Darstellung naturgetreuer Graslandschaften in Echtzeit

Patrick Horlebein, Peter Lyko, Sandra Barbara Ohmayer, Kai-Alexander Schubert

Hochschule Darmstadt

Abstract

Das Darstellen von lebensechten Szenerien gehört seit langer Zeit zu einer der größten Interessen auf dem Gebiet der Computergrafik. Es existieren viele Methoden, die darauf abzielen natürliche Phänomene zu visualisieren, sodass sie für den Betrachter realistisch erscheinen. Aufgrund seiner geometrischen Komplexität, war das Darstellen von einzelnen Grashalmen, eine kaum erfüllbare Aufgabe. Durch neueste Hard- und Softwaretechnik können heute neue Methoden eingesetzt werden, um einen höheren Grad an Realismus zu erzielen.

Dieses Paper beschreibt Methoden mit denen eine realistisch aussehende Grasfläche in Echtzeit animiert und gerendert werden kann. Die grundlegende Idee besteht darin, die parallelen Recheneigenschaften einer GPU zu nutzen, um lebensnähere Szenarien darzustellen, anders als es mit Methoden der Fall ist, die derzeit in Spielen und Simulationen angewendet werden. Um dies zu erreichen, beruht die beschriebene Methode auf der Verwendung von Geometry Shadern und erzielt dabei echtzeitfähige Ergebnisse.

1. Einleitung

Gras ist neben Bäumen und Sträuchern ein weiteres weit verbreitetes Vegetationselement. Es tritt in den verschiedensten ökologischen Regionen auf und ist somit ein zentrales Element fast jeder Landschaft.

Somit ist die Darstellung von virtuellem Gras für die Computergrafik von hohem Interesse und das sowohl für den Einsatz in der Film-, als auch Spieleindustrie. Besonders für die Spieleindustrie hat die rasante Entwicklung der Grafikhardware im Consumerbereich neue Wege eröffnet. Über alle Generationen von Grafikchips hinweg haben sich verschiedenste Verfahren entwickelt, welche die Simulation dieses Teils der Flora zum Ziel hatten.

Dieses Paper soll eine neue Methode beschreiben, mit der sehr detaillierte Grashalme in Echtzeit animiert und dargestellt werden können. Dabei sollen auch verschiedene Level-of-Detail - Methoden präsentiert werden, mit denen der Einsatz von Grashalmen in großen Mengen in Landschaftsaufnahmen ermöglicht werden kann. In einer abschließenden Bewertung dieses neuen Verfahrens, soll auch ein Vergleich zu anderen Methoden gezogen werden und zwar in Hinblick auf die Qualität der Darstellung, und deren Einsatzfähigkeit in Echtzeit-Anwendungen.

2. Grundlagen

Die Entwicklung der Visualisierung von Gras und ähnlichem Bodenbewuchs lässt sich in drei Kategorien einteilen.

Die einfachste Methode ist die Verwendung so genannter Billboards, Flächen, die früher stets orthogonal zur Blickrichtung der Kamera ausgerichtet waren und nun auch frei im Raum stehen können. Diese Flächen werden anschließend mit Bildern von ganzen Grasbüscheln oder zumindest mehreren Grashalmen texturiert. Das Verwenden von Alphablending ist hierbei von entscheidender Bedeutung, weil nur die Teile des Polygons dargestellt werden sollen, die Teil des Bewuchses sind. Billboards können nur sehr eingeschränkt animiert werden, da es sich in der Regel um einfache Vierecke handelt. Lediglich die freien, d.h. nicht mit dem Boden verankerten Punkte sind beweglich und können auf Wind oder eine Kollision reagieren.

Als Weiterentwicklung wurden drei oder mehr Flächen sternförmig angeordnet, um die ursprünglichen sich sichtbar mitdrehenden Billboards abzulösen. Mit dieser Technik wird die Illusion nur offenbart, wenn die Grasbüschel von oben betrachtet werden. Außerdem gibt es mehr Punkte, die für Animationen genutzt werden können.

Einerseits können alle oberen Punkte des Sterns gleichmäßig, andererseits die äußeren Punkte ungleichmäßig verschoben werden, was die Polygonflächen verändert. Zuletzt können die Büschel entweder getrennt oder gemeinsam animiert werden (Abb. 1).

