



UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Generierung, Rendering und Simulation von Gras und Wiesen

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Nico Ell

Erstgutachter: Prof. Dr.-Ing. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Zweitgutachter: M. Sc. Bastian Kraye
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im März 2018

Aufgabenstellung für die Bachelorarbeit

Nico Ell

(Mat. Nr. 211 200 036)

Thema: Generierung, Rendering und Simulation von Gras und Wiesen

Gras spielt eine wichtige Rolle bei der Erzeugung von realistischen virtuellen Naturlandschaften. Aufgrund der hohen geometrischen Komplexität birgt die Darstellung von und die Interaktion mit Wiesen in Echtzeit verschiedene Herausforderungen. Häufig findet eine Approximation der feinen Geometrie durch bildbasierte Verfahren statt. Die sich rasant entwickelnde Hardware in der Computergraphik ermöglicht jedoch die Verwendung von rechenaufwändigeren, geometriebasierten Verfahren zur Darstellung einzelner Grashalme. Sowohl bildbasierte, als auch geometriebasierte Ansätze sind mit eigenen Möglichkeiten und Herausforderungen verbunden. Diese gilt es in dieser Arbeit zu untersuchen.

Ziel dieser Arbeit ist es, sich in bestehende Verfahren zur Darstellung von Gras und Wiesen einzuarbeiten. Auf Basis der Recherche soll schließlich ein Verfahren konzipiert und umgesetzt werden, welches abhängig von dem Detaillierungsgrad (Level of Detail) geometriebasierte und bildbasierte Techniken kombiniert. Schwerpunkt bildet dabei die Generierung verschiedener Detailstufen, sowie deren Darstellung und Simulation. Einhergehend soll das Verfahren auf die Grenzen und Möglichkeiten in der Echtzeitanwendung untersucht werden. Eine realitätsnahe Darstellung ist nur optional angedacht.

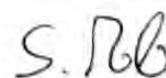
Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche und Analyse bestehender Gras Rendering Verfahren
2. Entscheidung für und Einarbeitung in ein grundlegendes Framework
3. Konzeption und Implementierung einer Echtzeitanwendung
 - Konzeption eines Verfahrens mit Augenmerk auf verschiedenen Detailstufen, deren Darstellung und Simulation
 - Implementation des konzipierten Verfahrens mit dem zuvor gewählten Framework
4. Evaluation der erreichten Darstellung und Performanz der Anwendung
5. Dokumentation

Koblenz, den 12.09.2017



Nico Ell



Prof. Dr. Stefan Müller

Danksagung

Zunächst möchte ich Herrn Prof. Dr.-Ing. Stefan Müller danken, der mit seinen interessant und anschaulich gestalteten Vorlesungen meine Begeisterung für die Computergraphik weiter gestärkt hat. Das kreative doch zugleich systematische Lösen spannender Probleme, mit dabei das Rendering von Gras, macht dieses Fachgebiet für mich ungemein faszinierend.

Besonderer Dank gilt meiner Mutter Gabi, die mich stets unterstützt hat und mir ihr Vertrauen schenkte sowie meiner Schwester Sandra, die mir auch in den stressigsten Zeiten ein Lächeln bereiten konnte. Auch meiner Freundin Nichakan gilt besonderer Dank, sowohl für das mehrmalige Korrekturlesen, als auch für die nötige Motivation und mentale Unterstützung.

Weiterhin danke ich meiner Tante Brigitte und meinem Freund Dominik für das Korrekturlesen und Günter Barth für das Bereitstellen von Naturfotografien.

Kurzfassung

Für das Erscheinungsbild fruchtbarer Landschaften ist Gras maßgeblich verantwortlich. Dessen Darstellung wird in Echtzeitanwendungen häufig durch die bildbasierte Approximation der hohen geometrischen Komplexität ermöglicht. Unlängst gewinnt die rechenintensive aber detailreiche geometrische Darstellung vieler einzelner Grashalme an Beliebtheit. Das Ziel dieser Bachelorarbeit ist die nahtlose Kombination von geometrischen Grashalmen und bildbasierten Billboard Repräsentationen. Dabei sind Grashalme mit benutzerdefinierter geometrischer Form die Basis für das Generieren der Texturen für Billboards. Beide Darstellungen verwenden ein einheitliches Modell für die Simulation. Somit können mehrere Windquellen und hunderte Kollisionsobjekte von beliebiger Form auf die gesamte Wiese in gleicher Weise einwirken. Neben einer Berücksichtigung des Speicherbedarfs findet eine Analyse von Performance-limitierenden Faktoren der Berechnungen auf der GPU statt.

Abstract

Grass is largely responsible for the look and feel of fertile landscapes. Its rendering in real-time applications is often made possible by using image-based approximations of the high geometric complexity. Recently, the computationally intensive yet high quality rendering of many single grass blades has become popular. The aim of this bachelor thesis is to seamlessly combine geometric grass blades with image-based billboard representations. Custom shaped grass blades are the basis for generating textures for billboards. Both representations use the same underlying simulation model and thus, multiple wind sources and hundreds of arbitrary shaped collision objects affect the entire meadow in the same way. Furthermore a low memory footprint is taken into account and an analysis of performance-limiting factors on GPU calculations is delivered.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation und Problematik	1
1.2	Aufbau der Arbeit	3
2	Grundlagen	4
2.1	Rendering Methoden	4
2.1.1	Geometriebasierte Methoden	5
2.1.2	Bildbasierte Methoden	8
2.1.3	Volumetrische Methoden	11
2.2	Level of Detail	12
2.2.1	Räumliche Aufteilung	14
2.3	Simulation	15
2.3.1	Prozeduraler Wind	15
2.3.2	Physikalisch-basierte Windsimulation	17
2.3.3	Kollisionen	18
2.4	Beleuchtung	21
2.5	Verteilung von Gras	22
2.6	Unity Engine	23
3	Umsetzung	25
3.1	Übersicht des Verfahrens	25
3.2	Kontrollierbarkeit von Gras und Wiese	26
3.3	Level of Detail	28
3.3.1	Unterteilung des Terrains	28
3.3.2	Kombination von Geometrie und Billboards	29
3.3.3	Tessellation	33
3.4	Gras Modellierung	34
3.4.1	Gras-Modell	34
3.4.2	Gras-Kontur	35
3.5	Simulation	36
3.5.1	Simulation in einer Textur	36
3.5.2	Ablauf der Simulation	37
3.5.3	Berechnung der Kontrollpunkte	37
3.5.4	Schwerkraft	39
3.5.5	Wind	39
3.5.6	Steifheit	41
3.5.7	Kollisionen	42
3.6	Rendering	45
3.6.1	Geometrische Grashalme	45
3.6.2	Blüten	48
3.6.3	Generierung von Billboard Texturen	50
3.6.4	Billboards	51

3.7	Beleuchtung	51
4	Ergebnisse und Bewertung	53
4.1	Performance	53
4.1.1	Beschreibung der Tests	53
4.1.2	Rendering Performance	55
4.1.3	Simulation Performance	56
4.2	Vergleich mit bestehenden Arbeiten	57
4.2.1	Geometrisches Gras	57
4.2.2	Level of Detail	58
4.2.3	Wind	60
4.2.4	Kollisionen	61
4.3	Verbesserungsmöglichkeiten	64
5	Fazit und Ausblick	65
	Glossar	A
	Abkürzungsverzeichnis	D
	Literatur	E

1 Einleitung

1.1 Motivation und Problematik

Mehrere Mars Expeditionen offenbarten einen Blick auf eine staubige und steinige Landschaft. Der Blick auf die Landmasse der Erde offenbart hingegen eine große Vielfalt an Landschaften, die abgesehen von Wüsten zumeist von üppiger Vegetation atemberaubender Artenvielfalt geprägt sind. Ob riesige Bäume, Büsche, Sträucher, kleine Gräser bis hin zu Moosen, alle haben die enorme Komplexität ihrer Form und Struktur und die schier unendliche Anzahl ihres Vorkommens gemein.

Diese und weitere Phänomene der Natur einzufangen und realitätsnah nachzubilden ist seit jeher ein bedeutendes und herausforderndes Ziel der Computergrafik. Mit dem *Texture Mapping* von Catmull [Cat74] können 1974 zweidimensionale Texturen auf dreidimensionale Objekte abgebildet werden. Erweiterungen wie *Bump Mapping* [Bli78] oder *Displacement Mapping* [Coo84] ermöglichen einige Jahre später auch eine stärkere räumliche Wirkung. Und mit *Perlin Noise* von Perlin [Per85] können 1985 die nötigen Texturen computergeneriert werden. Eine überzeugende Darstellung einer steinigen Marsoberfläche ist damit schon möglich. Für die Darstellung der detaillierten Geometrie von Blättern und Gräsern verwenden Reeves und Blau [RB85] ebenfalls im Jahr 1985 ein Partikelsystem, das u.a. im ersten Animationskurzfilm „The Adventures of André & Wally B.“ von „Pixar“ verwendet wurde. Abbildung 1 zeigt einen Ausschnitt des Kurzfilms, in dem Gras aus einzelnen Partikeln besteht, die entlang einer Flugbahn ausgerichtet werden.



Abb. 1: Szene im Wald aus „The Adventures of André & Wally B.“ [Smi07]

Beim Rendering von Animationsfilmen kann die Berechnung einzelner *Frames* mehrere Stunden oder Tage dauern. Die Darstellung in *Echtzeit* erfordert jedoch die Berechnung von mindestens 15 *Frames* pro Sekunde (FPS) [AHH08, p. 1] und wurde damit erst viel später und nur durch Approximationen möglich. So verwenden Perbet und Cani im Jahr 2001 nur in unmittelbarer Nähe des Betrachters sehr einfaches geometrisches Gras und approximieren dieses in größeren Entfernungen durch zweidimensionale Texturen [PC01]. Ein bemerkenswertes Ergebnis erreichen Bouatouch, Boulanger und Pattanaik mit einer ausgefeilten Kombination aus geometrischem Gras und einer volumetrischen Darstellung, bestehend aus mehreren vertikalen und horizontalen texturierten Schichten [BBP06]. Lange Zeit sind bildbasierte oder volumetrische Darstellungsmethoden vorherrschend und geometrisches Gras wird höchstens sparsam eingesetzt. Mit immer besserer Hardware gewinnt geometrisches Gras jedoch an Beliebtheit, sodass Fan et al. 2015 eine ganze Wiese bestehend aus Millionen einzelner Grashalme darstellen können [Fan+15].

Neben der Darstellungsweise spielen auch andere Faktoren eine Rolle, denn erst eine realitätsnahe Beleuchtung und das Wirken von Kräften durch Wind oder Kollisionen lassen eine Wiese lebendig wirken. Sogar der frische Geruch von Gras hat einen positiven Effekt auf das Realitätsempfinden virtueller Graslandschaften [Brk+09]. Anfangs wird auf eine korrekte Beleuchtung und eine Kollisionsbehandlung verzichtet und Wind wird durch einfache Animationen dargestellt. Auch hier verschieben sich die Grenzen des technisch Machbaren und Verfahren zur dynamischen Beleuchtung [Bou08] oder der Kollisionserkennung mit bildbasiertem Gras [ORK09] und geometrischem Gras [Fan+15] werden vorgestellt. Im Jahr 2016 setzen Lee et al. schon auf eine Flüssigkeitssimulation [Lee+16] zur Simulation vom geometrischem Gras, das im Wind weht.

Doch eine physikalisch korrekte Simulation und die geometrische Darstellung von Gras bleibt weiterhin rechenintensiv. Beim praktischen Einsatz in Echtzeitanwendungen muss zudem Rechenleistung für andere Systeme übrig bleiben. Eine Approximation der Geometrie ist deshalb durchaus sinnvoll, denn in größeren Entfernungen werden einzelne Grashalme so klein, dass es nicht nötig ist, diese Millionen Mal zu rendern [ZLZ09]. Das wird auch von Autoren der Arbeiten als zukünftige Arbeit erwähnt, die nur geometrisches Gras einsetzen [Fan+15; JW17].

Die Kombination von geometrischem Gras in der Nähe des Betrachters mit bildbasiertem Gras in der Ferne ist mit besonderen Herausforderungen verbunden, denn ein Übergang soll möglichst nicht sichtbar sein. Dazu ist es nicht nur nötig, die verschiedenen Darstellungen weich zu überblenden, diese sollen auch das gleiche Gras anzeigen [PC01] und den gleichen Einfluss durch Kräfte aufweisen. Hier setzt die vorliegende Arbeit an und präsentiert eine Analyse relevanter Vorgehensweisen in der Literatur, beschreibt die Umsetzung eines Verfahrens zur Kombination geometrischer

und bildbasierter Darstellungsmethoden und diskutiert die visuelle Qualität und Performance der erreichten Ergebnisse.

1.2 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen anhand von Verfahren aus der Literatur vorgestellt. Zunächst wird auf verschiedene Rendering Methoden und auf damit verbundene Vor- und Nachteile eingegangen. Abschnitt 2.2 befasst sich mit Möglichkeiten zur Kombination der Methoden und der räumlichen Aufteilung eines Terrains. Die Unterschiede zwischen physikalischen Simulationen und prozeduralen Approximationen, die Klärung der Terminologie sowie konkrete Beschreibungen von Wind- und Kollisionssimulationen finden sich in Abschnitt 2.3. Die realistische Beleuchtung von Gras und Wiesen hat in der vorliegenden Arbeit nur sekundäre Bedeutung, einige einfache Vorgehensweisen werden in Abschnitt 2.4 angeschnitten. Zuletzt werden Möglichkeiten zur benutzerdefinierten Verteilung von Gras in Abschnitt 2.5 aufgezeigt und das verwendete *Framework* vorgestellt.

Kapitel 3 präsentiert das entwickelte Verfahren und beginnt mit einer Übersicht des Gesamtsystems in Abschnitt 3.1. Anschließend werden die einzelnen Abläufe im Detail vorgestellt. Verschiedene Methoden zur Kontrolle des Aussehens und der Ausbreitung von Gras sind in Abschnitt 3.2 beschrieben. Der folgende Abschnitt 3.3 zum *Level of Detail*(LOD)-System stellt einen Schwerpunkt der Arbeit dar und behandelt neben der Unterteilung des Terrains die nötigen Schritte zur nahtlosen Kombination von geometrischem und bildbasierten Gras. Eine Technik zum benutzerdefinierten Formen des geometrischen Grasses sowie das zentrale Modell zur Simulation aller Grasdarstellungen ist in Abschnitt 3.4 zu finden. Ein weiterer Schwerpunkt ist der Abschnitt 3.5 zur Simulation, die neben einer flexiblen Kollisionsbehandlung auch das Einwirken von Wind, Schwerkraft und einer Erholungswirkung auf das Gras-Modell berücksichtigt. Weiterhin wird eine Maßnahme zur Reduktion des Speicherbedarfs vorgestellt. Auch das Rendering ist zentraler Bestandteil der Arbeit und wird in Abschnitt 3.6 für alle Grasdarstellungen beschrieben. Dabei inbegriffen ist das Generieren der *Billboard* Texturen und eine zusätzliche Methode zum Darstellen von Blüten. Ein einfaches Beleuchtungsmodell wird in Abschnitt 3.7 beschrieben.

Die erreichten Ergebnisse werden in Kapitel 4 durch eine tiefgehende Analyse der Performance und den visuellen Vergleich mit bestehenden Arbeiten bewertet. Wichtige Erkenntnisse zur Steigerung der Performance und der Darstellungsqualität werden im Abschnitt 4.3 festgehalten. Das abschließende Fazit fasst die Forschungsergebnisse zusammen und liefert einen Ausblick auf zukünftige Arbeiten.

2 Grundlagen

Im Folgenden werden erforderliche Grundlagen für die Entwicklung der angestrebten Grassimulation erläutert. Dabei werden unterschiedliche Vorgehensweisen aus der Literatur vorgestellt und damit eventuell verbundene Vor- und Nachteile aufgezeigt.

2.1 Rendering Methoden

In der Literatur finden sich unterschiedliche Methoden zur Darstellung von Gras. Fast alle lassen sich in drei Kategorien aufteilen, die in den folgenden zugehörigen Abschnitten genauer betrachtet werden: geometriebasierte, bildbasierte und volumetrische Rendering Methoden [BBP06]. Diese werden durchaus miteinander kombiniert, um verschiedene Detailstufen bereitzustellen (siehe Abschnitt 2.2). Manche Autoren unterscheiden nicht weiter zwischen bildbasierten und volumetrischen Methoden (wie z. B. in [DZJ06; JW17]), sondern zählen auch letztere zu den bildbasierten Methoden. In der vorliegenden Arbeit wird die getrennte Betrachtung wie nach Decaudin und Neyret [DN04] und Bouatouch, Boulanger und Pattanaik [BBP06; Bou08] verwendet. Danach liegt der Unterschied darin, dass bei volumetrischen Methoden eine dreidimensionale Menge an Daten entweder mittels 3D-Texturen oder mehrerer 2D-Texturen abgebildet wird, wohingegen bildbasierte Methoden eine oder mehrere 2D-Texturen verwenden, denen jedoch keine dreidimensionale Menge zugrunde liegt.

Eine andere Möglichkeit, die nicht in diese Kategorisierung passt, stellt bspw. die „Screen-Space Grass“ Methode dar [Pan14]. Dabei wird Gras in einem *Post-Processing* Schritt (dt. Nachbearbeitungs-Schritt) durch vertikales Verwischen von Rauschen erzeugt. Jedoch stößt die Methode schnell an ihre Grenzen, da sie nur bei Blickrichtung gen Horizont überzeugende Ergebnisse liefert.

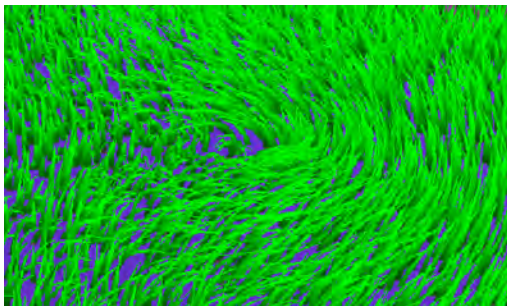
	[JW17]	[Lee+16]	[Fan+15]	[JW13]	[HRS13]	[CJ10]	[ORK09]	[HWJ07]	[BBP06]	[Wan+05]	[SKP05]	[Pe104]	[BLH02]	[PC01]
Renderingverfahren														
Geometrisch	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓
Bildbasiert	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✓	✓	✗	✓
Volumetrisch	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗

Tabelle 1: Übersicht verwendeter Rendering Methoden ausgewählter Arbeiten von 2001 bis 2017.

2.1.1 Geometriebasierte Methoden

Beim geometriebasierten Rendering von Gras wird dieses tatsächlich durch einzelne geometrische Objekte dargestellt. Durch die gezielte Modellierung, Texturierung und Beleuchtung einzelner Grashalme [BBP06] sowie der Möglichkeit Kräfteinwirkungen auf jeden Halm einzeln anzuwenden [JW17], ermöglicht diese Methode eine sehr präzise Darstellung von Gras. Das ist gewiss offensichtlich, da Gras in der Realität nun einmal aus einzelnen Grashalmen besteht. Die wesentlichen Schwierigkeiten der Methode gründen auf der enormen Menge an einzelnen Grashalmen, die in der Natur etwa auf einer Wiese wachsen und alle gerendert, modelliert und optional animiert werden müssen [Fan+15]. Für die Modellierung bieten sich prozedurale Methoden an, sodass mit möglichst wenig Aufwand detaillierte Grasoberflächen entstehen [Bou08].

Dennoch lässt sich in Tabelle 1 ein Trend hin zu geometriebasiertem Gras erkennen, welcher auf die Leistungssteigerung moderner Grafikkarten zurückzuführen ist. 2006 schreiben Bouatouch, Boulanger und Pattanaik, dass die verfügbare Rechenleistung noch nicht ausreicht, um Millionen einzelner Grashalme in Echtzeit darzustellen [BBP06]. Spätestens im Jahr 2015 gilt dies nicht mehr, wie Fan et al. mit dem Rendering von über einer Million Grashalme zeigen [Fan+15].



(a) Grashalme unter dem Einfluss von zirkulierendem Wind [PC01]



(b) Einzelne Grashalme in einer Wiese [Wan+05]

Abb. 2: Zwei frühe Beispiele zu geometriebasiertem Rendering von Gras.

Bereits 2001 verwenden Perbet und Cani [PC01] einfache 3D Geometrie zur Darstellung von Gras (siehe Abb. 2a), allerdings nur unmittelbar in Kameranähe. Einzelne Grashalme werden aus Ketten von Liniensegmenten konstruiert und als Grundlage für das Generieren von 2D-Texturen für weitere Detailstufen verwendet. Mögliche Haltungen der Grashalme in verschiedenen Neigungen werden in einer Physiksimulation vorberechnet und je nach Beeinflussung durch Wind angezeigt. Nach Kenntnis des Autors ist dies die erste Verwendung von dreidimensionalem geometrischem Gras im Echtzeit Rendering in der Literatur.

Wang et al. [Wan+05] nutzen ausschließlich ein geometrisches Gras Modell mit verschiedenen Detailstufen. Der Grashalm besteht aus mehreren Vierecken, die entlang von Skelett-Linien expandiert werden. Dieses Skelett wird zur Animation vom Gras im Wind verwendet (siehe Abb. 2b). Je weiter ein Grashalm von der Kamera entfernt ist, desto weniger Vierecke werden zu dessen Darstellung verwendet, bis er schließlich aus lediglich einem einzelnen Viereck besteht.



(a) Geometrisches Gras Modell von Bouatouch, Boulanger und Pattanaik [BBP06]



(b) Gras dargestellt mithilfe von Tessellation und Instanced-Rendering [JW13]

Abb. 3: Zwei weitere Beispiele zu geometriebasiertem Rendering von Gras.

Bouatouch, Boulanger und Pattanaik [BBP06] setzen auf eine Kombination geometrischer und volumetrischer Methoden. Ihre geometrischen Grashalme (siehe Abb. 3a) werden durch aneinandergereihte Vierecke modelliert, welche beidseitig gerendert werden. Die gewünschte Form entlang der Wuchsrichtung wird durch das mehrmalige Auswerten der Flugbahn eines Partikelsystems definiert, welches durch die Anziehungskraft beeinflusst wird. Eine halbtransparente Textur eines Grashalms wird auf die Geometrie projiziert, wobei der Alpha Kanal den Umriss des Grashalms definiert.

Mit dem Aufkommen neuer Funktionalitäten in den Standards von Grafikbibliotheken wie *Geometry-Shader*, Tessellation oder *Instanced-Rendering* boten sich auch neue Möglichkeiten für das Rendering von Gras.

Zur Modellierung von Grashalmen verwenden Jahrman und Wimmer [JW13] die Tessellation, um ein Viereck abhängig von der Entfernung zur Kamera in weitere Vierecke zu unterteilen. Die Form wird dabei durch eine Bézierkurve mit drei Kontrollpunkten bestimmt, welche an den relativen Positionen der unterteilten Vierecke ausgewertet wird. Ein Vorteil bei ihrer Methode ist, dass die Normale einfach berechnet werden kann. Wie bei vorangegangenen Methoden wird bei der Texturierung ein Alpha Kanal als Maske genutzt. Um weniger Speicherplatz für die Geometrie zu belegen, wird nur ein einzelner Ausschnitt (*Patch*) an geometrischen Daten erstellt und mittels *Instanced-Rendering* wiederholt gerendert. Das Ergebnis ist in Abb. 3a zu sehen.



(a) Detailliertes Gras von Fan et al. [Fan+15] (b) Darstellung von gezackten Löwenzahn Blättern, Auszug aus [JW17]

Abb. 4: Grafiken zu aktuellen geometrischen Methoden aus der Literatur.

Fan et al. [Fan+15] nutzen ebenfalls *Instanced-Rendering*, gehen bei der Generierung der Geometrie jedoch anders vor. Sie erzeugen in einem Vorverarbeitungsschritt eine Liste mit Geometrie bestehend aus mehreren vier-eckigen Segmenten für 16384 Grashalme und weisen einzelnen *Tiles* (hier quadratische Flächen in einem Gitter) eine Teilmenge dieser Liste zu. Im *Vertex-Shader* werden die Eckpunkte (engl. *vertices*) an die entsprechende Position der jeweiligen Gitterfläche verschoben. Dabei werden die Eckpunkte der einzelnen Segmente so ausgeweitet, dass sich der Grashalm nach oben hin verjüngt und kein Alpha Kanal als Maske benötigt wird. Dadurch entspricht die Geometrie tatsächlich der sichtbaren Form des Grasses, wie es in Abb. 4a zu sehen ist.

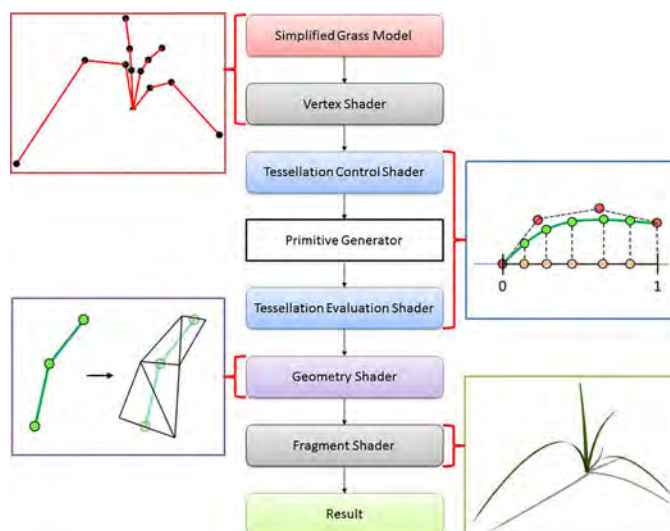


Abb. 5: Verwendung der Renderpipeline zur Generierung von Gras Geometrie [Lee+16]

In der Arbeit von Lee et al. [Lee+16] kommt eine Kombination aus Tessellation und *Geometry-Shader* zum Einsatz, um aus einem vereinfachten Gras Modell einen Grashalm herzustellen. Dieser wird modelliert durch vier Eckpunkte, welche als Kontrollpunkte einer Bézierkurve dienen. In der Tessellation-Phase werden die einzelnen Kurvensegmente je nach Entfernung zur Kamera unterteilt. Der *Geometry-Shader* erstellt durch Expandieren der Liniensegmente in Vierecke die endgültige Geometrie des Grashalms. In Abb. 5 ist die gesamte Pipeline dargestellt.

Die aktuellste betrachtete Arbeit von Jahrman und Wimmer [JW17] erweitert deren bereits vorgestellten Ansatz von 2013 [JW13]. Zur Definition der Form des Grashalms wird wieder eine quadratische Bézierkurve mit drei Kontrollpunkten herangezogen. Als Eingabe für die Tessellation wird anstelle eines Vierecks lediglich ein einzelner Punkt verwendet. Der *Primitive-Generator* erzeugt ein je nach Entfernung zur Kamera mehr oder weniger häufig unterteiltes Viereck, dessen Eckpunktkoordinaten entlang der Wuchsrichtung des Grashalms zur Auswertung der Bézierkurve verwendet werden. Um auf den Alpha Kanal als Maske verzichten zu können, wird eine prozedurale Methode unter Verwendung von mathematischen Funktionen vorgestellt, mithilfe derer die Kontur des Grashalms bestimmt wird. Neben einfachen Formen wird auch eine Funktion zur Darstellung von einem gezackten Löwenzahn Blatt vorgestellt (zu sehen in Abb. 4b), die für verschiedene Detailstufen entwickelt wurde. Anstatt auf *Instanced-Rendering* zu setzen, werden verschiedene *Culling* Verfahren in einem *Compute-Shader* angewendet, um die Anzahl darzustellender Grashalme zu verringern und damit die nötige Performanz zur Berechnung in Echtzeit zu erreichen.

