

Julian Hopp

**Ist es möglich eine realistische Simulation eines, durch Anwender-Interaktion beeinflussbares, Schwarmverhaltens in der Unity-Engine performant zu gestalten?**

GPD415

SAE Hamburg

Abgabe: 14.03.2016

Betreuender Fachlehrer: Fabio Anthony

(Wortanzahl)

# 1 Einleitung

"Mitte der 1980er Jahre entwickelt Craig Reynolds ein solches interaktives [Schwarmverhalten] Partikelsystem, nennt es aber - und in noch viel stärkerer Anlehnung an biologische Systeme - ganz anders. Reynolds ist ebenfalls Grafikdesigner und seinerzeit tätig für die Grafikabteilung der Firma Symbolics. Sein Schwarm-Animationsmoldell, das er unter dem Titel *Flocks, Herd, and Schools: A distributed behavioral model* veröffentlicht, klingt nicht nur wie ein Text aus dem Kontext der Verhaltensbiologie. Es wird auch in fast jedem Papier der späteren, computergestützten biologischen Schwarmforschung als eine Art >Urtext< zitiert." (Vehlken, 2012:315)

Auch wenn das von Craig Reynolds erstellte Schwarmverhalten nicht für die Spiele Industrie entwickelt würde, ist der Schwarm-Algorithmus für genau diese in vielen Situationen sehr nützlich.

"Allgemein bezeichnet man einen Schwarm als einen Zusammenschluss von Tieren in großer Zahl, ohne soziale Bindung oder Hierarchie. Der wohl wichtigste und bemerkenswerteste Punkt des Schwarmes ist die fehlende Hierarchie. Ein Schwarm besitzt keinen Anführer. Jedes Mitglied des Schwarmes ist absolut gleichrangig." (Metz, 2013:o.S.).

## 1.1 Themen Definition

"Schwarm: größere Anzahl sich [ungeordnet,] durcheinander wimmelnd zusammen fortbewegender gleichartiger Tiere, Menschen" (Dudenredaktion, 2006:913)

In dieser Arbeit werden die drei Grundlegenden "Steering Behaviours" begriffe von Craig Reynold beschrieben, "Separation", "Alignment" und "Cohesion", um eine natürliche Animation von Tierschwärmen in Unity zu simulieren. Anschließend wird in der Unity-Engine getestet ob dies Performant abläuft unter Anwender-Interaktion, desweiteren als Impact benannt.

## 1.2 These

**These: Ist es möglich eine realistische Simulation eines, durch Anwender-Interaktion beeinflussbares, Schwarmverhaltens in der Unity-Engine performant zu gestalten?**

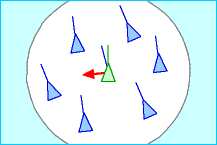
\*\*\* (muss überarbeitet werden) In dieser Facharbeit geht es darum, eine Anwender-Interaktion dazulegen, welches beispielsweise bei einem Hei Angriff auf einen Fischschwarm auftritt und die Reaktion des Schwarmes auf dieses. Der Schwerpunkt liegt darauf, dies Performant zu gestalten.

# 2. 1 Grundlagen des Flockings

In der Informatik beschreibt das Schwarmverhalten die Bewegung eines einzelnen Individuums, auch Boid oder autonomer Agent genannt, innerhalb eines Schwarms, sowie das daraus resultierende Verhalten aller Boids als Schwarm. (vgl. Buckland, 2005:118)

"Brian L. Partridge konnte bei Untersuchungen an Elritzen (ein kleiner Süßwasserfisch) zeigen, dass bei zwei Fischen stets einer führt und der anderen folgte) Der folgende passt in Tempo und Richtung dem Führer an, während der Führer in keiner Weise auf den anderen achtet. Dies mag wie ein Führungsprinzip aussehen, jedoch bricht es schon auseinander, sobald eine dritte Elritze hinzukommt. Jetzt richten sich die Tiere untereinander aus und einen Führer kann man nichtmehr ausmachen. Es entsteht das Schwarmprinzip: die Gesamtheit führt und jedes Individuum passt sich ihr an." (Kramper, 2010:2)

## 2.2 Alignment

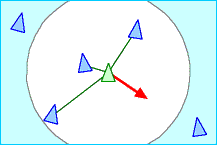


(Abbildung 1, Alignment)

Das Alignment ist dafür zuständig, dass die Boids sich in eine vorgegebene Richtung und an eine vorgegebene Geschwindigkeit orientieren und somit die Positionsänderung vorgenommen wird. Es wird beim Alignment dafür gesorgt, dass der individuelle Boid sich bei seinen Nachbarn Boids möglichst realistisch einreiht.

