Techniken zur Visualisierung von Eis- und Schnee-Landschaften mit programmierbarer Grafik-Hardware

Michael Horsch

Hochschule Darmstadt
Fachbereich Informatik

17. Februar 2008

Inhaltsverzeichnis

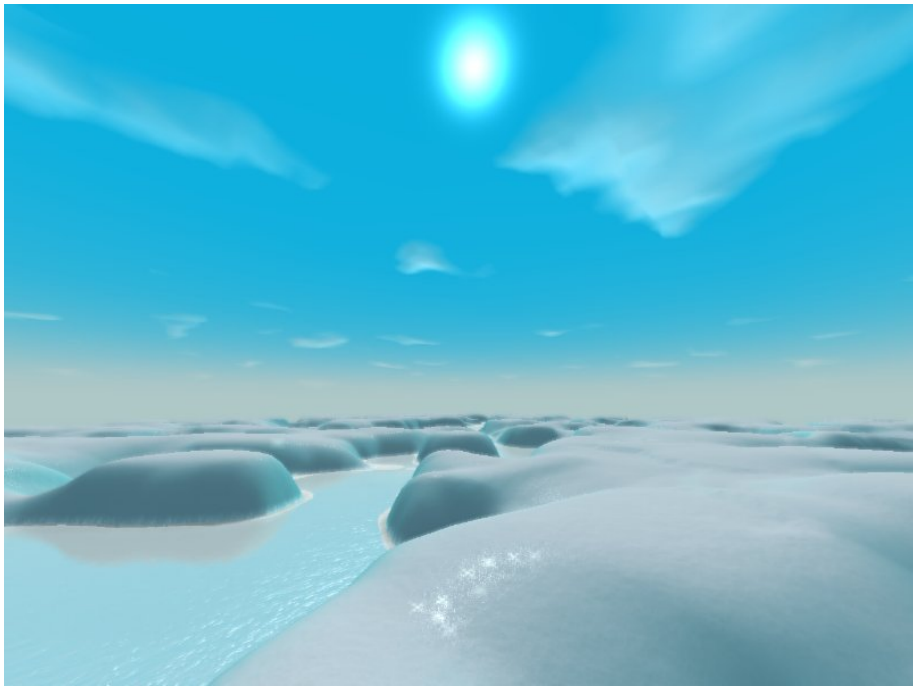

Abbildung 1: Ein fertiges Bild aus dem Demonstrationsprogramm

1 Einleitung

In dieser Arbeit werden Techniken zur Visualisierung von Eis- und Schnee-Landschaften mit programmierbarer Grafik-Hardware vorgestellt.
Solche Techniken werden vor allem für die realitätsnahe Simulation echter Landschaften benötigt, wie z.B. in GIS-Anwendungen oder Wettersimulationen. Aber auch Unterhaltungsmedien wie PC-Spiele können die Techniken nutzen um virtuelle Winter-Welten darzustellen.
Im Vorfeld der Arbeit wurde ein Demonstrationsprogramm entwickelt, in dem die hier vorgestellten Techniken erfolgreich implementiert wurden.

2 Anforderungen

Um die Landschaften realistisch darzustellen werden verschiedene Techniken für die verschiedenen Effekte benötigt. So muss die Landschaft selber,

Gewässer wie Flüsse oder Seen, der Himmel und Wettereffekte wie Schneefall und Blitze dargestellt werden um ein ganzheitliches Ergebnis zu erreichen.

Die hier vorgestellten Techniken sollen nur zwei Anforderungen erfüllen:

- Alle Effekte sollen mit Hilfe programmierbarer Hardware erzeugt und beschleunigt werden.
- Alle für die Visualisierung nötigen Daten sollen prozedural erzeugt werden

Die erste Anforderung ist nötig um eine Echtzeit-Darstellung bei interaktiven Frameraten der Landschaften zu gewährleisten. Die zweite Anforderung wurde rein willkürlich festgelegt. Da die Daten prozedural erzeugt werden, wird kein Speicherplatz auf der Festplatte benötigt und der Detail-Grad lässt sich frei festlegen. Weiterhin können auf diese Weise unendlich viele Variationen der Effekte erzeugt werden.

3 Stand der Technik

Die Darstellung von Landschaften ist ein sehr großes und aktives Forschungsgebiet, in dem bereits viele Techniken entwickelt worden sind. So findet man z.B. auf [1] eine große Auswahl an wissenschaftlichen Arbeiten über die Darstellung von Landschaften. Die dort vorgestellten Techniken behandeln jedoch immer nur ein spezielles Thema, während diese Arbeit eher als Sammlung von Konzepten bzw. Techniken anzusehen ist.
Für die Darstellung eines Gelände-Mesh gibt es verschiedene Techniken zur Beschleunigung. In [2] werden z.B. verschachtelte Gitter benutzt, deren Detail mit zunehmender Entfernung zum Betrachter abnimmt. So erreicht man eine hohe Detailstufe in der Umgebung des Betrachters. Auch sehr bekannt ist der ROAM-Algorithmus [3], bei dem in Echtzeit ein optimales Adaptives Mesh (Real-time-Optimally-Adapting-Mesh) zur Darstellung erzeugt wird. In fast allen Fällen beschränkt sich das Gelände jedoch auf Höhenfelder, d.h. Höhlen oder Bergüberhänge können nicht dargestellt werden. Um dieses Problem zu umgehen könnte man sog. Voxel benutzen, jedoch arbeitet