Während in einigen vorgestellten Arbeiten bereits verschiedene Rendering Methoden kombiniert werden, wird bei anderen nur geometrisches Gras eingesetzt. Jedoch erwähnen manche Autoren als weiterführende Arbeit eine mögliche Erweiterung um bildbasierte Methoden [Fan+15; JW17].

2.1.2 Bildbasierte Methoden

Bei bildbasierten Methoden ist die nötige Rechenleistung beim Rendering von Objekten unabhängig von deren geometrischen Komplexität [DZJ06]. Folglich ermöglichen diese übertragen auf Gras, viele Grashalme mit nur wenigen Polygonen darzustellen [Pel04]. Die einfachste Möglichkeit stellt das Darstellen einer flachen Gras-Textur auf dem Boden dar. Dies bietet zwar keinerlei Bewegungsparallaxe, doch wird es auch in aktuellen Arbeiten genutzt, da es schwierig ist, Gras dicht genug darzustellen, um den Boden nicht mehr hindurchsehen zu können [JW13].

Perbet und Cani [PC01] generieren mithilfe des bereits vorgestellten geometrischen Grasses 2D-Texturen, welche auf mehrere vertikal aneinandergereihter Polygone abgebildet werden. Diese werden reihenweise an-

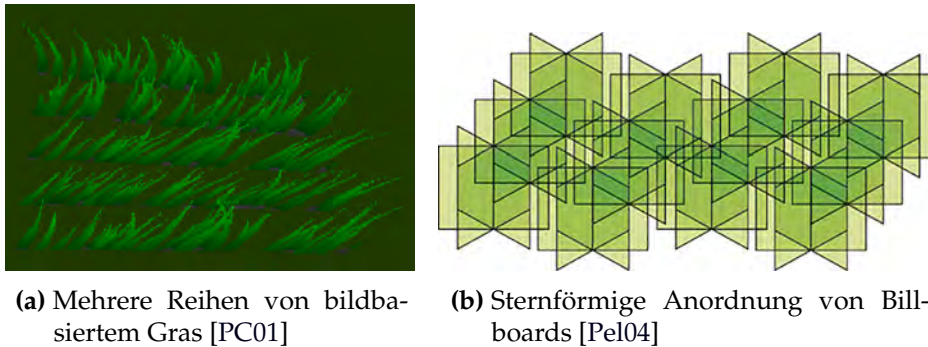


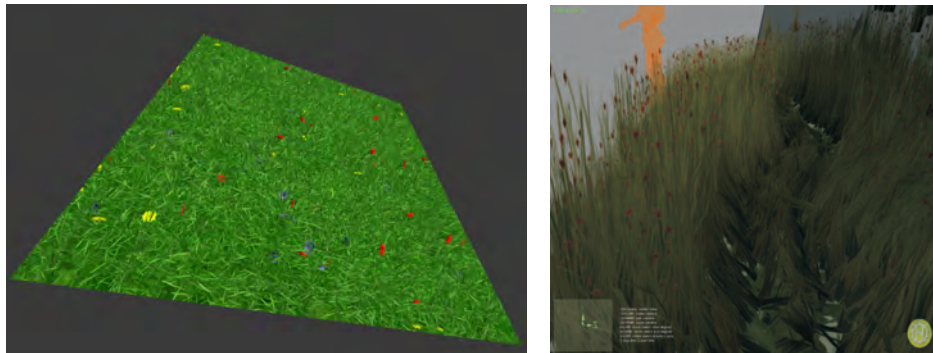
Abb. 6: Grafiken zu frühen bildbasierten Rendering Methoden von Gras

geordnet und nehmen eine von zwei möglichen, zueinander orthogonalen Orientierungen an. Es wird diejenige Orientierung gewählt, die möglichst nicht parallel zur Blickrichtung verläuft. Wie in Abb. 6a gut zu erkennen ist, offenbart sich die Reihenstruktur jedoch bei der Draufsicht. Weiter entfernt wird eine flache Gras-Textur auf das Terrain projiziert. Um den Übergang zwischen den Stufen weniger offensichtlich zu machen, wird die Textur in der Ferne nicht direkt auf dem Terrain, sondern mit einem Abstand entsprechend der durchschnittlichen Höhe des Grases abgebildet. Zu dem Verfahren gilt es anzumerken, dass die Einordnung, ob es sich um eine bildbasierte oder volumetrische Methode handelt, schwierig ist und in der Literatur unterschiedlich vorgenommen wird. Das liegt daran, dass die Abbildung der 2D-Texturen auf die Polygone wie bei anderen bildbasierten Verfahren funktioniert, die Texturen jedoch dynamisch unter Berücksichtigung der 3D-Geometrie erzeugt werden.

Um eine bessere Darstellungsqualität aus verschiedenen Blickrichtungen zu erreichen, schlägt Pelzer [Pel04] eine sternförmige Gruppierung von Billboards vor (siehe Abb. 6b). Eine Anordnung in Reihen sollte nach Pelzer nur bei automatisierter Kamerafahrt oder sehr weit entferntem Gras in Betracht gezogen werden.

Ein weiterer Ansatz ist die Verwendung einer „Bidirektionalen Textur Funktion“ wie etwa von Shah, Kontinnen und Pattanaik [SKP05]. Dabei wird vorab eine Menge von Bildern gerendert, die Gras aus unterschiedlichen Blickrichtungen und Lichtbedingungen zeigen. Bei Shah, Kontinnen und Pattanaik sind es 81 Blickrichtungen mit jeweils 21 verschiedenen Beleuchtungen. Zur Verringerung des enormen Speicherbedarfs werden die Bilder mit der „Principal Component Analysis“ komprimiert. Im *Fragment-Shader* wird die passende Farbe für den Pixel anhand verschiedener Parameter, wie z. B. Blick- und Beleuchtungsrichtung, rekonstruiert.

Habel, Wimmer und Jeschke [HWJ07] verwenden in ihrem Verfahren ausschließlich den *Fragment-Shader*, um auf einer bestehenden Geometrie ein vertikales Gitter aus „virtuellen Billboards“ zu erzeugen und in diesem



(a) Gras Rendering ausschließlich im *Fragment-Shader* [HWJ07]

(b) Billboard Darstellung von Gras mit Kollisionsbehandlung [ORK09]

Abb. 7: Zwei bildbasierte Methoden mit unterschiedlicher Billboard Darstellung

mittels *Raytracing* die Pixelfarbe zu bestimmen. Auf den Billboards wird dabei eine Gras-Textur mit Alpha Kanal als Maske projiziert. Als Vorteil geben Habel, Wimmer und Jeschke die einfache Integration in bestehende Systeme an. Ähnlich wie bei der Arbeit von Perbet und Cani wird die gleichmäßige Anordnung bei steilen Kamerawinkeln sichtbar. Auch durch eine zusätzliche horizontale Grasschicht kann diese nicht gänzlich versteckt werden (siehe Abb. 7a).

Orthmann, Rezk-Salama und Kolb [ORK09] generieren und animieren Billboards prozedural mithilfe mehrerer *Geometry-Shader*. In einem Vorverarbeitungsschritt werden diese von der *Graphics Processing Unit* (GPU) erstellt, wobei jedes als Punkt-Primitiv durch die Renderpipeline gereicht wird und eine Orientierung, Position, den Kollisionszustand sowie einen Index zugewiesen bekommt. Letzterer wird verwendet, um die Textur des Billboards in einem 2D-Textur Array zu adressieren. Es wird ein Konzept zur Kollisionsbehandlung vorgestellt, bei dem besonders darauf geachtet wird, die Billboards und damit auch die Gras-Textur nicht stark zu verzerren (siehe Abb. 7b). Die endgültige Geometrie eines Billboards kann unter Krafteinwirkung aus mehreren Vierecken bestehen, allerdings ohne Gewährleisten der Längenerhaltung. Weiterhin wird *Alpha to Coverage* verwendet, um Antialiasing bei halbtransparenten Texturen ohne Tiefensortierung zu ermöglichen.

Auch Chen und Johan [CJ10] verwenden Billboards mit unterschiedlich detaillierten Repräsentationen, wählen und erzeugen diese jedoch je nach Eigenschaft des Grases vorab. Solche für kurzes oder steifes Gras bestehen aus nur einem Segment, während Billboards für langes oder weiches Gras aus mehreren Segmenten bestehen.

Eine ähnliche Methode der prozeduralen Generierung im *Geometry-Shader* verwenden Hempe, Rossmann und Sondermann [HRS13]. Auch hier

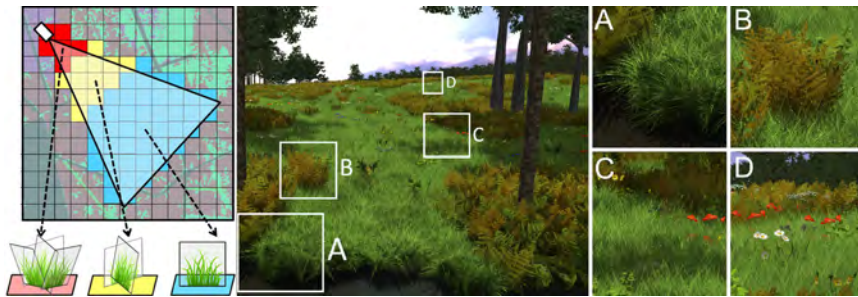


Abb. 8: Unterschiedliche Anordnung von Billboards je nach Entfernung zur Kamera [HRS13]

wird jedes Gras-Objekt als Punkt-Primitiv durch die Renderpipeline erreicht. Anders als bei Orthmann, Rezk-Salama und Kolb [ORK09] bestehen Billboards immer nur aus einem Viereck, Gras-Objekte werden je nach Entfernung zur Kamera jedoch unterschiedlich dargestellt. Wie in Abb. 8 illustriert, werden nahe der Kamera vier gekreuzte Billboards mit unterschiedlichen Winkeln, bei mittlerer Entfernung zwei gekreuzte und bei größerer Entfernung der Kamera zugewandte Billboards verwendet.

2.1.3 Volumetrische Methoden

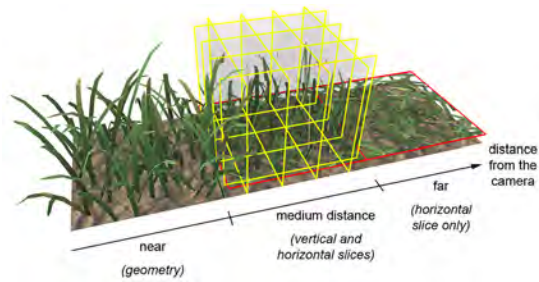
Im Gegensatz zu den flachen Billboards bieten volumetrische Methoden eine perfekte Parallaxe [DN04]. Sie werden 1989 erstmals zum Darstellen von Fell [KK89], was einige Parallelen zur Grasdarstellung aufweist, mittels Raytracing verwendet. Es sind weniger volumetrische Methoden zum Gras Rendering in der Literatur zu finden, als etwa geometrische oder bildbasierte Methoden. Ein möglicher Grund dafür könnte in der Arbeit von Orthmann, Rezk-Salama und Kolb [ORK09] zu finden sein. Danach sei die Kollisionserkennung und -behandlung mit volumetrischen Darstellungen „awkward to handle“ [ORK09, p. 65] (zu Deutsch etwa: schwierig zu handhaben). Ein anderer Grund könnte nach Auffassung des Autors der vorliegenden Arbeit die durch *Geometry-Shader* und Tessellation einfacher gewordene prozedurale Generierung von Billboard-Geometrie sein.

Bakay, Lalonde und Heidrich [BLH02] beschreiben in ihrer Arbeit ein Verfahren, in dem das Terrain mehrmals entlang der Oberflächennormale extrudiert wird, wodurch mehrere Schichten unterschiedlicher Höhe entstehen. Beim Rendern (von unten nach oben) wird eine Textur ausgewertet, in deren Alpha Kanal die Höhe des Grases an der jeweiligen Position gespeichert ist. Nur wenn die Höhe der Schicht kleiner als die Höhe des Grases ist, wird an der Stelle die in der Textur gespeicherte Farbe angezeigt. So entstehen je nach Anzahl an Schichten, mehr oder weniger detaillierte Grashalme (siehe Abb. 9a).

Von Bouatouch, Boulanger und Pattanaik [BBP06] werden neben der



(a) Volumetrische Grasdarstellung mit 16 Schichten [BLH02]



(b) Kombination von geometrischen und volumetrischen Rendering Methoden [BBP06]

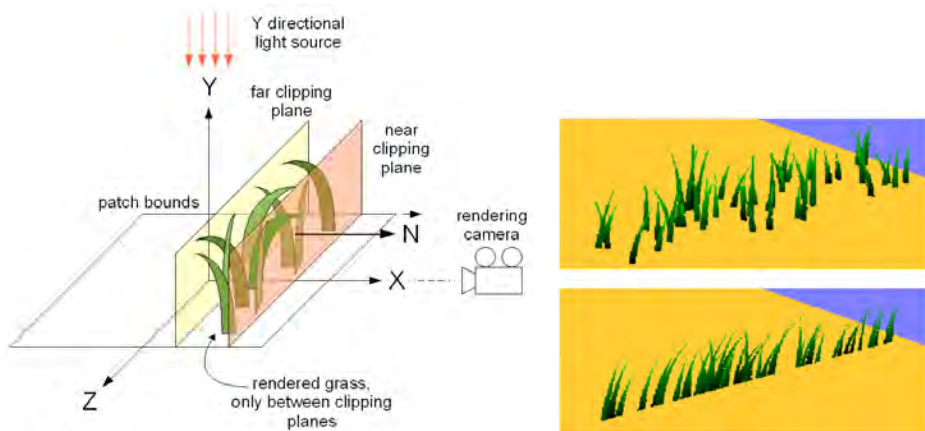
Abb. 9: Unterschiedliche volumetrische Darstellungen von Gras

geometrischen Methode auch zwei volumetrische Methoden für Detailstufen weiter entfernt von der Kamera verwendet (siehe Abb. 9b). Nach dem geometrischen Gras werden zunächst mehrere vertikale, entlang von drei orthogonalen Achsen ausgerichteten Schichten mit einer zum Terrain parallel verlaufenden Schicht kombiniert. Auf diese wird mittels Bidirektionaler Textur Funktion, abhängig von Blick- und Lichtrichtung, Gras projiziert. Sehr weit von der Kamera entfernt wird nur noch die zum Terrain parallel verlaufende Schicht verwendet.

2.2 Level of Detail

Das Verwenden eines LOD Systems ist unabdingbar, wenn Gras in großen Mengen über weite Strecken verteilt und in Echtzeit realitätsnah dargestellt werden soll. In Kameranähe ist es wünschenswert, die Geometrie einzelner Grashalme zu erkennen, um die volle Parallaxe zu ermöglichen. In größerer Entfernung ist es hingegen nicht nötig, Millionen von Grashalmen einzeln zu rendern [ZLZ09]. Viele Grashalme bedecken dann einen einzelnen Pixel [BBP06], weshalb ausschließlich geometrische Methoden verwendende Verfahren oft Probleme mit Aliasing lösen müssen (wie etwa [JW13; JW17]).

Das Verwenden verschiedener Rendering Methoden für verschiedene Entfernungen liegt folglich nahe. Die unterschiedlichen Methoden wurden im Abschnitt 2.1 bereits vorgestellt und ein Überblick der verwendeten Kombinationen lässt sich Tabelle 1 entnehmen. Entscheidend bei jeder Kombination ist, dass ein weicher Übergang zwischen den einzelnen Detailstufen geschaffen wird. Wie Perbet und Cani bereits 2001 erkennen, ist die Grundlage dafür die Fähigkeit, in verschiedenen Detailstufen das gleiche Gras anzuzeigen [PC01]. Im Folgenden wird deshalb betrachtet, wie bestehende Arbeiten verschiedene LOD Repräsentationen generieren, deren möglichst nahtlosen Übergang realisieren und wie die räumliche Aufteilung gehandhabt wird.



(a) Erstellen einer Gras-Textur mit einer orthographischen Kamera [BBP06]

(b) Geometrische und zugehörige bildbasierte Ansicht von Gras [PC01]

Abb. 10: Geometrisches Gras als Grundlage für niedrigere Detailstufen

In den beiden Arbeiten [PC01; BBP06] wird das geometrische Gras zum Generieren von Texturen für niedrige Detailstufen verwendet. Es wird mit einer orthographischen Kamera in eine Textur gerendert, wobei die *Near* und *Far Clipping Planes* (dt. vordere und hintere Schnittebenen) der Kamera so gewählt werden, dass genau der Bereich zwischen zwei Gras-Schichten abgebildet wird (siehe Abb. 10a). Perbet und Cani nutzen dieses System gleichzeitig zum nahtlosen Überblenden. Sie speichern die relativen Positionen der Grashalme in den generierten Texturen, um im Übergang der beiden Darstellungen das 3D-Gras an die entsprechenden Koordinaten des Grashalms in der Textur zu verschieben (siehe Abb. 10b). Dabei wird die Wurzel des Grashalms bewegt, was nach jedoch kaum auffalle [PC01]. Bouatouch, Boulanger und Pattanaik [BBP06] erreichen ein äußerst überzeugendes Resultat (siehe Abb. 11), indem sie im Bereich der Überblendung die Dichte des Grases einer Detailstufe verringern, gleichzeitig die Dichte der anderen Detailstufen erhöhen und zusätzlich die Transparenz vom Gras auf die Dichte der Detailstufe abstimmen.

In [JW13] wird ab einer bestimmten Distanz kein geometrisches Gras mehr angezeigt. Um eine sichtbare harte Kante zu vermeiden, wird das Gras in der Ferne langsam verkleinert.

Bei geometrischen Methoden findet teilweise eine Abstufung der Details innerhalb einer Detailstufe statt. In [JW13; Lee+16; JW17] wird hierzu der Tessellations-Faktor und damit die Anzahl an Unterteilungen der Gras-Geometrie abhängig von der Distanz zwischen Grashalm und Kamera variiert.

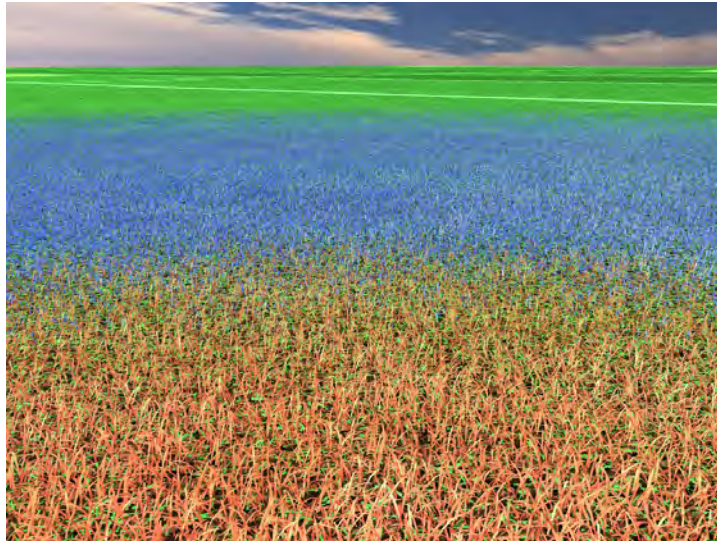


Abb. 11: Verdeutlichung der verschiedenen Detailstufen und deren Übergang durch Einfärbung [BBP06]

2.2.1 Räumliche Aufteilung

Eng verbunden mit LODs sind räumliche Datenstrukturen (engl. *spatial data structures*) und *Frustum Culling*, um nicht sichtbare Bereiche möglichst schnell und effizient auszusortieren. Zumeist wird das Terrain durch ein quadratisches Gitter aufgeteilt. Die dadurch entstehenden Bereiche, im Folgenden genannt *Tiles* (dt. Kacheln), können durch *Bounding Volumes* umschlossen werden. In [JW13] wird aus *Bounding Boxes* rekursiv eine Hierarchie erstellt, indem immer zwei *Bounding Boxes* zu einer nächst größeren kombiniert werden, bis das gesamte Grasfeld von einer einzelnen *Bounding Box* umschlossen ist. Bouatouch, Boulanger und Pattanaik [BBP06] kombinieren *Bounding Spheres* in einer „Quadtree“ Datenstruktur. Hierbei werden die *Bounding Spheres* der *Tiles* als Blätter eines Baums verwendet. Knoten vereinen vier benachbarte *Bounding Spheres* der Ebene darunter bis hin zur Wurzel. Orthmann, Rezk-Salama und Kolb verwenden eine Erweiterung der *Quadtree* Datenstruktur ins Dreidimensionale mit *Bounding Boxes*: die „Octree“ Datenstruktur [ORK09].

Bei all diesen Verfahren wird in jedem *Frame* getestet, ob die *Bounding Volumes* der Hierarchie das Kamera *Frustum* schneiden und damit sichtbar ist. Beginnend mit der obersten Ebene wird bei erfolgreichem Test die nächst tiefere Ebene behandelt [JW13]. Neben der Anwendung zum Aus-sortieren nicht sichtbarer Geometrie werden *Bounding Volumes* auch bei der Kollisionsbehandlung genutzt.

2.3 Simulation

Eine Simulation von Gras und Wiesen kann vielfältig ausfallen. Die biologische Entwicklung von Pflanzen, etwa durch *L-Systeme* [PL90], oder die Ausbreitung und Verteilung von Pflanzen in einem Ökosystem [Deu+98], ist nicht Teil dieser Arbeit. Betrachtet wird die Reaktion von Gras auf Kollisionen und Wind.

In frühen vorgestellten Arbeiten wird nur eine sehr einfache Repräsentation von Wind verwendet, denn die Herausforderung lag in der Darstellungsqualität von Gras. Erst später werden auch Kollisionen behandelt oder aufwendige Verfahren wie etwa Flüssigkeitssimulationen verwendet. Grundlegend kann zwischen physikalisch-basierter Simulation und prozeduralen Approximationen unterschieden werden [Bou08]. In der vorliegenden Arbeit wird unter dem Begriff der Simulation sowohl die physikalisch-basierte Simulation betrachtet, als auch die prozedurale Approximation von physikalischen Phänomenen.

Zur Simulation wird ein zu simulierendes Modell benötigt. Deshalb muss das Gras-Modell nicht nur eine effiziente Darstellung ermöglichen, sondern auch für die Simulation und ggf. die Synthese von Geometrie geeignet sein – und das im Optimalfall für verschiedene Detailstufen. So sind häufig nicht alle Parameter eines Modells simuliert und werden anderweitig kontrolliert, z. B. die Verteilung von Gras (siehe Abschnitt 2.5).

2.3.1 Prozeduraler Wind

Perbet und Cani [PC01] verwenden ein Konzept mit „Wind-Primitiven“ als Sender und mit den Gras-Modellen verknüpften Empfängern. In einer animierten 2D-Textur (zu sehen in Abb. 12a) werden Art und räumliche Ausdehnung der Wind-Primitive als Maske gespeichert. Die Haltung und Neigung von Gras, dessen Empfänger innerhalb einer Maske liegt, wird entsprechend der durch das Wind-Primitiv beschriebenen Funktion der Zeit animiert. Genauer handelt es sich um einen Vektor, der die Neigungsrichtung beschreibt und dessen Betrag als Index zur Wahl einer vorberechneten Haltung dient. Es werden Wind-Primitive für einen Wirbelwind, Windstoß und Lufthauch mit verschiedenen Prioritäten definiert, da nur ein Primitiv zur gleichen Zeit aktiv sein kann. Wichtig an dem Verfahren ist, dass die Wind Primitive direkt die Reaktion von Gras auf den Wind definieren und nicht etwa die Wirkung von Wind auf ein Vektorfeld.

Wenn nicht auf vorberechnete Daten zurückgegriffen wird, muss beachtet werden, dass Gras unter einer Kräfteinwirkung nicht seine Länge verändert. Bei dem volumetrischen Verfahren aus [BLH02] wird zur Wind Animation beim Extrudieren der Schichten neben der Normale zusätzlich ein Wind Vektor berücksichtigt. Dieser wird zunächst auf die Normale projiziert, um nur den Windeinfluss der senkrecht zur Normale verläuft zu

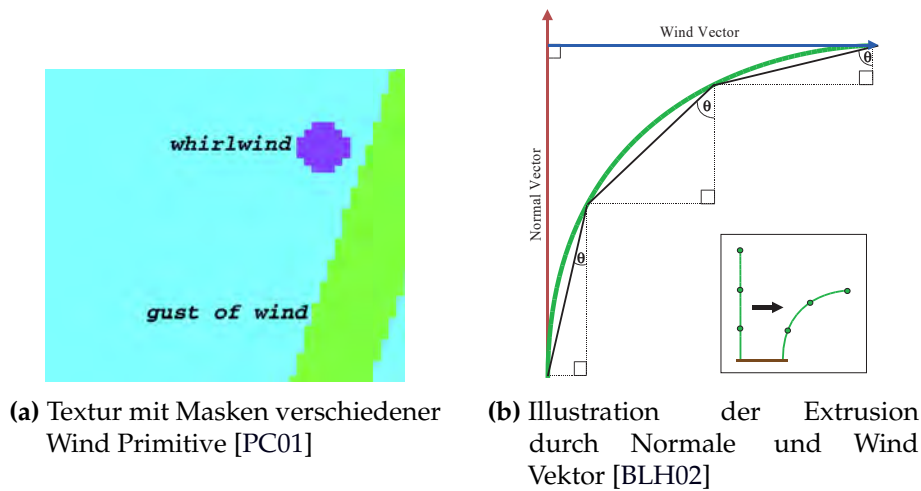


Abb. 12: Veranschaulichung von Vorgehensweisen bei prozeduraler Windapproximation

verwenden. Anschließend wird unter Berücksichtigung einer Wind Intensität der Anteil der Extrusion entlang der Normale und entlang des Wind Vektors berechnet, so dass die Länge von Gras zwischen den Schichten gleich bleibt. Abb. 12b illustriert dieses Vorgehen.

Für die gekreuzten Billboards schlägt Pelzer [Pel04] eine prozedurale Berechnung des Windes vor. Er benutzt die Position des Schnittpunkts der gekreuzten Billboards, einen Zeitstempel und eine Windrichtung und -stärke als Argumente für Sinus und Kosinus Funktionen, um die oberen Eckpunkte der Billboards zu bewegen.

Chen und Johan [CJ10] betrachten die Wiese aus Gras als ein Kontinuum, in dem wellenförmige Ausbreitung von Energie simuliert wird. Das Gras Modell wird beschrieben durch einen Biegewinkel θ an der Wurzel und einer Biege-Geschwindigkeit ω_g . Die Reaktion auf Wind wird zunächst sehr einfach gehandhabt, indem auf den Biegewinkel und die Biege-Geschwindigkeit etwas Rauschen hinzugefügt wird. Andere Windarten, wie etwa solcher verursacht durch einen Helikopter, werden durch die Kollision mit Objekten dargestellt. Im anschließenden Ausbreitungsprozess werden die Änderungen von Biegewinkel und Biege-Geschwindigkeit auf benachbartes Gras ausgebreitet, wodurch auch eine „Gras-Gras Interaktion“ erreicht wird. Chen und Johan erzielen durch diese wellenförmige Ausbreitung sehr ansprechende Ergebnisse bei der Darstellung von Wind, jedoch wirkt der Effekt bei der Kollision mit Objekten sehr unrealistisch (siehe Abb. 13a).