" Alignment is a behavior that causes a particular agent to line up with agents close by." (Pemmaraju, 2013:o.S.).

## 2.3 Separation

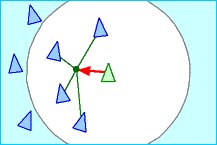


(Abbildung 2, Separation)

Die Separation sorgt dafür, dass die Boids untereinander einen vorgegebenen Abstand einhalten. Dadurch kann die Separation genutzt werden, eine Kollisionserkennung nicht notwendig zu machen, um so Performance zu sparen (siehe Abb.2).

"Separation is the behavior that causes an agent to steer away from all of its neighbors." (Pemmaraju, 2013:o.S.).

## 2.4 Cohesion

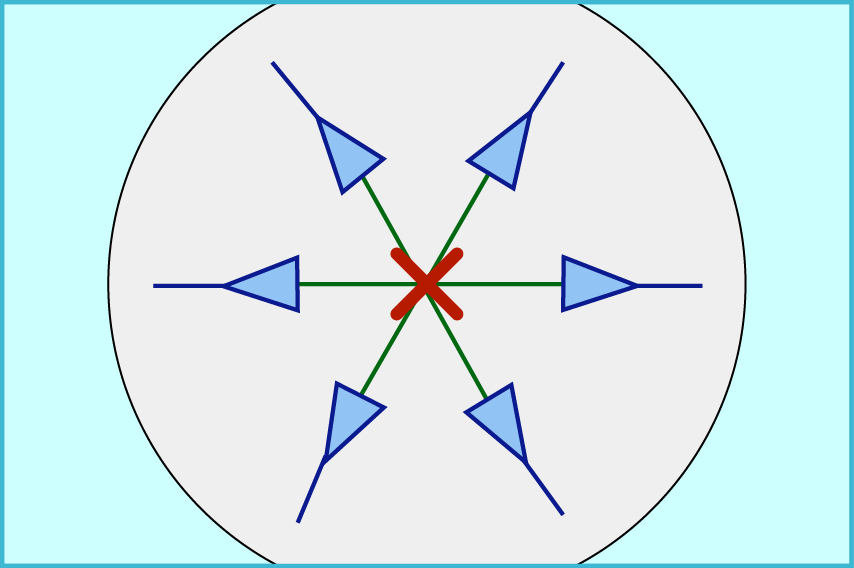


(Abbildung 3, Cohesion)

Die Cohesion wird verwendet, um zu verhindern, dass die Boids sich zu weit voneinander entfernen, so dass der Schwarm keine Boids verliert. Die Verbindung von Cohesion und Separation wirkt sich somit auf den Abstand zwischen den Boids aus und die Geschwindigkeit der Distanzveränderung zwischen den Boids im Schwarm (siehe Abb.3).

"Cohesion is a behavior that causes agents to steer towards the "center of mass" - that is, the average position of the agents within a certain radius." (Pemmaraju, 2013:o.S.).

## 2.5 Impact



(Abbildung 4, selbst erstellt)

Die Impact Simulation ist dafür verantwortlich, um Interaktion mit dem Schwarm realistisch zu gestalten. Dies würde unter anderem bei einem Haiangriff auf einen Fischschwarm zu finden sein (siehe Abb.4).

Für die Berechnung des Impacts helfen Tools wie etwa das Unity interne *NavMesh* (besonders im 2D bereicht) und die *Vecto*r Klassen, auf welche im Kapitel 4 noch genauer eingegangen wird.

# 3 Methodik

Bevor der Schwarm und die Performance getestet werden kann, muss untersucht werden, wie ein Schwarm auf einen Impact reagieren würde. In der Tierwelt ist dies besonders gut zu beobachten. Wenn zum Beispiel in einem Fischschwarm ein Fremdkörper, wie ein Hai oder ein Stein auf der Wasseroberfläche, auftaucht, zersprengt sich der Schwarm sofort. Er braucht dann einige Zeit, bis er sich wieder zusammen fügt.

