Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Parameterkurven haben Eigenschaften, die sich nur schlecht auf Papier bzw. zweidimensional darstellen lassen. In der Unity Umgebung besteht die Möglichkeit mittels Pfadanimation die Parameterkurven darzustellen. Diese Projektarbeit zeigt exemplarisch drei verschiedene Parameterkurven, die in der Virtuellen Realität dargestellt werden, um Eigenschaften der Kurven erlebbar zu machen.

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 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 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 Betriebsystem 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.

2.4 Beweisverfahren

2.5 Resolution

2.6 Hornklauseln

2.7 Berechenbarkeit und Komplexität

2.8 Anwendungen und Grenzen

Kapitel 3

Szenen

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

3.1 Circle

Die Szene enthält den einen Kreis, der beim Start erzeugt wird. Dieser wird mit konstanter Geschindigkeit umlaufen. Der Kreis ist mit kleinen Pfosten markiert, die eine Gasse bilden.

3.1.1 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

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 $f(t) = \cos(t)$ definiert. Für Z gilt $g(t) = \sin(t)$ analog. In diesem Koordinatensystem beschreibt Y die Hochachse.

3.2 Quantoren und Normalform

3.3 Resolution

3.4 Automatische Theorembeweiser

3.5 Anwendungen

Kapitel 4

Grenzen der Logik

4.1 Das Suchraumproblem

4.1.1 Problemstellung

Bei der Suche nach Beweisen existieren, bei fast jedem Schritt, bis zu unendlich viele Möglichkeiten Inferenzregeln anzuwenden. Nimmt man den Worst-Case an, so müssen alle Möglichkeiten ausprobiert werden. Dies führt zu einem Zeitaufwand, welcher diesen Prozess nicht sinnvoll umsetzbar macht.
Es gibt jedoch Möglichkeiten mit diesem extrem anwachsenden Suchraum umzugehen. So zeigt sich, dass Menschen, obwohl viel langsamer in der Ausführung von Inferenzen, durchaus schneller beim Lösen schwieriger Probleme sein können. Dies resultiert aus Faktoren wie Intuition, Lemmas (Hilfssätzen) und Erfahrung, durch welche der Suchraum extrem verkleinert werden kann und somit eine deutliche Zeitersparnis mit sich bringen.
Das Ziel muss es also sein diese Faktoren in irgendeiner Form auf die Maschine übertragen werden.

4.1.2 Lösungsansätze

Die Idee ist nun, die zuvor genannten Eigenschaften auf Maschinen zu übertragen. Dazu kommen verschiedene Ansätze in Frage. So können zum Beispiel Heuristiken integriert werden. Durch diese wird versucht die Intuition nachzubilden.
Um dies zu erreichen, wird maschinelles Lernen eingesetzt. Es werden aus vorangegangenen erfolgreichen Beweisen Klauselpaare als positiv oder negativ gespeichert und es wird versucht, aus diesen Trainingsdaten ein Programm zu erzeugen, welches Klauselpaare heuristisch bewerten kann.

Ein weiterer Ansatz ist, durch interaktive Systeme den Menschen bei der Beweisführung zu unterstützen. So bleibt die Kontrolle über die Beweisführung zwar beim Menschen, aber es können zum Beispiel Algebraprogramme eingesetzt werden. Diese können für den Menschen schwierige mathematische Vorgänge übernehmen ohne die intuitive Lenkung der Beweisführung durch den Menschen aufzugeben.

4.2 Entscheidbarkeit und Unvollständigkeit

Durch die Prädikatenlogik erster Stufe gibt es korrekte und vollständige Kalküle und Theorembeweiser. Man kann demnach bei einer wahren Aussage in endlicher Zeit Beweisen, dass sie tatsächlich wahr ist. Dies gilt jedoch nicht für unwahre Aussagen, da die Menge der allgemeingültigen Formeln der Prädikatenlogik erster Stufe halbentscheidbar ist.
Ist eine Aussage nicht allgemeingültig, so kann es sein, dass der Beweiser nicht hält. Dies ist unpraktisch, da für eine bereits als wahr bekannte Aussage kein Beweis mehr notwendig ist. Für eine möglicherweise unwahre jedoch schon. Die Prädikatenlogik erweist sich als zu mächtige Sprache, um noch entscheidbar zu sein.
Möchte man jedoch eine Logik höherer Stufe quantifizieren, so ergibt sich schnell das Problem, dass die Vollständigkeit einer Logik sofort verloren geht, wenn man sie auch nur minimal erweitert.
Kurt Gödel hat in diesem Kontext den Gödelschen Unvollständigkeitssatz bewiesen. Dieser besagt, dass jedes Axiomensystem für die Natürlichen Zahlen mit Addition und Multiplikation (die Arithmetik) unvollständig ist. Das heißt, es gibt in der Arithmetik wahre Aussagen, die nicht beweisbar sind [vgl. S.68].

