlage dieser Informationen kann die VideoPose3D-Ausgabe wie folgt analysiert werden.

```
[
    [
Hip         [3.4432923712302e-06, -5.299251279211603e-05, 2.9700197501369985e-06],
RightHip    [-0.09168196469545364, -0.009060420095920563, -0.0021563211921602488],
RightKnee   [-0.0939699113368988, 0.08705682307481766, -0.39008039236068726],
RightAnkle  [-0.08387982845306396, 0.2300644814968109, -0.6484818458557129],
LeftHip     [0.09168145060539246, 0.009052958339452744, 0.0021666456013917923],
LeftKnee    [0.08563882112503052, 0.10286296904087067, -0.38635390996932983],
LeftAnkle   [0.011247910559177399, 0.24902282655239105, -0.5970525741577148],
Spine       [0.027421778067946434, -0.14503924548625946, 0.14652109146118164],
Thorax      [0.05896963179111481, -0.35549163818359375, 0.2500664293766022],
Neck        [0.06549239158630371, -0.37734776735305786, 0.1858377605676651],
HeadEndSite [0.06791260838508606, -0.4785730838775635, 0.21560028195381165],
LeftShoulder [0.13811060786247253, -0.3279905915260315, 0.24326232075691223],
LeftElbow   [0.13364863395690918, -0.11146578192710876, 0.2775162160396576],
LeftWrist   [0.13312441110610962, -0.008068231865763664, 0.1832500696182251],
RightShoulder [-0.029369406402111053, -0.33788517117500305, 0.23068532347679138],
RightElbow  [-0.03643470257520676, -0.1259974241256714, 0.24413317441940308],
RightWrist  [-0.007702089846134186, -0.025032667443156242, 0.14593833684921265]
    ],
    ... more frames
```

Abbildung 2: Joint Index anhand Poseify Beispiel

Für das Skelett wurde das Format Human Motion 36 (H36M) entschieden, da das H36M-Format speziell für die menschliche Motion Capture entwickelt wurde und bietet eine umfassende Darstellung.

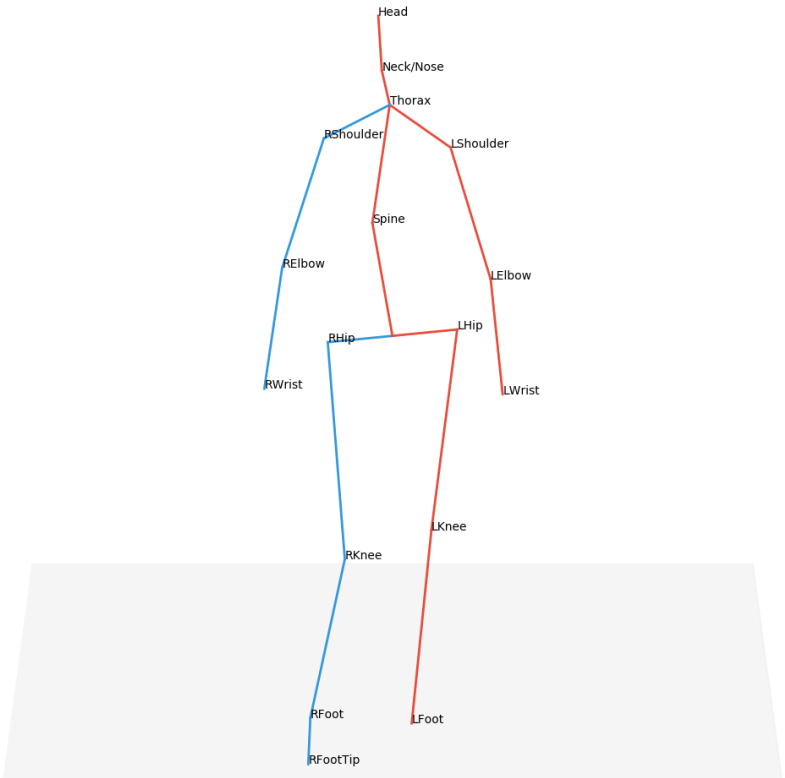


Abbildung 3: H36M Beispielt

Nicht FBX!