"All the boids can be moving in one direction at one moment, and then the next moment the tip of the flock formation can turn and the rest of the flock will follow as a wave of turning boids propagates through the flock. Reynolds’ implementation is leaderless in that no one boid actually leads the flock" (Bourg, 2004:86)

## 3.1 Engine Auswahl

"Unity wird üblicherweise als Game-Engine (deutsch Spiel-Engine) bezeichnet, und das ist auch korrekt. Beschreibender wäre aber *Entwicklungswerkzeug zur Erstellung von 2D- und 3D-Spielen und -Anwendungen für eine Vielzahl unterschiedlicher Zielplattformen einschließlich Virtual-Reality-Umgebung.* Streng genommen umfasst die Game-Engine an sich vor allem die einzelnen für Spiele notwenigen Systeme (wie Grafik, Physik, Audio, Steuerung und Skripting) sowie teilweise auch weiterführende Systeme, beispielsweise Netzwerkunterstützung oder auch das Speichern von Spielständen. Bis auf Letzteres (das Speichern von Spielständen) hat Unity das auch alles mit dabei."(Chittesh, 2015:1)

Es wird die Unity Engine (Version 5.3.2) benutzt, da Unity durch diverse Möglichkeiten dem Benutzer das Implementieren erleichtert. Besonders erwähnenswert ist der Profiler, mit dem sich die Performance permanent über ein extra Fenster kontrollieren lässt (Unity, 2015:o.S.).

Desweiteren bietet Unity eine zuverlässige und offizielle Dokumentation \* ([http://docs.unity3d.com](http://docs.unity3d.com/))

Ein weiterer wichtiger Punkt ist die kostenfreie Nutzung von Unity 3D für unkommerzielle Projekte.

## 3.2 Programmiersprache

Um die Performance des Schwarms unter Anwender-Interaktion vergleichen und testen zu können, muss diese erst implementiert werden. Durch die eben beschriebene Engine-Auswahl, gibt es nun die Programmiersprachen Java, C#, UnityScript und Boo zum Implementieren des Schwarmverhaltens. Aus diversen Gründen wurde hier die Programmiersprache C# gewählt. Unter anderem weil C# fortlaufend weiterentwickelt wird, \*\*

## 3.3 Variablen Benennung

"Für die Benennung von Variablen sollten Sie sich eine bestimmte Konvention aneignen, um Unklarheiten hinsichtlich Ihrer selbst definierten Variablen von vorherein zu vermeiden" (Frischalowski, 2008:52)

Jedoch, trotz eigen angeeignetem Style, ist es wichtig die Common Language Infrastructure einzuhalten.

"ISO/IEC 23271:2003 defines the Common Language Infrastructure (CLI) in which applications written in multiple high-level languages may be executed in different system environments without the need to rewrite the applications to take into consideration the unique characteristics of those environments." (Iso, 2003:o.S.).

Es gibt diverse Gründe, warum es wichtig ist die CLI einzuhalten. Etwa, dass Variablen nicht nur durch unterschiedliche Groß- und Kleinschreibung deklariert werden, sondern auch mit verschiedenen Namen, da Case Insensitive Programmiersprachen diese andernfalls nicht unterscheiden können.

"Bezeichner die sich nur durch die Groß-/Kleinschreibung unterscheiden, können die Wiederverwendung von Klassen behindern wenn Sie in Ihren Anwendungen auch andere Sprachen wie Visual Basic nutzen, bei denen die Groß-/Kleinschreibung keine Rolle spielt."

(Frischalowski, 2008:52).

## 3.3 Entwicklungsumgebung

Um die C# Bibliotheken und Anwendungen zu erstellen, wird in der Theorie lediglich ein Compiler und ein Texteditor benötigt. Jedoch beschleunigt eine professionelle Entwicklungsumgebung das Entwicklungstempo und den erhöht Komfort.

"Visual Studio ist die Anwendung zur Entwicklung in C# oder auch in anderen .NET-sprachen, wie Managed C++ oder Visual Basic. Aber auch klassische Sprachen wie C oder C++ lassen sich sehr gut mit Visual Studio von Microsoft entwickeln." (Wurm, 2010:99).

Aus verschiedenen Gründen wurde die Microsoft Visual Studio Community 2015 Edition gewählt. Diese Entwicklungsumgebung wird ab Unity 5.2 kostenfrei mitgeliefert und stetig aktualisiert (Unity, 2015:o.S.).

Zusätzlich wird das Plug-In ReSharper genutzt, um die Naming Convention zu beschleunigen und zu vereinfachen.

