

Technische Dokumentation VR-  
Visualization  
***Virtual Reality, Master Informatik***

Matthias Haselmaier, Andreas M. Brunnet

# Vorwort

Im Rahmen der Veranstaltung *Virtual Reality* wurde eine VR-Anwendung auf Basis der Unity 3D-Engine entwickelt, mit deren Hilfe eine Betrachtung von Machine Learning Daten im Raum ermöglicht wird. Hierbei hat der Anwender die Möglichkeit innerhalb der als Scatterplots visualisierten Daten umher zu wandern. Darüber hinaus lässt sich ein für den Nutzer interessanter Datenpunkt mittels VR üblicher Selektiergestik auswählen. Die Attribute und Werte des selektierten Datenpunkts werden relativ zu seiner Position dem Nutzer dargestellt. Über das Hervorheben des selektierten Punktes in sämtlichen Scatterplot-Matrizen hat der Anwender direkte Übersicht über die Verteilung des Punktes innerhalb des Datensets.

## Software und Bibliotheken

### Verwendete Software und Bibliotheken

**MiddleVR** ([\[mvvr\]](#)): Dient der Entkopplung von VR-Hardware und der Applikation. Als Middleware abstrahiert MiddleVR Eingaben durch Hardware wie der HTC Vive Pro und stellt diese auf Seite der Anwendung als verallgemeinerte Schnittstelle zur Verfügung.



Während der Navigation in UI Komponenten fiel auf, dass Tasteneingaben nicht *debounced* werden. So werden beispielsweise Checkboxes schwer steuerbar. Auch andere UI Komponenten waren unter diesen Bedingungen nicht zufriedenstellend nutzbar. Entsprechend wurde innerhalb der zuständigen MiddleVR Klasse *VRCanvas* die Abfrage von Tastenevents auf *Toogled* umgestellt, um so ein *pressed / released* Verhalten zu erhalten.

**IATK** ([\[iatk\]](#)): Das Toolkit stellt eine Reihe von Funktionalitäten für das Arbeiten mit Daten zur Verfügung. Unter anderem ist ein CSV-Parser enthalten, der im Gegensatz zu einem üblichen CSV-Parser Wertnormalisierungen unabhängig vom vorliegenden Datentyp vornimmt.



Es musste eine Anpassung am Parser vorgenommen werden. Dort wird in der Methode *GetOriginalValue* ein Float-Wert als Index für ein Array genommen. Hierbei wird die implizite Typkonvertierung und die daraus resultierende Abrundung zum Problem, da an diesem Punkt ein Runden zur nächsten Ganzzahl erfolgen sollte. Dieser Fehler wurde mittels *Mathf.Round* behoben.

Die Möglichkeiten des Frameworks zur Datenvisualisierung wurden nach ersten Tests mit dem Framework verworfen, da eine Eigenimplementierung der nötigen Visualisierungskomponenten mit den nötigen Interaktionsmöglichkeiten, wie der Datenpunktauswahl (siehe [\[Kurzeinleitung\]](#), [\[datapoint\\_selection\]](#)) somit umsetzbar wurde.

### Verworfen Software und Bibliotheken

**VRTK** ([\[vrtk\]](#)): Wie auch MiddleVR dient VRTK der Entkopplung von Eingabegeräten und der Applikation. Eine erste Integration in das Projekt verlief ohne größere Probleme, sodass das Testen

der Anwendung mittels Maus und Tastatur (Modus: Simulation) möglich war. Probleme traten jedoch beim Versuch der Integration einer Vive-Pro Konfiguration auf, da VRTK seit einiger Zeit nicht mehr aktualisiert wird. Entsprechend kam es zu Komplikationen mit der aktuellen API der Vive-Pro (SteamVR, [\[svr\]](#)). Entsprechend wurde als alternative Lösung MiddleVR erfolgreich evaluiert und im Projekt statt VRTK integriert.