Nachdem ich mich für das Skeleton-Format entschieden hatte, bestand der nächste Schritt darin, herauszufinden, wie man die Animationen in Blender einbinden kann. Ursprünglich war die Idee ein FBX zu generieren, leider wurde während der Recherche klar, dass die Erstellung einer FBX-Datei ziemlich schwierig sein würde und den Zugriff auf die Blender Python API erfordert (was der Hauptweg zur Erstellung von FBX-Dateien über Code zu sein scheint). Da es sich um ein zeitkritisches Projekt handelt, wurde beschlossen, keine FBX-Dateien für den Import in Blender zu verwenden, sondern stattdessen eine BVH-Datei zu erstellen, die einfacher zu erstellen ist und keinen Zugriff auf Blender erfordert, was bedeutet, dass man Blender nicht installieren muss, um dieses Skript ausführen zu können.

Dies hat zur Folge, dass ein vollautomatischer Import in Unity nicht möglich ist und der Benutzer von Poseify das FBX immer manuell in Blender erstellen muss.

BVH

Um eine BVH-Datei zu erstellen, war es zunächst wichtig zu verstehen, was eine BVH-Datei ist und wie sie aufgebaut ist.

BVH (Biovision Hierarchy) ist ein Dateiformat, das häufig zur Speicherung von Motion-Capture-Daten im Bereich der Computeranimation und Biomechanik verwendet wird. Es stellt die hierarchische Struktur eines Skelettsystems dar und erfasst die Gelenkrotationen im Zeitverlauf, um animierte Charaktere oder Modelle zu erstellen.

Das BVH-Dateiformat besteht aus zwei Hauptabschnitten: dem Abschnitt *HIERARCHY* und dem Abschnitt *MOTION*. Der Abschnitt *HIERARCHY* definiert die Skeletthierarchie, einschliesslich der Joints, ihrer Eltern-Kind-Beziehungen und der Reihenfolge der Rotationskanäle. Jeder Joint wird durch seinen Namen, seinen Offset (relative Position zum übergeordneten Joint) und seine Rotationskanäle (z. B. Euler-Angle oder Quaternionen) definiert. Es kann als eine baumähnliche Struktur verstanden werden. Der Abschnitt *MOTION* enthält die Animationsdaten. Er gibt die Gesamtzahl der Frames, die Framerate und die Rotationswerte für jeden Joint in jedem Frame an. Die Bewegungsdaten stellen in der Regel die Rotationen in Form der angegebenen Kanäle (z. B. X-Rotation, Y-Rotation, Z-Rotation) für jeden Joint in jedem Frame dar.

Hier ist ein vereinfachtes Beispiel für eine BVH-Datei:

```
HIERARCHY
ROOT Hip
{
    OFFSET 0.0 0.0 0.0
    CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
    JOINT RightHip
    {
        OFFSET -0.11174092805377328 0.0 0.0
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightKnee
        {
            OFFSET 0.0 0.0 -0.38685768687911826
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightAnkle
            {
                OFFSET 0.0 0.0 -0.2587473037751311
                CHANNELS 3 Zrotation Xrotation Yrotation
                End Site
                {
                    OFFSET 0.0 -0.10349892151005245 0.0
                }
            }
        }
    }
    JOINT LeftHip
    {
        OFFSET 0.11174092805377328 0.0 0.0
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftKnee
        {
            OFFSET 0.0 0.0 -0.38685768687911826
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT LeftAnkle
            {
                OFFSET 0.0 0.0 -0.2587473037751311
                CHANNELS 3 Zrotation Xrotation Yrotation
                End Site
                {
                    OFFSET 0.0 -0.10349892151005245 0.0
                }
            }
        }
    }
    ....
}
MOTION
Frames: 145
Frame Time: 0.03333333333333333
0.0 0.0 -0.24907581906518317 -6.907192391803101 44.4355496133378 -167.54721337052837 -2.7889506701304003 -31.165399209274298 -5.051455976500698 0.07439
-4.5737397158518434e-07 5.5878217608551495e-08 -0.24907570856885286 -7.225323193429277 44.20601453802567 -167.36815872034236 -2.7464783717610297 -30.79
4.51727828476578e-08 3.227637535019312e-08 -0.2490756033694197 -7.633477821340642 44.20136201937805 -167.14203196996894 -2.6046176065607307 -30.4982104
...
```

Abbildung 4 Vereinfachtes Beispiel einer BVH-Datei

In diesem Beispiel definiert der Hierarchieabschnitt eine einfache Skelettstruktur mit "Hip" als root und zwei Beinen mit ihren jeweiligen Fussgelenken. Der Bewegungsabschnitt spezifiziert Animationsdaten für 145 Frames, mit einer Framerate von etwa 30 Frames pro Sekunde. Die Rotationswerte für jedem Joint in jedem Frame folgen nach dem Kopf des Bewegungsabschnitts.