## 3.4 Versionsverwaltung

Als professionelles Versionsmanagement mit Cloud Speicherung wurde Github gewählt, um das Schwarmverhalten auf verschiedenen Rechnern nicht nur zu synchronisieren, sondern auch testen zu können. Somit ist sichergestellt, dass jederzeit eine funktionsfähige Version besteht und diese, falls benötigt, nur noch aktualisiert werden muss. Als grafische Benutzeroberfläche zur Bedienung des Versionskontrollsystems wurde SourceTree genutzt.

# 4 Durchführung

Um die Performance des Algorithmus testen zu können, muss dieser Algorithmus vorerst in Unity Implementiert werden.

Hierfür bietet Unity viel Unterstützung, beispielsweise das *NavMesh* und den *NavMesh* Agent. Das *NavMesh* bietet unter anderem die Möglichkeit, eine bereits vorhandene Oberfläche für den *NavMesh* Agent begehbar zu gestalten. Dies ist gerade im 2D Bereich sehr von Vorteil. Die *NavMesh* Agent Klasse besitzt die Funktion *Destination*, die es dem Agent erlaubt, zum Zielobjekt zu laufen.

Auch nimmt Unity dem Benutzer eine Vielzahl an mathematischen Berechnungen ab. Besonders wichtig hierbei sind die Vektor-Klassen, vor allem im 3D Bereich. -\* (Warum ist es vor allem im 3D Bereich wichtig/gut? Wenn möglich -> Zitat)

Mit der Funktion *velocity* wird die Bewegungsgeschwindigkeit ermittelt. Diese Geschwindigkeit kann zur Überprüfung der erlaubten Maximalgeschwindigkeit des Boids genutzt werden. Die *velocity* wird gegen die vorher initialisierte Maximalgeschwindigkeit geprüft und falls überschritten, zurück auf die diese gedrosselt. Zum Drosseln muss die Funktion *Velocity* mathematisch mit der in der Funktion vorhandenen Funktion *normalized* normalisiert werden. Anschließend wird diese wieder, unter Berücksichtigung der *deltaTime*, mit der Maximalgeschwindigkeit multipliziert. Auf die *deltaTime* wird anschließend noch genauer eingegangen.

Noch zu überprüfen im Bezug auf die Geschwindigkeit, ist die ebenfalls vorher initialisierte Mindestgeschwindigkeit. Hierbei wird ganz ähnlich, wie im Absatz zuvor, zuerst überprüft, ob die Mindestgeschwindigkeit unterschritten wurde. Falls dies eintrifft, wird diese auf die Mindestgeschwindigkeit angehoben. Dazu wird wieder die Magnitude durch das mathematische Normalisieren ermittelt und mit der Mindestgeschwindigkeit multipliziert.

Das Unity Keyword *Time* ist ein Interface, welches genutzt wird, um Information bezüglich der Zeit zu erhalten. Die Funktion *deltaTime* dieses Interfaces sollte in Unity in der Regel mit allem multipliziert werden, dass einmal pro Frame etwas Addiert oder Subtrahiert, um die mathematische Berechnung Frameunabhängig macht. "Use this function to make your game frame rate independent." (Unity, 2015:o.S.)

Gerne genutzt im Bereich der *Time*-Klasse, ist die *Coroutine*. Die *Coroutine* Funktion ermöglicht es, die Ausführung im Skript solange zu verzögern, bis die Bedingung der *Coroutine* erfüllt ist. Diese Bedingung muss als *Yield* der *WaitForSeconds*-Funktion deklariert werden.

Falls die Interaktion des Schwarmes es benötigt, ist es möglich, über die *LookAt* Funktion die Boids in Richtung des Impacts gucken zu lassen. Das ermöglicht es, anschließend mit dem negativen lokalen X Wert der *Transform.forward* Funktion, die Bewegung einer Flucht zu simulieren. Ebenfalls ist es möglich mit der *LookAt* Funktion, die Kamera Dynamisch auf den Schwarmmittelpunkt zu richten.

Ein weiteres nützliches Tool im bezug auf die Performance ist das *Level of Detail* (LOD). Hierbei handelt es sich um eine Technik, die es ermöglicht Performance zu sparen indem Tries reduziert werden. Je weiter das Objekt von der aktuell Aktivierten Kamera entfernt ist, um so niedriger wird das *Level of Detail* gesetzt und umso mehr Triangels werden reduziert (vgl. Unity, 2015:o.S.).