moderne Grafik-Hardware mit Polygonen. Ein Voxel-Volumen kann zwar polygonalisiert werden, dieser Vorgang ist aber selbst auf moderner Hardware noch langsam. Der Vorteil von Höhenfeldern besteht darin, dass viele Berechnungen stark vereinfacht werden können.
Die Darstellung des Himmels wird in den meisten Anwendungen mit einer sog. Sky-Box realisiert. Diese benötigen jedoch spezielle Texturen, d.h. die Daten werden nicht prozedural erzeugt und können somit nicht automatisch variiert werden. In [4] wird dagegen eine Technik zur prozeduralen Erzeugung von Wolken und dem Himmel vorgestellt. Oft wird der Himmel auch in Wolken und Atmosphäre unterteilt und getrennt behandelt. In [5] wird ein analytisches Modell für Tageslicht vorgestellt welches auf den Techniken aus [6] basiert. Wolken werden oft mit zum Betrachter orientierten Partikel-Systemen in Form einer Wolke realisiert. Solche Techniken benötigen jedoch viel Rechenzeit.
Auch für die Darstellung von Gewässern gibt es eine Vielzahl an Publikationen, so wird auf [7] ein Algorithmus vorgestellt mit dem ruhiges fließendes Wasser dargestellt werden kann. Ein ähnlicher, aber komplexerer Wasser-Effekt wird in [8] vorgestellt, bei dem die Wellen über eine FFT erzeugt werden.
Zur Darstellung von Blitzen existieren leider kaum Arbeiten. In [9] wird jedoch ein relativ realistisches Modell vorgestellt.
Der Schneefall wird in allen bekannten Anwendungen über ein Partikel-System erzeugt. Mit Partikel-Systemen können ohne viel Aufwand viele Effekte erzeugt werden, jedoch kann sich eine sehr hohe Anzahl an Partikeln negativ auf die Ausführungsgeschwindigkeit auswirken. Daher muss hier immer ein Kompromiss zwischen Leistung und Qualität gemacht werden.
Das Programm [10] ist ein Landschaftsgenerator welcher in etwa die gleichen Anforderungen hat wie die im folgenden vorgestellten Techniken. Der Quellcode ist nicht öffentlich verfügbar, aber allein durch Beobachtungen der Ausgaben des Programmes kann man viele Rückschlüsse auf die Implementierung bzw. die implementierten Konzepte ziehen.
Für die prozedurale Berechnung von Daten zu Darstellung von Landschaften wird häufig ein koherentes Rauschen als Basis benutzt. Durch die Auf-

summierung von Rauschen mit unterschiedlichen Frequenzen erhält man ein fraktales Rauschen. Mit verschiedenen Parametern und mathematischen Operationen kann eine Vielzahl an Texturen mit unterschiedlichen Eigenschaften berechnet werden. Bekannte Rauschtypen sind z.B. fBm-Fraktale oder Perlin-Noise.

4 Rauschen

Die Rauscherzeugung ist ein wichtiger Bestandteil für die Darstellung der gesamten Szene. So wird sowohl das Höhenfeld für das Gelände als auch alle verwendeten Texturen über verschiedene Rauschfunktionen erzeugt.
Als Basis wird für jede Rauschfunktion Perlin-Noise verwendet. Durch Veränderung der Ausgabewerte können bereits viele Variationen erzeugt werden.
Für das Gelände wird eine invertierte Ridged-Multi-Perlin-Noise-Funktion verwendet. Ohne die Invertierung würde eine mondähnliche Landschaft mit Kratern entstehen, die jedoch nicht für Schneelandschaften geeignet ist. Mit der Invertierung entstehen dagegen eher rundliche Landschaften. Genau dieser Effekt wird benötigt, da auf den meistens eher flachen Schneelandschaften oft ein Wind weht durch den Schnee und Eis abgetragen wird. Eine reine Perlin-Noise-Funktion eignet sich dagegen sehr gut für Gebirgsketten.
Für die Erzeugung von Texturen werden Variationen von Perlin-Noise benutzt. Insgesamt gibt es für die Texturen 3 unterschiedliche Rauschfunktionen: Perlin-Noise, Perlin-Noise-Absolutewerte und exponentiertes Perlin-Noise. Diese 3 Typen reichen für alle Effekte aus. Zusätzlich werden für diese Texturen die zugehörigen Normal-Maps berechnet.

5 Visualisierung des Geländes

Für die Visualisierung des Geländes wird zuerst ein quadratisches Mesh mit konstanter Unterteilung auf der XZ-Ebene erzeugt. Für jeden Vertex wird mit einer invertierten Ridged-Multi-Perlin-Noise-Funktion die Y-Koordinate berechnet. Durch die Invertierung entsteht das Aussehen einer flachen Landschaft mit Flußläufen und Seen. Außerdem wird für jeden Vertex die Nor-