BVH-Dateien können importiert und in verschiedenen Animationsprogrammen und Game-Engines verwendet werden, um 3D-Charaktere oder Modelle auf der Grundlage der erfassten Bewegungsdaten zu animieren.

Mit all den gesammelten Informationen über Motion-Capture-Daten war ich bereit, mit der Implementierung des Python-Skripts zu beginnen.

Python

Die Implementierung basiert auf den folgenden GitHub-Repositories:

- https://github.com/KevinLTT/video2bvh
- https://github.com/svolokh/MocapToFbx

Die Möglichkeit, meinen Code auf diese Repos zu stützen, hat die Entwicklungszeit erheblich verkürzt.

Erklärung des Python-Skripts

Die folgenden Python-Files wurden in Rahmen dieses Projektes erstellt / bearbeitet:

estimate_pose.py *geändert*

Das Hauptskript für die Estimation, das so bearbeitet wurde, dass es nach einer erfolgreichen Estimation nun das Skript output2bvh.py aufruft, um die Motion Capture Data zu verarbeiten. Dieses Skript ist nur für Poseify relevant.

output2bvh.py *neu*

Dieses Skript ist der Einstiegspunkt für die Erstellung der Motion Capture Data und erhält ein Numpy-Array mit Keypoints von estimate_pose.py (wenn es im Kontext von Poseify ausgeführt wird). Es führt einige Vorverarbeitungen mit der Hip, dem linken und dem rechten Ankle durch und gibt diese Daten dann an h36m_skeleton.py weiter

h36m_skeleton.py *neu*

h36m_skeleton.py ist für die Erstellung des Skeletts verantwortlich, indem es die Initial Offsets und Euler-Angles für jedes Gelenk berechnet.

bvh_helper.py *neu*

Ein Skript, das die bvh-Datei auf der Grundlage des Ergebnisses von h36m_skeleton.py schreibt.

math3d.py *neu*

Ein Hilfsskript für 3D-Matheoperationen.

Poseify Integration

Der Plan war, einen C#-Code zu schreiben, der dasselbe tut wie das Python-Skript aus dem vorherigen Kapitel. Leider war es nicht möglich, dies im Zeitrahmen dieses Projekts vollständig zu implementieren. Von der Logik her ist es identisch mit dem Python-Skript, aber insgesamt viel effizienter, da C# dank des MathNet-Pakets sehr leistungsfähige Vektor-/Euler-Winkel-Operationen bietet. Da der C#-Code unvollständig ist, wird das vorherige Python-Skript verwendet, um die BVH-Datei auf dem Poseify-Server zu erstellen. Das neue Python-Skript wird unmittelbar nach der Pose-Estimation ausgeführt. Die neue BVH-Datei wird dann in der Poseify-Datenbank gespeichert (Nur für neue Estimations werden BVH-Dateien generiert).

Der Benutzer kann die BVH-Datei im Ansichtsmodal mit der Schaltfläche ".bvh download" herunterladen.