In den Arbeiten [HRS13; JW13; Fan+15] wird auf prozedurale Windsimulation ähnlich der aus [Pel04] zurückgegriffen. In [HRS13; Fan+15] werden dabei zusätzlich Zufallswerte als *Offset* verwendet, um einen einheitliches Aussehen zu vermeiden.

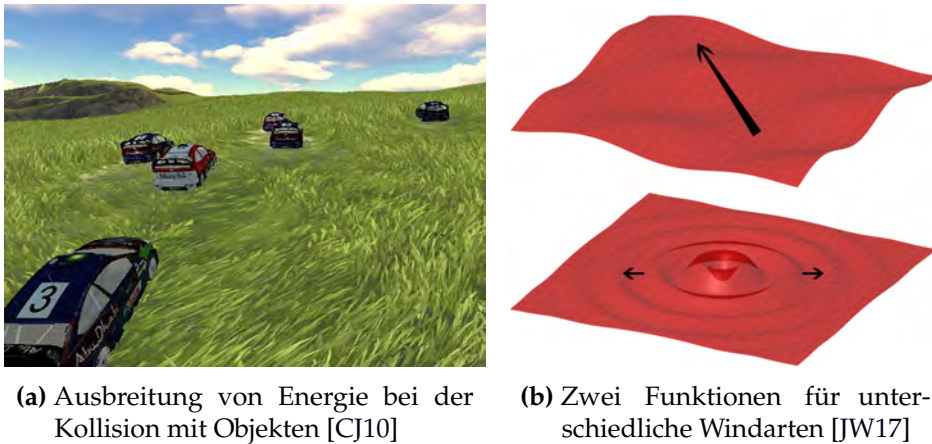


Abb. 13: Kraftausbreitung nach [CJ10] und verschiedene Windfunktionen nach [JW17]

Jahrmann und Wimmer [JW17] verwenden auch analytische Funktionen zur Darstellung von Wind, der sich in Wellen durch den Raum bewegt. Zur Auswertung der Kraft, die auf einen Grashalm wirkt, werden jedoch weitere Faktoren berücksichtigt. Aufrecht stehendes Gras und solches, dessen Oberfläche orthogonal im Wind steht, soll stärker vom Wind beeinflusst werden. Außerdem modellieren sie verschiedene Windfunktionen für Wind, der in eine bestimmte Richtung weht und Wind, der sich von einer bestimmten Position ausbreitet (siehe Abb. 13b).

2.3.2 Physikalisch-basierte Windsimulation

Bei Wang et al. [Wan+05] wird ähnlich wie in [PC01] ein Maske verwendet, um zu bestimmen, welches Gras durch Wind beeinflusst wird. Sie verwenden jedoch ein 2D Windfeld, beschrieben durch Windgeschwindigkeit und Windrichtung, und berechnen dessen physikalische Wirkung auf das Gras Modell. Die Windgeschwindigkeit V_w setzt sich dabei zusammen aus einer durchschnittlichen Geschwindigkeit \bar{V}_w und einem zufälligen Anteil \tilde{V}_w . Die Windrichtung ist ein 2D Vektor $\phi = (\phi_x, \phi_y)$, die Wang et al. für drei verschiedene Windarten als Funktion definieren: Wirbelwind, Lufthauch und eine leichte Brise. Die Berechnung der Reaktion von Gras auf den Wind beruht auf dem Hookeschen Gesetz, dem Impulserhaltungssatz und dem Kräftegleichgewicht. Das Gras-Modell besteht aus einem Skelett, welches aus einzelnen, durch Knoten verbundenen Segmenten besteht. Für diese Knoten werden vertikale und horizontale Verschiebungen berechnet, sowie eine Drehung um die Gras Normale (die Ruheposition) mit der Windrichtung. Die Windkraft F_g , die auf einen Grashalm wirkt, ist:

$$F_g = \rho * S * V_w^2 \quad (1)$$

S ist der Flächeninhalt der Anlaufläche des Windes:

$$S = L_{grass} * W_{grass} * \sin(\Delta\alpha_{w-g}) \quad (2)$$

Es gilt, ρ ist die Winddichte (Masse der Luft pro Volumen), L_{grass} ist die Länge, W_{grass} die Breite des Grashalms und $\Delta\alpha_{w-g}$ ist der Winkel zwischen Windrichtung und der Richtung des Grashalms. Nach dem hookeschen Gesetz ist die Elastizität eines Grashalms am Knoten i :

$$F_i = \mu_i * \theta_{iy} \quad (3)$$

Dabei ist θ_{iy} die vertikale Verschiebung (also die Biegung entlang der Wuchsrichtung) eines Segments und μ_i der Steifheitskoeffizient, der große Werte an der Wurzel des Grases und kleine an der Spitze annimmt. Wang et al. leiten schließlich Gleichungen zur Bestimmung der vertikalen (θ_{iy}) und horizontalen (θ_{ix}) Verschiebung und der Stärke der Grasdrehung (γ_i) eines Segmentes her. Da diese die Anteile der einzelnen Knoten berechnen, werden die Gleichungen für jeden Knoten i von der Wurzel bis zur Spitze ausgewertet und die Ergebnisse dabei aufsummiert.

$$\theta_{iy} = \rho * L_{grass} * W_{grass} * \sin(\alpha_{w-g}) * V_w^2 / \mu_{iy} \quad (4)$$

$$\theta_{ix} = \rho * L_{grass} * W_{grass} * \cos(\alpha_{w-g}) * V_w^2 / \mu_{ix} \quad (5)$$

$$\gamma_i = \rho * L_{grass} * W_{grass} * V_w^2 * \cos^2(\beta_{w-g}) / \mu_\gamma \quad (6)$$

Hier ist α_{w-g} der Winkel zwischen Windrichtung und der Richtung des aktuellen Segments, β_{w-g} der Winkel zwischen Windrichtung und der Gras Normale. Sowohl μ_{ix} als auch μ_γ werden in [Wan+05] nicht näher definiert.

Lee et al. [Lee+16] verwenden 2016 eine physikalische Flüssigkeitssimulation in einem volumetrischen Gitter zum Erzeugen eines Windfelds. Die Reaktion vom Gras auf den Wind wird durch drei verschiedene Bewegungsmuster modelliert: das Schwingen, Biegen und Drehen. Die Ergebnisse der Simulation sind zwar sehr überzeugend, jedoch sind sie nur auf einer kleinen Wiese echtzeitfähig. Ab 100.000 Grashalmen sinken die FPS auf etwa 30 [Lee+16].

2.3.3 Kollisionen

Bei der Behandlung von Kollisionen muss zunächst eine Kollision erkannt werden und anschließend muss das Gras auf die Kollision reagieren. Dabei kann bei der Kollisionserkennung zur Leistungssteigerung eine grobe Erkennung vorangestellt werden, um die Menge an zu testenden Kollisionen bei der exakteren Erkennung zu verringern. Wie beim Wind kommen auch bei der Kollisionsbehandlung Methoden mit variierender physikalischer Exaktheit zum Einsatz. Der Einfachheit halber bezeichnen Kollisionsobjekte

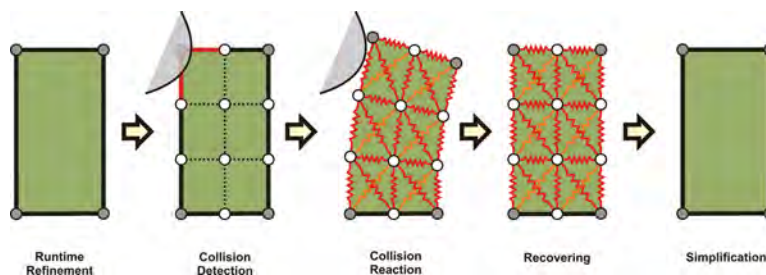


Abb. 14: Ablauf der Kollisionsbehandlung nach Orthmann, Rezk-Salama und Kolb [ORK09]

im Verlauf der weiteren Arbeit genau solche Objekte, die eine Kollision mit Gras auslösen sollen.

Orthmann, Rezk-Salama und Kolb [ORK09] verwenden ein Stoff-Modell mit physikalischen Federeigenschaften, um bei der Kollisionsreaktion das Billboard Gras nicht unschön zu deformieren. Bei der Kollisionserkennung wird zunächst ein grober Test zwischen den *Bounding Boxes* der Kollisionsobjekte und der *Octree* Datenstruktur auf der *Central Processing Unit* (CPU) durchgeführt. Bei einer möglichen Kollision wird die genaue Erkennung auf der GPU ausgeführt. Dabei erzeugt der *Geometry-Shader* die passende *Bounding Sphere* eines Billboards und testet, ob ein Schnittpunkt mit den *Bounding Spheres* der Kollisionsobjekte vorliegt. Fällt dieser Test positiv aus, wird ein weiterer, noch genauere Test mit einer Tiefen *Cube-Map* des Kollisionsobjekts durchgeführt, um zu bestimmen, ob die Billboard Eckpunkte innerhalb des Kollisionsobjekts liegen. Die Tiefen *Cube-Map* wird erstellt, indem der Abstand zwischen den Flächen einer das Objekt einschließenden Box und dem Objekt, sowie die entsprechende Normale des Objekts in einer Textur gespeichert werden. Genaueres zu diesem Test ist der Arbeit [ORK09] zu entnehmen. Wenn alle Tests erfolgreich waren, wurde eine Kollision erkannt und die Reaktion erfolgt, indem der Eckpunkt entlang der Normale des Kollisionsobjekt aus diesem heraus bewegt wird. Anschließend werden die anderen Eckpunkte des Billboards durch Auswerten von Federkräften zwischen den Eckpunkten ebenfalls bewegt, so dass keine unschöne Deformation entsteht. Jedes Billboard speichert eine Erholungszeit, die beim Eintreten einer Kollision auf 1 gesetzt wird. Sobald keine Kollision mehr erkannt wird, befindet sich ein Billboard im Erholungsprozess. In jedem *Frame* wird die Erholungszeit verringert und zur Interpolation zwischen der deformierten und der ursprünglichen Position eines Eckpunkts verwendet. Sobald die Erholung abgeschlossen ist, wird das Billboard wieder in der einfachen Darstellung, bestehend aus lediglich vier Eckpunkten, dargestellt. Der Prozess der Kollisionsbehandlung ist in Abb. 14 veranschaulicht.

Chen und Johan [CJ10] bestimmen zunächst die Fläche eines Kollisions-

objekts innerhalb des Gitter aus *Gras-Patches*. Bei der Kollisionsreaktion wird die Form der Fläche und die Bewegung des Objekts berücksichtigt. Gras wird entlang der Bewegungsrichtung gebogen und an den Seiten der Fläche zusätzlich weg von dem Objektzentrum bewegt.

In [HRS13] wird die Kollisionserkennung auf der CPU ausgeführt. Die Kollisionsobjekte werden in einer *Octree* Datenstruktur zusammengefasst und auf Kollision mit den *Gras-Patches* getestet, welche bei erfolgreichem Test als aktiv markiert werden und einen Zeitstempel zugewiesen bekommen. Die Kollisionskraft wird berechnet und an die GPU weitergereicht, wo die Reaktion auf Kollisionen erfolgt. Ähnlich wie bei bereits vorgestellten Arbeiten [Wan+05; ORK09] wird ein „Masse-Feder-System“ eingesetzt, um die Kraftauswirkung auf Geometrie zu berechnen. Nach einer bestimmten Zeit ohne erkannte Kollision wird ein *Gras-Patch* wieder deaktiviert.

Auch Fan et al. [Fan+15] aktivieren einzelne *Gras-Patches* zur Kollisionsreaktion auf der CPU und deaktivieren diese wieder nach einer gewissen Zeit. Zur Berechnung der Reaktion verwenden sie eine Methode basierend auf dem „Verlet Algorithmus“, die für die Haarsimulation entwickelt wurde und komplett auf der GPU behandelt werden kann. Dabei wird der Schnittpunkt zwischen dem Kollisionsobjekt und den Eckpunkten des Grashalms bestimmt und unter Berücksichtigung der harten Nebenbedingung der Längenerhaltung und der weichen Nebenbedingung Biegung und Drehung in mehreren iterativen Schritten so gelöst, dass kein Schnittpunkt mehr vorliegt. Genaueres ist den Arbeiten [Fan+15; HH12] zu entnehmen.

Jahrman und Wimmer [JW17] wenden die Kollisionserkennung direkt auf der GPU im *Compute-Shader* für alle sichtbaren *Gras-Patches* an, ohne eine grobe Variante voranzustellen oder einzelne *Patches* nach einer bestimmten Zeit deaktivieren zu müssen. Dabei werden die Spitze und der Mittelpunkt der dem Grashalm zugrundeliegenden Bézierkurve mit den Kollisionsobjekten auf einen Schnittpunkt getestet. Es werden nur Tests mit Kugeln durchgeführt, weshalb komplexe Objekte durch Kugeln approximiert werden müssen, welche alle zur Kollisionserkennung herangezogen werden. Wenn ein Schnittpunkt vorliegt, verschiebt sich der Punkt der Bézierkurve auf die nächstgelegene Position auf der Kugeloberfläche. Die quadrierte Länge dieser Verschiebung wird als Kollisionsstärke gespeichert und in den folgenden *Frames* allmählich verringert, um den Grashalm sich von der Kollision erholen zu lassen. Um zu verhindern, dass Gras durch den Boden hindurch gedrückt wird, verwenden sie einen einfachen Test mit der zur Normale des Grashalms orthogonalen Fläche.

Einen anderen Ansatz der Kollisionsbehandlung von Gras verwendet „NVIDIA“, wie in [Mak15] präsentiert. Kollisionsobjekte werden von einer Kamera gerendert und durch Auswerten der Tiefenwerte zur Kollisionserkennung mit Gras verwendet (siehe Abb. 15). Die Kollisionsreaktion erfolgt durch die Bewegung der Gras-Kontrollpunkte entsprechend der Bewegungsrichtung des Kollisionsobjektes und der Eindringtiefe des Grashalms

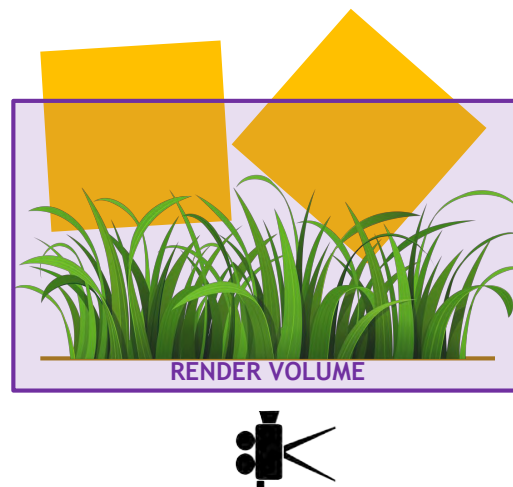


Abb. 15: Als Grundlage zur Kollisionserkennung wird eine orthographische Kamera zum Rendern von Kollisionsobjekten verwendet [Mak15] (bearbeitet als Druckversion mit weißem statt schwarzem Hintergrund).

in das Kollisionsobjekt. Da der Konferenz keine Tagungsberichte folgen, wird ein ähnliches Verfahren von Wang et al. [Wan+09] betrachtet, das der Visualisierung von Reifenabdrücken in einem Terrain dient. Hier wird eine orthographische Kamera unter den niedrigsten Punkt des Terrains positioniert und so ausgerichtet, dass das gesamte Terrain von der Kamera abgedeckt wird. Kollisionsobjekte, in dem Fall die Reifen eines Fahrzeugs, werden gerendert und die Tiefenwerte in der Tiefentextur gespeichert. Indem diese Tiefenwerte mit der Höhenkarte des Terrains verglichen werden, kann bestimmt werden, ob die Reifen des Fahrzeugs eine Deformation des Terrains hervorrufen.

2.4 Beleuchtung

Die Schwierigkeit bei der realistischen Beleuchtung von Gras ist auf die große Menge an Grashalmen und deren Eigenschaften zurückzuführen. Ein Grashalm ist sehr dünn und deshalb lichtdurchlässig, weshalb eine Volumenstreuung (engl. *subsurface scattering*) berücksichtigt werden muss. Es werden sehr viele Schatten geworfen, vor allem in Bodennähe gelangt weniger Licht und es kommt zu vielen Licht Wechselwirkungen. Deshalb ist die globale Beleuchtung (engl. *global illumination*) ein wichtiger Faktor. All diese Eigenschaften beim Echtzeit Rendering zu berücksichtigen gestaltet sich als schwierig. Da die Beleuchtung in der vorliegenden Arbeit kein Schwerpunkt ist, werden nur einfache Beleuchtungsmodelle in Betracht gezogen und Schatten werden nicht berücksichtigt (mehr dazu siehe [Bou08; Bao+11]).

Wenn ein einfaches Beleuchtungsmodell verwendet wird, ist die Texturierung ein entscheidender Faktor der Darstellungsqualität [Fan+15]. Beim bildbasierten Rendering ist es offensichtlich, dass die Wahl der Textur die Grundlage für ein ansprechendes Ergebnis ist. Die bereits vorgestellten Bidirektionalen Textur Funktionen können die Qualität noch steigern, da sie Beleuchtungsinformationen vorberechnen können (siehe [SKP05; BBP06]). Zur dynamischen Beleuchtung empfehlen Colditz et al. bei der bildbasierten Darstellung von Bäumen mit Billboards die Verwendung von *Normal Maps* [Col+05].

Orthmann, Rezk-Salama und Kolb beschreiben ein Verfahren zur dynamischen globalen Beleuchtung der Gras-Billboards [ORK09]. Dabei werden Informationen zur Umgebungsverdeckung und zum Lichteinfall in einem das Gras umhüllenden Volumen berechnet und in zwei 2D-Textur Arrays gespeichert. Die Informationen werden für jeden Eckpunkt ausgelesen und im *Fragment-Shader* mit der Farbe der Gras-Textur verrechnet.

Boulanger [Bou08] verwendet das „Lambert Beleuchtungsmodell“ zur Beleuchtung diffuser Oberflächen und erweitert es um eine ambiente Komponente und eine Transmissions-Komponente. Das geometrische Gras wird dabei beidseitig gerendert. Zur Approximation der Umgebungsverdeckung (engl. *ambient occlusion*) greift er auf die Variante aus [RB85] zurück. Dabei sinkt der Einfluss von ambientem Licht in Bodennähe und ist größer an der Spitze des Grashalms. Die Transmissions-Komponente approximiert das Durchstrahlen von Licht durch das Gras und wird berechnet, indem die Beleuchtung der Rückseite des Grashalms mit einbezogen wird. Auch Fan et al. [Fan+15] setzen auf eine solche Approximation der Volumenstreuung.

2.5 Verteilung von Gras

In der Natur wachsen auf einer Wiese verschiedene Spezies von Gras und Pflanzen mit variierender Größe, Neigung und Drehung und es gibt unbewachsene oder karge Flächen (siehe Abb. 16). Da keine Ökosystem Simulation stattfindet, soll eine manuelle Kontrolle dieser Faktoren möglich sein.

Boulanger [Bou08] verwendet nur eine Art von Gras und kontrolliert dessen räumliche Ausbreitung durch eine *Density-Map* (dt. Dichte-Karte), die als Wahrscheinlichkeitsverteilung agiert. Jeder geometrische Grashalm bekommt einen Schwellwert zugewiesen, der im *Fragment-Shaders* mit der Dichte an der jeweiligen Position verglichen wird. Ist die Dichte größer als der Schwellwert, wird das Fragment verworfen. In [HRS13] wird ein solcher Ansatz um verschiedene Vegetationsarten erweitert. Der Farbkanal bestimmt dabei die Art der Vegetation und die Intensität bestimmt die Dichte. [JW13] stellen verschiedene Vegetationsarten nur durch eine entsprechende Änderung der Farbe dar, welche aus einer das Terrain überspannenden Textur gelesen wird. In einer weiteren Textur speichern sie neben der Dichte



Abb. 16: Eine naturbelassene Wiese auf einem Hang mit üppiger Vegetation.

te u. a. auch die Höhe vom Gras, um einen weicheren Übergang zwischen bewachsenen und unbewachsenen Regionen zu schaffen.

In [JW17] wird jedem einzelnen Grashalm eine eigene Position, Höhe, Breite, Wuchsrichtung und ein Steifheitskoeffizient zugewiesen und in einem Buffer gespeichert. Außerdem ist es möglich, verschiedene Grasarten darzustellen, die durch analytische Funktionen definiert sind. Die Auswahl der passenden Funktion geschieht über „Shader Subroutinen“ und wird in der Diplomarbeit [Jah16] genauer erklärt.

2.6 Unity Engine

„Unity“ wird von „Unity Technologies“ entwickelt [Teco] und wird hauptsächlich für die Entwicklung von Videospielen für verschiedene Plattformen verwendet. Neben einem *Framework*, welches verschiedene Bereiche der Videospieldentwicklung abdeckt, bietet Unity auch eine Entwicklungsumgebung in Form des „Unity Editors“ [Teco]. Zur Programmierung wird die Programmiersprache C# von „Microsoft“ verwendet [Teco]. Für die Grafik Programmierung können unterschiedliche Grafikbibliotheken verwendet werden, wie z. B. *OpenGL* oder *Direct3D* (verschiedene Versionen) [Teco]. Unity stellt hier eine eigene Zugriffsebene dazwischen, sodass nicht direkt mit der Programmierschnittstelle der Grafikbibliotheken kommuniziert wird. *Shader* werden in der entsprechenden Sprache *GLSL* oder *HLSL* programmiert aber enthalten zusätzliche deklarative Anweisungen für die Unity Engine [Teco].

Die Wahl des *Frameworks* und der Entwicklungsumgebung fiel für die

2 GRUNDLAGEN

Entwicklung der vorliegenden Arbeit aus mehreren Gründen auf Unity. Es enthält alle für die Entwicklung notwendigen Funktionalitäten und stellt für die 3D Programmierung wichtige Grundlagen bereit, wie z. B. dem Laden von Texturen und Modellen, einer Kamera oder einer Schnittstelle zur Benutzereingabe, weshalb sich die Entwicklung gezielt auf die Generierung, das Rendering und die Simulation von Gras und Wiesen konzentrieren kann. Weiterhin ist Unity für den persönlichen Gebrauch kostenlos [Teco].

3 Umsetzung

Es folgt die Vorstellung des entwickelten Verfahrens, das verschiedene Techniken der zuvor vorgestellten Verfahren verwendet und kombiniert. Zunächst soll ein Überblick des gesamten Ablaufs geschaffen werden. Anschließend werden die einzelnen Vorgehensweisen und Modelle im Detail vorgestellt.

Als Grafikbibliothek wird in der vorliegenden Arbeit *Direct3D 11* und als Shader Programmiersprache *HLSL* verwendet. Um eine ähnliche Syntax zu verwenden, werden folgende Funktionen eingeführt:

$$\text{normalize}(\mathbf{x}) := \frac{\mathbf{x}}{\|\mathbf{x}\|} \quad (7)$$

$$\text{ceil}(x) := \lceil x \rceil \quad (8)$$

$$\text{floor}(x) := \lfloor x \rfloor \quad (9)$$

$$\text{frac}(x) := \begin{cases} x - \lfloor x \rfloor & x \geq 0 \\ x - \lceil x \rceil & x < 0 \end{cases} \quad (10)$$

$$\text{clamp}(x, a, b) := \begin{cases} a & x < a \\ b & x > b \\ x & \text{sonst} \end{cases} \quad (11)$$

$$\text{min}(a, b) := \begin{cases} a & a < b \\ b & \text{sonst} \end{cases} \quad (12)$$

$$\text{max}(a, b) := \begin{cases} a & a > b \\ b & \text{sonst} \end{cases} \quad (13)$$

$$\text{lerp}(a, b, t) := a + (b - a) * t \quad (14)$$

Wird `clamp`, `min` oder `max` mit Vektoren statt Skalaren verwendet, werden diese komponentenweise behandelt.

3.1 Übersicht des Verfahrens

Das Verfahren lässt sich unterteilen in Arbeitsschritte die zu Beginn einmal ausgeführt werden, der Vorverarbeitung, und in solche die pro *Frame* ausgeführt werden, der Echtzeitverarbeitung. Weiterhin kann zwischen der Ausführung auf der CPU und der GPU unterschieden werden. In Abb. 17 ist der Ablauf mit den genannten Unterteilungen illustriert. In der Vorverarbeitung werden verschiedene benötigte Daten erzeugt und Texturen für die Billboard Darstellung gerendert, die räumliche Datenstruktur wird aufgebaut, die Simulations Texturen werden mit initialen Daten gefüllt und die Kollisionserkennung wird vorbereitet. Die Echtzeitverarbeitung aktualisiert

Winddaten und Kollisionsobjekte, ermittelt sichtbare Gras-Patches und deren LOD, simuliert diese und rendert sie schließlich in den entsprechenden Detailstufen.

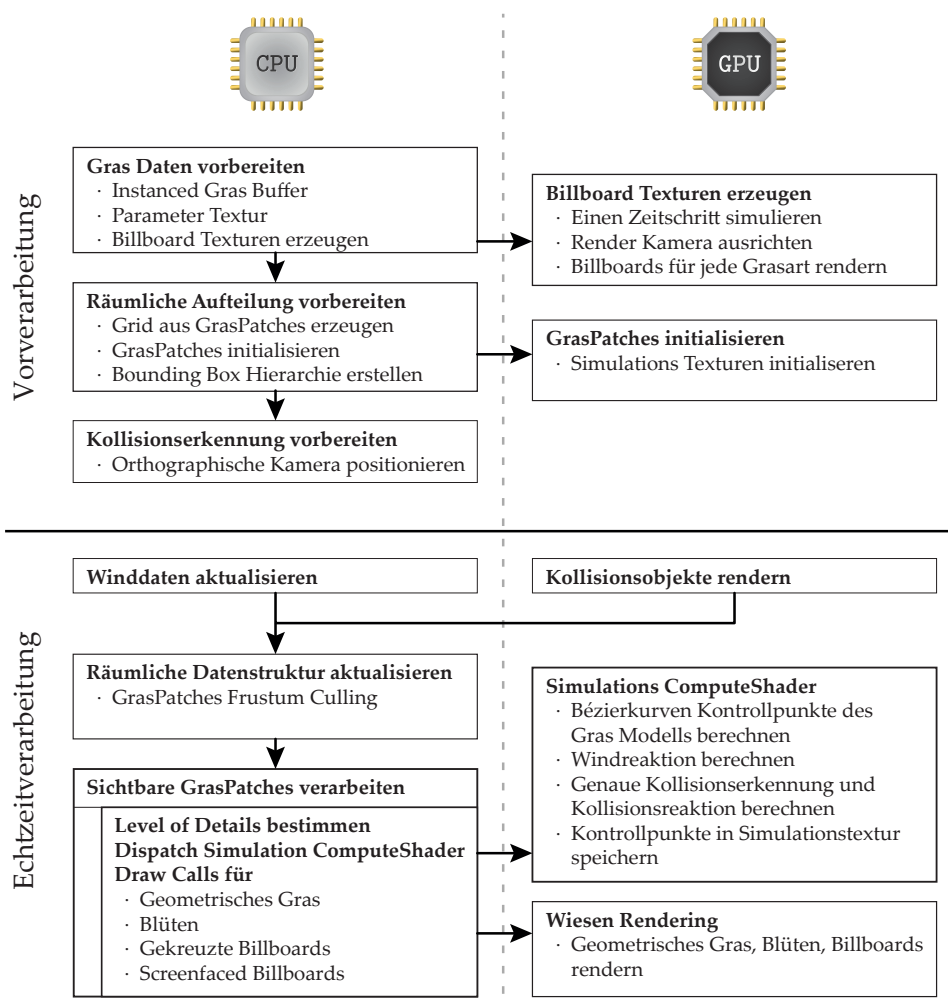


Abb. 17: Übersicht des gesamten Verfahrens

3.2 Kontrollierbarkeit von Gras und Wiese

Zur Kontrolle des Aussehens von Gras und dessen Ausbreitung auf einer Wiese werden verschiedene Parameter verwendet. Zur Definition der räumlichen Ausdehnung der Wiese wird eine *Bounding Box* benötigt. Die relative Position auf der xz -Ebene innerhalb der *Bounding Box* dient als 2D-Texturkoordinate und ist im Folgenden definiert als $uv_{\text{Global}} = [u \ v]$ mit Werten in $]0, 1[$. Eine Höhenkarte stellt Höhenwerte, eine *Normal Map* mit 32 Bit *float* Genauigkeit stellt Normalen in Weltkoordinaten bereit. Als Quelle für

die *Bounding Box* und die Texturen dient ein Terrain, das ebenfalls für die Darstellung des Untergrunds genutzt wird. Es wird das Unity eigene Terrainsystem verwendet.