Auf Abbildung 5 ist das Objekt mit voller Triangel Anzahl zu sehen (LOD 0). Je Höher die LOD, des so niedriger die Triangel Anzahl.



(Abbildung 5, Gebäude Mesh mit LOD der Stufe 0)



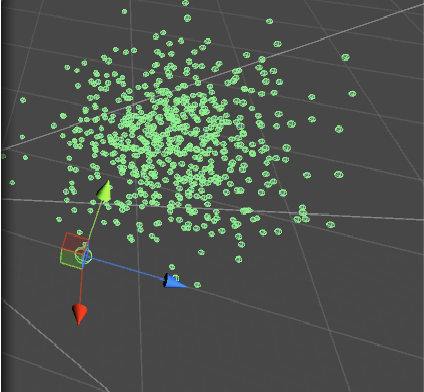
(Abbildung 6, Gebäude Mesh mit LOD der Stufe 1)

Bei Abbildung 6 wurden viele Triangels eingespart, deren Details bei einer weiter Distanzierten Kamera nichtmehr sichtbar wären.

Desweiteren kann bei einem hohen LOD der Collider deaktiviert werden, was sehr zugute der Performance sein kann.

## 4.1 Praxis Test

Ein Optischer Test zeigt, ob die Implementierung des Schwarm-Algorithmus erfolgreich zum gewünschten Ziel geführt hat.



(Abbildung 7, Schwarm in der Unity Scene)

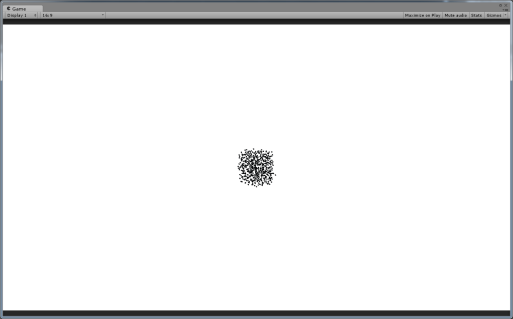
Auf Abbildung 7 zu sehen ist der Versuch der Implementierung eines möglichst realen Schwarms, mit einer Schwarmgröße von 600 Boids im 3D Bereich. Jeder Boid besitzt einen Collider Component, einen Rigidbody Component, einen Mesh Filter Component, einen Mesh Renderer Component, einen Shader Component und einen Transform Component.



(Abbildung 8, Unity Profiler mit 600 Boids)

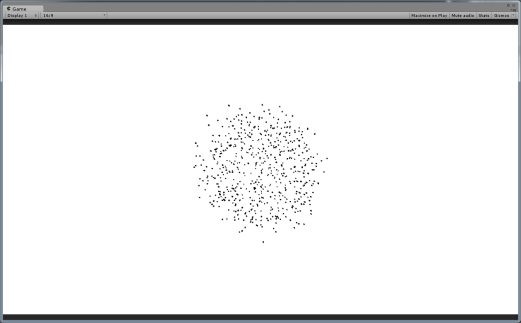
Abbildung 8 zeigt die 600 aktiven Rigidbodys der vorherigen Abbildung und deren Kontakten per Frame. Wie auf dem Diagramm zu sehen ist, gibt es durchschnittlich 10 bis 60 Kontakte unter allen Boids pro Frame. Wenn der Schwarm sich konstant in eine gleichbleibende Richtung bewegt, kommt es zu den zu sehenden durchschnittlichen 10 Kontakten, bei raschen Bewegungsänderungen erhöht sich der Kontakt per Frame auf 60.

Abbildung 4, aus dem Kapitel 2.5 Impact, stellt die Anwender-Interaktion dar. Der Schwarm simuliert auf Kommando ein Szenario, wie beispielsweise einen Fischschwarm, der von einem Hai angegriffen wird. Dieses Kommando kann unter anderem ein Maus-Klick in den Schwarm sein, oder das Auslösen eines Hotkeys.



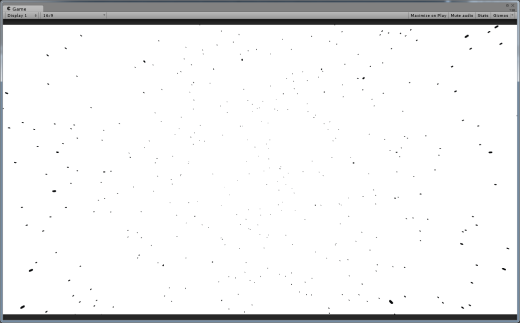
(Abbildung 9, Schwarm Status: Ausgangsposition)

Auf Abbildung 9 bleibt der Schwarm noch zusammen, da noch keine Interaktion passiert ist. Dies ist die Ausgangsposition mit 10 bis 60 Kontakten pro Frame.