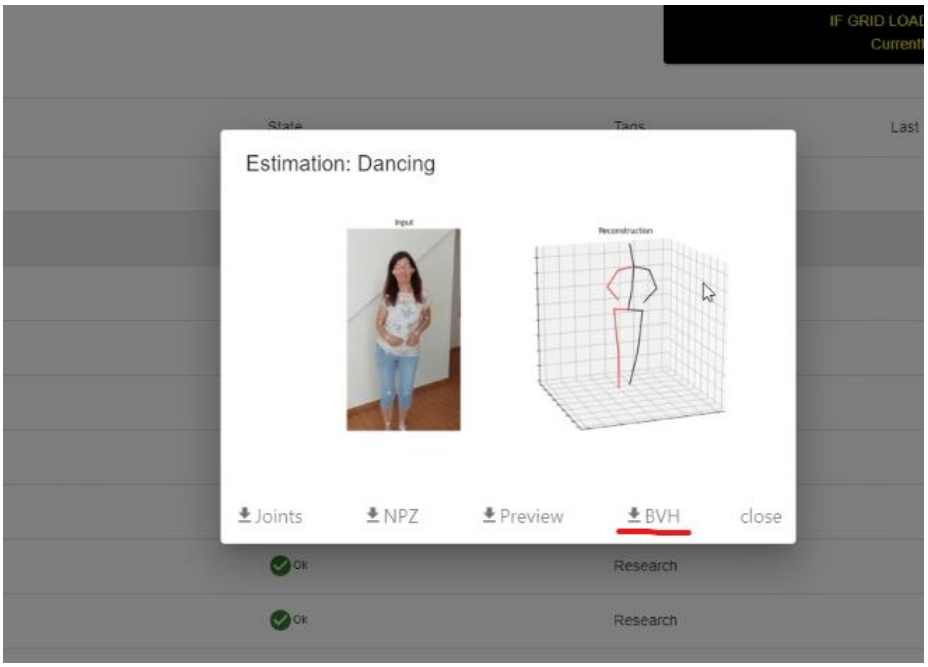


Abbildung 5: UI Download Button für BVH

Die komplette Code-Änderung an Poseify kann im folgendem Pull-Request gefunden werden: https://github.com/fierc3/poseify/pull/98/files

Auswertung / Evaluierung

Zu Beginn dieses Projekts wurden 3 Bewertungskategorien definiert.

Die *Bewertung der Animationsqualität*, *der Workflow-Effizienz* und der *Automatisierungsvorteile*.

Bewertung der Animationsqualität

Die Bewertung der Animationsqualität erfolgte anhand vom visuellen Vergleich zwischen der Reconstruction Video, Blender und Unity.

Hier das Ergebnis der Animation Dancing.