male berechnet, sowie die Texturkoordinaten. Danach kann das Gelände zeilenweise gezeichnet werden. Da das Demonstrationsprogramm räumlich begrenzt ist, wurde der Einfachheit halber auf die Verwendung einer Technik zur Beschleunigung des Zeichenvorgangs verzichtet.
Für das Shading des Geländes gibt es verschiedene Ansätze. So könnte man einfach in Abhängigkeit von der Höhe des Geländes unterschiedliche Materialberechnungen vornehmen. An tiefen Stellen könnte sich z.B. Kies befinden, an mittelhohen Stellen Gestein und an hohen Stellen dann Schnee. Dieses Vorgehen sieht aber nicht in allen Fällen realistisch aus. In einer Eis- und Schnee-Landschaft würde man an ebenen Stellen Schnee erwarten, da dieser an steilen Abhänge hinunter rutschen bzw. abprallen würde. An den steilen Stellen würde man wohl eher hartes Eis oder Gestein finden, das nur langsam abgetragen wird. Mit anderen Worten heißt das, dass man in Abhängigkeit der Steigung an einem bestimmten Punkt zwischen Schnee und Eis interpolieren will. Für diese Aufgabe werden die Normalen benutzt. Da die Normalen die Orientierung der Geländeoberfläche angeben, das Gelände auf der XZ-Ebene liegt und die Normalen normalisiert sind, eignet sich die Y-Komponente der Normalen als Metrik für die Steilheit. D.h. die Y-Komponente der Normalen wird als Interpolationsfaktor benutzt um zu bestimmen an welchen Stellen sich Schnee befindet und an welchen Eis. Um die Schärfe des Übergangs von Schnee zu Eis zu kontrollieren wird der Wert noch potenziert. Wird kein weicher Übergang gewünscht, könnte auch ein Schwellwert für die Steilheit festgelegt werden.
Für das Eis-Shading wird eine exponentierte Perlin-Noise Textur verwendet. Durch diese entstehen je nach Grauwert an vereinzelten Stellen Verunreinigungen bzw. Lufteinschlüsse. Diese Textur wird dann nur noch mit einem leuchtenden Türkis moduliert, wodurch das Eis ein bisschen so aussieht als würde es das Licht streuen. Für die Beleuchtungsberechnungen werden beim Eis die Vertex-Normalen benutzt.
Das Verfahren für die Darstellung des Schnees ist dagegen etwas komplexer. Hier wird zusätzlich eine Normal Map benutzt. Da die Normal Map über das Gelände gestreckt wird, ist der Detail-Grad abhängig von der Texturauflösung. Um einen viel höheren Detail-Grad zu erreichen wird die Nor-

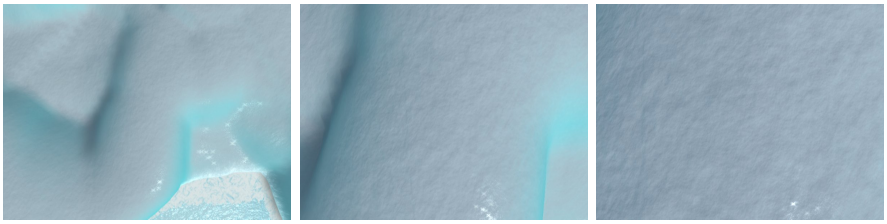

Abbildung 2: Der hohe Detail-Grad des Schnees wird durch die Aufsummierung von Normal Maps für unterschiedliche Frequenzen erreicht.

mal Map im Shader mehrfach für unterschiedliche Frequenzen geladen und der Mittelvektor gebildet. Auf diese Weise ist der Detail-Grad frei bestimmbar (siehe Abb. 2). Um die endgültigen Normale zu bestimmen wird die im Shader aus der Normal Map berechnete Normale noch mit der Vertex-Normale gemittelt.

Die Berechnung des Nebels erfolgt im gleichen Shader. In Abhängigkeit von den Tiefenwerten der Pixel wird zwischen einer Nebelfarbe und der Nebellosen Szene interpoliert. So sind Objekte im Vordergrund klar erkennbar, während Objekte in der Ferne an Farbe und Kontrast verlieren und dadurch vernebelt wirken. Die Dichte des Nebels ist von der Bewölkung abhängig.

An den Grenzen zu Gewässern soll sich Eis bilden. Hierfür wird einfache eine Noise-Textur in Abhängigkeit von der Entfernung zur Wasseroberfläche zum Schnee bzw. zum Eis dazuaddiert. Dadurch wird der Übergang vom Gewässer zur Landschaft weicher.

Für die Beleuchtung des Geländes wird das Phong-Shading verwendet. Unregelmäßigkeiten im Glanzlicht werden durch Modulation mit einer Specular Map erzeugt. Eine Specular Map ist eine Grauwert-Textur mit der bestimmt wird welche Stellen wie stark reflektieren. Das Glitzern des Schnees wird mit einem Post-Processing-Effekt simuliert (Abbildung 3). Zuerst wird dafür das Glanzlicht in eine Textur gezeichnet. Diese Textur wird dann verkleinert und weichgezeichnet um Aliasing-Artefakte zu verhindern. Als nächstes wird im Shader überprüft ob das Glanzlicht über einem bestimmten Schwellwert liegt. Tut es das, wird ein kreuzförmiger Mittelwert-Filter auf die Textur angewendet, anderfalls wird die Pixel-Farbe auf Schwarz gesetzt. Dieses Er-