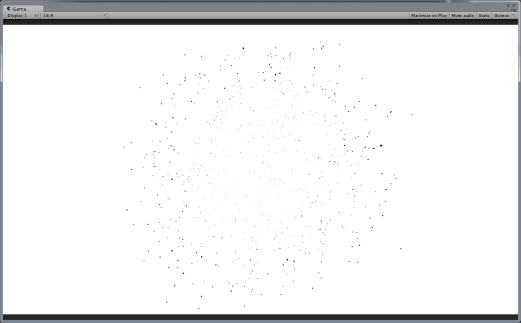
(Abbildung 10, Schwarm Status: Impact)

Die Anwender-Interaktion passiert: Der Schwarm entfernt sich vom Eintrittsort des Impacts (siehe Abb. 10). Ab hier gibt es keine Boid Kontakte pro Frame, bis hin zu Abbildung 13, da sich die Boids auch untereinander voneinander entfernen.



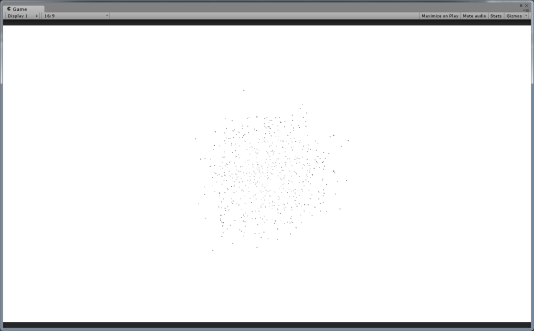
(Abbildung 11, Schwarm Status: Maximale Flucht Distanz erreicht)

Die individuellen Boids haben ihren maximalen Fluchtpunkt erreicht. Dies kann auf verschiedene Weisen passieren, zum Beispiel, wenn die Boids eine Abstandsberechnung zum Impact haben, oder über eine *Coroutine* (siehe Abb.11).



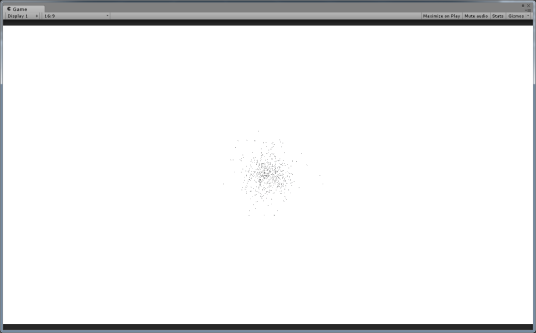
(Abbildung 12, Schwarm Status: Rückkehr)

Die Boids bewegen sich individuell erneut Richtung Schwarm-Mittelpunkt (siehe Abb. 12).



(Abbildung 13, Schwarm Status: Geschwindigkeit Drosslung)

Der Mittelpunkt des Schwarms wurde von den Boids fast erreicht (siehe Abb. 13). Dadurch kommt es zu einer Geschwindigkeit Drosslung, womit vermehrter Kontakt zwischen den Boids wieder möglich sind.



(Abbildung 14, Impact abgeschlossen)

Die Ausgangsposition ist fast erreicht und es pendeln sich wieder die 10 bis 60 Kontakte pro Frame ein. Die Anwender-Interaktion ist somit abgeschlossen (siehe Abb. 14).

-\*LOD anmachen und beschreiben http://docs.unity3d.com/Manual/LevelOfDetail.html

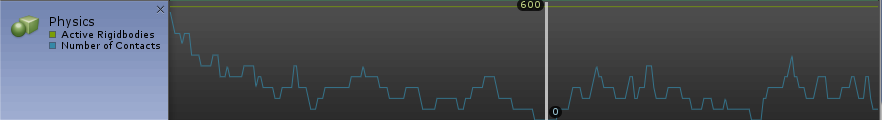
Zuletzt im Bereich des Praxis Tests sollten die Quality Settings nicht vernachlässigt werden. Unter *Menu: Edit > Project Settings > Quality* können diese eingestellt werden. In diesen Quality Settings kann geregelt werden, ob das Projekt zugunsten der Optik an Performance verlieren soll, oder andersrum.