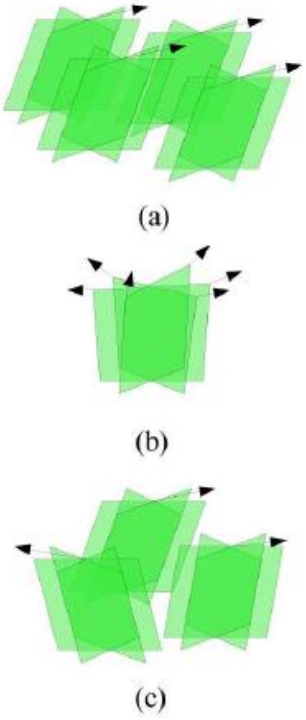

Abbildung 1: Animationsarten bei Billboards mit mehreren Ebenen ([B 08] S. 39 ff)

Während bisher, aufgrund mangelnder Rechenkapazität, versucht wurde, die Geometrie quantitativ stark zu beschränken, befasst sich die neuste Methode mit dem bewussten Erzeugen von Geometrie zum Darstellen einzelner Halme. Erst der 2006 von NVIDIA vorgestellte Geometry Shader ermöglicht dieses geometrisch komplexe Vorgehen, da er es erlaubt neue Geometrien zur Szene hinzuzufügen, während die Renderingpipeline durchlaufen wird. Ohne diese Dynamik wäre der Speicherbedarf dieses Ansatzes um ein Vielfaches höher. Zusätzlich wäre es nötig, Schritte, die parallelisierbar sind, sequentiell durchzuführen. Die Idee Geometry Shader zur Grasdarstellung zu verwenden, präsentierte unter anderem Edward Lee in seiner Masterarbeit "Realistic Real-Time Grass Rendering" [L 10].

Seine Arbeit dient dabei in einigen Aspekten als Grundlage dieses Papers. Es wurde die Methodik zum Erstellen der Grashalme auf der Grafikkarte als Vorlage genutzt, sowie das Konzept des Level–of-Detail bei einzelnen Halmen. Die Verteilung der Grashalme findet jedoch anhand eines verbesserten Chunk-Algorithmus statt, der sich an dem Heightmap-Konzept von Strugar Filips Paper "Continuous Distance-Dependent Level of Detail for Rendering Heightmaps“ [S 10] orientiert.

Um die Quantität der Geometrie zu reduzieren wird mit dem Level–of-Detail, kurz LOD, gearbeitet; d.h., dass Objekte in Abhängigkeit der Entfernung zur Kamera und ihrer Wichtigkeit in der Szene in verschiedenen Detailstufen angezeigt werden. Das Detail der Grashalme wird durch die Anzahl ihrer Glieder bestimmt, so bestehen Grashalme, die sich nahe am Betrachter befinden, aus mehreren Gliedern während weiter entfernte Halme wenig oder gar nicht unterteilt werden (Abb. 2).

Zusätzlich werden weiter entfernte Objekte verschwommen dargestellt, um ihren mangelnden Detailgrad zu verschleiern. Damit soll gewährleistet werden, dass eine maximale Menge Grashalme erzeugt werden kann und trotzdem noch eine ausreichend hohe Bildrate erreicht wird.

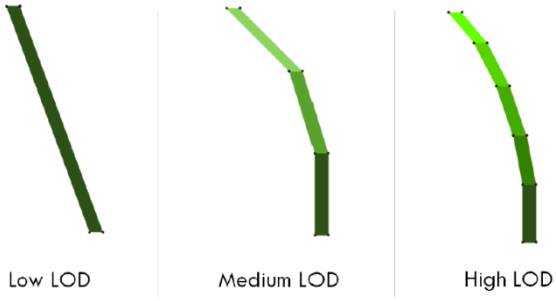

Abbildung 2: Level-of-Detail Darstellung von Halmen

Dieser Ansatz wurde in diesem Paper aufgegriffen und mit einigen Vereinfachungen, aber auch anderen Erweiterungen umgesetzt. Dazu zählen Elemente, die der Szene mehr Dynamik verschaffen, wie zum Beispiel die Darstellung von Insekten.