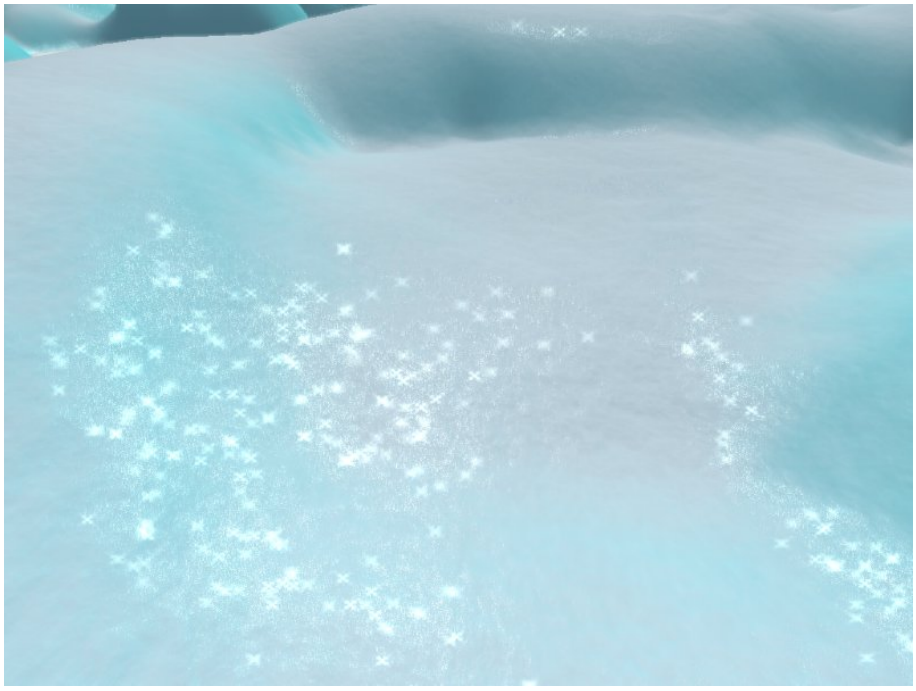

Abbildung 3: Schneeglitzern als Post-Processing Effekt

gebnis wird dann wiederum in eine Textur gezeichnet, die zum Schluß in einem Post-Processing Schritt zur restlichen Szene addiert wird.

Um Schatten zu erzeugen stehen verschiedene Verfahren zur Auswahl: Stencil Shadows, Shadow Mapping und Ray Marching. Stencil Shadows eignen sich gut für große Areale, kosten jedoch viel Leistung, besonders wenn weiche Schatten gewünscht sind. Mit Shadow Mapping können zwar einfach weiche Schatten erzeugt werden, jedoch ist es nicht für große Areale geeignet. Dies hat mehrere Gründe: Die Schattenauflösung ist abhängig von der Auflösung der Shadow Map, es gibt Aliasing-Artefakte, die Tiefenwerte in der Shadow Map können nicht gleichmäßig über da Gelände verteilt werden und wegen mangelnder Präzision können große Areale nicht in eine einzelne Shadow Map gezeichnet werden. Diese Probleme werden zwar von verschiedenen Variationen des Algorithmus adressiert, aber es wird mehr Leistung benötigt und/oder die Lösungen sind nicht robust. Aus diesen Gründen bie-

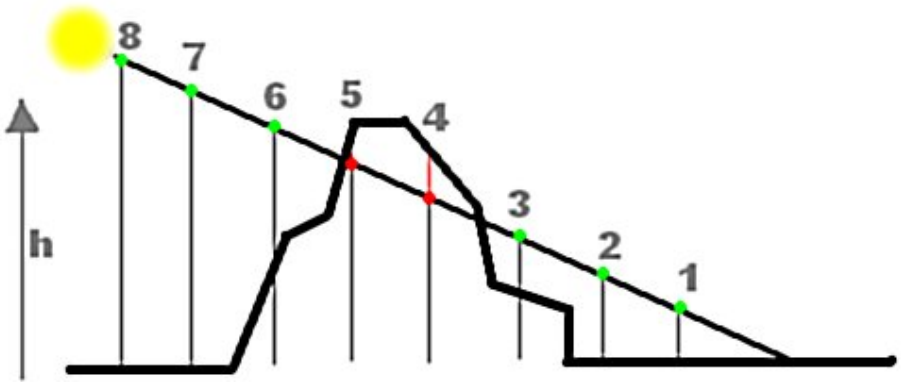

Abbildung 4: Ray Marching Beispiel: Beginnend bei Punkt 1 wird die Höhe des Strahls zur Sonne in regelmäßigen Abständen mit der Höhe des Geländes verglichen. Ist die Höhe des Strahls an einer der Stellen kleiner als die des Geländes (Punkt 4 und 5), liegt der Punkt im Schatten.

tet sich Ray Marching an. Hierbei wird ein Strahl entlangmarschiert und bei jedem Schritt wird überprüft ob sich an der aktuellen Position ein Objekt befindet. Dieses Vorgehen kann zwar je nach Detail-Grad viel Leistung kosten, dafür ist die Berechnungszeit konstant und auch mit niedrigen Detail-Einstellungen lassen sich gute Ergebnisse erzielen, zudem ist es für große Areale geeignet und es können einfach weiche Schatten erzeugt werden. Um die Schatten für das Gelände zu erzeugen wird das Ray Marching im Shader mit Hilfe einer Height Map des Geländes benutzt. D.h. von jedem Pixel in der Height Map wird ein Strahl in Richtung Sonne gebildet, ist die Y-Koordinate des Strahls bei einem Schritt kleiner als die Höhe des Geländes bei diesem Schritt, liegt der Punkt im Schatten, siehe Abbildung 4. Das Ergebnis wird in einer Textur gespeichert. Durch die Texturfilterung haben die Schatten automatisch weiche Kanten, um mögliche Treppeneffekte zu entfernen kann aber noch zusätzlich ein Unschärfe-Filter angewendet werden. Die Schatten-Textur wird dann einfach über das Gelände gelegt.