Frame 1 ✅

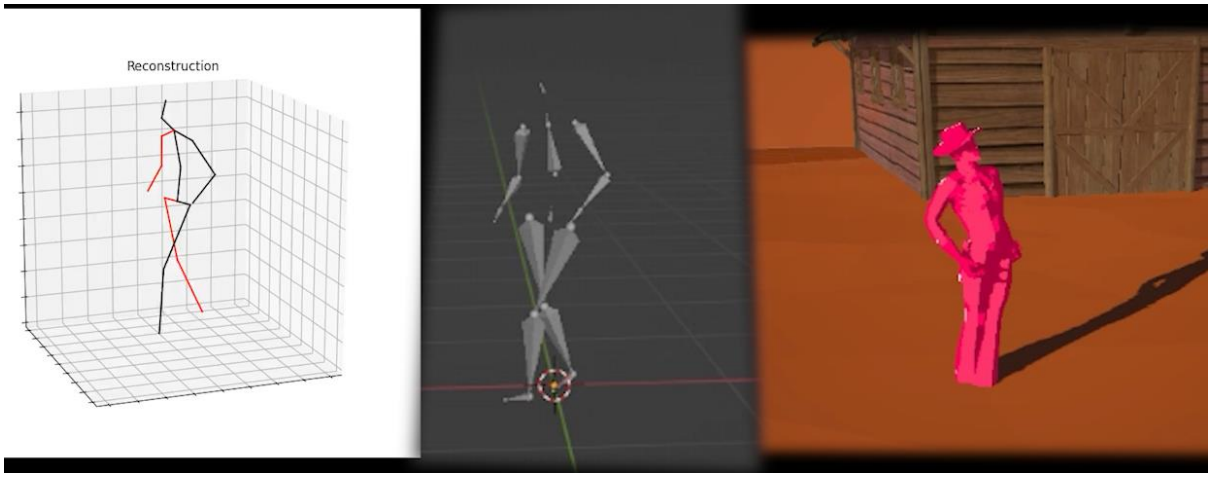


Abbildung 6: Frame 1 Resultat

Frame 18 ✅

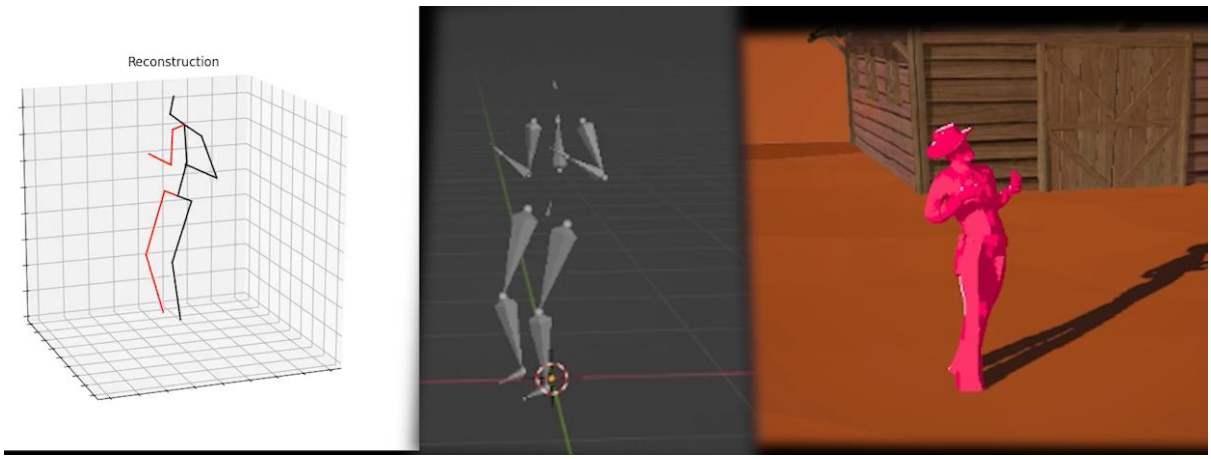


Abbildung 7: Frame 18 Resultat

Frame 60 ✅

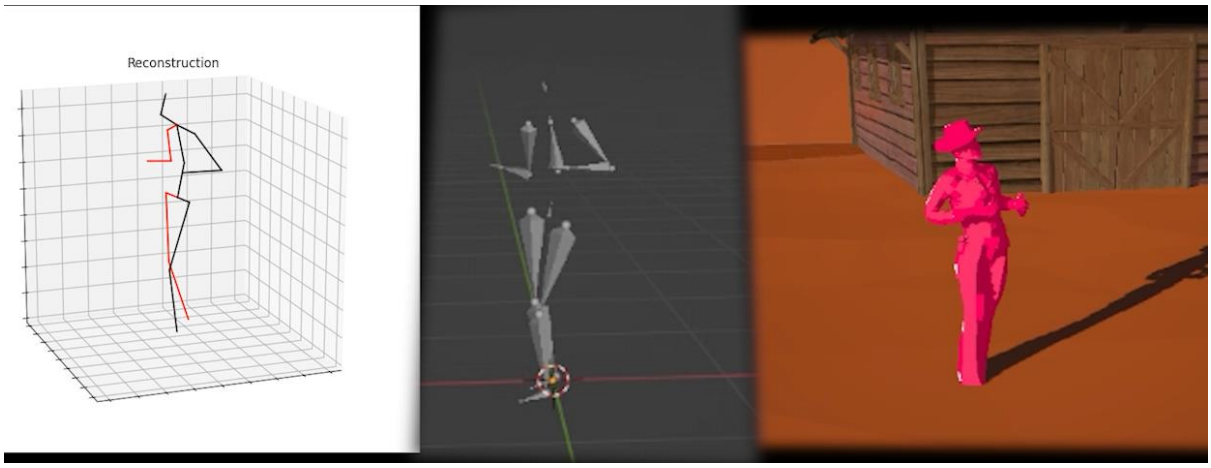


Abbildung 8: Frame 60 Resultat

Frame 80 ✅

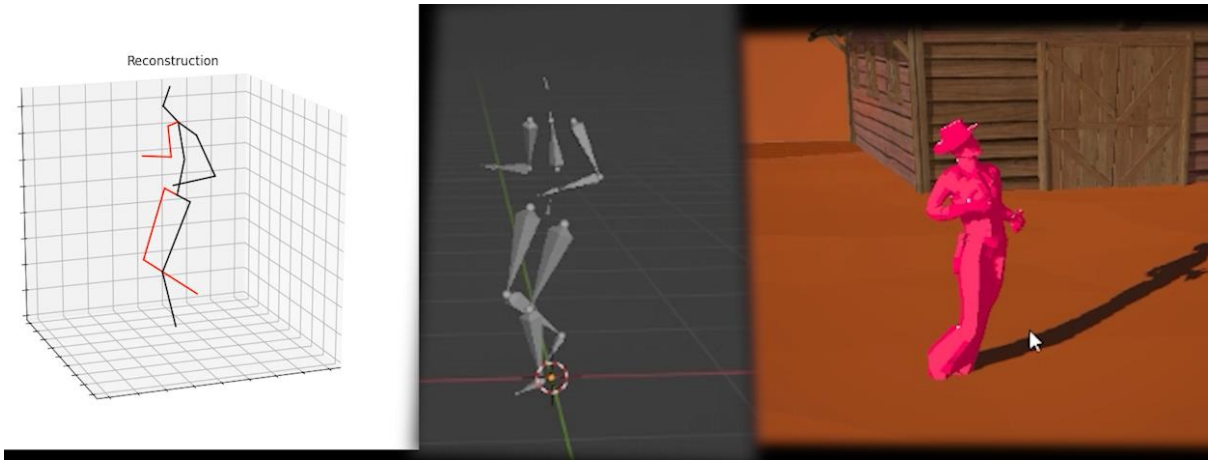


Abbildung 9: Frame 80 Resultat

Frame 110 ✅

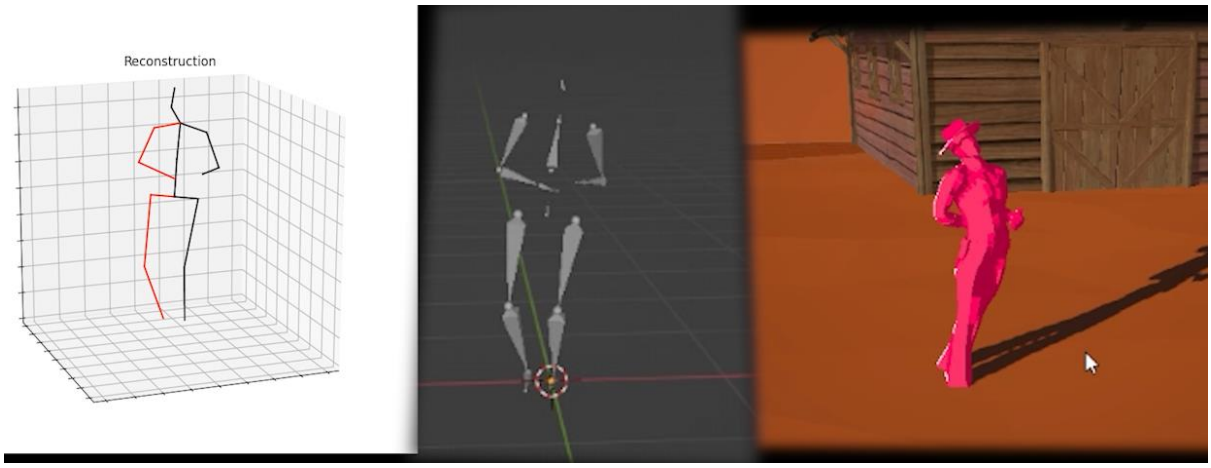


Abbildung 10: Frame 110 Resultat

Frame 275 ❌

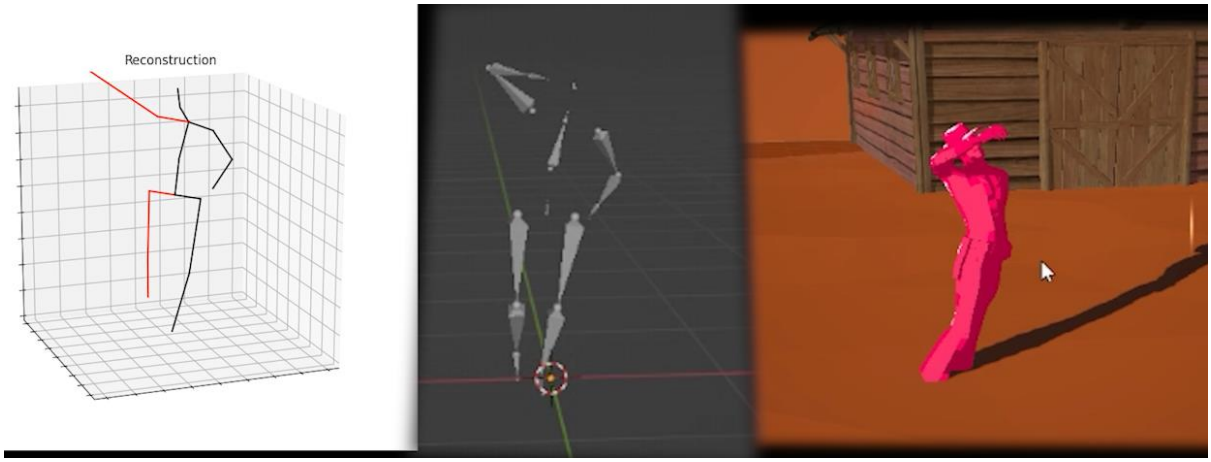


Abbildung 11: Frame 275 Resultat

Anhand vom Beispiel Dancing sieht man, das die Konvertierung von VideoPose3D-Ausgaben zu Unity sehr akkurat ist. Jedoch sehen die Animationen manchmal ein bisschen komisch aus, dass ist aber vor allem, weil VideoPose3d-Ausgaben manchmal sehr unnatürlich aussehen oder auch nicht reproduzierbar sind mit einem menschlichen Skelett (Siehe *Frame 275*). Was beim Testen klar wurde, dass