In einer benutzerdefinierten *Gras-Map* wird in 2 Kanälen eine Dichte und ein Höhenfaktor gespeichert, jeweils mit Werten in $]0, 1[$. Die Dichte beeinflusst die Menge an Gras an einer Position und wird in der Tessellations-Phase ausgewertet (siehe Unterabschnitt 3.6.1). Der Höhenfaktor skaliert die Höhe von Gras um einen weichen Übergang zwischen bewachsenen und unbewachsenen Regionen zu schaffen.

Ähnlich dem Prinzip aus [Fan+15] wird eine Liste vorbereitet mit Informationen zu Grashalmen, aus der später einzelne *Patches* eine Teilmenge zugeordnet bekommen. Jedes Element der Liste enthält eine zufällige 2D-Koordinate und eine zufällige Grasart als positiven Integer Wert. Die Koordinate stellt die relative Position auf der xz -Ebene innerhalb eines *Patches* dar und ist im Folgenden definiert als $uv_{\text{Local}} = [u \ v]$ mit Werten in $]0, 1[$. Beim Ermitteln einer Grasart, welche als Index für ein 2D-Textur Array dient, werden unterschiedliche Wahrscheinlichkeiten verschiedener Arten berücksichtigt. Die Länge der Liste n_b muss groß genug sein, um ausreichend Elemente für die maximale Anzahl an Grashalmen gp_{max} bzw. die maximale Anzahl an Billboards bp_{max} pro *Patch* bereitzustellen. Sie kann mit der folgenden Formel berechnet werden, wobei der Faktor if größer als 1 gewählt werden soll, um sichtbare Wiederholungen von Position und Grasart zu vermeiden:

$$n_b = \max(gp_{\text{max}}, bp_{\text{max}}) * if \quad (15)$$

Genauer handelt es sich bei der Liste um einen strukturierten Buffer mit Lesezugriff (*StructuredBuffer*), dessen Inhalt nach dem Befüllen in der Vorverarbeitung nicht mehr geändert wird. Dieser Buffer wird im Folgenden bezeichnet als *Instanced-Buffer*.

Als weitere Parameter für Gras gibt es die Breite, Höhe, den Biegefaktor und die Rotation. Anders als bei [JW17] werden diese nicht pro Grashalm gespeichert, sondern in einer weiteren Textur, um weniger Speicherplatz zu benötigen. Diese Parameter-Textur speichert in 4 Kanälen zufällige Werte für die Breite, Höhe, den Biegefaktor und die Rotation (in $]0, 2\pi[$) mit 32 Bit *float* Genauigkeit. Mit Ausnahme der Rotation liegen die Zufallswerte in benutzerdefinierten Intervallen. Jedes *Gras-Patch* bekommt bei der Initialisierung einen zufälligen Parameter-Offset uv_{Offset} mit Werten in $]0, 1[$ zugewiesen. Die Summe $uv_{\text{Offset}} + uv_{\text{Local}}$ dient als Texturkoordinate für die Parameter-Textur. Die Textur-Adressierung muss deshalb auf *umwickeln* (engl. *wrap*) oder *spiegeln* (engl. *mirror*) gesetzt werden. Als Auflösung der Textur hat sich mindestens die doppelte Auflösung der Simulations-Textur als geeignet herausgestellt. So fallen mehrere Pixel der Parameter-Textur auf einen Pixel der Simulations-Textur und unterschiedliche *Patches* tasten aufgrund des *Offsets* unterschiedliche Parameter ab. Da es sich bei diesen um

Zufallswerte handelt, entstehen bei einer Unterabtastung keine sichtbaren Artefakte.

3.3 Level of Detail

Eine räumliche Aufteilung durch ein quadratisches Gitter unterteilt das Terrain in einzelne *Gras-Patches*. Daraus wird eine Hierarchie aufgebaut, die für das spätere Frustum *Culling* herangezogen wird. Sichtbaren *Gras-Patches* werden abhängig von der Entfernung zur Kamera eine oder mehrere Grasdarstellungen zugewiesen, von denen es drei verschiedene gibt:

1. Geometrische Grasdarstellung in Kameranähe
2. Gekreuzte Billboards in mittlerer Entfernung
3. Der Kamera zugewandte Billboards (i. F. *screen-facing* Billboards) in weiter Ferne

Die verschiedenen Darstellungen werden miteinander überblendet, sodass der Übergang kaum sichtbar ist. Es können Blüten auf den Spitzen von Grashalmen dargestellt werden, die in den Billboard Darstellungen ebenfalls zu sehen sind. Für geometrische Grashalme und Blüten gibt es nochmals unterschiedliche Detailstufen, die durch den Tessellierungsgrad kontrolliert werden. Um Gras unabhängig von seiner Quantität zu simulieren, wird die Simulation in einem kleineren Maßstab durchgeführt. Dieses Vorgehen wird in Unterabschnitt 3.5.1 getrennt betrachtet.

3.3.1 Unterteilung des Terrains

In Abb. 18 ist die Unterteilung eines 64×64 Terrains in *Gras-Patches*, die erzeugte Hierarchie und die Anwendung von Frustum *Culling* dargestellt. Zunächst wird das Terrain in quadratische Grundflächen der Seitenlänge $l = 8$ aufgeteilt und eine an der Fläche das Terrain einschließende *Bounding Box* wird erstellt. Jedes *Patch* bekommt die relative 2D-Koordinate uv_{Patch} zugewiesen, um später die globale Texturkoordinate uv_{Global} bestimmen zu können.

Weiterhin wird eine *Patch Model Matrix* M_P zur Transformation von Patchkoordinaten in Weltkoordinaten definiert:

$$M_P = \begin{bmatrix} l & 0 & 0 & \mathbf{b}_x \\ 0 & t_h & 0 & \mathbf{t}_y \\ 0 & 0 & l & \mathbf{b}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (16)$$

Dabei ist \mathbf{b} der Minimalpunkt der *Bounding Box* eines *Patch*, also der mit den niedrigsten Achsen-Koordinaten. Die Höhe des Terrains ist t_h und \mathbf{t} ist

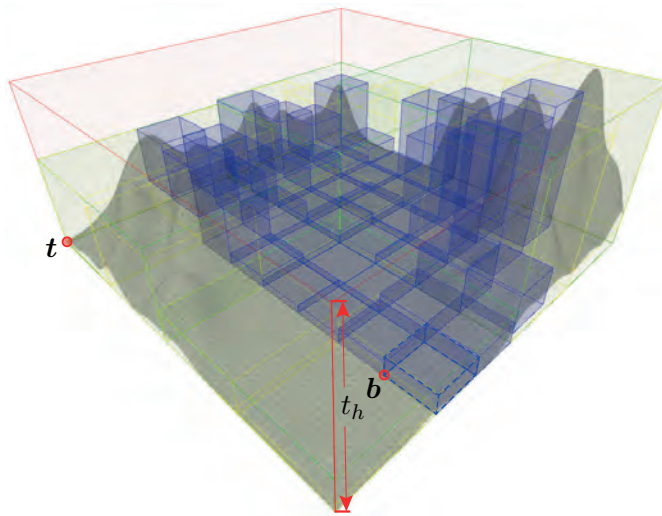


Abb. 18: Sichtbare *Patches* (blau) und die *Bounding Box* Hierarchie Ebenen (gelb, grün, rot)

der Minimalpunkt der *Terrain Bounding Box* (siehe Abb. 18). Eine Rotation ist nicht vorgesehen, da es sich um *Axis-Aligned* (dt. entlang der Achsen ausgerichtet) *Bounding Boxes* handelt.

Hierarchie und Frustum Culling Die Hierarchie wird rekursiv erzeugt, indem nach Möglichkeit immer vier *Patches* zusammengefasst und deren *Bounding Boxes* von einer neuen *Bounding Box* umschlossen werden. Dies wird solange wiederholt, bis eine einzelne *Bounding Box* das gesamte Terrain umschließt. Um die Höhe des Grases beim *Frustum Culling* miteinzubeziehen, wird die *Bounding Box* von jedem *Patch* in alle Richtungen um die maximale Höhe von Gras erweitert. Zu Beginn jedes *Frames* wird die Hierarchie von oben nach unten abgearbeitet, dabei wird getestet, ob eine *Bounding Box* innerhalb des Kamera *Frustums* liegt oder ob ein Schnittpunkt vorliegt. Die Kinder einer Hierarchieebene werden nur weiter getestet, wenn der Test erfolgreich war. Das Ergebnis davon ist ebenfalls in Abb. 18 zu sehen. Nur *Gras-Patches*, die den Test bestanden haben (blau), werden simuliert und gerendert.

3.3.2 Kombination von Geometrie und Billboards

Die verschiedenen Grasdarstellungen werden in benutzerdefinierten Entfernungen zur Kamera eingesetzt. Als Methode zur Überblendung wird dabei die Grasdichte und Grashöhe modifiziert. Ein *Gras-Patch* kann gleichzeitig mit mehreren Grasdarstellungen gerendert werden. Jede Grasdarstellung wird pro *Patch* mit einem einzelnen *Draw-Call* (dt etwa: Malaufruf) mehre-

rer *Instances* (Anzahl instanzierter Aufrufe beim *Instanced-Rendering*) gerendert. Diese Anzahl an *Instances* kontrolliert die Dichte von Gras und wird auf der CPU für jedes *Patch* pro *Frame* berechnet. Die GPU skaliert die Höhe von dem Gras, welches sich gerade in der Überblendung befindet und entfernt Grashalme, die bereits ausgeblendet wurden oder zu klein sind. Dieses *Culling* von unerwünschtem Gras geschieht im *Hull-Shader*, indem der Tessellations-Faktor auf null gesetzt wird. Da die Anzahl der *Instances* bereits auf der CPU begrenzt wird, muss weniger Gras auf der GPU aussortiert (engl. *culled*) werden als etwa bei Jahrman und Wimmer [JW17], die alle Grashalme im *Compute-Shader* einzeln betrachten und ggf. aussortieren. Die Transparenz anzupassen ist für das entwickelte Verfahren ungeeignet, da so der Vorteil des geometrischen Grasmodells, ohne Transparenz auszukommen (siehe Unterabschnitt 3.4.2), verloren ginge.

Zur Unterscheidung von Grasdarstellungen werden im Folgenden die Subskripte *geo* (geometrisch), *cro* (gekreuzt, engl. *crossed*) und *scr* (*screen-facing*) verwendet.

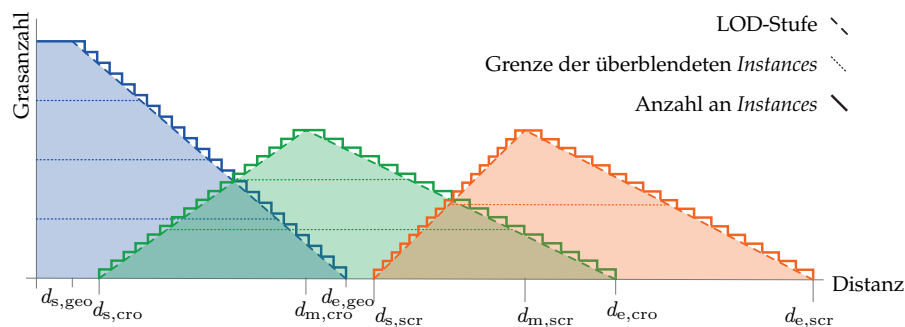


Abb. 19: Visualisierung des LOD Verfahrens, dem Bestimmen der Anzahl an *Instances* und der weichen Übergänge. Geometrisches Gras ist blau, gekreuzte Billboards grün und *screen-facing* Billboards orange gefärbt.

Berechnung der LOD Stufen Zunächst gilt es einige Variablen zu definieren, die vom Benutzer kontrolliert werden:

- n : Die maximale Anzahl *Instances* einer Grasdarstellung.
- d_s, d_m, d_e : Die Start-, Mittel- und Enddistanz der Überblendung einer Grasdarstellung.
- k : Die Anzahl an *Instances* die gleichzeitig überblendet werden.
- Die Anzahl an Gras pro *Instance* von geometrischen Gras ist $v_{geo} = l^2$, die für Billboard Darstellungen ist $v_{cro} = v_{scr} = l$.

Die Berechnung der Anzahl an *Instances* und der Höhenüberblendung an einem Punkt in der Welt leitet sich aus einem einzelnen Wert ab, genannt

LOD-Stufe (siehe Legende in Abb. 19). Diese wird in Abhängigkeit der Distanz von einem Punkt zur Kamera, der maximalen Stufe x und weiteren benutzerdefinierten Variablen berechnet. Für geometrisches Gras ist die LOD-Stufe definiert als:

$$\text{lod}_{\text{geo}}(d, x) = \text{lerp}\left(x, 0, \text{clamp}\left(\frac{d - d_{\text{s,geo}}}{d_{\text{e,geo}} - d_{\text{s,geo}}}, 0, 1\right)\right) \quad (17)$$

Für gekreuzte Billboards gilt:

$$\begin{aligned} \text{lod}_{\text{cro}}(d, x) = & \frac{n_{\text{cro}}}{k_{\text{cro}}} - \text{lerp}\left(x, 0, \text{clamp}\left(\frac{d - d_{\text{s,cro}}}{d_{\text{m,cro}} - d_{\text{s,cro}}}, 0, 1\right)\right) \\ & + \text{lerp}\left(0, x, \text{clamp}\left(\frac{d - d_{\text{m,cro}}}{d_{\text{e,cro}} - d_{\text{m,cro}}}, 0, 1\right)\right) \end{aligned} \quad (18)$$

Die Gleichung für *screen-facing* Billboards entspricht der Gleichung 18 unter Verwendung der benutzerdefinierten Variablen für eben solches.

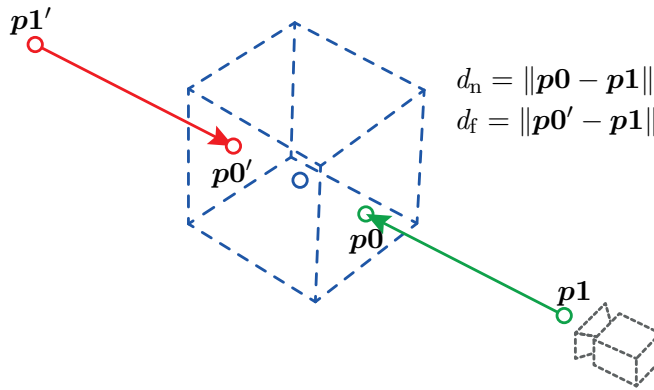


Abb. 20: Berechnung der Entfernungen d_n und d_f

Berechnung der Instances auf der CPU Zur Berechnung der Anzahl von *Instances* für die jeweiligen Grasdarstellungen eines *Patches* werden zuerst zwei Entfernungen auf der CPU berechnet, wie in Abb. 20 illustriert. Die kleinste Distanz d_n ist die, von der *Bounding Box* zur Kamera und die Distanz d_f ist die, vom entgegengesetzten Punkt auf der *Bounding Box* zur Kamera. Für d_n ist dies die Länge des Vektors von der Position der Kamera zum, der Kamera am nächstgelegenen Punkt auf der *Bounding Box*. Für d_f ist es die Länge des Vektors von der Kameraposition zu dem Punkt, der am nächsten zu der am Mittelpunkt der *Bounding Box* punktgespiegelten Kameraposition liegt. Der nächstgelegene Punkt $p0$ auf einer *Bounding Box* zu einem beliebigen Punkt $p1$ kann berechnet werden, indem $p1$ auf den Minimal- und Maximalpunkt der *Bounding Box* beschränkt wird [Sza17, pp. 166–167] (siehe Gleichung 11).

Anschließend kann die Anzahl der *Instances* für die Grasdarstellungen berechnet werden. Nur für Billboard Darstellungen werden beide Distanzen benötigt, da diese ein- und ausgeblendet werden können. Die Anzahl an *Instances* (in Abb. 19 dargestellt als durchgängige Linie) für geometrisches Gras c_{geo} ist:

$$c_{\text{geo}} = \text{ceil}(\text{lod}_{\text{geo}}(d_n, n_{\text{geo}})) \quad (19)$$

Die für gekreuzte Billboards c_{cro} ist:

$$c_{\text{cro}} = \max(\text{ceil}(\text{lod}_{\text{cro}}(d_n, n_{\text{cro}})), \text{ceil}(\text{lod}_{\text{cro}}(d_f, n_{\text{cro}}))) \quad (20)$$

Auch hier berechnet sich die Anzahl an *Instances* für *screen-facing* Billboards c_{scr} analog zu den gekreuzten.

Behandlung auf der GPU Im *Hull-Shader* wird die LOD-Stufe für jeden Grashalm bzw. für jedes Gras-Billboard berechnet. Da mehr als eine *Instance* gleichzeitig überblendet werden soll, wird die maximale LOD-Stufe durch die Anzahl gleichzeitig zu Überblendender *Instances* k dividiert. Zusätzlich wird die Dichte s aus der *Gras-Map* mit einbezogen. Es wird die exakte Entfernung zwischen Gras und der Kamera d_{grs} verwendet. Es kommt zum *Culling*, wenn die *Instance ID* id (der Index des aktuellen *Instanced Draw-Calls*) dividiert durch k größer als die LOD-Stufe ist. Das entspricht in Abb. 19 dem Bereich überhalb der gestrichelten Linie der LOD-Stufe.

Die Höhe wird von dem Gras angepasst, welches sich in der aktuell überblendeten Gruppe von *Instances* befindet. Formal ausgedrückt ist diese Bedingung erfüllt wenn gilt:

$$\text{floor}\left(\text{lod}\left(d_{\text{grs}}, \frac{n}{k}s\right)\right) = \text{floor}\left(\frac{id}{k}\right) \quad (21)$$

In Abb.19 entspricht das dem Gras, das zwischen der gestrichelten Linie und der nächsten gepunkteten Linie liegt. Der Faktor trf_{grs} , mit dem die Höhe des Grasses in diesem Fall multipliziert wird, entspricht dem Nachkommastellenanteil der LOD-Stufe:

$$\text{trf}_{\text{grs}} = \text{frac}\left(\text{lod}\left(d_{\text{grs}}, \frac{n}{k}s\right)\right) \quad (22)$$

Unterschreitet dieser einen bestimmten Schwellwert, wird Gras ebenfalls aussortiert, um Aliasing zu verhindern.

Qualität der Überblendung Indem nicht das gesamte Gras einer Darstellung gleichzeitig überblendet wird, ist diese Überblendung weniger offensichtlich. Zwar sieht eine solche Überblendung (siehe Abb. 21a) getrennt betrachtet weicher aus, doch ermöglicht eine Überblendung von nur weniger *Instances* (siehe Abb. 21b) eine dichtere Darstellung der Wiese ohne

dabei den Wechsel der Grasdarstellung zu offenbaren. Je kleiner der Wert k gesetzt wird, desto schneller wird die Größe von Gras im Überblendungsbereich geändert, weshalb ein zu kleiner Wert schließlich zu sichtbarem Erscheinen und Verschwinden von Gras führt. Eine alternative Vorgehensweise wäre, die Anzahl von Gras pro *Instance* zu erhöhen. Im Vergleich zum vorgestellten Verfahren würde dabei mehr Gras von der GPU aussortiert werden müssen. Die Auswirkungen der beschriebenen und alternativen Vorgehensweise auf die Performance werden in Abschnitt 4.1 untersucht.

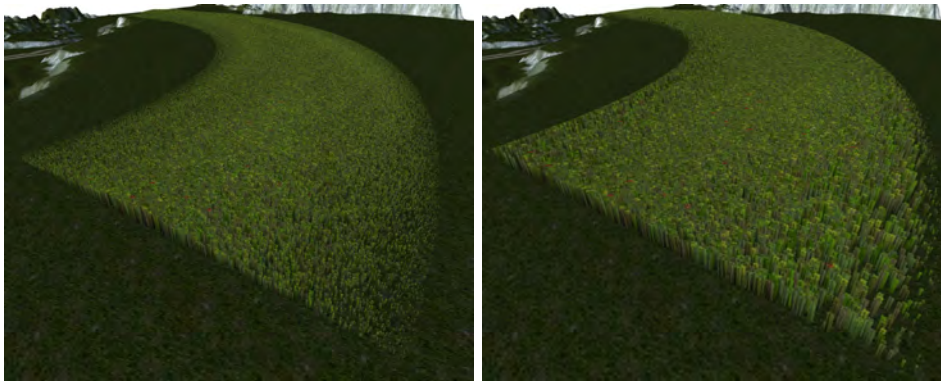
(a) Überblendung mit $k_{\text{cro}} = 192$ (b) Überblendung mit $k_{\text{cro}} = 32$

Abb. 21: Ergebnis des LOD Systems bei gekreuzten Billboards mit $n_{\text{cro}} = 192$ und unterschiedlicher Anzahl gleichzeitig überblendeter *Instances* k_{cro}

3.3.3 Tessellation

Die Tessellations-Faktoren von geometrischem Gras tf_{grs} und Blüten tf_{bls} kontrollieren deren Detailgrad. Sie lassen sich wie in [Jah16] durch benutzerdefinierte Werte kontrollieren:

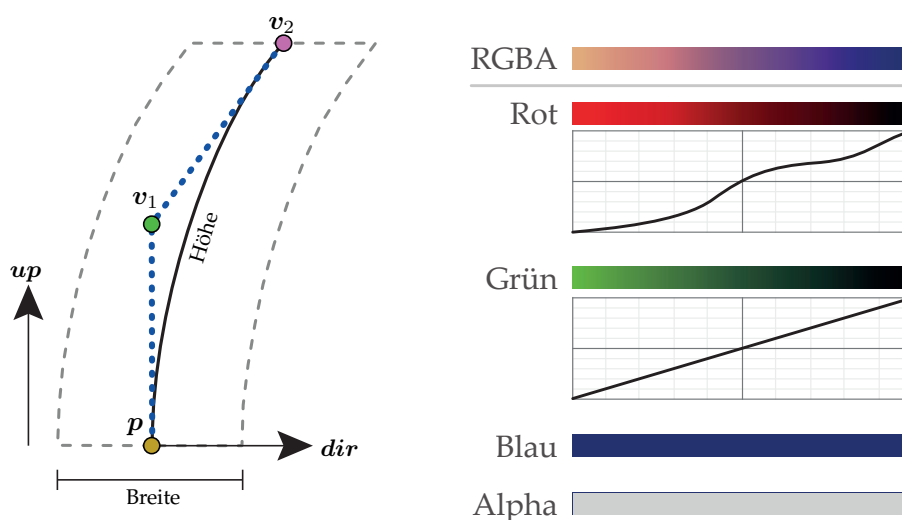
- $tf_{\text{min}}, tf_{\text{max}}$: Der minimale und maximale Tessellations-Faktor.
- td_{min} : Die Distanz, bis zu der der maximale Tessellations-Faktor verwendet wird.
- td_{max} : Die Distanz, ab der der minimale Tessellations-Faktor verwendet wird.

Zwischen den beiden Entfernungen (jeweils die Entfernung von der Kamera zum Grashalm bzw. zur Blüte) wird der Faktor linear interpoliert. Bei Blüten muss aufgrund ihrer Modellierung sichergestellt werden, dass tf_{bls} gerade ist.

3.4 Gras Modellierung

Für geometrisches Gras wird ein Modell basierend auf dem Gras-Modell aus [JW17] verwendet. Die Grundlage stellt eine quadratische Bézierkurve mit drei Kontrollpunkten dar, die zur Simulation herangezogen wird. Diese ist gut geeignet, da sie gleichzeitig für Billboards verwendet werden kann. So kann für geometrisches und bildbasiertes Gras das gleiche Modell und folglich die gleiche Simulation verwendet werden.

3.4.1 Gras-Modell



(a) Definition des Gras-Modells durch eine quadratische Bézierkurve und weiterer Attribute, nach [JW17].

(b) Informationen zur Kontrolle der Form eines Grashalms, sowie Beleuchtungseigenschaften werden in einer 1D-Textur gespeichert.

Abb. 22: Präsentation des Gras-Modells und zugehöriger Attribute

Das Gras-Modell ist in Abb. 22a illustriert. Den Kern stellt die Bézierkurve dar, bestehend aus der Position p auf dem Boden in Weltkoordinaten, der Spitze $p + v_2$ und dem Kontrollpunkt zur Biegung $p + v_1$. In der vorliegenden Arbeit sind v_1 und v_2 folglich Richtungsvektoren, die addiert mit p die Kontrollpunkte in Weltkoordinaten ergeben. Der up Vektor bestimmt die Wuchsrichtung und ist stets der Einheitsvektor von v_1 . Die Breite w bestimmt die maximale Breite eines Grashalms, die Höhe h beschreibt die Länge von Gras und wird zur deren Erhaltung verwendet. Der Biegefaktor b zwischen 0 und 1 beeinflusst, wie stark sich Gras biegen kann. Größere Werte bedeuten dabei eine stärkere Biogsamkeit. Die Ausrichtung von Gras wird durch den Vektor dir bestimmt, entlang dem Gras in seiner Breite

expandiert wird. Dieser lässt sich mithilfe des Rotationsparameters α berechnen:

$$\begin{aligned} \mathbf{tmp} &= \text{normalize}([\sin(\alpha) \quad \sin(\alpha) + \cos(\alpha) \quad \cos(\alpha)]) \\ \mathbf{dir} &= \text{normalize}(\mathbf{tmp} \times \mathbf{up}) \end{aligned} \quad (23)$$

Für geometrisches Gras werden die Werte für h , w , b und α aus der Parameter-Textur gelesen. Für Billboards werden einige Änderungen eingeführt. Sie benötigen eine andere Breite und Ausrichtung. Beim Generieren der Billboard Texturen (siehe Unterabschnitt 3.6.4) wird das Seitenverhältnis ar von Breite zu Höhe der Textur berechnet. Die Breite w von Billboards lässt sich damit berechnen mit:

$$w = \|\mathbf{v}_2\| * ar \quad (24)$$

Gekreuzte Billboards bestehen aus drei einzelnen Billboards $i = 0 \dots 2$ mit der gleichen Bézierkurve. Der Rotationsparameter aus der Parameter-Textur wird dabei manipuliert, um die Ausrichtung zu kontrollieren:

$$\alpha_i = \alpha + \frac{2}{3}\pi i \quad (25)$$

Die *screen-facing* Billboards werden so ausgerichtet, dass der Vektor \mathbf{dir} orthogonal zur Kamera (Kameraposition sei $\mathbf{cam}_{\text{pos}}$) ist:

$$\begin{aligned} \mathbf{cam}_{\text{dir}} &= \mathbf{p} - \mathbf{cam}_{\text{pos}} \\ \mathbf{dir} &= \text{normalize}(\mathbf{up} \times \mathbf{cam}_{\text{dir}}) \end{aligned} \quad (26)$$

3.4.2 Gras-Kontur

Die von Jahrman und Wimmer in [JW17] eingeführten analytischen Funktionen zur geometrischen Definition der Gras-Kontur haben den Vorteil, dass weniger Fragmente als etwa bei der Verwendung von Texturen mit Alpha Kanal als Maske betrachtet werden müssen [JW17]. Trotzdem können verschiedene Formen von Gras realisiert werden. Die Gestaltung dieser Funktionen gestaltet sich in der Praxis jedoch schwierig, nicht zuletzt weil diese für verschiedene Detailstufen entwickelt werden müssen, um einen Verlust von Details bei der diskreten Abtastung zu vermeiden.

