Virtual Reality

Mathematische Konzepte prototypisch visualisieren

Johannes Schmitt

Letzte Änderung: 19. Dezember 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Im Rahmen der Vorlesung Virtual Reality des Wintersemesters 18/19 wurden am Beispiel von Parameterkurven in der Ebene prototypisch dargestellt. Parameterkurven haben Eigenschaften, die sich nur abstrakt auf Papier bzw. zweidimensional darstellen lassen. In der UnityEngine besteht die Möglichkeit die Parameterkurven mittels Virtual Reality darzustellen. Diese Projektarbeit zeigt exemplarisch drei verschiedene Parameterkurven, um Eigenschaften der Kurven erlebbar zu machen. Die Idee lässt sich erweitern und könnte zukünfitg auch der Illustration von Dingen wie Ebenen im R^3 dienen.

Kapitel 2

Verwendete Technologie

2.1 Unity

Unity ist eine Grafik-Engine, mit deren Hilfe 3D-Anwendungen realisiert werden. Es exisitert ein integrierter Editor, der ähnlich einem 3D-Grafik Programm, dazu verwendet wird die entsprechende Umgebung zu gestalten. Die fertige Anwendung bzw. das Spiel können für unterschiedlichste Plattformen von Android bis hin zu Linux erzeugt werden. Diese Anwedungen sind unabhängig von der Unity Entwicklungsumgebung lauffähig. Es wird ein Kompilat mit Abhänigkeiten und Ressourcen erzeugt.

2.1.1 SciptEngine

Unity stellt eine ScripEngine zur Verfügung, mit deren Hilfe derer die Umgebung und die darin vorhandenen Objekte beinflusst werden kann. Als Sprache für die Skripte kommt C# zum Einsatz. Die Logik und der Ablauf der Ereignisse in der Anwendung werden durch Skripte festgelegt. Als Framework kommt das *Mono Framework* zum Einsatz, eine Open Source Implementierung von Microsofts *.NET Framework*.

2.1.2 Prefabs

Prefabs - PreFabricates, sind bereits vorgefertigte Objekte, die Instanziiert werden müssen. Dies kann auf verschiedenen Wegen geschehen, in diesem Projekt wurden die Prefabs im Unity Editor erzeugt und per Script instantiiert.

2.1.3 SteamVR Plugin für Unity

Es existiert ein Plugin für die HTC Vive, eine VR-Brille. Dieses Plugin ist über den in Unity integrierten Asset Store zu beziehen. Es beinhaltet unter anderem sogenannte

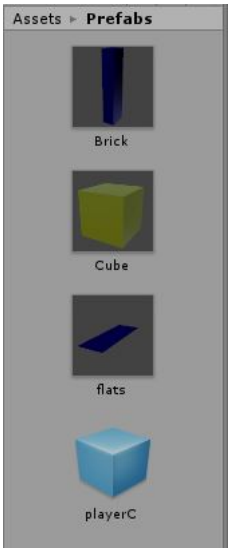

Abbildung 2.1: *Im Projekt verwendete Prefabs*

Prefabs, vorgefertigte Objekte, die in eine Szene eingesetzt werden können. Damit die Vive inklusive ihrer Controller funktioniert müssen die Prefabs [Steam VR] und [Camera Rig] in die Szene eingesetzt werden.

2.2 JetBrains Rider

Die Standart Entwicklungsumgebung für Unity ist *Microsoft Visual Studio 2017 in der Community Edition (kostenlos)*. Grundsätzlich ist jede IDE die C# unterstüzt geeignet. Visual Studio 2017 und JetBrains Rider bieten zudem die Möglichkeit, sich an den Unity Prozess anzuhängen und einen Debugger einzusetzen.
In diesem Projekt wurde Rider verwendet, da im Schwerpunkt auf Computern mit den Betriebssystem MacOS X entwickelt wurde und die Rider IDE in dieser Umgebung performanter und agiler ist.

2.3 HTC Vive

Das Zielmedium der Anwendung ist die HTC Vive, eine VR-Brille. Die Konfiguration an der Hochschule beinhaltet eine Brille, zwei Controller (einer Pro Hand) und die Sensoren, um die Position und Ausrichtung des Spielers im Raum zu erfassen. Der eingesetzte PC ist ein potenter Spielecomputer mit entsprechender Grafikhardware.