4.3 Ein Beispiel

Ein fundamentales Problem der Logik und passende Lösungsansätze, zeigt das folgende Beispiel [vgl. S. 71,72]

Es sei gegeben:
1. Tweety ist ein Pinguin
2. Pinguine sind Vögel
3. Vögel können fliegen

In Prädikatenlogik 1 ergibt sich als Wissensbasis:
$pinguin(tweety)$
$pinguin(x) \Rightarrow vogel(x)$
$vogel(x) \Rightarrow fliegen(x)$

Es lässt sich nun $fliegen(tweety)$ ableiten, jedoch können Pinguine nicht fliegen. Demnach gilt:
$pinguin(x) \Rightarrow \neg fliegen(x)$

daraus kann $\neg fliegen(tweety)$abgeleitet werden, was jedoch im Widerspruch zu fliegen(tweety) steht. Hier erkennen wir die Eigenschaft der Monotonie. Die Definition der Monotonie besagt, dass eine Logik monoton ist, wenn für eine beliebige Wissensbasis WB und eine beliebige Formel φ die Menge der aus WB ableitbaren Formeln eine Teilmenge der aus WB $\bigcup \varphi$ ableitbaren Formeln ist [vgl. S. 70].
Wird die Formelmenge also erweitert, wächst die Menge der beweisbaren Aussagen monoton. Demnach führt eine Erweiterung der Wissensbasis nie zum Ziel. Es muss also die falsche Aussage 3, Vögel können fliegen, durch eine präzisere Aussage ersetzt werden. Die neue Aussage heißt nun "Vögel außer Pinguine können fliegen". Aus dieser neuen Wissensbasis ergeben sich neue Klauseln:
$pinguin(tweety)$
$pinguin(x) \Rightarrow vogel(x)$
$vogel(x) \bigwedge \neg pinguin(x) \Rightarrow fliegen(x)$
$pinguin(x) \Rightarrow \neg fliegen(x)$

Der vorher entstandene Widerspruch ist nun behoben. Jedoch bleibt das Problem der erweiterbaren Wissensbasis weiter bestehen. Erweitert man nun diese um einen weiteren Typ Vogel so lässt sich mit der neuen Wissensbasis keinerlei Aussage über die Flugeigenschaften der neuen Vogelart machen. Es muss zuerst definiert werden, dass die neue Vogelart kein Pinguin ist. Folgt man dieser Logik weiter, so muss für jede Vogelart welche fliegen kann definieren, dass sie keine Pinguine sind. Jedoch muss man dann noch für alle anderen nicht fliegenden Vögel ebenfalls die Ausnahmen vergeben. Eine Lösung dieses Problems ist die Einführung einer Default-Logik, welche Default-Regeln etabliert. In diesem Beispiel wäre "Vögel können fliegenßolch eine Regel. Nun müssten nur noch alle Ausnahmen definiert werden und alle anderen würden automatisch mit dem Default-Wert ausgestattet.

4.4 Modellierung von Unsicherheit

Wie in dem vorher angeführten Beispiel gezeigt, gibt es Logiken, welche nicht sinnvoll mit wahr/falsch modellierbar sind. Es bietet sich an mit Wahrscheinlichkeiten zu arbeiten. so können Aussagen mit einer Wahrscheinlichkeit versehen werden um Ausnahmen darzustellen. Um komplexe Anwendungen mit vielen Variablen zu modellieren können Bayes-Netze verwendet werden. Zuletzt wurde außerdem noch die Fuzzy-Logik entwickelt um unscharfe Variablen modellieren zu können.
Vergleicht man die verschiedenen Logikformalismen ergibt sich folgendes Bild:

Formalismus	Anzahl der Wahrheitswerte	Wahrscheinlichkeiten ausdrückbar
Aussagenlogik	2	nein
Fuzzy-Logik	∞	nein
diskrete Wahrscheinlichkeitslogik	n	ja
stetige Wahrscheinlichkeitslogik	∞	ja

Literaturverzeichnis