Um die genannten Vorteile der analytischen Funktionen zu behalten und dabei die Gestaltung von Gras-Konturen zu vereinfachen, wird eine 1D-Textur als *Lookup Tabelle* eingesetzt (siehe Abb. 22b). Der rote Kanal enthält die Werte zur Definition der Gras-Kontur und stellt damit das Äquivalent zu den analytischen Funktionen dar. Die 1D-Textur hat eine Größe entsprechend dem maximalen Tessellations-Faktor tf_{max} , maximal jedoch 64, was dem Hardwarelimit des Tessellations-Faktors entspricht [CoroJd].

Mittels *Mip Mapping* und trilinearere Texturfilterung kann die dem Tessellations-Faktor entsprechende Detailstufe der Textur ausgewertet werden.

Dies ermöglicht einen weichen Übergang zwischen verschiedenen Detailstufen, welcher beim Verwenden analytischer Funktionen von [JW17] nicht möglich ist. Jahrman und Wimmer [JW17] schlagen eine Korrektur der Grasbreite vor, um dünne Grasspitzen in der Ferne breiter werden zu lassen. In der vorliegenden Arbeit wird diese Korrektur im Voraus in den *Mip-Map* Stufen gespeichert. Die LOD Stufe zum Abtasten der Textur tx_{lod} lässt sich berechnen mit:

$$\begin{aligned} tx_{\text{mip}} &= 1 + \text{floor}(\log_2(tf_{\text{max}})) \\ tx_{\text{lod}} &= \text{lerp}\left(tx_{\text{mip}} - 1, 0, \text{clamp}\left(\frac{tf_{\text{grs}}}{tf_{\text{max}}}, 0, 1\right)\right) \end{aligned} \quad (27)$$

Dabei ist tx_{mip} die Anzahl an *Mip-Maps* der 1D-Textur.

Die weiteren Kanäle der Textur enthalten Werte für die 3D-Verschiebung (Grün), den diffusen Reflexionsgrad (Blau) und die Lichtdurchlässigkeit (Alpha).

3.5 Simulation

Im Gegensatz zu der Arbeit [JW17] ist das Gras-System in der vorliegenden Arbeit nur für die Verwendung auf einem Terrain vorgesehen und nicht auf beliebigen Oberflächen. Dies grenzt die Anwendungsmöglichkeiten ein, ermöglicht aber einige Optimierungen. Grashalme müssen nicht einzeln vorliegen und über die zugrundeliegende Geometrie verteilt werden. Stattdessen kann der in Abschnitt 3.2 vorgestellte *Instanced-Buffer* eingesetzt werden, um zusammen mit den Höhenwerten der Höhenkarte und der Matrix M_P die Position p zu ermitteln. Die beiden Vektoren v_1 und v_2 sind hingegen vom Standort abhängig und können nicht instanziiert vorliegen.

3.5.1 Simulation in einer Textur

In [JW17] simuliert Jahrman jeden einzelnen Grashalm, der in einem sichtbaren *Patch* liegt. Die Kontrollpunkte werden für jeden Grashalm einzeln gespeichert. Die Echtzeitfähigkeit kann dabei erzielt werden, indem Grashalme, welche keinen visuellen Beitrag leisten, in einem *Compute-Shader* aussortiert werden. Nach anfänglichen Tests stoß dieses Vorgehen, angewendet auf ein großes Terrain (256×256) mit 1.024 *Gras-Patches* jedoch an seine Grenzen. Wenn bspw. in Kameranähe 32.768 Grashalme dargestellt und die beiden Kontrollpunkte (jeweils ein Array mit 3 Elementen des Datentyps *float*) für jeden Grashalm einzeln gespeichert würden, ergäbe sich ein Speicherbedarf von:

$$1.024 * 32.768 * 2 * 3 * 4\text{Byte} = 805.306.368\text{Byte} \approx 805\text{MB}$$

Stattdessen werden für jedes *Gras-Patch* zwei quadratische 2D-Texturen mit 32 Bit *float* Genauigkeit eingeführt (i. F. genannt Simulationstexturen),

welche die Vektoren v_1 und v_2 speichern. Die Auflösung der Texturen wird durch die benutzerdefinierte Seitenlänge s bestimmt, wobei sich die doppelte bis vierfache Seitenlänge von einem *Patch* l als ausreichend herausgestellt hat. Eine Seitenlänge von $s = 4 * l = 32$ resultiert unter den zuvor beschriebenen Testbedingungen (mit dem Unterschied, dass mit 4 Kanälen pro Textur gerechnet wird) in einem Speicherbedarf von:

$$1.024 * (32 * 32) * 2 * 4 * 4\text{Byte} = 33.554.432\text{Byte} \approx 34\text{MB}$$

Bei einer Seitenlänge von $s = 16$ sind es noch etwa 8MB.

Durch dieses Vorgehen müssen pro *Patch* nur s^2 Grashalme simuliert werden, wodurch die Simulation von der Quantität an Gras entkoppelt wird. Jede 2D-Koordinate im Intervall $]0, 1[$ kann mit dem Höhenwert der Höhenkarte und der Matrix M_p in die Position auf dem Boden des Terrains p transformiert werden. Die Simulation wird an den diskreten Abtastpunkten der Textur ausgeführt und die resultierenden Vektoren von der Position zu den Kontrollpunkten in den beiden Simulationstexturen gespeichert. Beim Rendering vom Gras wird die 2D-Koordinate uv_{Local} aus dem *Instanced-Buffer* als Texturkoordinate verwendet, um die Vektoren aus den Simulationstexturen zu lesen und die Kontrollpunkte zu bestimmen.

3.5.2 Ablauf der Simulation

Die Simulation wird in einem *Compute-Shader* ausgeführt. Dabei gibt es insgesamt s^2 *Threads* pro *Compute-Shader Dispatch* (die Benachrichtigung an die GPU, den *Compute-Shader* auszuführen). Die Simulationstexturen werden in der Vorverarbeitung mit initialen Werten befüllt, die das Gras im Ruhezustand darstellen. Die 2D-Texturkoordinate uv_{Local} berechnet sich mit der Division der *Dispatch Thread ID* durch s . Die globale 2D-Koordinate uv_{Global} wird berechnet durch lineare Interpolation der minimalen und maximalen 2D-Koordinaten des aktuellen *Patches*, mit uv_{Local} als Wert zur Interpolation. Pro *Frame* wird die Simulation für sichtbare *Patches* ausgeführt. Dabei werden auf die Grashalme wirkende Kräfte berechnet und unter Einhaltung bestimmter Einschränkungen die resultierenden Vektoren v_1 und v_2 .

3.5.3 Berechnung der Kontrollpunkte

Die Berechnung der Kontrollpunkte der Bézierkurve folgt der Definition in [JW17]. In der vorliegenden Arbeit wird mit den Richtungsvektoren von p zu den Kontrollpunkten statt mit den Kontrollpunkten in Weltkoordinaten gerechnet.

Die Höhe eines Grashalms h wird um den Höhenfaktor aus der *Gras-Map* skaliert. Ein Vektor $gdir$ repräsentiert eine Anziehungskraft. Damit Gras auf schrägem Terrain nicht orthogonal zu diesem wächst und Fell

ähnelt, wird zur Berechnung von up neben der Terrain Normale n_t auch der Vektor $gdir$ mit einbezogen:

$$up = \text{normalize}(n_t - 0,5 \text{ normalize}(gdir)) \quad (28)$$

Die Richtung zur Spitze v_2 befindet sich im Ruhezustand bei:

$$v_2 = hup \quad (29)$$

Kräfte werden nur auf diese Spitze v_2 angewendet. Der Vektor v_1 wird abhängig von v_2 so berechnet, dass er parallel zu up ist. Dazu wird v_2 auf die Ebene projiziert, zu der up senkrecht steht. Die Länge dieses Vektors g_{up} wird mit der Höhe h in Relation gesetzt, wobei durch den konstanten Faktor 0,05 sichergestellt wird, dass v_1 mindestens eine kleine Biegung des Grashalms bewirkt. Formal gilt:

$$\begin{aligned} g_{up} &= v_2 - up(v_2 \cdot up) \\ v_1 &= hup \max\left(1 - \frac{\|g_{up}\|}{h}, 0,05 \max\left(\frac{\|g_{up}\|}{h}, 1\right)\right) \end{aligned} \quad (30)$$

Weiterhin wird in [JW17] ein approximatives Vorgehen zur Erhaltung der Länge eines Grashalms beschrieben. Die verschiedenen Kräfte wirken dabei zunächst ohne Einschränkung auf den Vektor v_2 , der anschließend so korrigiert wird, dass die Länge der Bézierkurve etwa der Grashöhe h entspricht. Hierzu berechnet Jahrmann die ungefähre Länge der Bézierkurve L und korrigiert die Abstände zwischen den Kontrollpunkten. Als Formel zur Approximation der Länge einer Bézierkurve n -ten Grades verwendet er die aus [Gra97]:

$$L = \frac{2L_0 + (n - 1)L_1}{n + 1} \quad (31)$$

L_0 ist der Abstand vom ersten zum letzten Kontrollpunkt und L_1 die Summe aller Abstände zwischen aufeinanderfolgenden Kontrollpunkten. Die korrigierten Vektoren v'_1 und v'_2 werden berechnet mit:

$$\begin{aligned} r &= \frac{h}{L} \\ v'_1 &= rv_1 \\ v'_2 &= r(v_2 - v_1) \end{aligned} \quad (32)$$

Es muss sichergestellt werden, dass die Position $p + v_2$ nicht unter der Oberfläche des Terrains liegt. In [JW17] ist up orthogonal zum Untergrund ausgerichtet, so dass die Bedingung erfüllt ist, wenn gilt:

$$up \cdot v_2 \geq 0 \quad (33)$$

In der vorliegenden Arbeit kann dies unter Verwendung der Terrain Normale \mathbf{n}_t sichergestellt werden:

$$\mathbf{v}_2 = \mathbf{v}_2 - \mathbf{n}_t \min(\mathbf{n}_t \cdot \mathbf{v}_2, 0) \quad (34)$$

Indem die Gleichungen 34, 32 und 30 nacheinander angewandt werden, befindet sich der Grashalm wieder in einer gültigen Position.

3.5.4 Schwerkraft

Die Reaktion von Gras auf die Schwerkraft soll bewirken, dass der Vektor \mathbf{v}_2 seinen Ruhezustand verlässt und das Gras eine gebogene Haltung einnimmt. Hierzu wird das Prinzip der Schwerkraft, repräsentiert durch eine Richtung und Fallbeschleunigung aus [JW17] verwendet. Der normalisierte Vektor \mathbf{gdir} stellt die Richtung der Schwerkraft und dessen Betrag die Beschleunigung dar. Damit der Vektor \mathbf{v}_2 nicht nur entlang der Richtung der Schwerkraft verschoben wird, wird zusätzlich entlang des Vektors \mathbf{front} verschoben. Die resultierende Auswirkung der Schwerkraft \mathbf{g} auf \mathbf{v}_2 berücksichtigt auch die Höhe h und den Biegefaktor b :

$$\begin{aligned} \mathbf{front} &= \mathbf{dir} \times \mathbf{up} \\ \mathbf{g}_F &= \frac{1}{4} \|\mathbf{gdir}\| \mathbf{front} \\ \mathbf{g} &= hb (\mathbf{gdir} + \mathbf{g}_F) \end{aligned} \quad (35)$$

3.5.5 Wind

Die Windsimulation setzt sich zusammen aus dem Verarbeiten verschiedener Windquellen auf der CPU zu Beginn jedes *Frames*, dem Summieren der Winde auf der GPU und unmittelbar darauf und dem Auswerten des Einflusses auf das Gras.

Vorbereitung der Windquellen Um verschiedene Quellen und Arten von Wind zu ermöglichen, wird ein System von Windebene eingeführt. Jede Ebene bekommt eine Windart zugewiesen (eine ganze Zahl vom Datentyp `int`) und stellt zwei Vektoren mit jeweils vier Elementen bereit, die für das Speichern von Windart-abhängigen Daten genutzt werden. Eine Windquelle belegt somit eine Windebene, wobei mehrere Windquellen einer Art möglich sind. Die maximale Anzahl an Windebene wird im Vorfeld vom Benutzer definiert, da zum Übertragen der Windebene auf die GPU ein Buffer fester Größe erzeugt wird. Deaktivierte Windquellen bzw. ungenutzte Windebene werden durch die Windart `-1` markiert und ans Ende des Buffers sortiert. Zu Beginn jedes *Frames* werden alle Windquellen aktualisiert und die neuen Daten in den Buffer geschrieben. Für die vorliegende

Arbeit werden zwei verschiedene Windarten verwendet, was jedoch keine Limitierung des Systems auf nur wenige Windarten impliziert.

Ein gerichteter Wind, basierend auf dem „directional wind“ aus [Jah16] stellt die Windart 0 dar. Auf der CPU werden die Windrichtung und Windstärke λ_{wd} , ein *Offset* t_{wd} und ein dreielementiger Vektor \mathbf{f}_{wd} mit Frequenzen bestimmt. Die Windrichtung ändert sich nach einer zufälligen Dauer (die in einem benutzerdefinierten Intervall liegt) zu einer neuen zufälligen Richtung. Dabei wird zwischen der alten und neuen Windrichtung linear interpoliert, damit starke Änderungen der Richtung nur bei kurzem zeitlichen Abstand der Richtungsänderung sichtbar sind. Gleiches gilt für die Windstärke. Auf das *Offset* t_{wd} wird die vergangene Zeit seit dem letztem *Frame* in Sekunden addiert. Die Frequenzen werden vom Benutzer definiert. Nach Aktualisieren der Werte werden diese in den Buffer geschrieben.

Ein Wirbelwind ist die Windart 1. Er ist verbunden mit einer Position in Weltkoordinaten \mathbf{p}_{ww} , einem Radius r_{ww} und einer Windstärke m_{ww} . Die Windstärke wird wie beim gerichteten Wind bestimmt, der Radius vom Benutzer definiert. Auch diese Daten werden nach dem Aktualisieren in den Buffer geschrieben.

Summieren der Winde auf der GPU Im *Compute-Shader* wird für jeden Grashalm der Buffer mit Windebene durchlaufen und abhängig von der Windart der Ebene eine entsprechende Auswertung vorgenommen. Dabei wird die Windrichtung und Windstärke, die auf den Punkt in der Welt einwirkt von allen Windebene addiert \mathbf{w} . Sobald als Windart die -1 in einer Windebene eingetragen ist, kann der Schleifendurchlauf abgebrochen werden.

Für den gerichteten Wind wird ein Windeinfluss i_{wd} berechnet, mit dem Wind λ_{wd} multipliziert und das Produkt auf \mathbf{w} addiert. Die Formel für den Windeinfluss i_{wd} stammt von Jahrman [Jah16], wird jedoch um benutzerdefinierte Frequenzen erweitert:

$$i_{wd} = 1 - \frac{1}{3} \max(\cos(\mathbf{f}_{wdx}(\mathbf{p}_x + \mathbf{p}_z) + t_{wd}) + \sin(\mathbf{f}_{wdy}(\mathbf{p}_x + \mathbf{p}_y) + t_{wd}) + \sin(\mathbf{f}_{wdz}(\mathbf{p}_y + \mathbf{p}_z) + t_{wd}), 0) \quad (36)$$

Der Wirbelwind hat nur Einfluss auf das Gras, dessen Position auf dem Boden \mathbf{p} im Radius der durch Position \mathbf{p}_{ww} und Radius r_{ww} definierten Kugel liegt. Ist dies der Fall, erzeugt der Wirbelwind einen Wirbel, der Grashalme um die y -Achse wirbelt und weg vom Zentrum drückt:

$$\begin{aligned} d\mathbf{w} &= \text{normalize}(\mathbf{p} - \mathbf{p}_{ww}) \\ i_{ww} &= \text{lerp}\left(1, 0, \text{clamp}\left(\frac{\|d\mathbf{w}\|}{r_{ww}}, 0, 1\right)\right) \\ \mathbf{w}_{xz} &= \mathbf{w}_{xz} + i_{ww}m_{ww} \left(\frac{3}{4} [d\mathbf{w}_z \quad -d\mathbf{w}_x] + \frac{1}{4} d\mathbf{w}_{xz}\right) \end{aligned} \quad (37)$$

Reaktion von Gras auf Wind Mit dem Vektor w steht die Windrichtung und -stärke fest. Um den Einfluss auf den Grashalm zu berechnen, werden drei Kriterien nach [Jah16] beachtet:

1. Aufrecht auf dem Terrain ausgerichtetes Gras wird stärker beeinflusst.
2. Wind hat stärkeren Einfluss, wenn er senkrecht auf die Grasfläche wirkt.
3. Biegsames Gras wird stärker beeinflusst.

Das erste Kriterium wh lässt sich berechnen mit dem Skalarprodukt der Terrain Normale und dem Vektor v_2 :

$$wh = ||v_2|| \cdot n_t \quad (38)$$

Das zweite Kriterium ergibt sich aus dem Skalarprodukt der Windrichtung und des Vektors $front$, der auf der Grasfläche steht:

$$wd = ||w|| \cdot front \quad (39)$$

Der Biegefaktor b vom Gras stellt den Wert für das dritte Kriterium. Damit lässt sich die Kraft des Windes auf das Gras berechnen mit:

$$wind = wh * wd * b * w \quad (40)$$

Das Verändern von Breite und Ausrichtung beim Rendering von Billboards führt zu einer Diskrepanz zwischen der Darstellung und der Simulation, die jedoch hingenommen wird.

3.5.6 Steifheit

Die Steifheit beschreibt eine Kraft basierend auf dem *hookeschen Gesetz*, welche die Spitze vom Gras in Richtung seiner Ruheposition drückt und so eine Erholung von Gras bewirkt [JW17]. Neben dem Biegefaktor wird die Steifheit durch die Stärke von Kollisionen cs beeinflusst. Dadurch wird ermöglicht, dass sich Gras von heftigen Kollisionen langsamer erholt [Jah16]. Die Berechnung der Steifheit st ist wie folgt definiert:

$$st = (hup - v_2) \left(1 - \frac{b}{2}\right) \max(1 - cs, 0) \quad (41)$$

Bevor die Kollisionsbehandlung erfolgt, werden die bis dahin berechneten Kräfte auf die Spitze v_2 addiert und mit der Zeit Δt multipliziert, die seit dem letzten *Frame* vergangen ist, gefolgt von der Korrektur der Vektoren v_1 und v_2 nach den Gleichungen 34, 32 und 30.

$$v_2 = v_2 + \Delta t (g + wind + st) \quad (42)$$

3.5.7 Kollisionen

Zur Kollision wird ein Verfahren eingeführt, das auf den Arbeiten [Wan+09; Mak15] basiert. Eine orthographische Kamera wird so unter dem Terrain platziert, dass ihr Ansichtsvolumen das gesamte Terrain und darauf wachsendes Gras umhüllt. Nur Objekte, die als Kollisionsobjekte markiert sind, werden zu Beginn jedes *Frames* aus der Sicht der orthographischen Kamera in eine Textur gerendert. Die Textur besitzt 4 float Kanäle und hat eine Auflösung entsprechend der summierten Auflösung einer Simulationstextur aller *Patches*. Im *Pixel-Shader* wird in den Alpha Kanal der Tiefenwert a und in die RGB-Kanäle der Geschwindigkeitsvektor v geschrieben. Die Kollisionserkennung und -behandlung findet im *Compute-Shader* statt.

Unabhängig davon findet eine Kollisionsbehandlung zwischen Kollisionsobjekten und Terrain statt. Dafür wird das von Unity bereitgestellte System verwendet.

Rendering der Kollisionsobjekte Grundsätzlich spielt die Form von Kollisionsobjekten keine Rolle, da sowohl konvexe und konkave Geometrie gerendert werden kann. Es bietet sich an, dass Kollisionsobjekte beim Rendering der Kollisions-Textur durch ein Objekt mit geringer Anzahl an Eckpunkten dargestellt wird (siehe Abb. 23). Als Geschwindigkeitsvektor v wird in der vorliegenden Arbeit die Geschwindigkeit des Objekts verwendet. Eine Erweiterung ist möglich, indem die relative Geschwindigkeit jedes Eckpunkts in die Kollisions-Textur geschrieben wird, um auch den Einfluss von Objektrotationen mit einzubeziehen. Der Tiefenwert entspricht der z -Komponente des Eckpunktes nach der perspektivischen Transformation und liegt im Intervall $]0, 1[$, mit dem Wert 0 an der *Far* und 1 an der *Near Clipping Plane*. An der Stelle ist anzumerken, dass *Direct3D* eine zeilendominierte Reihenfolge verwendet [Coro]f].

Kollisionserkennung Im Gegensatz zu anderen Verfahren ist es beim beschriebenen Vorgehen nicht nötig, eine grobe Kollisionserkennung voranzustellen. Im *Compute-Shader* der Simulation wird nach dem Anwenden von Schwerkraft, Wind und Steifheit getestet, ob eine Kollision vorliegt. Dazu wird zuerst der Kontrollpunkt $p + v_2$ getestet und bei fehlgeschlagenem Test der mittlere Punkt $p + m$ der Bézierkurve. Dieser wird nach [JW17] wie folgt berechnet wird:

$$m = \frac{1}{2}v_1 + \frac{1}{4}v_2 \quad (43)$$

Zunächst wird die Position des zu testenden Punktes (i. F. beschrieben für $p + v_2$) im Ansichtsvolumen der orthographischen Kamera berechnet. Dazu werden die Weltkoordinaten des Punktes mit der Ansichtsmatrix V (engl. *view matrix*) und der Projektionsmatrix P (engl. *projection matrix*) der

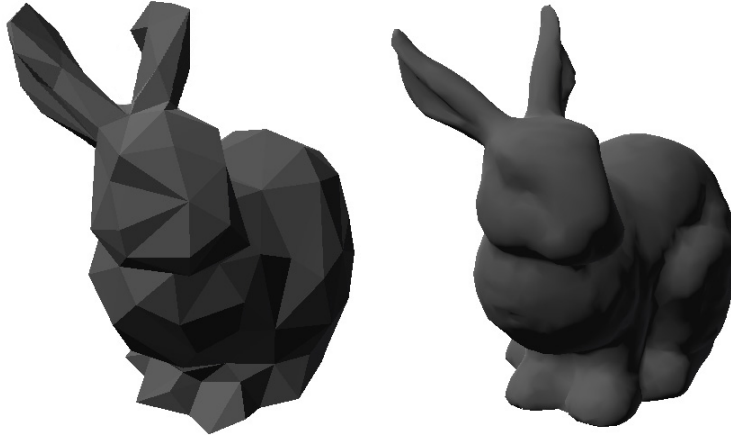


Abb. 23: Modell des *Stanford Bunny* [Lev14] zum Rendering als Kollisionsobjekt (links) und zum normalen Rendering in der Szene (rechts)

Kamera multipliziert und so skaliert, dass sie als 2D-Koordinate für die Kollisions-Textur herangezogen werden können:

$$\begin{aligned} \mathbf{v}_2' &= (\mathbf{p} + \mathbf{v}_2) \mathbf{VP} \\ v_{2'xy} &= \frac{1}{2}(v_{2'xy} + 1) \end{aligned} \quad (44)$$

Mit $v_{2'xy}$ kann der Tiefenwert a aus der Kollisions-Textur gelesen werden. Ist dieser größer als $v_{2'z}$, wird angenommen, dass es eine Kollision gibt. Der Test für den mittleren Punkt der Bézierkurve folgt dem gleichen Prinzip. Wurde für beide Punkte keine Kollision erkannt, ist die Bearbeitung der Simulation für das aktuell betrachtete Gras abgeschlossen. Für den Fall, dass sich ein Kollisionsobjekt unter dem Terrain befindet, kann bei Bedarf ein weiterer Vergleich mit dem Tiefenwert der Gras Position \mathbf{p} eingeführt werden. In der vorliegenden Arbeiten verursachen auch solche Objekte eine Kollision.

Kollisionsreaktion In Abb. 24 ist die Reaktion auf die Kollision illustriert. Für die Berechnung der Kraft einer Kollision wird zuerst die Eindringtiefe c_p des Grasses in das Kollisionsobjekt bestimmt. Um zu große Eindringtiefen von Kollisionsobjekten unterhalb des Terrains zu vermeiden, wird als maximale Größe der Eindringtiefe die Höhe des Grashalms verwendet. Es sei c_h die Höhe des Kameravolumens in der Welt.

$$c_p = \min(c_h(a - v_{2'z}), h) \quad (45)$$

Die Richtung der resultierenden Kollisionskraft setzt sich zusammen aus einer Richtung entlang der Bewegungsrichtung des Kollisionsobjekts \mathbf{v} und

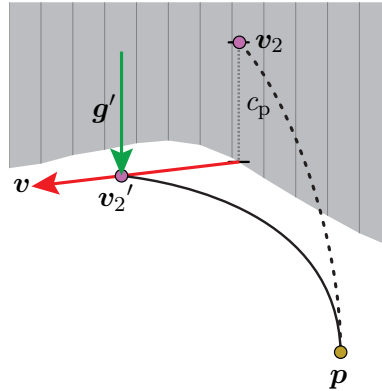


Abb. 24: Illustration der Kollisionserkennung zwischen einem Kollisionsobjekt (grau) und Gras sowie der Kollisionsreaktion

entlang der Schwerkraft g . Die resultierende Auswirkung auf v_2 ist damit:

$$v_2' = v_2 + \frac{c_p}{h} (v + g) \quad (46)$$

Wie in der Arbeit [JW17] kann nur der Vektor v_2 modifiziert werden. Deshalb wird auch in der vorliegenden Arbeit die Reaktion auf eine Kollision mit dem mittleren Punkt $p + m$ durch Multiplikation der resultierenden Kollisionskraft mit dem Faktor 4 auf den Vektor v_2 übertragen [JW17]. Anschließend erfolgt erneut die Korrektur der Vektoren v_1 und v_2 nach den Gleichungen 34, 32 und 30.

Die für die Steifheit benötigte Stärke der Kollision cs wird im Alpha Kanal einer Simulationstextur gespeichert. Zu Beginn der Simulation wird der Wert gelesen und abhängig von b , Δt und einem benutzerdefinierten Faktor cr verringert:

$$cs = \max (cs - (1.0 - b) cr \Delta t, 0) \quad (47)$$

Wird eine Kollision erkannt, wird die neue Stärke auf die bestehen Kollisionsstärke addiert und der aktualisierte Wert in der Simulationstextur gespeichert. Die Stärke einer Kollision entspricht der Länge des Vektors von v_2 vor der Kollision zu v_2' nach der Kollision und Korrektur.