3. Konzept

Die Idee ist, jeden Grashalm dynamisch, während des Rendervorgangs, an einer vorher definierten Stelle, der Wurzel, zu erzeugen.

Die Dynamik dieser Technik schont zwar den Speicher der Grafikkarte, belastet aber deren Prozessoren. Durch die parallel rechnende Architektur der Grafikkarte, ist es nötig die Grashalme jeweils unabhängig behandeln zu können.

Dies ist kein Problem, da jeder Halm ein eigenständiges Objekt ist und ohne Wissen über die anderen Halme erstellt werden kann. Jeder Grashalm besteht aus mehreren Segmenten, damit er in der Animation durch den Wind gebogen werden kann. Es können sowohl für die Grashalme als auch für den Boden Level-of-Detail-Algorithmen verwendet werden, um die Performanz des Renderings zu steigern. Hierfür wird abhängig vom Abstand des Objektes zur Kamera der Detailgrad reduziert.

Aus dem ursprünglich ebenen Boden wird durch eine Höhenkarte und damit verbundenes Vertex-Displacement eine Hügellandschaft.

Als Hintergrund dient eine Skybox, die passend zur Beleuchtung einen Tag-Nacht-Wechsel vollzieht. Die Kamera ist frei beweglich und abrundend ziehen einige Schmetterlinge ihre Bahnen.

4. Realisierung

4.1 Grashalme

Die Grundlage für die dynamische Erzeugung der Grashalme ist der Geometry Shader. Dieser Teil der Rendering-Pipeline ermöglicht es, zu vorhandenen Primitiven neue hinzuzufügen.

Jeder Grashalm besteht aus bis zu fünf Segmenten, die jeweils aus einem Viereck bzw. zwei Dreiecken zusammengesetzt sind. Bei der Erzeugung des Halmes entstehen aus einem Punktprimitiv, der Wurzel, bis zu fünf Segmente. Für jedes Segment werden also aus einem Vertex bis zu 12 Vertices generiert, da sich zwei Vierecke immer zwei Vertices teilen. Wie viele Segmente ein Grashalm hat, hängt von der Distanz zum Betrachter ab. In der ersten, dem Betrachter nächsten Stufe, hat jeder Halm fünf Segmente. In der zweiten sind es drei Segmente und in der dritten Stufe eines.

Das dynamische Erzeugen von Geometrie führt allerdings dazu, dass sich Operationen, die per Vertex durchgeführt werden müssen, vom Vertex Shader in den Geometry Shader verlagern. Da der Vertex Shader in der Renderingpipeline vor Letzterem ausgeführt wird, reicht er lediglich alle Vertices (Wurzeln) an den Geometry Shader weiter.

Dieser muss zusätzlich zu den Transformationen in die verschiedenen Räume, die Normalen und Texturkoordinaten berechnen. Die Texturkoordinaten werden je nachdem, wie viele Segmente es gibt, so vergeben, dass sich die Textur über alle Segmente von der Wurzel bis zur Spitze erstreckt.

Durch die dynamische Erzeugung, eröffnen sich jedoch auch eine Reihe von Vorteilen, die den Speicherbedarf betreffen, wie im Konzept bereits angedeutet wurde. Der erste Vorteil ist, den Videospeicher der Grafikkarte nicht mit den Datensätzen (Position, Normale, Texturkoordinate) aller Grasvertices belasten zu müssen, sondern nur mit denen der Wurzeln. Zum Zweiten enthalten diese Datensätze lediglich die Position, da Normale und Texturkoordinate dynamisch berechnet werden.

Um den Grashalm an der Spitze nicht eckig, sondern rund darzustellen, wird zusätzlich zur normalen Textur eine Alphatextur benutzt, um festzustellen, welche Pixel zu setzen sind und welche aus der Pipeline entfernt werden müssen. Dieses Verfahren wird als *Alpha to Coverage* bezeichnet.

Die Animation des Grashalms setzt sich aus zwei Funktionen zusammen. Vorerst muss beachtet werden, wie sich ein Grashalm im Wind verhält. Es zeigt sich, dass die Beeinflussung an der Spitze des Halms am höchsten ist. Dies wurde durch eine Parabel mit dem Wertebereich $[0 \leq x \leq 1]$ realisiert.