Kapitel 3

Szenen

Im vorliegenen Projekt wurde für jede Parameterkurve eine eigene Szene erstellt, damit die einzelnen Kurven gegeneinander abgegrenzt sind.

3.1 Circle

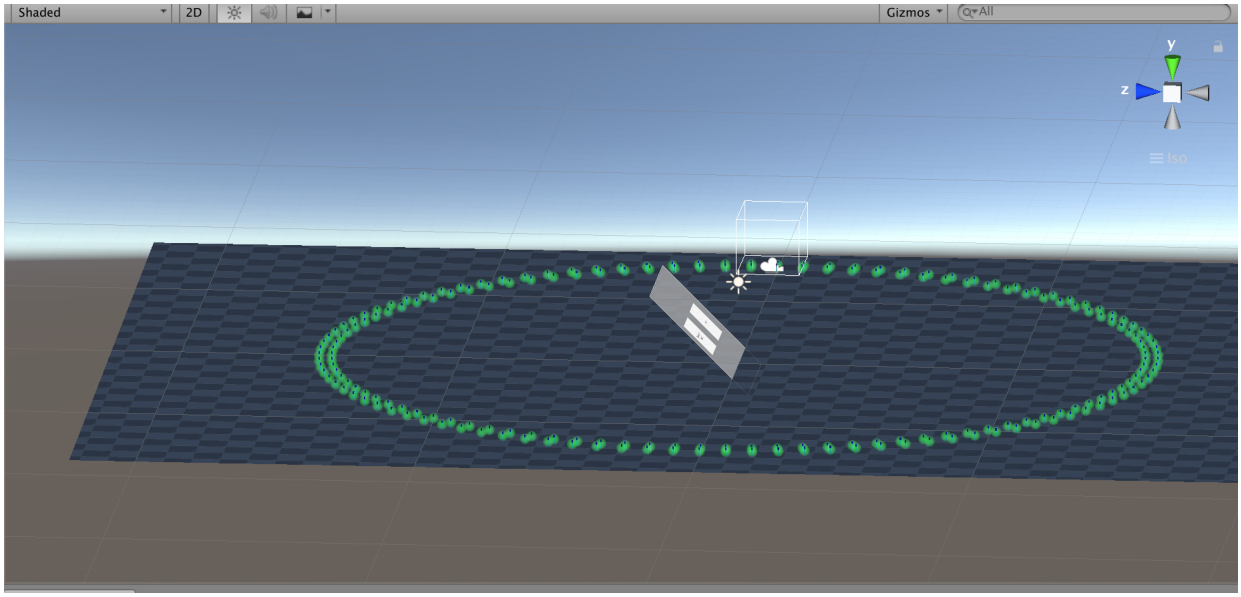

Abbildung 3.1: *Ansicht Circle Scene*

Die Szene enthält den einen Kreis, der beim Start erzeugt wird. Dieser wird mit konstanter Geschwindigkeit umlaufen. Der Kreis ist mit kleinen Pfosten markiert, die eine Gasse bilden, durch die sich der Spieler automatisch bewegt.

3.1.1 3-D Elemente

Plane: Die Grundfläche der Szene ist eine 4,5 x 3,5 Meter großes 3D-Objekt vom Typ *Plane*.
Capsule: Der Spieler wird mit Hilfe eines GameObjects in Form einer Capsule repräsentiert. Das Camera Rig ist mit Hilfe eines Scripts an dieses Objekt gebunden. Die Capsule hat die Markierung: Player.
Brick: Bricks sind Prefabs, die zur Laufzeit erzeugt werden. Ein einzelner Brick ist konfiguriert, ein Script instanziert Kopien davon.
PlayerC: PlayerC Objekte liegen ebenfalls als Prefab vor. Sie werden in Abhängigkeit der Bricks erzeugt und beschreiben die Positionen, die während der Pfadanimation verwendet werden. PlayerC sind auch GameObjects, haben aber keinen Mesh-Renderer also sind sie unsichtbar. Sie sind als "waypoint"getagged.
Canvas: Auf der Canvas befinden sich zwei Buttons, die dem Szenenwechsel dienen. Die Canvas befindet sich in der Mitte des Kreises. Zur Auswahl einer anderen Szene mit dem Laserpointer des linken Controllers auf den Button zeigen und das Touchpad des Controllers klicken.