einfache Animationen gut funktionieren können, aber auch nur für Animationen, die entweder im Hintergrund verwendet werden oder die von einem Animator von Hand verbessert werden.

Workflow-Effizienz

Der Arbeitsablauf ist insgesamt sehr effizient. Wenn der Benutzer Erfahrung mit Poseify hat, muss er keine zusätzliche Arbeit leisten, sondern nur auf die Schaltfläche BVH herunterladen klicken. Von diesem Moment an hat er seine Motion-Capture-Daten und kann sie in Blender importieren.

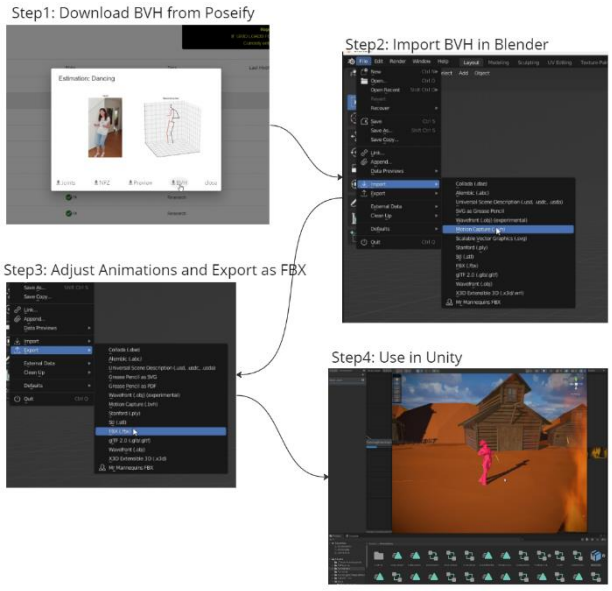


Abbildung 12: Poseify zu Unity

Automatisierungsvorteile

Leider reichte die Zeit nicht aus, um einen vollautomatischen Workflow zu FBX zu implementieren, und der Benutzer muss die BVH-Daten immer erst in Blender importieren.

Die Möglichkeit, diesen Teil zu überspringen, wäre von Vorteil, aber gleichzeitig muss der Benutzer wahrscheinlich Anpassungen von Hand vornehmen, z. B. die Animationen verfeinern, die Skalierung korrigieren, die Drehung anpassen usw.

Fazit

Ich habe bei diesem Projekt sehr viel gelernt, vor allem in Bezug auf das Skelett und die Dateiformate. Ich bin ein bisschen enttäuscht, dass die C#-Implementierung nicht rechtzeitig fertig wurde, aber ich freue mich darauf, den Rest des Sommers daran zu arbeiten.

Was die fehlende FBX-Generierung angeht, fühle ich mich nicht so schlecht. Denn in den meisten Fällen wird der Benutzer ohnehin Anpassungen an der Animation vornehmen müssen, dann genügt eine BVH-Datei. Es wird in der Zukunft interessant sein, wenn mehr Algorithmen zur Pose-Estimation (vielleicht mehr akkurate) in Poseify integriert werden, vielleicht können dann die Motion-Capture-Daten direkt in einem Programm wie Unity / Unreal Engine usw. verwendet werden. Wenn dies der Fall ist, ist eine direkte FBX-Generierung sicher sehr interessant.

Anhang