## Architektur der Anwendung

Zur Visualisierung der Daten stellt die VR-Anwendung vier Komponenten zur Verfügung, welche hierarchisch aufgebaut sind. Die *Visualizer*-Komponente stellt die Schnittstelle zwischen der UI und der internen Datenstruktur dar. Sie ist dafür zuständig, die Daten aus einer CSV-Datei einzulesen, der UI die nötigen Informationen bereitzustellen und die vom Benutzer gewählte *ScatterplotMatrix* zu erstellen. Die *ScatterplotMatrix*-Komponente wird als Kind des *Visualizers* erstellt. Sie ist für das Erzeugen und Positionieren der einzelnen *Scatterplots* verantwortlich. Die *Scatterplot*-Komponente erstellt ein 3D-Diagramm, in dem die drei übergebenen Dimensionen für alle *DataPoints* gegeneinander aufgetragen werden. Die *DataPoint*-Komponente stellt einen Datenpunkt als Kugel im *Scatterplot* dar. Wie die Szene nach dem Importieren einer CSV-Datei aussieht, ist auf [Abb. 1](#) gezeigt. Im Folgenden werden die Komponenten genauer beschrieben.

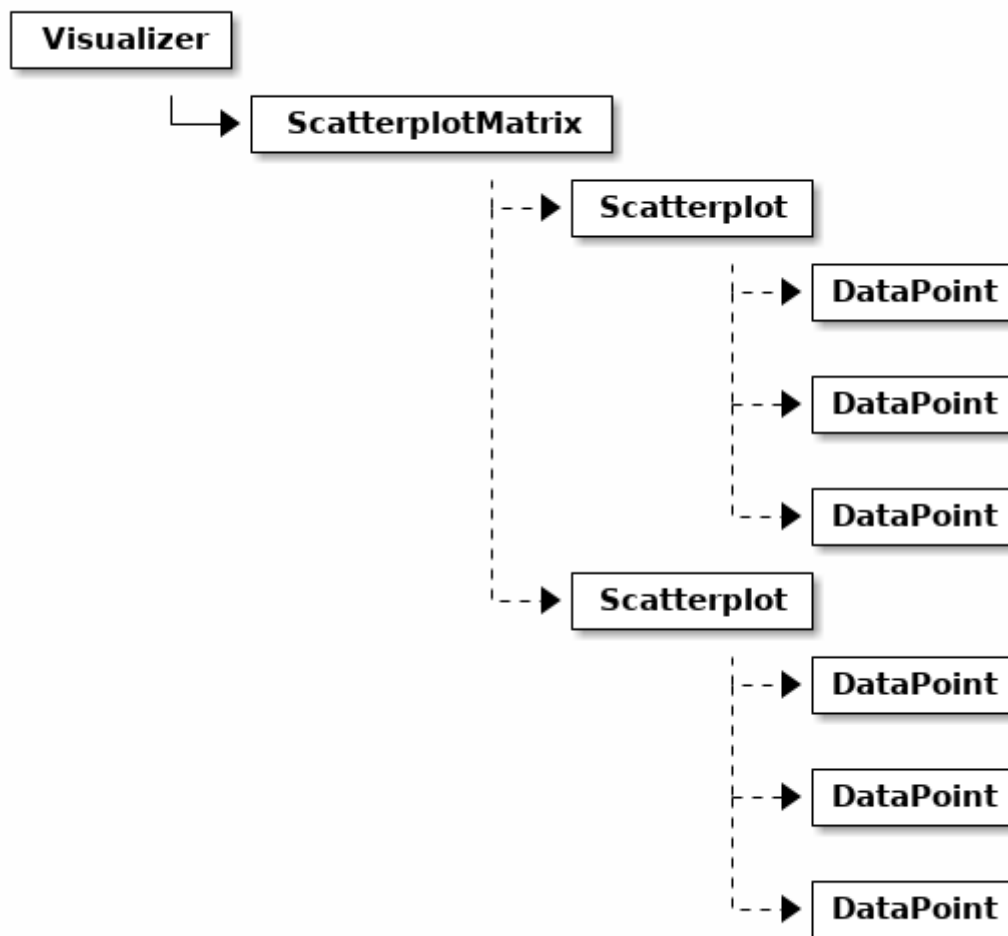


Abbildung 1. Szene

# Visualizer

Die Schnittstelle des *Visualizer* besteht aus drei Methoden und einem Attribut. Die Methode *LoadDataSource(TextAsset)* wird vom *ImportDialog* aufgerufen, wenn der Benutzer im DropDown-Menü eine neue CSV-Datei auswählt. Neben dem eigentlichen Laden der CSV-Datei werden zusätzlich alle möglichen 3er-Kombinationen der in der Datei enthaltenen Dimensionen gebildet. Hierzu wird *CalculatePossibleScatterplots()* aufgerufen, welche ein zweidimensionales *int*-Array zurück gibt. Die erste Dimension dieses Arrays stellt dabei die einzelnen Kombinationen der Dimensionen der CSV-Datei dar. Die zweite Dimension hat die Größe 3 und besteht aus den drei Indizes der jeweiligen Dimension der CSV-Datei. Der *ImportDialog* erhält die 3er-Kombinationen als *string* -Array über die Methode *GetPossibleScatterplots()*. Somit können dem Benutzer die möglichen *Scatterplot*-Varianten angezeigt werden. Wird der *Importieren*-Button angeklickt, ruft der *ImportDialog* die Methode *CreateScatterplotMatrix(int[])* auf. Über den Parameter werden die Indizes der vom Benutzer gewünschten *Scatterplots* übergeben. Es wird dann eine neue *ScatterplotMatrix* instantiiert. Wurde vorher bereits eine *ScatterplotMatrix* erzeugt, wird diese gelöscht. Es ist somit zu jeder Zeit immer nur eine *ScatterplotMatrix* in der Szene. Über das Attribut *pointSize* kann die Größe der *DataPoints* angepasst werden. Der *ImportDialog* inkrementiert bzw. dekrementiert diesen Wert, wenn der Benutzer die entsprechenden Buttons klickt.