3.1.2 Scripte

To do (1)

Instantiate.cs Dieses Script dient dazu, den Kreis und die PlayerC Punkte zu erzeugen. In der Awake()-Methode werden mittels einer For-Schleife die Bricks und die PLayerC Objekte erzeugt. Die Plane wird anhand ihres Namens „Boden“ gesucht. Relativ zum Boden wird die Y - Höhe der Bricks berechnet. Zusätzlich werden die Positionen der einzelnen PlayerC in ein Array geschrieben, dies dient der späteren Pfadanimation. Es befinden sich noch zwei weitere Funktionen im Script. Diese werden bei jedem Brick bzw. PlayerC aufgerufen. Die X-Koordinate einer Paramterkurve wird über die Funktion $f(t) = \cos(2 * \pi * t)$ definiert. Für Z gilt $g(t) = \sin(2 * \pi * t)$ analog. Das Parameterintervall ist [0,1]. In diesem Koordinatensystem beschreibt Y die Hochachse.
Movement.cs Dieses Script bewegt den Spieler auf dem Pfad. Dazu werden das Player-GameObject und alle waypoints in der Szene gesucht und adressierbar gemacht. Wenn die Liste nicht leer ist, dann wird der Spieler und dessen Ziel auf das erste PlayerC gesetzt. *Hinweis: Wenn die Liste leer ist, dann kommt es zu einer NullPointer Exception. Jedoch stürzt die Entwicklungsumgebung nicht ab, sondern sie verahrrt im Pause-Modus.*
In der FixedUpdate()-Methode wird nun die Sicht des Spielers auf den nächsten Wegpunkt ausgerichtet. Danach wird der Spieler auf dieses Ziel zubewegt. Sobald der

Spieler eine Mindestdistanz zu einem Wegpunkt unterschreitet, wird der Wegpunkt inkrementiert und der Spieler weiter bewegt.

CameraController.cs Der CamerController sorgt dafür, das die Kamera (in diesem Fall das SteamVR Camera Rig) an der Capsule haftet. Beim Instanzieren, wird der Spieler über den Tag „Player“ angesprochen und die Position der Camera auf die Position des Spielers gesetzt.

Um eine Isometrische oder Third-Person-View zu erzeugen müsste die Kamera einen Offset zur Position des Spielers haben. Dies hat sich im hier vorliegenden Kontext als nicht zweckmäßig erwiesen.

SteamVR Scripts Das SteamVR Plugin bietet bereits fertige Scripte zur Nutzung der Controller. In dieser Scene werden *SteamVR_Tracked_Controller* und *SteamVR_Laserpointer*.

GraphScene.cs Dieses Script ist dem Button, der zur Szene mit der Semikubischen Parabel führt angehängt. Es ruft den SceneManager von Unity auf und übergibt die gewünschte Szene an die Anwendung.

SpiralScene.cs Dieses Script ist dem Button, der zur Szene mit der Spirale führt angehängt. Es ruft den SceneManager von Unity auf und übergibt die gewünschte Szene an die Anwendung.