6 Visualisierung des Himmels

Der Himmel wird mit der auf [4] vorgestellten Technik berechnet. Einzelne Schritte werden leicht angepasst. Der Vorteil dieser Technik ist die Einfachheit der Berechnungen bei plausiblen Ergebnissen ohne große Leistungseinbußen.
Die Geometrie des Himmels ist eine Kugel. Die Kugel wird auf der y-Achse runterskaliert um die Größenverhältnisse realistisch zu gestalten. Eine Kugel im Maßstab 1:1 würde nicht in das View Frustum passen, die Präzision des Tiefenpuffers würde zu sehr darunter leiden. Für jeden Vertex wird eine Normale generiert. Als Texturkoordinaten wird die XZ-Koordinate des Vertex benutzt. So entsteht ein planarer UV-Raum, wodurch spätere Berechnungen vereinfacht werden können.
Die Farbe der Atmosphäre wird durch die Normale bestimmt. Der Himmel soll eine blaue Farbe haben, in der Entfernung soll er aber durch Dunst und Nebel verdeckt werden. Wieder wird die Y-Komponente der Normale als Interpolationsfaktor benutzt.
Die Wolken werden über Texturen erzeugt. Zuerst werden ein paar Perlin-Noise Texturen die sich in unterschiedliche Richtungen bewegen gemittelt. Um die Dichte der Wolken zu berechnen wird das Ergebnis in Abhängigkeit von einem Bewölkungs- und einem Dichte- Parameter potenziert. Dadurch bleiben nur sehr helle Stellen erhalten die sehr an Wolken erinnern.
Um die Sonne zu erzeugen wird das Skalarprodukt zwischen Normale und Richtung zur Sonne berechnet und potenziert um ein klarere Abtrennung zum Himmel zum bekommen.
Für die Beleuchtungsberechnung wird diffuses Phong-Shading verwendet. Da die Wolken erst im Shader berechnet werden, müssen die Normalen der Wolken ebenfalls dort dynamisch berechnet werden. Die Lichtstreuung in den Wolken wird um Leistung zu sparen nicht berechnet, wodurch die Wolken leider weniger realistisch wirken.

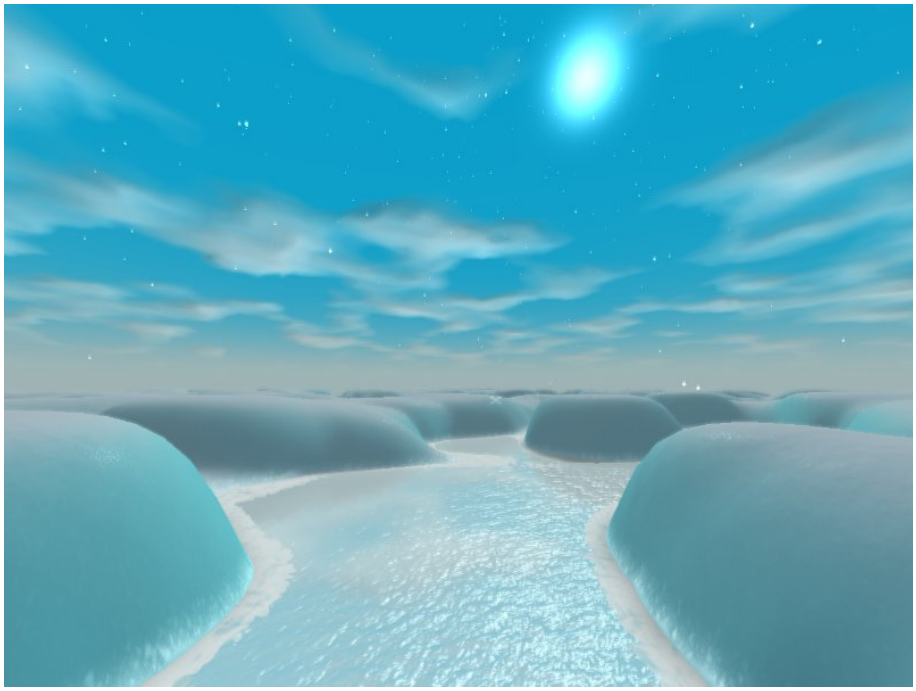

Abbildung 5: Die Bewölkung des Himmels, die Dichte der Wolken und die Position der Sonne sind frei einstellbar