## ScatterplotMatrix

Die *ScatterplotMatrix* kapselt und erzeugt *Scatterplots*. Nachdem eine *ScatterplotMatrix* erzeugt wurde sollte immer die Methode *Initialize(CSVDataSource, int[], float)* aufgerufen werden. Durch den Aufruf dieser Methode wird die Referenz auf die eingelesene CSV-Datei gesetzt, die gewünschten *Scatterplots* angegeben und die initiale Größe der *DataPoints* definiert. In ihr wird die private Methode *CreateScatterplots(CSVDataSource, int[],)* aufgerufen. Hierdurch werden die angegebenen *Scatterplots* erzeugt. Zum Erzeugen der *Scatterplots* wird die Coroutine *CreateScatterplotsCoroutine(GameObjec, CSVDataSource, int[], int)* verwendet. Mit dem Frame wird ein *Scatterplot* erzeugt. Diese Variante wurde gewählt, damit die Anwendung während dem Erzeugen der *Scatterplots* nicht einfriert. Die letzte Methode der Komponente *ScatterplotMatrix* ist *SelectDataPoint(int)*. Sie wird von einem *DataPoint* mit seinem Index aufgerufen, wenn er vom Benutzer ausgewählt wird. Aus der *ScatterplotMatrix* wird der Index des jeweiligen *DataPoints* an alle *Scatterplots* weitergegeben. Somit benachrichtigt ein ausgewählter *DataPoint* über die *ScatterplotMatrix* alle *Scatterplots* und diese können den *DataPoint* mit dem gegebenen Index hervorheben.

## Scatterplot

Einem *Scatterplot* sind immer drei Spalten der CSV-Datei zugeordnet, welche über die Attribute *xDim*, *yDim* und *zDim* bestimmt sind. Wie auch schon bei der *ScatterplotMatrix* sollte nach dem Erzeugen eines *Scatterplot* immer die Methode *Initialize(CSVDataSource, float, float, float, int, int, int)* aufgerufen werden. Hierbei bestimmen die ersten beiden *float*-Werte die Position des *Scatterplots* in der Ebene, der letzte *float*-Wert bestimmt die initiale *DataPoint* Größe. Die letzten drei *int*-Werte definieren die drei bereits angesprochenen Spalten der CSV-Datei. Durch den Aufruf von *Initialize(..)* werden außerdem die Axen des *Scatterplots* beschriftet und die einzelnen *DataPoints* erzeugt. Wurde ein *DataPoint* vom Benutzer ausgewählt, wird die Methode *SelectDataPoint(int)* aufgerufen (vgl. [ScatterplotMatrix](#)). Der *Scatterplot* iteriert über alle seiner

*DataPoints* und hebt den durch den Index bestimmten *DataPoint* hervor.