3.2 Spiral

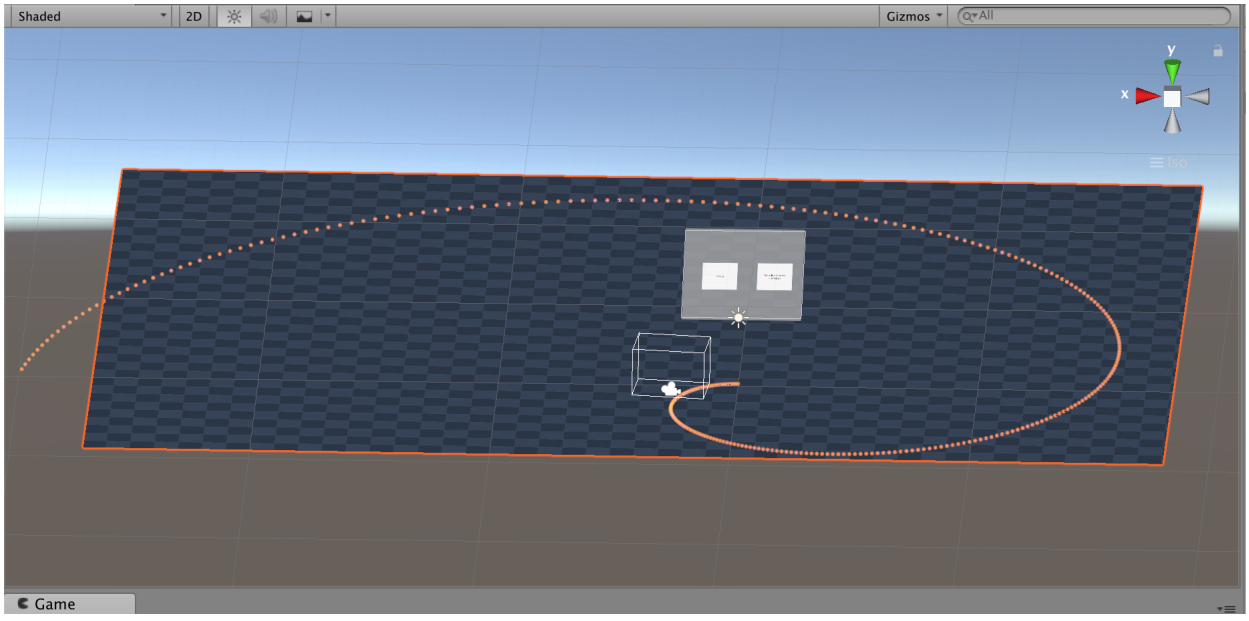

Abbildung 3.2: *Ansicht Spiral Scene*

Die Szene enthält den eine Spirale, die beim Start erzeugt wird. Diese wird mit kon-

stanter Geschindigkeit abgefahren. Die Spirale ist mit leuchtenden Quadraten auf dem Boden markiert. Bei Erreichen der letzten Koordinate bewegt sich der Spieler wieder in Richtung des Startpunktes.

3.2.1 3D - Elemente

Plane: Die Grundfläche der Szene ist eine rotierte 4,5 x 3,5 Meter großes 3D-Objekt vom Typ *Plane*.
Capsule: Der Spieler wird mit Hilfe eines GameObjects in Form einer Capsule repräsentiert. Das Camera Rig ist mit Hilfe eines Scripts an dieses Objekt gebunden. Die Capsule hat die Markierung: Player.
Brick: Bricks sind Prefabs, die zur Laufzeit erzeugt werden. Ein einzelner Brick ist konfiguriert, ein Script instanziert Kopien davon.
PlayerC: PlayerC Objekte liegen ebenfalls als Prefab vor. Sie werden in Abhängigkeit der Bricks erzeugt und beschreiben die Positionen, die während der Pfadanimation verwendet werden. PlayerC sind auch GameObjects, haben aber keinen Mesh-Renderer also sind sie unsichtbar. Sie sind als "waypoint"getagged.
Canvas: Auf der Canvas befinden sich zwei Buttons, die dem Szenenwechsel dienen. Die Canvas befindet sich in der Mitte des Kreises. Zur Auswahl einer anderen Szene mit dem Laserpointer des linken Controllers auf den Button zeigen und das Touchpad des Controllers klicken.

3.2.2 Scripts

Instantiate_Spiral.cs **To do** (2) Dieses Script dient dazu, die Spirale und die PlayerC Punkte zu erzeugen. In der Awake()-Methode werden mittels einer For-Schleife die Bricks und die PLayerC Objekte erzeugt. Die Plane wird anhand ihres Namens „Boden“ gesucht. Relativ zum Boden wird die Y - Höhe der Bricks berechnet. Dies geschieht analog zum Kreis.
Ein Vector3-Array wird mit den Koordinaten der PlayerC Objekte gefüllt, damit die Pfadanimation darüber durchgeführt werden kann.
Es befinden sich noch zwei weitere Funktionen im Script. Diese werden bei jedem Brick bzw. PlayerC aufgerufen. Die X.Koordinate einer Paramterkurve wird über die Funktion $f(t) = t * \cos(2\pi t)$ definiert. Für Z gilt $g(t) = t * \sin(2\pi t)$ analog. Das Parameterintervall ist [0,1]. **To do** (3) In diesem Koordinatensystem beschreibt Y die Hochachse. Die Rückgabewerte der Funktionen f und g sind auf 10% skaliert, damit die Spirale in den Bereich der Plane passt.