7 Visualisierung von Gewässern

Da in einer Landschaft auch Gewässer vorkommen, sollen diese auch dargestellt werden können. Hiefür wird die in [7] vorgestellte Technik mit leichten Veränderungen benutzt. Mit ihr können die wichtigsten optischen Eigenschaften wie Reflexion und Refraktion simuliert werden. Als Geometrie wird eine einfache Ebene benutzt, die Wasserhöhe kann frei eingestellt werden. Die Normalen werden mit einer Normal Map bestimmt.
Um Reflexionen zu erzeugen wird ein auf Texturen basierter Ansatz benutzt. Bei einer perfekt planaren Spiegelung kann die Geometrie an der XZ-Ebene gespiegelt werden um die reflektierte Geometrie zu darzustellen. Eine negative Skalierung der Y-Achse sorgt für die nötige Spiegelung. Zusätzlich muss die Geometrie noch unterhalb der Wasserhöhe abgeschnitten werden, da sonst auch die unter der Wasseroberfläche Geometrie gespiegelt werden würde. Die gespiegelte Geometrie wird dann in eine Textur gezeichnet und vom Betrachter aus wieder auf die Wasser-Ebene projeziert.
Die Refraktion wird auf ähnliche Weise erzeugt. Hier genügt es aber die normale Szene in eine Textur zu zeichnen. Auch diese Textur wird dann wieder auf die Wasser-Ebene projeziert.
Die Wellen werden mit Hilfe der Normal-Map berechnet. Da Wellen die Reflexion und Refraktion verzerren, wird im Shader das gleiche gemacht. Dazu wird die XZ-Komponente der Normale zu den Textur-Koordinaten für die Reflexions- und Refraktions-Textur addiert. So erhält man eine leicht gekräuselte Gewässeroberfläche. Für einen höheren Detail-Grad können wieder mehrere Normal-Maps mit unterschiedlichen Frequenzen aufaddiert werden um die Normale zu berechnen. Für fließendes Wasser können die Textur-Koordinaten entlang einer Richtung verschoben werden, für gefrorenes Wasser wird dieser Schritt natürlich nicht benötigt.
Reflexion und Refraktion werden dann in Abhängigkeit vom Fresnel-Faktor gemischt. Der Fresnel-Faktor gibt an, wieviel Licht in eine Richtung reflektiert bzw. gebrochen wird. Der Fresnel-Faktor ist sehr wichtig für den Realismus. Bei gefrorenem Wasser erhalten Reflexion und Refraktion noch eine weißliche Färbung.

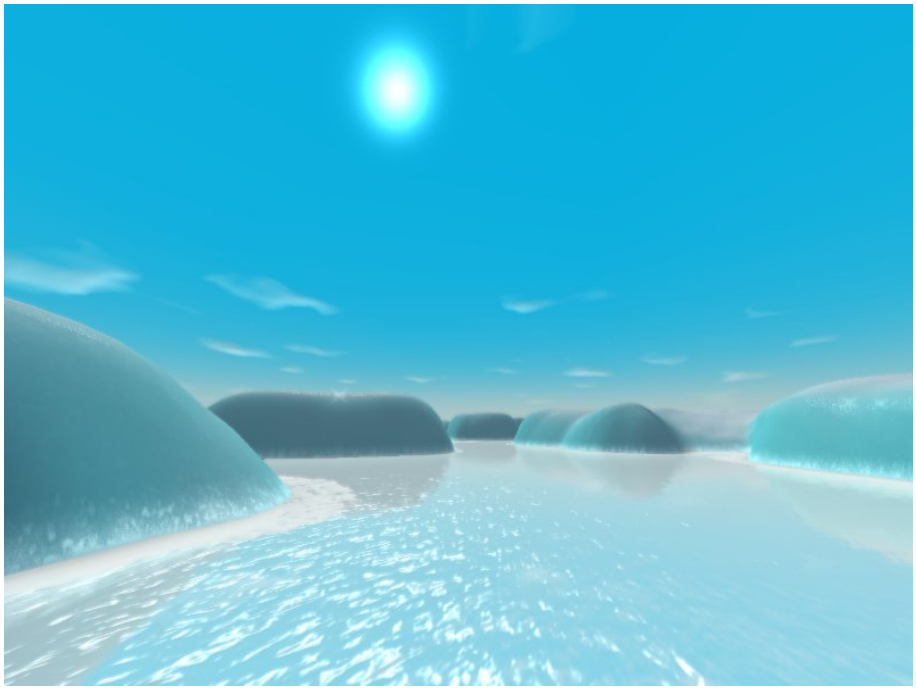

Abbildung 6: Wasser mit Reflexionen und Lichtbrechung

Zusätzlich bilden die Gewässer an den Grenzen zur Landschaft Eisschollen. Dazu wird zwischen einer Noise-Textur und dem Wasser in Abhängigkeit von der Tiefe des Wassers interpoliert.
Das Glanzlicht auf dem Wasser wird durch spekulares Phong-Shading berechnet. Die Schatten werden wie die Schatten des Geländes berechnet. Um Leistung zu sparen werden die Gewässer-Schatten zusammen mit Gelände-Schatten berechnet und in der gleichen Textur gespeichert.

8 Visualisierung von Schneefall

Für die Visualisierung des Schneefalls wird ein Partikel-System benutzt. Der erste Ansatz war aber den Schneefall als Post-Processing-Effekt zu realisieren. Hierfür wurden Versuche mit der in [11] vorgestellten Technik unternommen, die aber statt Schneefall Regenfall erzeugt. Jedoch stellte sich schnell

heraus, dass diese Technik nicht für Schneefall geeignet ist, da verschiedene Annahmen gemacht werden die zwar für Regenfall gelten, sich aber nicht auf Schneefall übertragen lassen. Daher wurde der Schneefall mit dem klassische Ansatz von Partikel-Systemen realisiert. Die Partikel werden dabei als Quadrate gezeichnet, die immer in Richtung zum Betrachter orientiert sind.