" Unity allows you to set the level of graphical quality it will attempt to render. Generally speaking, quality comes at the expense of framerate and so it may be best not to aim for the highest quality on mobile devices or older hardware since it will have a detrimental effect on gameplay." (Unity, 2015:o.S.).

Viele Einstellungen sind eher individuell auf der Projekt zuzuschneiden, etwa das Anti-Aliasen (AA). Auch hier muss vorab eingeschätzt werden, ob der Visuelle unterschied die Performance Senkung rechtfertigt. Beim AA oder auch Kantenglättung genannt, berechnet die Grafikkarte ein höher aufgelöstes Bild und verrechnet dieses mit Ihrer wirklichen Auflösung. Dies erzeugt ein weicheren Übergang zwischen der Umgebung und dem Objekt.

" Anti aliasing improves the appearance of polygon edges, so they are not “jagged”, but smoothed out on the screen. However, it incurs a performance cost for the graphics card and uses more video memory (there’s no cost on the CPU though). The level of anti-aliasing determines how smooth polygon edges are (and how much video memory does it consume)." (Unity, 2015:o.S.).

# 5 Ergebnisse



(Abbildung 15, Unity Profiler bei Impact)

Nach Implementierung des Schwarm-Algorithmus ist zu sehen, dass bei der Anwender-Interaktion die Kollision pro Frame nahezu gleich null ist.

Daher muss Unity intern keine zusätzlichen physikalischen Berechnungen machen, was der Performance zugutekommt.

Wie in Kapitel 4.1 schon angesprochen, und auf Abbildung 8 zu sehen, sind im bei realistischen verhalten des Schwarmes durchschnittlich 10 bis 60 Kontakte pro Frame, bei aktiviertem collider component.

Diese Ergebnisse wurden auf verschiedenen Rechnern erzielt und überprüft.

## 5.1 Auswertung

Bei einer Schwarmgröße von 600 Boids mit den Unity Standard Objekten, gibt es zur Runtime keinerlei Veränderungen. Sichtbar wird dies erst bei höherer Anzahl an Boids. Bei hoher Boid Anzahl kann es beim Projekt Start zu Ladeverzögerungen kommen. Hierbei hilft jedoch das Unity interne *SceneManager.LoadSceneAsync*, oder das *Object Pooling*. Seid Unity 5.3 ersetzt das *SceneManager.LoadSceneAsync* das nun Obsolete *Application.LoadLevelAsync.* Mit der *SceneManager.LoadSceneAsync* Funktion ist es möglich, eine Unity Scene asynchron im Hintergrund laden zu lassen. (Unity, 2015:o.S.).

Es wird empfohlen beim *Instantiaten* der Boids zu achten, dass diese jeweils mit einem Abstand erstellt werden, um anfängliches physikalische Berechnungen durch Kollision verringert werden.

Beim abrupten Richtungswechsel des Schwarmes ist es möglich, dass es zu vermehrter Kollision kommt. Daher ist es empfehlenswert diesen Richtungswechsel möglichst langsam und geschmeidig zu gestalten.

## 5.2 Tabellen & Diagramme

# 6 Zusammenfassung

## 6.1 Fazit

\*\*\*\*\*\*

Da Unity durchaus eine Leistungsfähige Engine ist, und Schwarm-Algorithmus

Die Implementiert des Schwarm-Algorithmus war erfolgreich, und durch Optische Tests belegt.

Inhaltsverzeichnis

[1 Einleitung 2](#_Toc445601088)

[1.1 Themen Definition 2](#_Toc445601089)

[1.2 These 2](#_Toc445601090)

[2. 1 Grundlagen des Flockings 2](#_Toc445601091)

[2.2 Alignment 3](#_Toc445601092)

[2.3 Separation 3](#_Toc445601093)

[2.4 Cohesion 3](#_Toc445601094)

[2.5 Impact 4](#_Toc445601095)

[3 Methodik 4](#_Toc445601096)

[3.1 Engine Auswahl 4](#_Toc445601097)

[3.2 Programmiersprache 5](#_Toc445601098)

[3.3 Variablen Benennung 5](#_Toc445601099)

[3.3 Entwicklungsumgebung 5](#_Toc445601100)

[3.4 Versionsverwaltung 6](#_Toc445601101)

[4 Durchführung 6](#_Toc445601102)