3.3 Graph

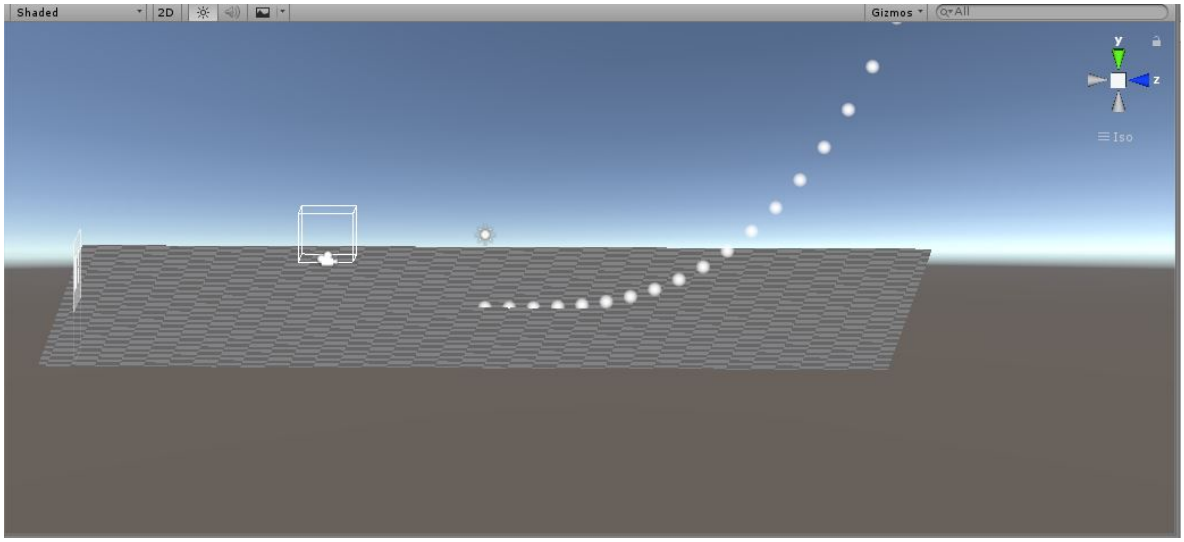

Abbildung 3.3: *Ansicht Graph Scene*

Die Szene enthält den ein Graphen, der beim Start erzeugt wird. Der Graph wird durch leuchtende Würfel dargestellt und ersteckt sich von (0,0,0) in „den Himmel“.
Eine Besonderheit dieser Szene ist die Bewegung des Spielers. Im Gegensatz zu den anderen beiden Szenen ist es dem Spieler überlassen, einen geeigneten Standort für die Betrachtung des Graphen zu wählen. Dies geschieht über Teleportation.
Mit dem linken Controller, der einen Laserstrahl emittiert wird das Ziel gewählt und mit einem Druck auf den Trigger wird ein Teleport ausgelöst.
To do (4)

3.3.1 3D - Elemente

Plane: Die Grundfläche der Szene ist eine rotierte 4,5 x 3,5 Meter großes 3D-Objekt vom Typ *Plane*.
Capsule: Der Spieler wird mit Hilfe eines GameObjects in Form einer Capsule repräsentiert. Das Camera Rig ist mit Hilfe eines Scripts an dieses Objekt gebunden. Die Capsule hat die Markierung: Player.
Cube: Cubes sind Prefabs, die zur Laufzeit erzeugt werden. Ein einzelner Cube ist konfiguriert, ein Script instanziert Kopien davon.
PlayerC: PlayerC Objekte liegen ebenfalls als Prefab vor. Sie werden in Abhängigkeit der Bricks erzeugt und beschreiben die Positionen, die während der Pfadanimation verwendet werden. PlayerC sind auch GameObjects, besitzen aber keinen Mesh-Renderer also sind sie unsichtbar. PlayerC Prefabs sind als "waypoint"getagged.

Canvas: Auf der Canvas befinden sich zwei Buttons, die dem Szenenwechsel dienen. Die Canvas befindet sich in der Mitte des Kreises. Zur Auswahl einer anderen Szene mit dem Laserpointer des linken Controllers auf den Button zeigen und das Touchpad des Controllers klicken.