Jeder Partikel hat nur zwei Eigenschaften: Die Position und die Flugrichtung. Die Flugrichtung kann als Parameter angegeben werden. Für jeden Partikel wird sie leicht variiert. Auch die Richtung des Windes beeinflusst die Flugrichtung. Bei klarem Wetter mit wenig Wind fällt der Schnee eher steil herab, während bei einem Sturm der Schnee schneller fliegt und dadurch in einem flacheren Winkel herunterfällt.

Am Anfang wird die Position zufällig initialisiert. Die möglichen XZ-Koordinaten werden dabei auf die Dimensionen des Geländes beschränkt. Die mögliche Y-Koordinaten werden auch auf eine bestimmte Höhe eingeschränkt, so wirkt der Schnee als würde er aus den Wolken fallen. Die Partikel fliegen bis sie unter der Gewässeroberfläche sind. Danach werden die Partikel wieder neu initialisiert.

Das Partikel System benutzt eine feste Zahl an Partikeln, wobei jedoch auch nur ein Teil davon dargestellt werden kann. Da es ohne Wolken kein Schnee geben kann, setzt der Schneefall erst überhalb eines Schwellwertes für die Bewölkung ein. Mit zunehmender Bewölkung wird auch die Zahl der Schnee-Partikel erhöht. Der Schneefall passt sich auf diese Weise dem Wetter an.

Da Schnee als rundliche Partikel wahrgenommen wird, wird im Shader die Distanz zur Mitte des Partikels berechnet. Mit einer Normal-Map wird diese Distanz verzerrt um Unregelmäßigkeiten zu erzeugen. Der endgültige Wert wird dann benutzt um mit Alpha-Testing und Blending Pixel außerhalb einer festgelegten Distanz durchsichtig zu machen. So entstehen rundliche Schneeflocken.