Qualität der Kollisionsbehandlung Die vorgestellte Kollisionsbehandlung erzielt in den meisten Fällen einen optisch guten Eindruck einer Kollision (siehe Abb.25). Kollisionsobjekte, die zu klein sind und keinen der beiden getesteten Punkte auf dem Gras zu berühren, werden nicht erkannt. Außerdem wird nicht sichergestellt, dass Gras nach der Reaktion auf die Kollision tatsächlich außerhalb eines Kollisionsobjekts verweilt. Für Terrains mit sehr großen Höhenunterschieden, könnte die Genauigkeit der Tie-

fenwerte nicht ausreichend sein. Im Unterabschnitt 4.2.4 wird das erzielte Ergebnis genauer betrachtet.



Abb. 25: *Stanford Bunny* [Lev14] hinterlässt eine Spur von niedergedrücktem Gras

3.6 Rendering

Das Rendering der verschiedenen Grasdarstellungen funktioniert sehr ähnlich, weshalb zunächst der vollständige Renderingprozess geometrischer Grashalme beschrieben wird, gefolgt vom Rendering der Blüten. Danach kann das Erstellen der Billboards Texturen beschrieben und auf Besonderheiten bei deren Rendering eingegangen werden. Die Beleuchtung im *Pixel-Shader* wird in Abschnitt 3.7 gesondert betrachtet.

3.6.1 Geometrische Grashalme

Zum Rendering werden *Vertex-Shader*, *Hull-Shader*, *Domain-Shader* und *Pixel-Shader* der Renderpipeline von *Direct3D 11* programmiert. Es wird nun vorgestellt, was in diesen Phasen geschieht.

Vertex-Shader Der Vertex-Buffer enthält $v_{\text{geo}} = l^2$ Dummy Punkt-Primitive. Im *Vertex-Shader* werden nur die vom *Input-Assembler* erzeugten Werte für *Vertex ID* vid und *Instance ID* iid verwendet, um den Index i_b zum Zugriff auf den *Instanced-Buffer* (siehe Abschnitt 3.2) zu berechnen. Dazu wird jedem *Patch* beim Erstellen ein zufälliger Startindex i_s zugewiesen. Es gilt:

$$i_b = i_s + v_{\text{geo}} * iid + vid \quad (48)$$

Damit kann uv_{Local} und die Grasart aus dem *Instanced-Buffer* gelesen werden.

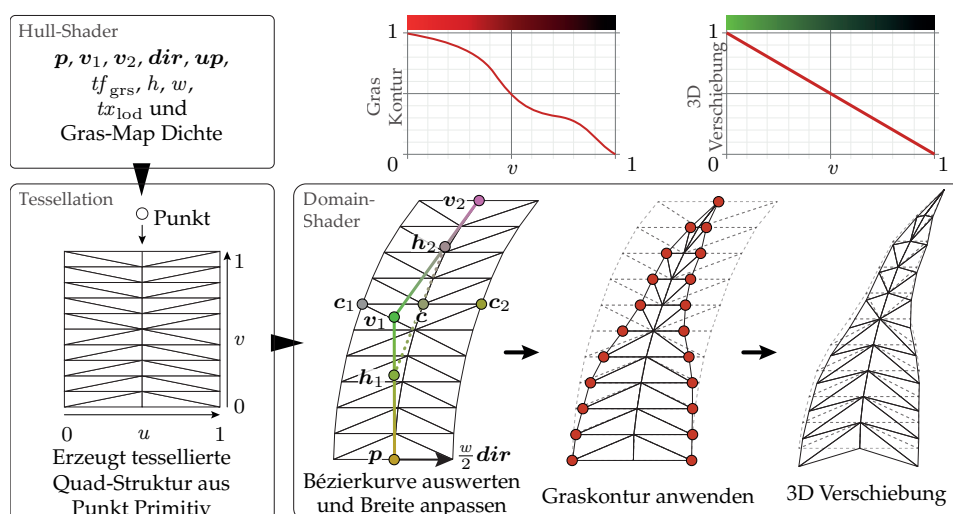


Abb. 26: Generierung der Geometrie eines Grashalms in der Tessellations-Phase

Hull-Shader und Tessellation Der *Hull-Shader* berechnet den Tessellations-Faktor tf_{grs} (siehe Abschnitt 3.3.3) für die Tessellation und kontrolliert damit die Anzahl an Unterteilungen eines Grashalms. In Abb. 26 ist das Vorgehen der drei Phasen zur Expansion des Gras-Modells illustriert. Im Unterschied zu [JW17] wird in der vorliegenden Arbeit das *Culling* in dieser Phase ausgeführt.

Für die Tessellation einer Quad-Struktur werden vier äußere und zwei innere Faktoren benötigt. Die beiden äußeren und der innere Faktor entlang der Vertikalen v entsprechen tf_{grs} . Soll ein Grashalm aussortiert werden (siehe Unterabschnitt 3.3.2), wird $tf_{grs} = 0$ gesetzt, wodurch die folgenden Phasen nicht ausgeführt werden. Die für die Horizontale u zuständigen äußeren Faktoren seien 2, der innere sei 1. Die Tessellation-Phase erzeugt aus der Eingabe (einem einzelnen Kontrollpunkt) den Tessellations-Faktoren entsprechende 2D-Ausgabekoordinaten u und v im Intervall $]0, 1[$. Für jede dieser Ausgabekoordinaten wird der *Domain-Shader* einmal aufgerufen. In Abb. 26 ist der Vorgang für $tf_{grs} = 10$ illustriert, wodurch 33 solcher Ausgabekoordinaten erstellt werden.

Weiterhin finden Berechnungen und Texturzugriffe statt, die nur einmal pro Grashalm ausgeführt werden müssen. Dazu gehören die Position p in Weltkoordinaten, die Entfernung d_{grs} von p zur Kamera sowie weitere beschreibende Parameter des Grashalms (siehe Abb. 26). Die Vektoren v_1 und v_2 werden mit dem Höhenfaktor zur LOD Überblendung trf_{grs} skaliert. Alle vom *Domain-Shader* benötigten Werte werden an diesen weitergereicht.

Domain-Shader In dieser Phase werden die ortsunabhängigen 2D-Ausgabekoordinaten so transformiert, dass ein Grashalm geformt und an der

entsprechenden Position in der Welt positioniert wird. Wie in [JW17] wird dazu die Bézierkurve mithilfe des „De-Casteljau-Algorithmus“ ausgewertet. Die v -Koordinate dient dabei als Wert zur Interpolation. Die Position auf der Kurve c (siehe Abb. 26) wird berechnet mit:

$$\begin{aligned} \mathbf{h}_1 &= \mathbf{p} + v\mathbf{v}_1 \\ \mathbf{h}_2 &= \mathbf{p} + \mathbf{v}_1 + v(\mathbf{v}_2 - \mathbf{v}_1) \\ \mathbf{c} &= \mathbf{h}_1 + v(\mathbf{h}_2 - \mathbf{h}_1) \end{aligned} \quad (49)$$

Die Punkte \mathbf{c}_1 und \mathbf{c}_2 werden durch eine Translation mit der Richtung und Breite bestimmt:

$$\begin{aligned} \mathbf{c}_1 &= \mathbf{c} - \frac{w}{2} \mathbf{dir} \\ \mathbf{c}_2 &= \mathbf{c} + \frac{w}{2} \mathbf{dir} \end{aligned} \quad (50)$$

Wie Jahrmann [Jah16] feststellt, ist es ein bedeutender Vorteil, dass sich mit dem „De-Casteljau-Algorithmus“ die Tangente \mathbf{t} einfach berechnen lässt. Zusammen mit der Bitangenten \mathbf{dir} kann die Normale \mathbf{n} mit dem Kreuzprodukt beider Vektoren bestimmt werden [JW17].

$$\begin{aligned} \mathbf{t} &= \text{normalize}(\mathbf{h}_2 - \mathbf{h}_1) \\ \mathbf{n} &= \mathbf{dir} \times \mathbf{t} \end{aligned} \quad (51)$$

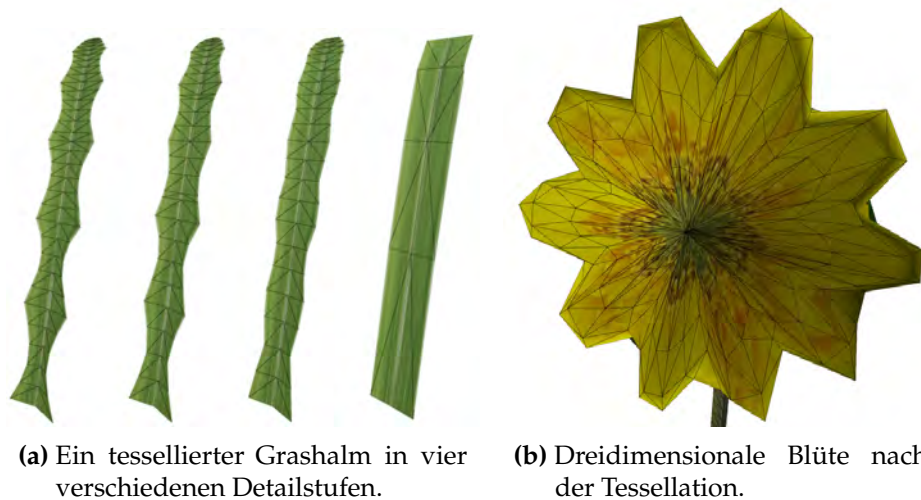
Für die Kontur wird die 1D-Textur, beschrieben in Abschnitt 3.4.2 herangezogen. Da es mehrere Arten von Gras gibt, wird eine 1D-Textur für jede Grasart erstellt und in einem 1D-Textur Array angeordnet. Die Grasart gi dient der Wahl des passenden Array Elements und v als Texturkoordinate, i. F. gegeben als $\mathbf{tex}_{v,gi}$. Unter Verwendung von u und \mathbf{tex}_{v,gi_x} kann nun die Position des Eckpunkts $\mathbf{p}_{u,v}$ vom Grashalm (in Abb. 26 rot dargestellt) berechnet werden:

$$\mathbf{p}_{u,v} = \text{lerp}(\mathbf{c}_1, \mathbf{c}_2, 0.5 + (u - 0.5)\mathbf{tex}_{v,gi_x}) \quad (52)$$

Optional kann eine 3D-Verschiebung nach [JW17] angewendet werden. Diese zieht jedoch keine Änderung der Normale mit sich, weshalb sie nur bei Gras nahe der Kamera wahrgenommen wird. Berechnet wird sie durch Addition eines Anteils der Normale auf den Eckpunkt $\mathbf{p}_{u,v}$ und kann in der vorliegenden Arbeit durch die 1D-Textur kontrolliert werden:

$$\mathbf{p}'_{u,v} = \mathbf{p}_{u,v} + w * \mathbf{tex}_{v,gi_y} * |u - 0.5| * \mathbf{n} \quad (53)$$

In Abb. 27a ist ein einzelner geometrischer Grashalm in vier verschiedenen Detailstufen zu sehen. In niedrigen Detailstufen wird der Grashalm als Antialiasing Maßnahme breiter, indem die Werte der Kontur in kleineren *Mip-Maps* gegen 1 laufen.



(a) Ein tessellierter Grashalm in vier verschiedenen Detailstufen.

(b) Dreidimensionale Blüte nach der Tessellation.

Abb. 27: Geometrische Darstellung von Gras und Blüten.

3.6.2 Blüten

Zur Demonstration der Flexibilität des Prinzips, aus einer tessellierten Fläche Geometrie zu formen, sowie der Erweiterbarkeit des Gras-Modells mit einer Bézierkurve, wird ein Verfahren zum Rendering geometrischer Blüten vorgestellt. Für jeden Punkt auf der Bézierkurven ist die Position in der Welt berechenbar und mit der Tangenten t , der Bitangenten dir und der Normale n ein Koordinatensystem definiert. Damit lässt sich ein beliebiges Objekt an jede Position des Grashalms transformieren und entsprechend seiner Haltung ausrichten.

Zur geometrischen Darstellung einer Blüte wird, wie beim Grashalm, durch Tessellation eine Quad-Struktur erzeugt. Diese wird durch Auslesen von Funktionswerten aus einer 1D-Textur (float) in Form gebracht und auf die Spitze des Grashalms platziert. Um verschiedene Blütenarten zu ermöglichen, wird wieder ein Textur Array eingesetzt. Die Anzahl der Blüten kann durch Reduzieren der *Instances* oder durch das Verknüpfen mit bestimmten Grasarten eingeschränkt werden. Dazu kann wieder das *Culling* durch einen Tessellations-Faktor von 0 eingesetzt werden.

In Abb. 28 ist die Tessellation von einem Viertel der Blüte illustriert (links) und die Werte einer 1D-Textur für eine Blüte abgebildet (rechts). Zunächst wird wie beim Rendering von Grashalmen vorgegangen, jedoch wird die Bézierkurve nur einmal an deren Spitze ausgewertet. Der Tessellations-Faktor der Blüte tf_{bls} wird für alle äußeren und inneren Faktoren verwendet, er muss jedoch gerade sein, damit die Koordinate im Mittelpunkt $u = v = 0,5$ erzeugt wird. Die Koordinate zum Auslesen der Textur t_b wird so gewählt, dass sie an der mittleren Koordinate 0 und an den Rändern 1 ergibt. Wenn möglich, zeigt ein 2D-Einheitsvektor uv_{dir} von der Mitte in

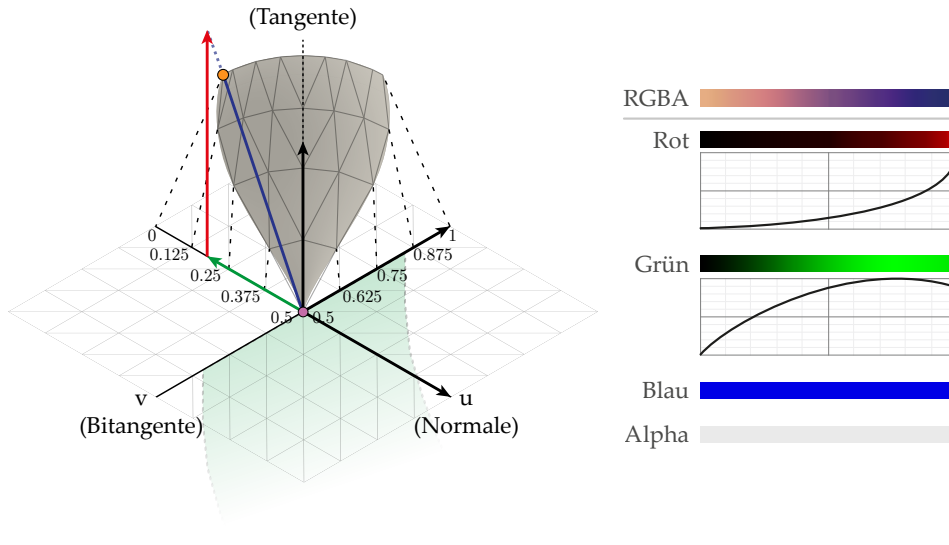


Abb. 28: Veranschaulichung der Erzeugung geometrischer Blüten durch Tessellation. Gezeigt wird ein Viertel der Blütegeometrie (links) und die verwendete 1D-Textur (rechts) als *Lookup Tabelle*.

Richtung des Randes:

$$t_b = \max(|2(u - 0,5)|, |2(v - 0,5)|)$$

$$\mathbf{uv}_{\text{dir}} = \begin{cases} \begin{bmatrix} 0 & 0 \end{bmatrix} & u = v = 0,5 \\ \text{normalize} \left(2 \begin{bmatrix} u - 0,5 & v - 0,5 \end{bmatrix} \right) & \text{sonst} \end{cases} \quad (54)$$

Der grüne Kanal der Textur stellt den Faktor g_b zur Translation entlang der Richtung \mathbf{uv}_{dir} bereit und der rote Kanal den Faktor r_b zur Translation entlang der Tangente. Der blaue Kanal ist ein allgemeiner Skalierungsfaktor b_b , der separat und durch Verwenden des skalierten Polarwinkels φ_b als Texturkoordinate gelesen wird.

$$\varphi_b = \begin{cases} 0 & u = v = 0,5 \\ 1 + \frac{1}{2\pi} \arctan2(\mathbf{uv}_{\text{dir}x}, \mathbf{uv}_{\text{dir}y}) & \text{sonst} \end{cases} \quad (55)$$

Die Position eines Eckpunkts \mathbf{p}_b in Weltkoordinaten lässt sich nun berechnen mit:

$$\mathbf{uv}_{\text{dir}}' = g_b w \mathbf{uv}_{\text{dir}}$$

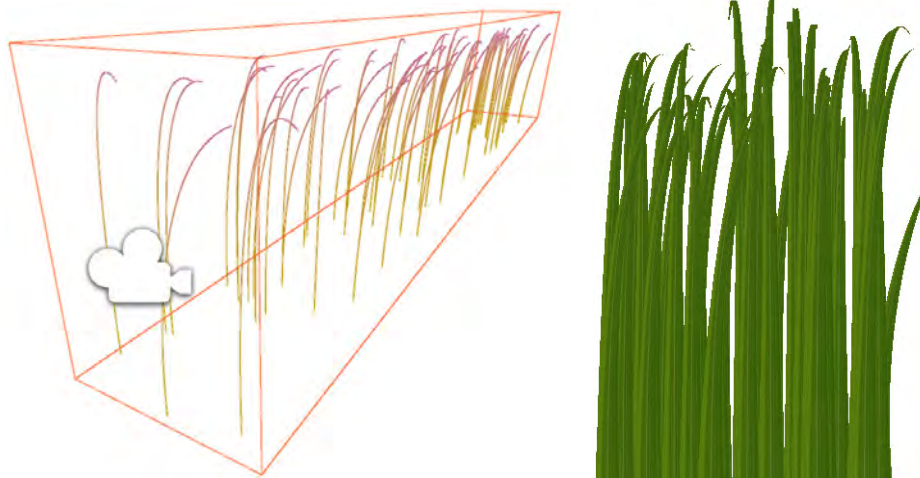
$$\mathbf{p}_b = \mathbf{p} + \mathbf{v}_2 + b_b (\mathbf{uv}_{\text{dir}x} * \mathbf{n} + \mathbf{uv}_{\text{dir}y} * \mathbf{dir} + r_b * \mathbf{t}) \quad (56)$$

Da keine Ableitung gegeben ist, muss zur Berechnung der Oberflächennormale zunächst die Steigung zwischen zwei Punkten berechnet werden. Zum Verringern von Aliasing können weiter entfernte Blüten der Kamera zugeneigt werden.

In Abb. 27a ist eine Blüte dargestellt, die nach dem vorgestellten Prinzip erstellt wurde.

3.6.3 Generierung von Billboard Texturen

Wie in den Arbeiten [PC01; BBP06] wird die geometrische Grasdarstellung von einer orthographischen Kamera in eine Textur gerendert, welche dann als Textur für Billboards verwendet wird. Dies geschieht einmal für jede Grasart als Vorverarbeitungsschritt.



(a) Die orthographische Kamera umschließt das Gras für das Rendern von Billboard-Texturen.

(b) Ergebnis des Rendering von Gras in eine Billboard-Textur.

Abb. 29: Eine Billboard-Textur wird in der Vorverarbeitung aus der geometrischen Grasdarstellung erzeugt.

Zunächst wird ein *Gras-Patch* auf flachem Grund initialisiert und die Simulation mit fixem Zeitschritt $\Delta t = 1$ ausgeführt. Die Vektoren v_1 und v_2 werden auf der CPU für die gewünschte Anzahl an Grashalmen in der Billboard-Textur aus den Simulationstexturen ausgelesen. Die Bézierkurve wird ausgewertet und die minimalen und maximalen Weltkoordinaten bestimmt, um das Ansichtsvolumen der orthographischen Kamera die Grashalme einschließen zu lassen (siehe Abb. 29a). Zusätzlich ist eine Anpassung des Ansichtsvolumens entsprechend der maximalen Grasbreite nötig. Anschließend wird jede Grasart einmal aus der Sicht der orthographischen Kamera unbeleuchtet gerendert (siehe Abb. 29b). Um keine zu dünne Geometrie darzustellen, wird Gras orthogonal zur Kamera in seiner Breite expandiert. Wenn Blüten mit einer Grasart verbunden sind, werden diese ebenfalls in die Billboard-Textur gezeichnet.

Die verschiedenen *Mip-Map* Stufen der Textur werden manuell und

Transparenz-erhaltend erzeugt, da die Transparenz im Alphakanal beim automatischen Erzeugen stark verwischt wird und bei niedriger Auflösung eine halbtransparente Fläche entsteht. Die Maße aller Texturen sind gleich und werden in einem 2D-Textur Array gespeichert, wobei die Grasart den Index darstellt.

3.6.4 Billboards

Das Rendering von Billboards unterscheidet sich nur in wenigen Punkten von dem der geometrischen Grasdarstellung. Da transparente Texturen auf die Billboards gezeichnet werden, wird wie bei [ORK09] *Alpha to Coverage* verwendet. Im Unterschied zu *Alpha-Blending* (dt. etwa Alpha-Mischung) muss keine Tiefensortierung der Geometrie erfolgen und im Unterschied zu *Transparency-Cutout* (dt. etwa Transparenz-Ausschnitt) entstehen dank *Multisampling* (dt. Mehrfachabtastung) weniger harte Kantenübergänge [KCS07].

Die Anzahl an Gras pro Patch bei Billboards ist mit $v_{\text{cro}} = v_{\text{scr}} = l = 8$ geringer als die von geometrischem Gras. Gras in gekreuzter Billboard Darstellung besteht jeweils aus drei gekreuzten Billboards, die an der gleichen Position in der gleichen Haltung stehen, jedoch sternförmig angeordnet sind (siehe Gleichung 25). Dazu gibt es $v_{\text{cro}} * 3$ Dummy Punkt-Primitive und der Index i_b zum Zugriff auf den *Instanced-Buffer* berechnet sich mithilfe einer Modulo Operation mod:

$$i_b = i_s + v_{\text{cro}} * iid + (vid \bmod v_{\text{cro}}) \quad (57)$$

Der Index für *screen-facing* Billboards wird berechnet wie der von geometrischem Gras, jedoch mit v_{scr} statt v_{geo} und es gibt nur v_{scr} Punkt-Primitive pro *Instance*. Die Ausrichtung ergibt sich nach Gleichung 26.

Wird ein Billboard nicht durch *Culling* aussortiert, erhält es einen Faktor von 1 für alle äußeren und inneren Faktoren der Tessellation. Es entsteht eine Quad-Struktur bestehend aus zwei Dreiecken. Die Bézierkurve wird für die Koordinaten im *Domain-Shader* wie beim geometrischen Gras ausgewertet. Die Breite wird nach Gleichung 24 berechnet. Die Position eines Eckpunktes ergibt sich aus der linearen Interpolation zwischen c_1 und c_2 mit u als Wert zur Interpolation.

3.7 Beleuchtung

Zunächst sind Texturen maßgeblich für die Darstellungsqualität verantwortlich. Für jede Grasart und jede Blüte ist eine Oberflächen Textur in einem weiteren Textur Array gespeichert. Zur Beleuchtung wird das Lambert Beleuchtungsmodell mit ambienter und Lichtdurchlässigkeits-Komponente von Boulanger [Bou08] verwendet und für eine einzige gerichtete Lichtquelle ausgewertet.

Der diffuse Reflexionsfaktor K_d und der Lichtdurchlässigkeitsfaktor γ werden aus der 1D-Textur der jeweiligen Grasart gelesen, sodass diese entlang der Wuchsrichtung variiert werden können. Für Blüten ist nur der Reflexionsfaktor variabel, der Lichtdurchlässigkeitsfaktor ist stets 1. Der Faktor der Umgebungsverdeckung A_o ist nahe am Boden hoch, an der Spitze niedrig und für weniger dichtes Gras niedriger. Die Intensität von ambienten Licht sei I_a , die der Lichtquelle I_d , deren Richtung \mathbf{L} und die Oberflächennormale \mathbf{N} . Eine Lichtabnahme oder Distanz zur Lichtquelle wird nicht berücksichtigt, da es sich um ein Sonnenlicht handelt. Es folgt die Strahlungsintensität I_r :

$$I_r = K_d A_o I_a + K_d I_d (\max(\mathbf{N} \cdot \mathbf{L}, 0) + \gamma \max(-\mathbf{N} \cdot \mathbf{L}, 0)) \quad (58)$$

4 Ergebnisse und Bewertung

In mehreren Performance Tests werden einzelne Techniken der Arbeit analysiert. Der Vergleich mit bestehenden Arbeiten untersucht die visuelle Qualität und die Möglichkeiten verwendeter Techniken. Ein Performance Vergleich mit bestehenden Arbeit ist aufgrund unterschiedlicher Testumgebungen nicht möglich. Anschließend werden Verbesserungsmöglichkeiten für verwendete Verfahren aufgezeigt.

4.1 Performance

Der Test unterteilt sich in die Betrachtung der Performance des Rendering und der Simulation. Hierzu werden Szenen in unterschiedlichen Qualitätsstufen und anderen variierenden Faktoren ausgewertet und miteinander verglichen. Alle Tests werden auf einem Heimcomputer mit *Windows 10 64-Bit* Betriebssystem, *NVIDIA GeForce GTX 1070* Grafikkarte, *Intel Core i5-4670 @ 3.40GHz* CPU und 16 GB Arbeitsspeicher bei einer Auflösung von 1920×1080 Pixeln und 8-fachem *Multisampling* ausgeführt. Zur exakten Analyse der Aktivität auf der GPU wird das „GPU-Nutzungstool“ von „Visual Studio“ verwendet. Die damit gemessenen Zeiten fallen höher aus, da die Aufnahme der Daten einen Mehraufwand darstellt. Die Daten sind dennoch von großem Wert, da sie die relativen Zeiten der einzelnen Aktivitäten auf GPU zeigen. Die gemessenen FPS werden ohne Aktivität des Tools ermittelt.

4.1.1 Beschreibung der Tests

Der Terrainuntergrund wird bei allen Tests ausgeblendet, um nur die Performance des entwickelten Gras-Systems zu testen. Es gibt 14 verschiedene Grasarten und 4 verschiedene Blütenarten. Die Testszene *A* (siehe Abb. 30) zeigt eine Gesamtansicht der Szene mit 1.024 sichtbaren *Gras-Patches*. Die Testszene *B* zeigt einen Ausschnitt in der Nahaufnahme, bei der 8 *Gras-Patches* im Kamera Frustums liegen, jedoch nur ein *Patch* vollständig zu sehen ist. Tabelle 2 zeigt die Parameter verschiedener Qualitätsstufen, die im Folgenden verwendet werden. Die Qualitätsstufen *Hoch*, *Normal* und *Niedrig* sollen einen Überblick der Performance des Systems liefern, durch variieren der Menge von Gras einzelner Darstellungsmethoden und der Detaillierung von geometrischem Gras. *Tess.A* und *Tess.B* dienen dem Vergleich der Detaillierungsstufe von geometrischem Gras durch Tessellation. Weiterhin wird der Einfluss von Kollisionsobjekten und der Auflösung von Simulationstexturen auf die Performance analysiert. Es gibt 3 verschiedene Kollisionsobjekte, diese sind in aufsteigender Anzahl der Eckpunkte: ein Würfel, eine Kugel und der *Stanford Bunny*. Daraus ergeben sich die 7 Test-szenarien und Ergebnisse in Tabelle 3.