3.3.2 Scripts

Instantiate_Curve.cs
Dieses Script dient dazu, den Graphen und die PlayerC Punkte zu erzeugen. In der Awake()-Methode werden diese mittels einer For-Schleife erzeugt.
Die Funktionen für f und g sind ebenfalls im Script. Diese werden bei jedem Brick bzw. PlayerC aufgerufen. Die X - Koordinate einer Parameterkurve wird über die Funktion $f(t) = t^2$ definiert. Für Z gilt $g(t) = t^3 - 3t$ analog.

Kapitel 4

Resümee

4.1 Diskussion

Die Darstellung von Parameterkurven ist in Unity gut möglich. Lediglich die Skalierung muss angepasst werden. Es gibt verschieden gut geeignete Parameterkurven. In diesem Projekt wurden ein Kreis, eine archimedische Spirale und eine Parabel gewählt. Der Kreis ist gut darstellbar, ebenso die Spirale. Eine Pfadanimation lässt sich relativ einfach implementieren.
Die Darstellung einer Parabel ist „instabil “. Das Intervall sollte geschickt ausgewählt werden. Im Allgemeinen lässt sich mit genug Zeit und Expertise ein Teil der mathematischen Konzepte modellieren bzw. darstellen. Der Spieler schlüpft in eine Art übergeordnete Rolle und kann die erzeugten Objekte frei, selbständig und wahlfrei betrachten.
Es stellte sich heraus, dass bei einer Pfadanimation am Kreis oder der Spirale die Darstellung und die Bewegung des Spielers intuitiv zweckmäßig implementiert wurden. Die Parabel hingegen stellt vor Probleme. Der Spieler würde sich ausschließlich nach oben bewegen.
Aus diesem Grund ist in der Szene der Parabel keine automatische Pfadanimation implementiert. Im Gegenteil, der Spieler ist in der Lage sich mittels Teleportation an eine beliebige Stelle auf der Plane der Szene zu bewegen. So kann die Form von einem beliebigen Punkt betrachtet werden.
Verscheidene Entscheidungen erwisen sich als unzweckmäßig:

1. Graph in XZ-Ebene darstellen.
   Bei der Darstellung der Parabel in der XZ-Ebene von Unity („etwa seitlich auf dem Boden liegend“) wurde deutlich, dass die Parabel zu schnell von der Ebene rutschte. Ihre Dimensionen wachsen sehr schnell.

Daraufhin wurde die Parabel senkrecht auf die Plane gestellt.

2. Eigenimplementierung eines Tracked_Object scripts.

   Die eigene Implementierung eines Scriptes zum verfolgen von Objekten endete in einer Sackgasse. Es wurde ein Script aus dem SteamVR Plugin verwendet.

4.2 Rückblick

- In der Nachbetrachtung wurde festgestellt, dass für dieses Thema zumindest Grundwissen über die Entwicklungsumgebung und Kentnisse in der Sprache C# vorausgesetzt werden sollten. Ist dies nicht der Fall, so ergeben sich große Schwierigkeiten bei der Umsetzung.

- Der Workload, vor allem im Bereich „Erlernen einer Programmiersprache in einem speziellen Kontext“, wurde stark überschätzt.

- Es wäre eleganter, wenn die Pfadgeneration mittels eines LineRenderers durchzuführen und darauf Meshes zu erzeugen. Dies würde helfen eine optisch ansprechendere Szene zu erstellen.

- Bei einer zukünftigen Anwendung würde ich die Szenen aus der VR-Sicht planen und früher mit der Vive testen.

- Bei der Wahl der IDE hätte ein früheres wechseln auf Rider die Entwicklung beschleunigt.

- Die Darstellung des Graphen in der entsprechenden Szene ist nicht gut gelungen, da die Dimensionen sehr schnell wachsen und eine entsprechde optische Metapher fehlt bzw. die Darstellung wenig detailliert ist.

Literaturverzeichnis

To do…

- ☐ 1 (p. 6): Scripts einzeln Beschreiben, pro Szene eine Liste der Verwendung
- ☐ 2 (p. 8): Spirale Beschreiben
- ☐ 3 (p. 8): Hier auch noch die 360 punkte erklären
- ☐ 4 (p. 9): Mglw Fahrstuhl einbauen