Ein wichtiger Teil der *Scatterplot* Komponente ist die Methode *OnMVRWandEnter(VRSelection)*. Diese Methode wird immer dann aufgerufen, wenn der Wand auf den *Scatterplot* trifft. In diesem Fall wird der Collider des *Scatterplots* deaktiviert und die Collider aller seiner *DataPoints* werden aktiviert. War zuvor ein anderer *Scatterplot* vom Wand ausgewählt, so werden die Collider seiner *DataPoints* wieder deaktiviert und sein eigener Collider aktiviert, damit er das nächste Mal wieder darauf reagieren kann. Durch dieses Vorgehen wird die maximale Anzahl an aktivierten Collidern in Unity begrenzt. Würde dies nicht so gehandhabt werden, kann es passiert, dass Unity eine Fehlermeldung wirft, wenn zu viele *DataPoints* erzeugt werden.

## DataPoint

Die Komponente *DataPoint* visualisiert eine Zeile der CSV-Datei als Kugel in einem Scatterplot. Auch nach dem Erzeugen eines *DataPoints* sollte immer die *Initialize(int, float, Vector3)* Methode aufgerufen werden. Durch den Index erhält der *DataPoint* seine Zuordnung zu einer Zeile der CSV-Datei, die Größe des *DataPoints* wird durch den *float*-Wert definiert und seine Position durch den *Vector3*. Außerdem wird bei der Initialisierung der Text des Attributsdialog gesetzt. Dieser wird angezeigt, wenn der Benutzer den *DataPoint* auswählt. Über die Methode *GetData()* können die Werte des *DataPoints* erhalten werden. Der Rückgabewert der Methode ist ein zweidimensionales *string*-Array. Die erste Dimension repräsentiert die drei Achsen des *DataPoints* in der zweiten wird der Identifizierer der CSV-Spalte mit dem Index 0 angegeben und der entsprechende Werte des *DataPoints* mit Index 1.

Wenn der Benutzer den *DataPoint* auswählt, wird die Methode *VRAction(VRSelection)* aufgerufen. Der *DataPoint* gibt dann seinen Index an die *ScatterplotMatrix* über die Methode *SelectDataPoint(int)* weiter (vgl. [ScatterplotMatrix](#)), damit alle *Scatterplots* benachrichtigt werden können.

## Programmablauf

Das folgende Diagramm stellt den Programmablauf beim erstmaligen Starten der Anwendung dar:

- Blub
  - Blub
- blub

Blub

1. Ich
  - a. Mag
    - i. Züge
2. du
  - a. magst
    - i. LKW

## Neue Subüberschrift

Blub [Mein Bereich](#)

*Beispiel 1. Mein Bereich*

Übernimmt Ascii Zeug

*Listing 1. Source Code*

```
public void Foo(){  
  
}
```

Und noch Ref auf Überschrift [Neue Subüberschrift](#).

## Quellen / Ressourcen

- [MiddleVR]: <https://www.middlevr.com/home>
- [IATK] Immersive Analytics Toolkit: <https://github.com/MaximeCordeil/IATK>
- [VRTK] Virtual Reality Toolkit: <https://vrtoolkit.readme.io>
- [SteamVR]: <https://steamcommunity.com/steamvr>
- [GitHub-Repository] Projekt Repository: <https://github.com/MHaselmaier/VR-Visualization.git>
- [Unity-Dokumentation]: <https://docs.unity3d.com/Manual/index.html>
- [MiddleVR-Dokumentation]: <https://www.middlevr.com/doc/current/>