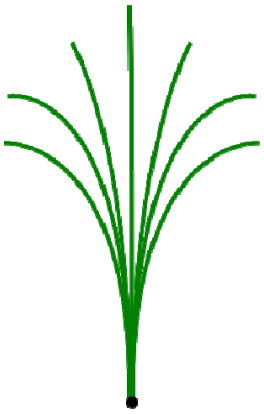

Abbildung 3: Bewegung eines Grashalms ([L 10] S. 42)

Der Wind wird über einen Windrichtungsvektor und die Windkraft geregelt. Sie steuern eine Sinusfunktion mit einer Zufallszahl sowie einer Phasenverschiebung für die Gruppierung von mehreren Grasblöcken. Somit wird ein Windfluss simuliert, bei dem der Wind oft wechselt und ein Wippen der Halme stattfindet (Abb. 3). Die Funktion wird bei der X- und Z-Achsenverschiebung des Halms normal angewendet, bei der Y-Achse jedoch im Betrag, da sich der Halm immer nach unten biegen soll.

4.2 Gelände

Als Grundlage für das Gras dient ein Gelände, welches zur Laufzeit aus einer Höhenkarte generiert wird. Neben dem Algorithmus zur Erstellung des Geländes, wird ein zweiter dazu verwendet, innerhalb der Polygone des Geländes, zufällige Punkte, die als Wurzeln dienen, zu setzten. Das Level-of-Detail wird auf diesem Gebiet durch die Anzahl der Grashalme pro Quadrat bestimmt, die je nach Entfernung zum Betrachter festgelegt ist. Um eine realistisch anmutende Graslandschaft umzusetzen, ist es zunächst wichtig eine Höhenkarte zu erstellen, die sich über eine möglichst große Fläche erstreckt. Dieser Punkt unterstreicht den schwierigsten Teil bei der Umsetzung einer Höhenkarte. Sie muss sparsam mit den verfügbaren Ressourcen umgehen und gleichzeitig genug Details beinhalten.

Prinzipiell ist eine Höhenkarte eine Ansammlung von Punkten mit einem bestimmten Wert, welcher der Höhe entspricht und einer horizontalen Anordnung im Raum. Dieser angesprochene Wert ist mit Hilfe einer Bitmap, welche nur aus Grauwerten besteht für alle vorhandenen Höhenpunkte gespeichert. Hierbei ist folgende Zuordnung festgelegt - Ein weißer Pixel entspricht dem höchst möglichen Punkt, ein schwarzer Pixel dem tiefsten. Alle dazwischen liegenden Grauwerte unterliegen einem linearen Verlauf. Die zugrunde liegende Bitmap wird nun verwendet um Vertices mit einer Höhe und einer horizontalen Ausrichtung in den Vertexbuffer zu schreiben. Ein solcher Vertex besteht unter anderem aus den drei Weltkoordinaten X, Y und Z.

Die Werte für X und Z werden durch einen festgelegten Parameter (Abstand) zunächst so gewählt, dass alle Punkte mit gleich bleibendem Abstand voneinander angeordnet werden. Der Parameter Y erhält seinen Wert aus der Grauwertkarte. Dadurch entsteht eine lückenhafte Fläche aufgebaut aus Punkten.