[4.1 Praxis Test 8](#_Toc445601103)

[5 Ergebnisse 10](#_Toc445601104)

[5.1 Auswertung 11](#_Toc445601105)

[5.2 Tabellen & Diagramme 11](#_Toc445601106)

[6 Zusammenfassung 11](#_Toc445601107)

[6.1 Fazit 11](#_Toc445601108)

[Literaturverzeichnis: 12](#_Toc445601109)

[Abbildungsverzeichnis: 13](#_Toc445601110)

# Literaturverzeichnis:

Bourg, David M. / Glenn, Seeman AI for Game Developers. Sebastopol: O'Reilly, 2004

Buckland, Mat Programming Game AI by Example. Texas: Wordware Publishing, 2005

Chittesh, Jashan Das Unity-Buch. 2D- und 3D-Spiele entwickeln mit Unity 5. Heidelberg: pdunkt.verlag, 2015

Dudenredaktion Duden. Mannheim: Die Dudenredaktion, April 2006

Frischalowski, Dirk Visual c# 2008 Einstieg für Anspruchsvolle. München: Pearson Studium, 2008

Iso 2013: Common Language Infrastructure

Online unter:

http://www.iso.org/iso/catalogue\_detail?csnumber=36769

(Zuletzt überprüft am 14.03.2016)

Kramper, Wolfgang Simulation von Schwarmverhalten. Berlin: Mensch und buch Verlag, 2010

Unity 2015: Profiler Physics

Online unter:

http://docs.unity3d.com/Manual/ProfilerPhysics.html

(Zuletzt überprüft am 04.03.2016)

Unity 2015: Profiler Physics

Online unter:

http://docs.unity3d.com/Manual/ProfilerPhysics.html

(Zuletzt überprüft am 01.03.2016)

Unity 2015: Unity 5.2 – your gateway to Unity Services

Online unter:

http://blogs.unity3d.com/2015/09/08/unity-5-2-easy-access-to-unity- services/

(Zuletzt überprüft am 03.03.2016)

Unity 2015: Time.deltaTime

Online unter:

http://docs.unity3d.com/ScriptReference/Time-deltaTime.html

(Zuletzt überprüft am 01.03.2016)

Unity 2015: Upgrade Guide 5.3

Online unter:

http://docs.unity3d.com/Manual/UpgradeGuide53.html (Zuletzt überprüft am 10.03.2016)

Unity 2015: Level of Detail

Online unter:

http://docs.unity3d.com/Manual/LevelOfDetail.html (Zuletzt überprüft am 16.02.2016)

Unity 2015: Quality Settings

Online unter:

http://docs.unity3d.com/Manual/class-QualitySettings.html

(Zuletzt überprüft am 04.02.2016)

Metz, Felix 2013: Schwarmverhalten in der Informatik

Online unter:

<http://felixmetzfacharbeit.de/schwarmverhalten.html>

(Zuletzt überprüft am 13.03.2016)

Pemmaraju, Vijay 2013: Implementation. The Three Simple Rules of Flocking Behaviors: Alignment, Cohesion, and Separation

Online unter:

http://gamedevelopment.tutsplus.com/tutorials/the-three-simple-rules-of- flocking-behaviors-alignment-cohesion-and-separation--gamedev-3444

(Zuletzt überprüft am 24.02.2016)

Vehlken, Sebastian Zootechnologien. Zürich: Diaphanes, 2012

Wurm, Bernhard Programmieren lernen! Bonn: Galileo Press, 2010

# Abbildungsverzeichnis:

Abbildung 1-3 Seite ???

Alignment, Separation, Cohesion

Online unter:

<http://www.red3d.com/cwr/boids/>

(Zuletzt überprüft am 08.03.2016)

Abbildung 4 Seite ???

Impact

Erstellt von Katharina Schneiders

Abbildung 5-6 Seite ???

Gebäude Mesh mit LOD der Stufe 0

Online unter:

<http://docs.unity3d.com/Manual/LevelOfDetail.html>

(Zuletzt überprüft am 08.03.2016)

Abbildung 7 Seite ???

Schwarm in der Unity Scene

Selbst erstellt

Abbildung 8 Seite ???

Unity Profiler mit 600 Boids

Selbst erstellt

Abbildung 9-14 Seite ???

Schwarm Status

Selbst erstellt

Abbildung 15 Seite ???

Unity Profiler bei Impact

Selbst erstellt