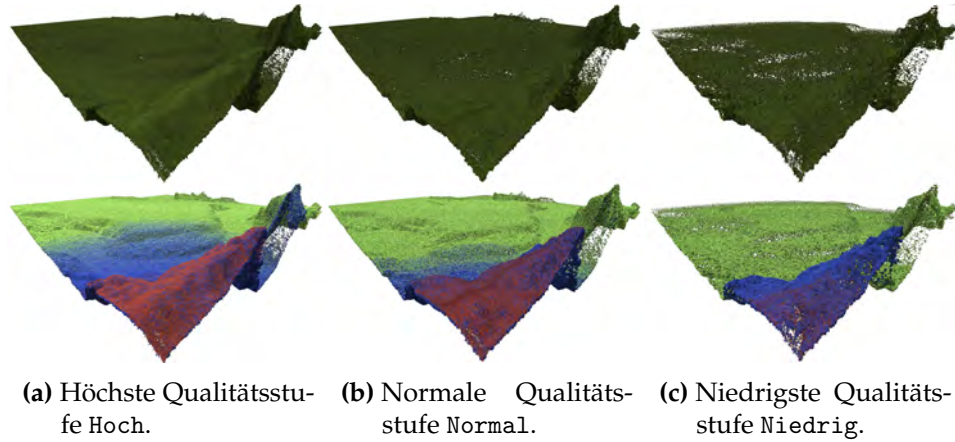


Abb. 30: Die Testszene A mit der Gesamtansicht eines bewachsenen Terrains ohne Untergrund in drei verschiedenen Qualitätsstufen, jeweils mit Visualisierung der Detailstufen. Rot zeigt geometrisches Gras, blau gekreuzte Billboards und grün *screen-facing* Billboards.

Parameter	Symbol	Hoch	Normal	Niedrig	Tess.A	Tess.B
Patch Größe	l	8	8	8	8	8
Min. Tessellations-Faktor	tf_{\min}	4	3	2	64	8
Max. Tessellations-Faktor	tf_{\max}	40	32	16	64	8
Min. Tessellation Distanz	td_{\min}	4	4	0	48	48
Max. Tessellation Distanz	td_{\min}	32	24	16	64	64
Geometrische <i>Instances</i>	n_{geo}	512	384	192	512	512
Überblendete <i>Instances</i>	k_{geo}	32	32	32	32	32
Startdistanz Überblendung	$d_{\text{s,geo}}$	6	4	0	48	48
Enddistanz Überblendung	$d_{\text{e,geo}}$	64	48	40	64	64
Gekreuzte Billboard <i>Instances</i>	n_{cro}	256	160	96	0	0
Überblendete <i>Instances</i>	k_{cro}	32	32	32	0	0
Startdistanz Überblendung	$d_{\text{s,cro}}$	12	12	4	0	0
Mitteldistanz Überblendung	$d_{\text{m,cro}}$	56	40	32	0	0
Enddistanz Überblendung	$d_{\text{e,cro}}$	144	104	64	0	0
Screen-facing Billboard <i>Instances</i>	n_{scr}	128	96	48	0	0
Überblendete <i>Instances</i>	k_{scr}	16	16	16	0	0
Startdistanz Überblendung	$d_{\text{s,scr}}$	64	48	40	0	0
Mitteldistanz Überblendung	$d_{\text{m,scr}}$	128	96	56	0	0
Enddistanz Überblendung	$d_{\text{e,scr}}$	512	384	256	0	0

Tabelle 2: Auflistung der Parameter verschiedener Qualitätsstufen

	1	2	3	4	5	6	7
Testszene	\mathcal{A}	\mathcal{A}	\mathcal{A}	\mathcal{A}	\mathcal{A}	\mathcal{B}	\mathcal{B}
Qualitätsstufe	Hoch	Hoch	Normal	Normal	Niedrig	Tess.A	Tess.B
$\approx\# \text{Gras}_{\text{geo}}$	327.168	327.168	120.384	120.384	32.640	261.888	261.888
$\approx\# \text{Gras}_{\text{cro}}$	215.328	215.328	61.172	61.172	12.416	0	0
$\approx\# \text{Gras}_{\text{scr}}$	759.096	759.096	467.112	467.112	117.292	0	0
Auflösung Simulation	16×16	16×16	16×16	64×64	16×16	16×16	16×16
Simuliertes Gras	262.144	262.144	262.144	4.194.304	262.144	2.048	2.048
# Würfel	0	1984	0	0	0	0	0
# Kugeln	0	1856	0	0	0	0	0
# <i>Bunny</i>	0	1920	0	0	0	0	0
\emptyset FPS	68,82	29,81	70,64	57,16	78,23	57,63	161,87
\emptyset <i>Frame</i> Zeit in ms	14,53	33,54	14,16	17,49	12,78	17,35	6,18
Rendering in ms	11,61	10,37	6,91	–	3,72	11,16	4,68
Simulation in ms	12,74	12,94	12,51	15,88	13,02	–	–

Tabelle 3: Auflistung der verschiedenen Testszenarien und deren Ergebnisse. Die FPS sind Durchschnittswerte von 10 Sekunden Messzeit.

4.1.2 Rendering Performance

Zunächst wird die allgemeine Performance vom Gras Rendering durch Vergleich der Szenarien 1, 3 und 5 betrachtet. Die FPS liegen selbst bei der höchsten Qualitätsstufe mit 68,82 deutlich über der 60 FPS Grenze. Die Anzahl der geometrischen Grashalme kann durch das LOD-System drastisch verringert werden, was sich auch der Visualisierung der LOD-Stufen in Abb. 30a, 30b und 30c entnehmen lässt. Die längste Zeit für das Rendering eines *Patches* in Szenario 1 beträgt $0,385ms$ und stammt von einem nah zur Kamera gelegenen *Patch* mit viel geometrischem Gras. In Szenario 6 beträgt die längste Zeit nur noch $0,060ms$.

Der Einfluss der Eckpunktanzahl kann durch den Vergleich eines einzelnen, vollständig sichtbaren *Patches* in Szenario 6 und 7 getestet werden, bestehend aus 32.768 geometrischen Grashalmen mit einem Tessellationsfaktor von je 64 bzw. 8. Daraus ergibt sich eine gesamte Eckpunktanzahl von 6.389.760 bzw. 884.735. In Szenario 6 beträgt die Zeit zum Rendern des einzelnen *Patches* $3,02ms$, in Szenario 7 $1,82ms$ (da hier mehr Pixel abgedeckt werden, sind die Zeiten höher als bei Szenario 1). Hieraus lässt sich ableiten, dass ein einzelner Eckpunkt $0,212ns$ in Anspruch nimmt und die von den Eckpunkten unabhängige Zeit bei $47,88ns$ liegt:

$$\frac{(3.019.789 - 1.852.424)ns}{(6.389.760 - 884.735)} = \frac{1.167.365ns}{5.505.024} \approx 0,212ns \quad (59)$$

Der benötigte Zeit für 4 Eckpunkte eines Billboards ist $0,848ns$, die für 195 Eckpunkte eines Grashalms mit maximalen Tessellierungsfaktor $41,24ns$. Eine Billboard Textur zeigt bereits 64 Grashalme zeigt.

Geometrisches Gras stellt folglich den zeitintensivsten Faktor dar und erzeugt dabei nur einen geringeren Eindruck der Dichte. Eine Abdeckung des gesamten Terrains nur mit diesem, wäre bei ähnlichem Eindruck der Dichte nicht mehr echtzeitfähig. Um den visuellen Vorteil von geometrischem Gras auf großem Terrain nutzen zu können, ist ein LOD-System unabdingbar.

Culling in der Tessellation-Phase Es soll überprüft werden, wie effizient das *Culling* in der Tessellation-Phase ist. Das Ziel ist herauszufinden, ob das vorgestellte Verfahren zur Reduktion der Anzahl an *Instances* einen nennenswerten Einfluss auf die Performance hat. Ohne diese Reduktion muss die GPU mehr Aussortieren.

Beim Rendering von Blüten wird erst auf der GPU entschieden, ob eine Blüte dargestellt werden soll oder nicht. Die Anzahl an *Instances* entspricht der von geometrischem Gras. Deshalb wird Szenario 6 verwendet und dabei der *Draw-Call* für Blüten erzwungen, welche alle im Shader aussortiert werden. Danach beträgt die durchschnittliche Zeit für das *Culling* von 512 *Instances* 37.120ns, daraus abgeleitet beträgt die für eine *Instance* 72,5ns. Die durchschnittliche Dauer der Berechnung der LOD-Stufen für alle Darstellungsmethoden auf der CPU beträgt hingegen 4.372ns.

Ohne jegliche Reduktion der *Instances* und damit dem alleinigen *Culling* durch die GPU betrüge die benötigte Zeit bei 512 *Instances* und 1024 *Patches* etwa 38ms. Die Reduktion der *Instances* ist folglich notwendig. Die Effektivität ist jedoch davon abhängig, wie genau die Anzahl der berechneten *Instances* an die tatsächliche Anzahl der einzelnen Grashalme angenähert werden kann. Je mehr *Patches* zwischen Start- und Ende der Überblendung liegen, desto weniger Gras muss von der GPU aussortiert werden.

4.1.3 Simulation Performance

Der Vergleich von Szenario 1, 3 und 5 zeigt, dass die Simulationsdauer unabhängig von der Menge an gerendertem Gras ist. In Szenario 5 wird jedoch mehr Gras simuliert als angezeigt. Insgesamt gibt es auch hier mehr mögliche Gräser als simulierte Gräser, wodurch das Speichern der Simulationsdaten in einer Textur weniger Speicherbedarf hat.

Bei näherer Betrachtung zeigt sich, dass das Behandeln der Simulationstexturen pro *Patch* und das separate *Dispatchen* des *Compute-Shaders* einen großen Mehraufwand mit sich führt. Für das Wechseln der an den *Compute-Shader* gebundenen Texturen werden für 1024 *Patches* 4,19ms benötigt, so dass etwa ein Drittel der Simulationsdauer nur damit beschäftigt ist. Das System der Simulationstexturen muss dahingehend geändert werden, dass weniger Aufrufe des *Compute-Shaders* nötig sind, z. B. indem eine Simulationstextur mehrere *Patches* abdeckt.

In Szenario 4 wird eine höhere Simulationsauflösung von 64×64 verwendet. Die 4,19ms zum Wechseln der Texturen bleibt gleich, sodass

sich für die Simulation von 64×64 Gräsern eine Dauer von $11,69ms$ ergibt. Hingegen benötigen 16×16 Gräser etwa $8,32ms$. Die Dauer steigt nicht proportional zur Anzahl der simulierten Gräser. Das ist ein Indiz dafür, dass der *Compute-Shader* nicht optimal ausgelastet ist und zu viele einzelne Ausführungen stattfinden.

Kollisionsbehandlung Zuletzt soll die Auswirkung von Kollisionen auf die Performance analysiert werden. In Szenario 2 wird eine enorme Menge an Kollisionsobjekten in die Szene geworfen und die FPS sinken auf 29,81. Die Dauer der Simulation und des Gras Renderings ändert sich dabei im Vergleich zu Szenario 1 kaum und liegt im Rahmen gewisser Schwankungen. Das Rendering der Kollisionsobjekte aus Sicht der Szenenkamera dauert $10,37ms$. Aus der Sicht der orthographischen Kollisionskamera dauert das Rendering in eine Textur mit 1024×1024 Pixel nur $2,40ms$. Der enorme Verlust an FPS ist damit auf das Zeichnen der Kollisionsobjekte auf dem Bildschirm und deren physikalischem Simulation auf der CPU zurückzuführen. Der zeitliche Mehraufwand bei der Behandlung vieler Kollisionen mit Gras ist mit $2,40ms$ gering.

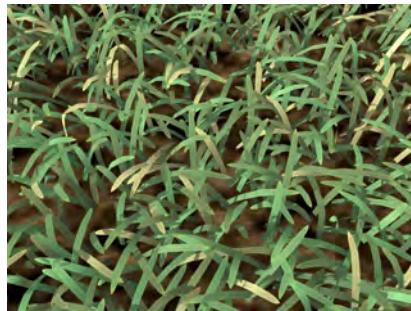
4.2 Vergleich mit bestehenden Arbeiten

Der Vergleich mit anderen Verfahren soll die visuelle Qualität von geometrischem Gras, der LOD Überblendung sowie Wind und Kollisionen bewerten und verschiedene Vor- und Nachteile aufzeigen.

4.2.1 Geometrisches Gras

In Abb. 31 sind vier verschiedene geometrische Grasdarstellungen abgebildet, darunter auch die der vorliegenden Arbeit. Im Vergleich zu den anderen Arbeiten ist das Gras in Abb. 31d am gleichmäßigsten ausgerichtet und benachbarte Grashalme wachsen relativ parallel und mit gleicher Höhe. Das gründet in der Verwendung von Simulationstexturen. Benachbarte Grashalme lesen ähnliche Werte für die Bézierkurve aus, womit Überschneidungen von Grashalmen nur bei höherem Gras möglich sind. Durch die wenigen Überschneidungen sieht das Gras weniger dicht aus, wiederum können durch die Simulationstexturen mehr Grashalme gerendert werden und dichteres Gras entsteht. Sehr offensichtliche Überschneidungen wie in Abb. 31c wirken jedoch nicht sehr realistisch. Insgesamt betrachtet erzeugt das Gras von Fan et al. in Abb. 31b und das der vorliegenden Arbeit in Abb. 31d die größte Verdeckung des Untergrunds.

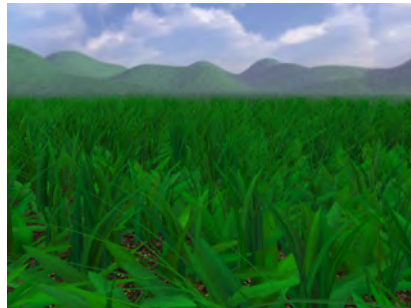
Die Schatten auf dem Gras und die Umgebungsverdeckung auf dem Untergrund in Abb. 31a erzeugen eine überzeugende Tiefenwirkung. In Abb. 31b sind es die Beleuchtung und die Glanzpunkte auf den Grashalmen, die einen realistischen Eindruck der Tiefe bewirken. In der vorliegenden



(a) Geometrisches Gras aus [BBP06] (bearbeitet: Ausschnitt einer Bildkomposition)



(b) Geometrisches Gras aus [Fan+15] (bearbeitet: Zugschnitt von 16:9 in 4:3 Format)



(c) Geometrisches Gras aus [Jah16]



(d) Geometrisches Gras der vorliegenden Arbeit

Abb. 31: Geometrisches Gras verschiedener Arbeiten in der Nahaufnahme.

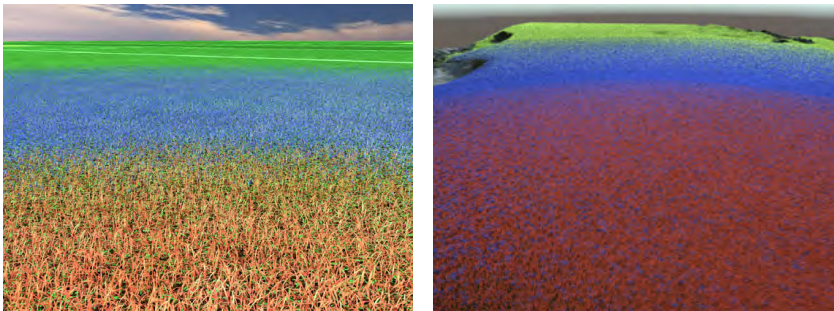
Arbeit kann ein Unterschied nur durch verschiedene Oberflächentexturen wahrgenommen werden, die Beleuchtung und Umgebungsverdeckung ist rudimentär.

Sowohl in Abb. 31c als auch in Abb. 31d sind verschiedene Formen von Gras zu sehen, die aufgrund der exakten geometrischen Modellierung scharfe Kanten haben. Die beiden anderen Arbeiten variieren nur die Farbe von Gras, nicht jedoch deren Form.

4.2.2 Level of Detail

Der direkte visuelle Vergleich von Billboard-Gras erweist sich als schwierig, da es in der vorliegenden Arbeit nur für die Ferne eingesetzt wird. Vielmehr ist die Überblendung des LOD-Systems entscheidender Faktor der Qualität. Hierzu wird das vorliegende System mit dem aus [BBP06] verglichen, da dies das aktuellste betrachtete Verfahren zur Kombination verschiedener Grasdarstellungen ist und sehr überzeugende Ergebnisse liefert.

Zunächst lässt sich in Abb. 32 erkennen, dass in der vorliegenden



(a) Übergänge zwischen Geometrie (rot), vertikalen Schichten (blau) und horizontalen Schichten (grün) [BBP06]

(b) Übergänge zwischen Geometrie (rot), gekreuzten Billboards (blau) und Billboards (grün) der vorliegenden Arbeit

Abb. 32: Visualisierung der weichen Übergänge zwischen verschiedenen Grasdarstellungen.

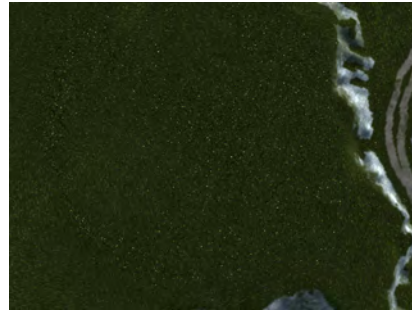
Arbeit viel mehr geometrisches Gras dargestellt wird als in der Arbeit von Bouatouch, Boulanger und Pattanaik im Jahr 2006 möglich war. Der Übergang zwischen geometrischem und volumetrischem Gras erscheint in Abb. 32a etwas weicher als in der vorliegenden Arbeit, da in [BBP06] die Transparenz zur Überblendung variiert wird. Da sich die Grenze des geometrischen Gras weiter von der Kamera entfernt hat, fällt dieser hier weniger ins Gewicht. Der Übergang zwischen gekreuzten und *screen-facing* Billboards in Abb. 32b ähnelt dem zwischen vertikalen Schichten und horizontalen Schichten in Abb. 32a.

Ein Vergleich einer Draufsicht offenbart einen Vorteil von horizontalen Schichten als Grasdarstellung für die Ferne. Während in Abb. 33a ein gleichmäßiger Eindruck entsteht, kommt es in Abb. 33b bei senkrechtem Blick auf Billboards zu Aliasing. Dies geschieht immer dann, wenn Geometrie sehr viel kleiner ist als ein Pixel.

Ein letzter Vergleich zum LOD-System zeigt Gras und Blüten in verschiedenen Darstellungsmethoden aus einem flacheren Winkel. In Abb. 33d sind alle drei Grasdarstellungen zu sehen. Eine sichtbare Grenze ist nicht direkt erkennbar, lediglich die Blüten sind in der Ferne nicht mehr deutlich zu sehen. Die Anforderung, dass Gras für alle Detailstufen gleich aussieht, ist somit teilweise erfüllt. Das Gras, das von *Instances* der geometrischen Darstellung stammt die größer als die Anzahl von bildbasierten *Instances* sind, wird in der bildbasierten Ansicht nicht dargestellt. An der Stelle, an der bspw. ein Billboard mit Löwenzahn steht, befindet sich jedoch auch in der geometrischen Darstellung Löwenzahn. Die Gleichheit der Simulation ist für alle Ansichten gegeben. Im Vergleich zu Abb. 33c aus [Bou08] erscheint das Gras in der vorliegenden Arbeit in der Ferne weniger flach. Für die-



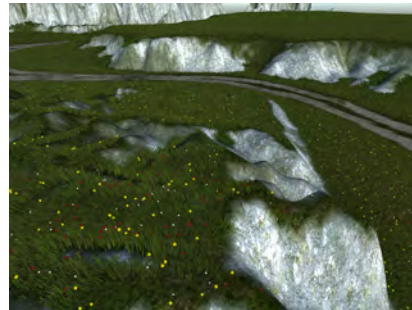
(a) Gras aus der Vogelperspektive [BBP06] (bearbeitet: Ausschnitt einer Bildkomposition).



(b) Billboards senkrecht zur Kamera erzeugen Aliasing Artefakte.



(c) Gras in einem Park mit unbewachsenen Flächen[Bou08] (bearbeitet: Ausschnitt einer Bildkomposition).



(d) Gras in einer Landschaft mit Felsen.

Abb. 33: Vergleich der Darstellungsqualität verschiedener Grasdarstellungen aus verschiedenen Perspektiven.

se Ansicht sind Billboards sehr gut geeignet. Mit der *Grass-Map* kann Gras nur auf bestimmten Flächen angezeigt werden. An Stellen ohne Gras wird dieses jedoch erst von der GPU aussortiert.

4.2.3 Wind

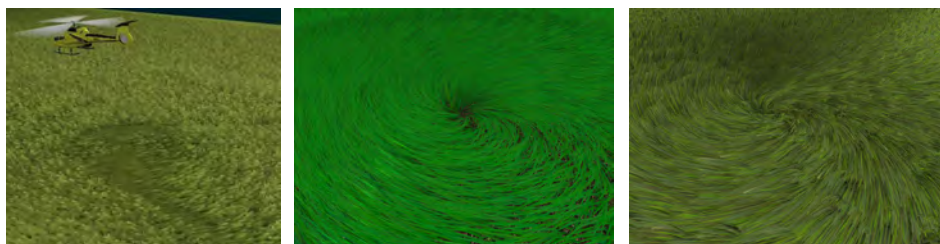
Der Wind in der vorliegenden Arbeit beruht auf der Auswertung eines einfachen, physikalische Eigenschaften imitierenden Modells. Die wirkenden Kräfte werden durch verschiedene Funktionen bereitgestellt. Das resultierende Bewegungsmuster ist dementsprechend sehr einfach und kann mit der visuellen Qualität von komplexeren Systemen wie etwa der Flüssigkeitssimulation von [Lee+16] nicht mithalten.

Das zu Beginn ausführlich beschriebene Modell von [Wan+05] ist dem vorliegenden insofern überlegen, als es auf physikalischen Gesetzen basiert und ein Schwingen mittels Elastizität simuliert. Die Anwendung dieses

Modells auf die vorliegende Arbeit wurde jedoch verworfen, da sich das Übertragen auf die Simulation eines einzigen Kontrollpunkts einer Bézierkurve als schwierig erwies.

Im Vergleich mit anderen prozeduralen Systemen stellen die eingeführten Windebene eine Erweiterung dar. Sie ermöglichen die gleichzeitige Behandlung mehrerer Windquellen und Windarten. In [JW17] werden verschiedene Windarten vorgestellt, jedoch kann nur eine zur gleichen Zeit aktiv sein. Die Arbeiten [HRS13; JW13; Fan+15; Pel04] verwenden eine einzelne Windart und werten diese nur als Verschiebung der Eckpunkte aus. Bei Perbet und Cani [PC01] können verschiedene Windarten in einer Szene aktiv sein, diese können jedoch nicht gleichzeitig auf Gras einwirken. Der Wind in [CJ10] erzielt gute Ergebnisse, da Energie zwischen Grashalmen weitergereicht wird. Andere Windarten werden dabei durch Kollisionen mit Objekten dargestellt.

In Abb. 34 sind drei ähnliche Windarten dargestellt. Der Wirbelwind in Abb. 34b und Abb. 34c weist die gleiche Charakteristik auf. Der Wind vom Helikopter in Abb. 34b breitet sich auch auf umliegende Gräser aus und hinterlässt eine Spur.



(a) Wind von einem Helikopter [CJ10] (b) Wirbelwind aus [JW17] (bearbeitet: Ausschnitt einer Bildkomposition) (c) Wirbelwind der vorliegenden Arbeit

Abb. 34: Vergleich von ähnlichen Windarten verschiedener Arbeiten.

4.2.4 Kollisionen

Zunächst stellt die vorgestellte Kollisionsbehandlung ein flexibles System dar. Es können Kollisionen mit jeder Geometrie ermittelt werden, ohne zusätzliche Schritte voranzustellen, wie etwa die Repräsentation eines Objekts durch Kugeln in [JW17]. Eine grobe Kollisionserkennung, wie sie bei den Verfahren in [ORK09; CJ10; HRS13; Fan+15] verwendet wird, ist nicht nötig, da die Kollisionsbehandlung auch mit vielen Objekten effizient abläuft. Auch ein Aktivieren einzelner *Patches* für eine Kollisionsbehandlung fällt damit weg. Die Erholung von Gras hängt von der Kollisionsstärke ab, und erfolgt nicht innerhalb einer festen Zeit wie in [ORK09; Fan+15].

Ein wichtiger Aspekt ist, dass die Kollision auf ein Gras-Modell angewandt wird, das sowohl für geometrisches als auch bildbasiertes Gras verwendet wird. Die Länge von Billboards bleibt dabei nicht exakt die gleiche und die obere und untere Kante bleiben stets parallel, sodass Billboards im Gegensatz zu solchen aus [BPB09] verzerrt werden können. Da Billboards nur in größeren Entfernungen eingesetzt werden, fällt dies in den meisten Fällen nicht auf.

Die Exaktheit der Kollisionserkennung ist geringer als die anderer Arbeiten. Ein Grund dafür ist die geringe Auflösung der Simulationstextur und der linearen Interpolation zwischen den Kontrollpunkten. Daneben ermöglicht die Technik der Kollisions-Textur keine Kollisionserkennung aus anderen Blickrichtungen als der der orthographischen Kamera. Das begrenzt die Anwendung auf ein Terrain. Zu kleine Kollisionsobjekte werden aufgrund des Tests an nur zwei Punkten der Bézierkurve nicht erkannt. Mit mehreren Iterationen bei der Kollisionsbehandlung könnte sichergestellt werden, dass Grashalme nach einer Reaktion nicht erneut in Kollisionsobjekte gelangen. Auch ein Test an mehreren Punkten der Bézierkurve ist denkbar. Um noch exaktere Ergebnisse zu bekommen, müsste jedoch auch die Simulationstextur durch eine exaktere Alternative ersetzt werden.

Ein großer Nachteil der Simulationstextur wird vor allem bei der Kollisionsbehandlung sichtbar. *Patches* außerhalb des Sichtfelds werden nicht simuliert. Beim Drehen der Kamera befindet sich demnach keine Spur an solchen Stellen, die bspw. von einer rollenden Kugel beeinflusst wurden.

Für den visuellen Vergleich sind verschiedene Kollisionsbehandlungen in Abb. 35 abgebildet. In Abb. 35b ist zu sehen, dass geometrisches Gras den Bällen sehr exakt ausweicht, sodass keine Grashalme in die Geometrie ragen. In der vorliegenden Arbeit ist dies nicht garantiert. Außerdem kann das Gras auch weiter vom eigentlichen Kollisionsobjekt weggebogen werden (zu sehen in Abb. 35e). Im Gegensatz zu Abb. 35b wird Gras in der vorliegenden Arbeit viel länger von einer Kollision beeinflusst und entspricht damit mehr dem Aussehen in Abb. 35c.

Die Kollisionsreaktion auf Billboards in Abb. 35a zeigt eine gut sichtbare Spur niedergedrückten Grases. Da die Geometrie nicht sehr hoch aufgelöst ist, sind harte Kanten zu sehen. Die Reaktion auf Billboards ist in Abb. 35f für die vorliegende Arbeit aus der Ferne zu sehen. Auch hier sind die Spuren der Kollisionsobjekte deutlich zu sehen. An den Seiten erkennt man jedoch, dass bei niedergedrücktem Gras der Boden zu sehen. Hier werden *screen-facing* Billboards entlang ihrer Ausrichtung verbogen und dabei so stark verzerrt, dass sie beinahe verschwinden. Bei der Arbeit [ORK09] wird dies durch ein „Masse-Feder-System“ verhindert [ORK09].



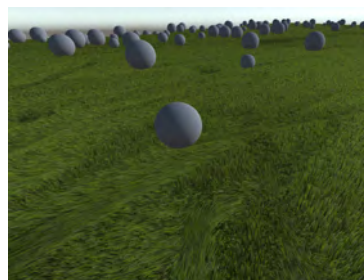
(a) Niedergedrückte Billboards nach Kollision [ORK09].