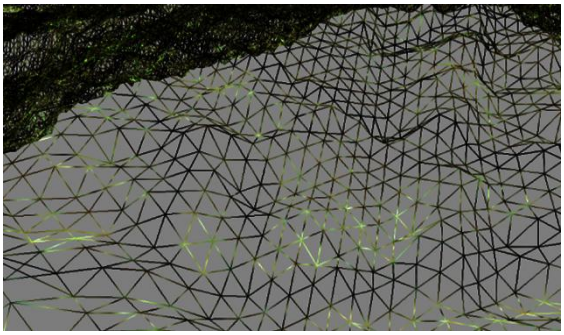

Abbildung 4: Wireframe der Höhenkarte

Um daraus eine Hügellandschaft zu erzeugen müssen alle Vertices miteinander verbunden werden (Abb. 4). Da in der Computergrafik alle visuellen Objekte durch Dreiecke dargestellt werden, muss auch beim Verbinden darauf geachtet werden, dass ausschließlich Dreiecke entstehen. Dies wird mit Hilfe des Indexbuffers umgesetzt. Alle Punkte, die in den Vertexbuffer geschrieben wurden, sind nach der Reihenfolge der Speicherung aufsteigend nummeriert. Nun muss ein Algorithmus verwendet werden um dem Indexbuffer zu signalisieren in welcher Reihenfolge er die einzelnen Vertices beim Zeichnen berücksichtigen soll. Dieser Algorithmus ist so konstruiert, dass er jeweils innerhalb von vier Punkten, die im Quadrat zueinander angeordnet sind, zwei Dreiecke beschreibt. Beinahe selbstverständlich müssen deshalb nahezu alle Punkte mehrmals im Indexbuffer stehen. Dies wird klar wenn man

bedenkt, dass ein Quadrat aus vier Punkten besteht und zwei Dreiecke aus sechs Punkten.

Die exakte Umsetzung entspricht folgendem Beispiel - Besteht eine Zeile Vertices aus 100 Elementen werden für die Behandlung der ersten vier Punkte die Vertices (0, 1, 100 und 101) verwendet. Die ersten sechs Einträge im Indexbuffer lauten dann:

[0] [101] [1] [0] [100] [101]

Das Ergebnis ist eine einfarbige Fläche, die ein Profil visualisiert, welches von der ihr zugrunde liegenden Grauwerttextur abhängt.

4.3 Wurzeln

Der Programmteil zur Erstellung der Wurzeln, fußt auf den Vertices des Geländes. Hierdurch wird sichergestellt, dass alle generierten Wurzeln bündig auf den Polygonen des Geländes aufliegen und keine Grashalme im Boden versinken oder in der Luft schweben.

Um eine faire Verteilung von Grashalmen sicherzustellen, wird in einem Parameter die Anzahl an Wurzelpunkten pro Polygon festgelegt. Zur tatsächlichen Erstellung von zufälligen Punkten, werden Baryzentrische Koordinaten verwendet.

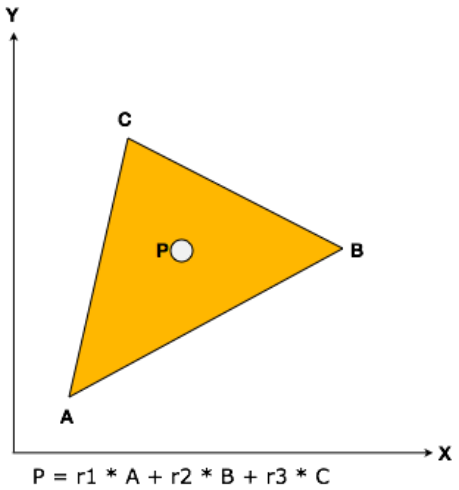

Abbildung 5: Baryzentrische Koordinaten

Diese helfen sicherzustellen, dass keine der zufälligen Koordinaten außerhalb der Fläche des betrachteten Dreiecks liegen (Abb. 5). Für einen zufälligen Punkt muss lediglich je eine zufällige Strecke auf zwei Geraden des Polygons ausgewählt werden, diese beiden Werte sind Teil der Baryzentrischen Koordinaten, welche besagen, dass alle drei Punkte der Dreiecks-Koordinaten in ihrer Summe eins ergeben müssen, damit der Rahmen des Dreiecks nicht verlassen wird. Es muss also sichergestellt werden, dass die Summe der beiden Zufallszahlen niedriger als eins ist um Platz für die dritte Koordinate zu lassen die hergeleitet werden kann.

Da im vorgestellten Programm alle Wurzeln statisch beim Programmstart generiert werden, ergeben sich bei zunehmender Kartengröße ansteigende Performanceprobleme in zweierlei Hinsicht - zum einen bezogen auf die benötigte Rechenzeit, die zur Erstellung der Wurzeln in Anspruch genommen wird, zum anderen wird durch die zunehmend massive Größe des Vertexbuffers die Pipeline der GPU überlastet.