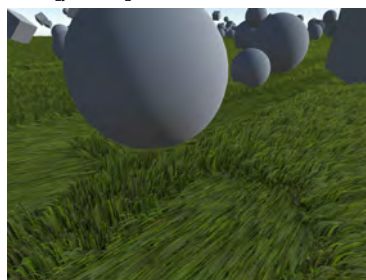
(b) Bälle kollidieren mit geometrischem Gras [Fan+15].



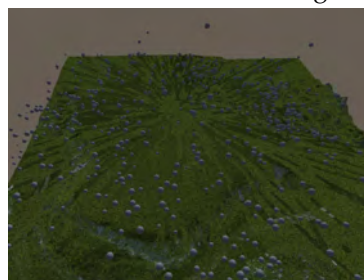
(c) Eine Hand hinterlässt niedergedrücktes Gras [JW17].



(d) Spuren von Kugeln nach der Kollision mit Gras verschiedener Darstellungen.



(e) Kollision mit verschiedenen Objekten in der Nahansicht.



(f) Hunderte Kugeln hinterlassen Spuren im bildbasierten Gras.

Abb. 35: Abbildungen verschiedener Kollisionsbehandlungen.

4.3 Verbesserungsmöglichkeiten

In diesem Abschnitt sollen die wesentlichen entdeckten Probleme des entwickelten Verfahrens festgehalten und Möglichkeiten zu deren Verbesserung gegeben werden.

- Bei der Simulation im *Compute-Shader* wird jedes *Patch* in einem separaten *Dispatch* bearbeitet, was zu einem gewaltigen Mehraufwand von etwa 30% führt. Zur Verbesserung kann ein Textur Array eingesetzt werden, welche die Simulationsdaten aller *Patches* enthält. Ein einzelner Aufruf des *Compute-Shaders* aktualisiert dann die nötigen Elemente.
- Nicht sichtbare *Patches* werden nicht simuliert, was bei einer Änderung der Blickrichtung sehr störend wirkt. Möglicherweise ist die Simulation nach der zuvor genannten Optimierung effizient genug, um sie unabhängig von der Sichtbarkeit auszuführen. Dabei kann die Aktualisierungsrate der Simulation außerhalb des Sichtfelds reduziert werden oder eine niedrigere Simulationsauflösung gewählt werden.
- Für die geometrische Grasdarstellung muss eine bessere Antialiasing Maßnahme eingeführt werden. Bisher wird nur die Breite in den verschiedenen *Mip-Map*-Stufen angepasst. Ein exaktes Verfahren soll stattdessen sicherstellen, dass die Gras-Geometrie groß genug ist, um alle Pixel abzudecken, die es berührt. In [Yu+12] ist ein solches Vorgehen für Haar-Rendering beschrieben.
- Billboards werden zu stark verzerrt und verursachen ebenfalls Aliasing Artefakte. Die Anwendung des formerhaltenden Verfahrens aus [ORK09] kann getestet werden, dies führt jedoch zu mehr Eckpunkten der Billboards. Als Antialiasing Maßnahme kann die Detailstufe der *screen-facing* Billboards durch eine volumetrische Darstellung ersetzt werden. Solche Billboards, die senkrecht zur Kamera stehen, können von der GPU aussortiert werden, wie geometrisches Gras in [JW17].

5 Fazit und Ausblick

In dieser Bachelorarbeit wurde auf Basis einer Recherche und Analyse bestehender Arbeiten eine Echtzeitanwendung entwickelt, die eine dicht bewachsene Wiese darstellt und die Reaktion von Wind und Kollisionen auf Gras simuliert. Ein *Level of Detail* System ermöglicht die dafür notwendige Performance durch Kombination einer geometriebasierten und zwei bildbasierter Darstellungsmethoden. Zwischen den Darstellungen wird ein Verfahren zur nahtlosen Überblendung eingesetzt, das zugleich die Arbeit der GPU reduziert. Die Geometrie aller Darstellungen wird erst zur Laufzeit in der Tessellation-Phase generiert, wozu ein einheitliches Gras-Modell herangezogen wird. Geometrisches Gras kann dabei durch ein leicht handhabbares Prinzip in vielfältiger Weise geformt werden. Zudem wird es als Grundlage zur Generierung von Texturen für die Billboards der bildbasierten Darstellung verwendet. Das Positionieren von geometrischen Blüten auf der Grasspitze demonstriert die Erweiterbarkeit des Gras-Modells.

Durch eine getrennte Behandlung der Simulation vom Rendering wird diese von der Quantität an Gras entkoppelt und Speicherplatz eingespart. Der Einfluss von Wind und Kollisionen wird auf das Gras-Modell angewandt und ist damit für alle Grasdarstellungen identisch beschrieben. Mit einem erweiterbaren Windsystem können viele Windquellen gleichzeitig verwendet werden. Die resultierende Auswirkung auf Gras ist nicht physikalischer Natur und erreicht keine visuelle Qualität von komplexeren Modellen. Zur Kollisionsbehandlung werden Kollisionsobjekte von einer orthographischen Kamera gerendert. Damit wird eine flexible und schnelle Kollisionserkennung mit Objekten jeglicher Form ermöglicht und eine ansehnliche Kollisionsreaktion von Gras erzielt. Die gewonnene Flexibilität geht dabei im Vergleich zu anderen analysierten Verfahren zu Lasten der Präzision.

Die Evaluation der entwickelten Anwendung hat deren Echtzeitfähigkeit und die positive Auswirkung des *Level of Detail*-Systems auf die Performance bestätigt. Auch die Effizienz der Kollisionsbehandlung konnte in einem Belastungstest belegt werden. Indessen wurden Schwächen bei der Performance und visuellen Qualität der Simulation sowie Probleme mit Aliasing beim Rendering aufgedeckt. Hierfür wurden konkrete Verbesserungsmöglichkeiten gegeben, die in zukünftigen Arbeiten berücksichtigt werden können.

Darüber hinaus gibt es verschiedene Möglichkeiten zur Steigerung der Performance. Mittels *Occlusion Culling* wie in [Mak15; JW17] können nicht sichtbare Grashalme aussortiert werden. Bei der Tessellation von Gras können die Faktoren an die resultierende Größe des Grashalms auf dem Bildschirm angepasst werden. *Gras-Patches* könnten abhängig von der Entfernung zu größeren Gruppen zusammengefasst werden, um die Anzahl an *Draw-Calls* zu reduzieren. Dabei muss jedoch die Integration in

das LOD- und Simulations-System berücksichtigt werden.

Zur Verbesserung der visuellen Qualität haben ein realistischeres Beleuchtungsmodell, Schatten, globale Beleuchtung sowie die Volumenstreuung oberste Priorität. Daneben kann eine gezieltere Kontrolle der Verteilung von Grasarten durch den Benutzer oder gar durch eine realistische Simulation eines Ökosystems wie in [Deu+98] eingesetzt werden. Zur Verbesserung der geometrischen Darstellung könnten Grashalme um zusätzliche Seitensprosse erweitert werden, die ähnlich der vorgestellten Blüten platziert werden können.

Auch eine Erweiterung auf ein unendlich großes Terrain ist denkbar. Hierzu muss die Technik der *Gras-Patches* und die Kollisionsbehandlung umgerüstet werden, sodass die Simulation und das Rendering korrekte Daten erhalten. Die Generation der Geometrie findet bereits zur Laufzeit auf der GPU statt.

Glossar

Alpha to Coverage Mit Alpha to Coverage können ohne Tiefensortierung weiche Kanten erzielt werden [KCS07]. 10, 51

Billboard Billboards sind einfache Modelle mit nur wenigen Polygonen, die mit einer Textur versehen werden. 3, 9–11, 16, 19, 22, 25, 27, 28, 30–35, 41, 45, 50, 51, 54, 55, 58–60, 62–65

Bounding Box Ein *Bounding Volume* mit einem Quader als Körper. In der vorliegenden Arbeit wird, wenn nicht explizit anders genannt, stets von einem entlang der Koordinatenachsen ausgerichteten Quader ausgegangen. 14, 19, 26–29, 31

Bounding Sphere Ein *Bounding Volume* mit einer Kugel als Körper. 14, 19

Bounding Volume Ein Bounding Volume ist ein nicht näher beschriebener geometrischer Körper, der ein Objekt umschließt. 14, A

Compute-Shader Ein Compute-Shader ist ein programmierbarer Shader von *Direct3D* [CoroJg] und *OpenGL* [con18a], der für allgemeine Berechnung auf der GPU verwendet werden kann. 8, 20, 30, 36, 37, 40, 42, 56, 57, 64

Cube-Map Eine Cube-Map besteht aus Bildern für die sechs Seiten eines Würfels und wird beim Cube-Mapping Verfahren [Gre86] für Umgebungsreflexionen verwendet. 19

Culling Culling bezeichnet in der Computergrafik das Aussortieren von Objekten oder Oberflächen auf Basis verschiedener Kriterien. 8, 30, 32, 46, 48, 51, 56, 65, B

Domain-Shader Der Domain-Shader ist Teil der Tessellation-Phase in *Direct3D* [CoroJd]. 45, 46, 51

Echtzeit Die Echtzeit ist ab einer Rate von 15 Bildern pro Sekunde gegeben, ab 72 Bildern pro Sekunde ist kein Unterschied mehr erkennbar [AHH08]. . 2, 5, 8, 12, 21

Fragment-Shader Der Fragment-Shader bearbeitet einzelne Fragmente die durch Rasterisierung erstellt wurden und ist Teil von *OpenGL* [conan]. Das Äquivalent in *Direct3D* ist der *Pixel-Shader*. C, 9, 10, 22

Frame Ein Einzelbild einer Anwendung. 2, 14, 19, 20, 25, 29, 30, 37, 39–42, 55

Framework Ein Framework (dt. Gerüst) stellt ein Grundgerüst für die Programmierung bereit.. 3, 23

Frustum Ein Frustum ist das kegelstumpfförmige Ansichtsvolumen einer perspektivischen Kamera. 14, 29, 53, B

Frustum Culling Frustum Culling ist ein *Culling* Verfahren, das Objekte aussortiert, die nicht im Sichtbereich eines Frustums liegen. 14, 28, 29

Geometry-Shader Der Geometry-Shader ermöglicht das Erzeugen neuer Geometrie und ist Teil von *Direct3D* [CoroJa] und *OpenGL* [con18b]. 6, 8, 10, 11, 19

Hull-Shader Der Hull-Shader ist Teil der Tessellation-Phase in *Direct3D* [CoroJd]. 30, 32, 45, 46

Input-Assembler Die erste Phase der *Direct3D* Renderpipeline [CoroJb], die Eingabe Primitive verarbeitet und für folgende Phasen zusammenstellt. 45

Instanced-Rendering Rendering unter Verwendung der *Instancing* Technik, die es ermöglicht Objekte mehrmals in einem einzelnen Rendereaufruf zu zeichnen [con17b]. 6–8, 30

L-System Die Lindenmayer Systeme sind eine mathematische Theorie zur Entwicklung und Modellierung von Pflanzen [PL90]. 15

Level of Detail Level of Detail steht wörtlich für den Detaillierungsgrad. Als LOD-Stufen werden hier verschiedene Detailstufen mit variierendem Detaillierungsgrad bezeichnet. Ein LOD-System steht für die Kombination verschiedener LOD-Stufen. D, 3, 65

Lookup Tabelle Beschreibt in der vorliegenden Arbeit eine Textur mit vorberechneten Werten, die zur Laufzeit ausgelesen werden. 35, 49

Mip Mapping Mip Mapping bezeichnet die Verwendung von *Mip-Maps* zur Texturfiltering [CoroJe]. 35

Mip-Map Mip-Maps sind eine Menge zusammenhängender Texturen, bei der jede den gleichen Inhalt in kleinerer Auflösung anzeigt [CoroJe]. 36, 47, 50, 64, B

Normal Map Eine Textur, die Oberflächennormalen zumeist im Tangentialraum speichert. In der vorliegenden Arbeit werden Normalen in Weltkoordinaten gespeichert.. 22, 26

- Patch** Ein Patch beschreibt in der vorliegenden Arbeit ein mit Gras bewachsenes *Tile*. 6, 20, 26–31, 36, 37, 42, 45, 50, 53–56, 61, 62, 64–66
- Pixel-Shader** Der Pixel-Shader wird nach der Rasterisierung ausgeführt und ist Teil von *Direct3D* [CoroJc]. Das Äquivalent in *OpenGL* ist der *Fragment-Shader*. 42, 45, A
- Primitive-Generator** Der Primitive Generator ist der nicht programmierte Teil der Tessellation, er erzeugt neue Primitive und ist Teil von *OpenGL* [con17a]. 8
- Raytracing** Raytracing (dt. Strahlenverfolgung) bezeichnet in der Computergrafik ein Verfahren zum Rendering durch das Aussenden von Strahlen. Es wird in der Regel nicht für Echtzeitanwendungen angewandt. 10, 11
- Shader** Ein Shader ist ein programmiertes Programm, das in einer programmierbaren Phase der Renderpipeline (in dieser Arbeit von *Direct3D* oder *OpenGL*) ausgeführt wird [conay]. C, 23, 25, 56, A
- Tessellation** Die Tessellation ermöglicht in der Computergrafik die Unterteilung von Polygonen zum Erzeugen einer höheren Detailstufe und ist Teil von *Direct3D* [CoroJd] und *OpenGL* [con17a]. C, 6, 8, 11, 13, 27, 30, 33, 35, 46, 48, 49, 51, 53–56, 65, A, B
- Tile** In der vorliegenden Arbeit wird die Begriffsdefinition nach Fan et al. [Fan+15] verwendet. Danach ist ein *Tile* ein Gitterelement, welches durch die Aufteilung eines Terrains in ein Gitter entsteht. C, 14
- Vertex-Shader** Der Vertex-Shader ist der erste programmierbare Shader der Renderpipeline, er arbeitet auf einzelnen Eckpunkten und ist Teil von *Direct3D* [CoroJc] und *OpenGL* [conov]. 7, 45

Abkürzungsverzeichnis

CPU *Central Processing Unit*. 19, 20, 25, 30, 31, 39, 40, 50, 53, 56, 57

FPS *Frames pro Sekunde*. 2, 18, 53, 55, 57

GPU *Graphics Processing Unit*. 10, 19, 20, 25, 30, 32, 33, 37, 39, 40, 53, 56, 60, 64–66

LOD *Level of Detail*. D, 3, 12, 14, 26, 30–33, 36, 46, 55–59, 65, 66, B, *Glossar: Level of Detail*

Literatur

- [AHH08] Tomas Akenine-Möller, Eric Haines und Naty Hoffman. *Real-Time Rendering 3rd Edition*. Natick, MA, USA: A. K. Peters, Ltd., 2008, S. 1045. ISBN: 987-1-56881-424-7.
- [Bao+11] G. Bao et al. "Realistic real-time rendering for large-scale forest scenes". In: *2011 IEEE International Symposium on VR Innovation*. März 2011, S. 217–223.
- [BBP06] Kadi Bouatouch, Kévin Boulanger und Sumanta Pattanaik. *Rendering Grass in Real Time with Dynamic Light Sources*. Research Report RR-5960. INRIA, 2006, S. 36.
- [BLH02] Brook Bakay, Paul Lalonde und Wolfgang Heidrich. "Real-time animated grass". In: *Proceedings of Eurographics (short paper) 27* (2002).
- [Bli78] James F. Blinn. "Simulation of Wrinkled Surfaces". In: *SIGGRAPH Comput. Graph.* 12.3 (Aug. 1978), S. 286–292.
- [Bou08] Kevin Boulanger. "Real-time Realistic Rendering of Nature Scenes with Dynamic Lighting". AAI3335335. Diss. Orlando, FL, USA: University of Central Florida, 2008. ISBN: 978-0-549-88752-2.
- [BPB09] K. Boulanger, S. N. Pattanaik und K. Bouatouch. "Rendering Grass in Real Time with Dynamic Lighting". In: *IEEE Computer Graphics and Applications* 29.1 (Jan. 2009), S. 32–41.
- [Brk+09] Belma R. Brkic et al. "Cross-modal Affects of Smell on the Real-time Rendering of Grass". In: *Proceedings of the 25th Spring Conference on Computer Graphics*. SCCG '09. Budmerice, Slovakia: ACM, 2009, S. 161–166. ISBN: 978-1-4503-0769-7.
- [Cat74] Edwin Earl Catmull. "A Subdivision Algorithm for Computer Display of Curved Surfaces." AAI7504786. Diss. 1974.
- [CJ10] K. Chen und H. Johan. "Real-time continuum grass". In: *2010 IEEE Virtual Reality Conference (VR)*. März 2010, S. 227–234.
- [Col+05] Carsten Colditz et al. "Real-time Rendering of Complex Photorealistic Landscapes Using Hybrid Level-of-Detail Approaches". In: *Conference for Information Technologies in Landscape Architecture, 2005*. 2005.
- [con17a] OpenGL Wiki contributors. *Tessellation*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. Nov. 2017. URL: <http://www.khronos.org/opengl/wiki/opengl/index.php?title=Tessellation&oldid=14135>.

- [con17b] OpenGL Wiki contributors. *Vertex Rendering*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. Sep. 2017. URL: http://www.khronos.org/opengl/wiki/opengl/index.php?title=Vertex_Rendering&oldid=13993.
- [con18a] OpenGL Wiki contributors. *Compute Shader*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. Jan. 2018. URL: http://www.khronos.org/opengl/wiki/opengl/index.php?title=Compute_Shader&oldid=14282.
- [con18b] OpenGL Wiki contributors. *Geometry Shader*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. Jan. 2018. URL: http://www.khronos.org/opengl/wiki/opengl/index.php?title=Geometry_Shader&oldid=14230.
- [conan] OpenGL Wiki contributors. *Fragment Shader*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. 2018 Jan. URL: http://www.khronos.org/opengl/wiki/opengl/index.php?title=Vertex_Shader&oldid=14097.
- [conay] OpenGL Wiki contributors. *Shader*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. 2015 May. URL: <http://www.khronos.org/opengl/wiki/opengl/index.php?title=Shader&oldid=12482>.
- [conov] OpenGL Wiki contributors. *Vertex Shader*. Hrsg. von OpenGL Wiki. Abrufdatum: 10.03.2018. 2017 Nov. URL: http://www.khronos.org/opengl/wiki/opengl/index.php?title=Fragment_Shader&oldid=14228.
- [Coo84] Robert L. Cook. "Shade Trees". In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), S. 223–231.
- [CoroJa] Microsoft Corporation, Hrsg. *Geometry Shader Stage*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/en-us/library/windows/desktop/mt787170\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/mt787170(v=vs.85).aspx).
- [CoroJb] Microsoft Corporation, Hrsg. *Input-Assembler Stage (Direct3D 10)*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/de-de/library/windows/desktop/bb205116\(v=vs.85\).aspx](https://msdn.microsoft.com/de-de/library/windows/desktop/bb205116(v=vs.85).aspx).
- [CoroJc] Microsoft Corporation, Hrsg. *Shader Stages (Direct3D 10)*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/de-de/library/windows/desktop/bb205146\(v=vs.85\).aspx](https://msdn.microsoft.com/de-de/library/windows/desktop/bb205146(v=vs.85).aspx).

- [CoroJd] Microsoft Corporation, Hrsg. *Tessellation – Überblick*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/de-de/library/windows/desktop/ff476340\(v=vs.85\).aspx](https://msdn.microsoft.com/de-de/library/windows/desktop/ff476340(v=vs.85).aspx).
- [CoroJe] Microsoft Corporation, Hrsg. *Texture Filterung with Mipmaps (Direct3D 9)*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/de-de/library/windows/desktop/bb206251\(v=vs.85\).aspx](https://msdn.microsoft.com/de-de/library/windows/desktop/bb206251(v=vs.85).aspx).
- [CoroJf] Microsoft Corporation, Hrsg. *Transforms (Direct3D 9)*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/en-us/library/windows/desktop/bb206269\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/bb206269(v=vs.85).aspx).
- [CoroJg] Microsoft Corporation, Hrsg. *Übersicht über Compute-Shader*. Abrufdatum: 10.03.2018. o.J. URL: [https://msdn.microsoft.com/de-de/library/windows/desktop/ff476331\(v=vs.85\).aspx](https://msdn.microsoft.com/de-de/library/windows/desktop/ff476331(v=vs.85).aspx).
- [Deu+98] Oliver Deussen et al. "Realistic Modeling and Rendering of Plant Ecosystems". In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. New York, NY, USA: ACM, 1998, S. 275–286. ISBN: 0-89791-999-8.
- [DN04] Philippe Decaudin und Fabrice Neyret. "Rendering Forest Scenes in Real-Time". In: *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*. Juni 2004, S. 93–102.
- [DZJ06] Q. Deng, X. Zhang und M. Jaeger. "Efficient Multiresolution of Foliage for Real-Time Rendering". In: *2006 Second International Symposium on Plant Growth Modeling and Applications*. Nov. 2006, S. 307–314.
- [Fan+15] Zengzhi Fan et al. "Simulation and Rendering for Millions of Grass Blades". In: *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. i3D '15. San Francisco, California: ACM, 2015, S. 55–60. ISBN: 978-1-4503-3392-4.
- [Gra97] Jens Gravesen. "Adaptive subdivision and the length and energy of Bézier curves". In: *Computational Geometry 8.1 (1997)*, S. 13–31.
- [Gre86] N. Greene. "Environment Mapping and Other Applications of World Projections". In: *IEEE Computer Graphics and Applications* 6.11 (Nov. 1986), S. 21–29.

- [HH12] Dongsoo Han und Takahiro Harada. "Real-time Hair Simulation with Efficient Hair Style Preservation". In: *Workshop on Virtual Reality Interaction and Physical Simulation*. Hrsg. von Jan Bender et al. The Eurographics Association, 2012. ISBN: 978-3-905673-96-8.
- [HRS13] Nico Hempe, Jürgen Rossmann und Björn Sondermann. "Generation and Rendering of Interactive Ground Vegetation for Real-Time Testing and Validation of Computer Vision Algorithms". In: *ELCVIA Electronic Letters on Computer Vision and Image Analysis* 12.2 (2013).
- [HWJ07] Ralf Habel, Michael Wimmer und Stefan Jeschke. "Instant Animated Grass". In: *Journal of WSCG* 15.1-3 (Jan. 2007). ISBN 978-80-86943-00-8, S. 123–128.
- [Jah16] Klemens Jahrmann. "Interaktives Gras Rendering in Echtzeit unter Verwendung moderner OpenGL Methoden". Diplomarbeit. Wien: Technische Universität Wien, 2016.
- [JW13] Klemens Jahrmann und Michael Wimmer. "Interactive Grass Rendering Using Real-Time Tessellation". In: *WSCG 2013 Full Paper Proceedings*. Hrsg. von Manuel Oliveira und Vaclav Skala. Plzen, CZ, Juni 2013, S. 114–122. ISBN: 978-80-86943-74-9.
- [JW17] Klemens Jahrmann und Michael Wimmer. "Responsive Real-time Grass Rendering for General 3D Scenes". In: *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '17. San Francisco, California: ACM, 2017, 6:1–6:10. ISBN: 978-1-4503-4886-7.
- [KCS07] Alexander Kharlamov, Iain Cantlay und Yury Stepanenko. "GPU Gems 3". In: *GPU Gems 3*. Hrsg. von Hubert Nguyen. Pearson Higher Education, 2007. Kap. Next-Generation SpeedTree Rendering, S. 69–92.
- [KK89] J. T. Kajiya und T. L. Kay. "Rendering Fur with Three Dimensional Textures". In: *SIGGRAPH Comput. Graph.* 23.3 (Juli 1989), S. 271–280.
- [Lee+16] Ruen-Rone Lee et al. "A simulation on grass swaying with dynamic wind force". In: *The Visual Computer* 32.6 (Juni 2016), S. 891–900.
- [Lev14] Mark Levoy. *The Stanford 3D scanning repository*. Hrsg. von Stanford University Computer Graphics Laboratory. Abrufdatum: 10.03.2018. Aug. 2014. URL: <http://graphics.stanford.edu/data/3Dscanrep/>.

- [Mak15] Evgeny Makarov. *NVIDIA Turf Effects: Massive Grass Rendering with dynamic simulation*. Präsentiert auf der GPU Technology Conference 2015. San Jose, Kalifornien, März 2015. URL: <http://on-demand.gputechconf.com/gtc/2015/presentation/S5748-Evgeny-Makarov.pdf>.
- [ORK09] J. Orthmann, C. Rezk-Salama und A. Kolb. "GPU-based Responsive Grass". In: *Journal of WSCG* 17 (Apr. 2009), S. 65–72.
- [Pan14] David Pangerl. "GPU Pro 5: advanced rendering techniques". In: Hrsg. von Wolfgang Engel. Hoboken, NJ: Taylor und Francis, 2014, S. 221–232.
- [PC01] Frank Perbet und Maric-Paule Cani. "Animating Prairies in Real-time". In: *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. I3D '01. New York, NY, USA: ACM, 2001, S. 103–110. ISBN: 1-58113-292-1.
- [Pel04] Kurt Pelzer. "GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics". In: *GPU Gems*. Hrsg. von Randima Fernando. Pearson Higher Education, März 2004. Kap. Rendering Countless Blades of Waving Grass, S. 107–121.
- [Per85] Ken Perlin. "An Image Synthesizer". In: *SIGGRAPH Comput. Graph.* 19.3 (Juli 1985), S. 287–296.
- [PL90] P. Prusinkiewicz und Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. New York, NY, USA: Springer-Verlag New York, Inc., 1990. ISBN: 0-387-97297-8.
- [RB85] William T. Reeves und Ricki Blau. "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems". In: *SIGGRAPH Comput. Graph.* 19.3 (Juli 1985), S. 313–322.
- [SKP05] Musawir A. Shah, Jaakko Kontinen und Sumanta Pattanaik. "Real-time Rendering of Realistic-looking Grass". In: *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. GRAPHITE '05. Dunedin, New Zealand: ACM, 2005, S. 77–82. ISBN: 1-59593-201-1.
- [Smi07] Alvy Ray Smith. "The Adventures of André & Wally B." In: *Pixar Short Films Collection, Volume 1*. Hrsg. von Pixar Animation Studios. United States: Walt Disney Studios Home Entertainment, Nov. 2007.
- [Sza17] G. Szauer. *Game Physics Cookbook*. Packt Publishing, 2017. ISBN: 9781787120815.

- [TecoJ] Unity Technologies. *Unity - Products*. Hrsg. von Unity Technologies. Abrufdatum: 10.03.2018. o.J. URL: <https://unity3d.com/de/unity>.
- [Wan+05] Changbo Wang et al. "Dynamic Modeling and Rendering of Grass Wagging in Wind: Natural Phenomena and Special Effects". In: *Comput. Animat. Virtual Worlds* 16.3-4 (Juli 2005), S. 377–389.
- [Wan+09] D. Wang et al. "Real-Time GPU-Based Visualization of Tile Tracks in Dynamic Terrain". In: *2009 International Conference on Computational Intelligence and Software Engineering*. Dez. 2009, S. 1–4.
- [Yu+12] Xuan Yu et al. "A Framework for Rendering Complex Scattering Effects on Hair". In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '12. Costa Mesa, California: ACM, 2012, S. 111–118. ISBN: 978-1-4503-1194-6.
- [ZLZ09] X. Zhao, F. Li und S. Zhan. "Real-Time Animating and Rendering of Large Scale Grass Scenery on GPU". In: *2009 International Conference on Information Technology and Computer Science*. Bd. 1. Juli 2009, S. 601–604.