Betrachten wir elementare LOD-Methoden, wie das Frustum-Culling, ergeben sich weitere Probleme mit der Verwendung eines monolithischen Vertexbuffers für alle Wurzeln. Das Frustum-Culling sieht vor, Objekte, die nicht im Sichtbereich des Betrachters liegen und daher keine Relevanz für die Szene darstellen, nicht in die Pipeline zu übertragen.

Da der für die Wurzeln verwendete Buffer statisch verwendet wird und somit nicht für eine nachträgliche Manipulation vorgesehen ist, ist das Programm darauf angewiesen, für jedes Bild sämtliche Wurzeln zu verarbeiten. Außerdem ist es nicht effizient für jeden Grashalm einen separaten Buffer anzulegen, deshalb wird die Wurzelkarte in kleine Quadrate unterteilt, welche in einer Baumstruktur sortiert werden. In dieser Struktur enthält jedes Quadrat einen Bruchteil der Wurzeln der gesamten Wurzelkarte, und verfügt über einen eigenen Vertexbuffer. Dabei wird jeder dieser Ausschnitte - im folgenden Chunk genannt - in 4 Quadranten unterteilt, welche die Kindsknoten dieses Chunks bilden. Diese Unterteilung wird bis zu einer angegebenen Maximaltiefe des Baums durchgeführt.

Zur Laufzeit können nun mittels der Position und Blickrichtung des Betrachters, die Chunks ausgewählt werden, die für die Szene von Relevanz sind. Die Baumstruktur erlaubt

es ebenfalls, in jeder Ebene von Chunks eine andere Detaildichte anzuwenden, sodass für weiter entfernte Bereiche, Chunks mit niedrigeren Detailstufen ausgewählt werden können. Dabei beinhalten Chunks, welche einen großen geometrischen Bereich abdecken, weniger Details, als kleine Chunks, die in unmittelbarer Nähe zur Kamera Verwendung finden. Die angewendeten Algorithmen zur Selektion der Chunks, orientieren sich dabei stark an Strugar Filips Algorithmen aus dem Paper "Continuous Distance-Dependent Level of Detail for Rendering Heightmaps“ (vgl. [S 10]).

4.4 Ambiente

Der Tag und Nachtwechsel der Szene wurde zum einen durch die Änderung der Lichtfarbe und zum anderen über die Texturen der Skybox realisiert. Der Wechsel nutzt die Kombination aus einer Sinus- und einer Modulofunktion. Für die Lichtberechnung wurde das Phong-Modell gewählt. Zur Auflockerung der Szene wurden Schmetterlinge, in Form von zwei Vierecken, eingefügt, die im Vertex Shader durch eine Sinusfunktion animierbar sind.

5. Ergebnisse

Nach einer zweimonatigen Realisierungsphase kam das Projekt zu folgendem Ergebnis (Abb. 6).

Abbildung 6: Wiese bei Sonnenuntergang

Es befinden sich zurzeit 6.029.312 Grashalme auf der Wiese; als Weichzeichner wurde eine Blur-Funktion implementiert (siehe Abb. 7 + 8).

Abbildung 7: Gras ohne Blur

Abbildung 8: Gras mit Blur

Bei dem unten genannten Testrechner ergeben sich folgende Werte:

Ausstattung	Auflösung	FPS
Intel i7 950, 12GB RAM, Sapphire Radeon HD 6850 1GB	1600x800	49

Tabelle 1: Ergebnistabelle mit Frames-per-Second-Auswertung

Es zeigt sich, dass ein sehr gutes Ergebnis erzielt werden konnte, sowohl optisch als auch technisch.

Werden die Ergebnisse mit dem Ausgangspaper von Edward Lee verglichen, ist erkennbar, dass es durch neue Hardware 2010/2011 einen Leistungsschub gab, und somit die Ergebnisse nicht mehr direkt vergleichbar sind. Anbei wurden zur Vollständigkeit die Werte für die jeweilige Gesamthalmzahl - (1) ~30.000 (2) ~90.000 - dargestellt.

Ausstattung	Auflösung	(1) FPS	(2) FPS
Intel Core 2 Duo P8600, 4GB RAM, NVIDIA GTX 260M	1680x1050	49	23
	1920x1200	44	21

Tabelle 2: Ergebnistabelle mit Frames-per-Second Auswertung von Edward Lee (Werte enthalten alle LOD Methoden und Animationen) (vgl. [L 10] S. 61 ff)

In einer vergleichsweise kurzen Bearbeitungszeit von nur 15 Wochen wurde ein Ergebnis herausgearbeitet, welches mit anderen Echtzeitanimationen auf diesem Gebiet vergleichbar ist. Dies ist der Tatsache zuzuschreiben, dass nicht nur das Gebiet der Grashalm-Visualisierung intensiv bearbeitet wurde, sondern auch andere Teilgebiete graphisch einem höheren Anspruch genügen. Zum Einen verbessert die animierte Skybox das Ergebnis, zum Anderen ist es durch die Verwendung der dynamischen Höhenkarte möglich, eine beliebige, bewachsene Hügellandschaft zu erstellen.

6. Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass mit Hilfe gängiger Methoden eine Landschaft aus Gras virtuell implementiert werden konnte. Die Methode zur grafischen Darstellung mit Hilfe des Geometry Shaders hat sich als effektiv erwiesen. Ebenso haben sich die Ansätze von Lee, Boulanger und Filip Strugar im Hinblick auf ein Level-of-Detail für einzelne Grashalme, zur Bewegung dieser durch eine Windsimulation, als auch zur Verbesserung der Höhenkarte im Sinne von Größe und Bildwiederholungsrate, als wirksam und effizient herausgestellt. Bei Verlängerung des Projektzeitraums könnten weitere Features, wie unendlich weite Grasflächen, realisiert werden. Zudem wäre es möglich, dass das Gras mit Objekten oder sich selbst kollidiert, was jedoch zu aufwendigen Berechnungen führen könnte. Eine weitere optische Verbesserung wäre der Schattenwurf, dieser ist jedoch sehr ressourcenbelastend, da für das gängige Shadowmapping ein kompletter Renderpass zusätzlich benötigt wird.

Dieses Verfahren eignet sich zunächst hauptsächlich für den Einsatz in den Unterhaltungsmedien. Werden allerdings die zuvor erwähnten anspruchsvollen Kollisionssysteme bedacht, lässt sich diese Technologie auch für komplexe Simulationen einsetzen. Es ist außerdem erwähnenswert, dass sich die hier vorgestellte Methode zwar gut zum Darstellen und Simulieren von großen Mengen einzelner Grashalme eignet, allerdings ebenso gut auch für die Simulation von Haar und Fell angepasst werden kann.

Jedoch wird der Einsatz in Echtzeitsystemen durch die hohen Anforderungen an die Hardware noch stark eingeschränkt. Neben der Darstellung der Grasfläche bleiben kaum weitere Rechenressourcen für andere Renderaufgaben. Ohne ausgereifte LOD-Methoden lässt sich Beispielsweise ein Einsatz in Architektursoftware vorstellen. Betrachtet man jedoch eine Weiterentwicklung mit einem starken Fokus auf die Kombination mit durchdachten LOD-Verfahren, ist es realistisch, dass die hier gezeigten Methoden in Echtzeitanwendungen integriert werden können und dort zu einer maßgeblichen Steigerung des Realismus beitragen können.

Literatur:

[BLH 02] Bakay, B, Lalonde, P, Heidrich, W 2002, ‘Real Time Animated Grass‘, paper to be presented at Eurographics Conference, The Eurographics Association.

[BW 06] Banisch, S & Wüthrich, CA 2006, ‘Making Grass and Fur Move‘, paper to be presented at the WSCG‘ 2006, The 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision‘ 2006, Bauhaus-University Weimar.

[B 08] Boulanger, K 2008, ‘Real-Time Realistic Rendering of Nature Scenes with Dynamic Lighting‘, PhD thesis, University of Rennes 1, France.

[L 10] Lee, E 2010, ‘Realistic Real-Time Grass Rendering‘, Master thesis, University of California, Los Angeles, 17 December.

[S 10] Strugar, F 2010, 'Continuous Distance-Dependent Level of Detail for Rendering Heightmaps', 11 July.