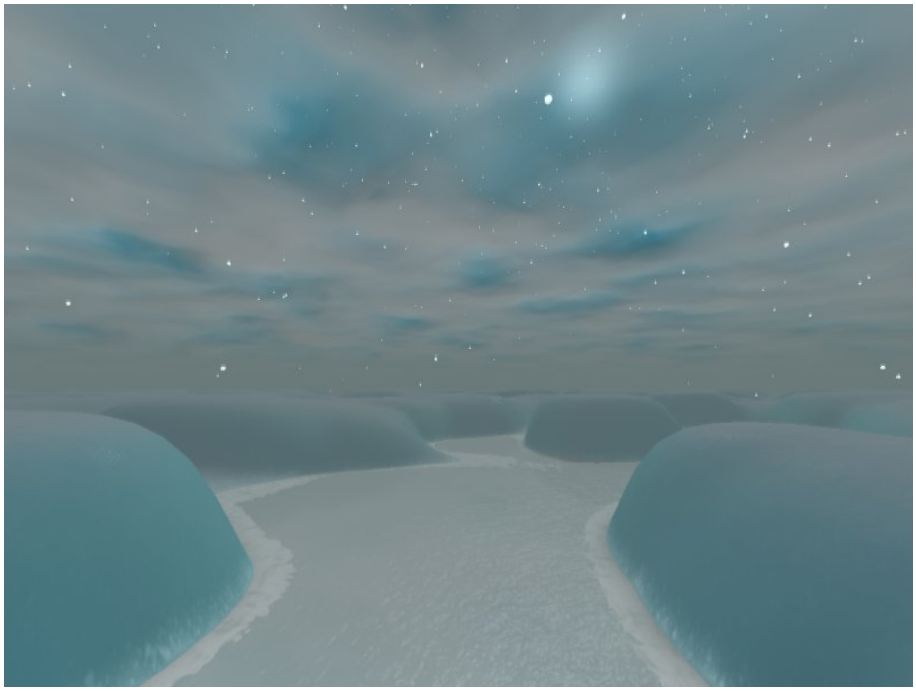

Abbildung 7: Ab einer bestimmten Bewölkungsstufe setzt der Schneefall ein

9 Visualisierung von Blitzen

Die Blitze werden als Post-Processing-Effekt realisiert.
Hierzu wird zuerst die Geometrie der Blitze erzeugt. Dazu werden zufällige Punkte entlang der Strecke vom Austrittspunkt der Blitzes zum Auftreffpunkt erzeugt und durch weiße Linien miteinander verbunden. Für jeden Blitzeinschlagen werden mehrere dieser Linienfolgen generiert.
Die Geometrie wird dann in eine Textur gezeichnet. Um das Glühen des Blitzes zu simulieren wird diese Textur dann weichgezeichnet und zum Ausgangsbild addiert. Da die Linien immernoch zu gerade für einen Blitz sind, werden sie weiterhin mit einer Normal-Map verzerrt. Dazu werden die Normalen in der Textur zu den Texturkoordinaten dazuaddiert. Die Komplexität der Blitze wird dadurch deutlich erhöht.
Blitze entstehen wie die anderen Wettereffekt bei starker Bewölkung. Für die Animation der Blitze wird nach einer zufällig festgelegten Zeit ein Blitz erzeugt und für eine kurze Zeit angezeigt. Während der Blitz angezeigt wird, ändert sich seine Form ein paar mal, damit er nicht zu statisch wirkt. Um die Wucht des Blitzeinschlags besser zu visualisieren, wird darauf verzichtet den Weg des Blitzes vom Himmel zur Erde (bzw. umgekehrt) zu animieren. Das sofortige Auftreffen sorgt für den nötigen optischen Knall [12].
Durch die Blitze wird die Szene zusätzlich beleuchtet. Für die Landschaft wird für die Beleuchtung die Distanz zur Mitte des Blitzes berechnet. Außerdem wird das Skalarprodukt zwischen XZ-Komponente der Normale und der Richtung zum Blitz berechnet, also quasi zweidimensionales Phong Shading. Dadurch kommt die Oberflächenstruktur des Geländes bei Blitzen sehr gut zur Geltung. Die Beleuchtung des Himmels funktioniert auf die gleiche Weise. Für die Schnee-Partikel wird dagegen nur die Entfernung zum Blitz als Faktor für die Beleuchtung benutzt, Details wären durch die Fallgeschwindigkeit des Schnees nicht sichtbar. Für die Gewässer wird der Blitz in die Reflexions-Textur gezeichnet.

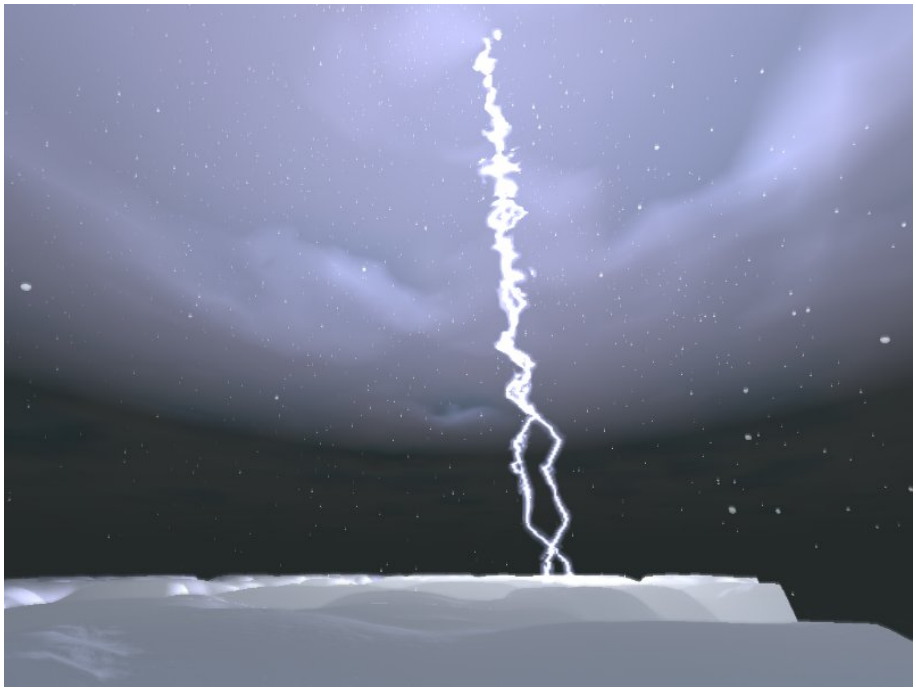

Abbildung 8: Ein Blitzeinschlag

10 Implementierung

Die Techniken wurden in einem Demonstrationsprogramm mit C++ und OpenGL implementiert. Shader wurden in der OpenGL Shading Language geschrieben.
Für das Rauschen werden zwei Generatoren benutzt, ein selbstentwickelter für die Texturen und libNoise[13] für das Gelände. Die Bibliothek libNoise ist wesentlicher komplexer als die selbstentwickelte Lösung und unterstützt mehr Rauschtypen.
Die Height Map für das Gelände hat eine Auflösung von 256*256 Pixeln, somit besteht es aus etwas mehr als 130000 Polygonen. Um den Zeichenvorgang zu beschleunigen wird eine OpenGL Display-Liste benutzt. Der Kugel die als Geometrie für den Himmel dient hat 1922 Polygone. Die Ebene der Gewässer wird als einfaches Quadrat gezeichnet. Blitze haben 50 Kontrollpunkte. Die Schnee-Partikel werden als Quadrate mit jeweils 2 Polygonen gezeichnet, das Partikel-System besteht aus 5000 Partikeln, also werden für den Schneefall nochmal 10000 Partikel gezeichnet. Für die verschiedenen Effekte müssen die Geometrien mehrmals gezeichnet werden. Das Gelände muss z.B. für die Reflexions- und Refraktions-Textur gezeichnet werden, für das Schneeglitzern und auch für die Blitze. Insgesamt werden für jedes Bild mehr als eine halbe Millionen Polygone gezeichnet.
Auf aktueller Hardware läuft das Demonstrationsprogramm durchschnittlich mit mehr als 75 Bildern pro Sekunde, hier wird man von der Vertikalen Synchronisation gebremst. Auf einer etwa 4-5 Jahre alten Hardware läuft das Demonstrationsprogramm immernoch mit etwa 10-20 Bildern pro Sekunde.

11 Fazit

Die vorgestellten Techniken eignen sich alle zur Visualisierung von Eis- und Schneelandschaften bei interaktiven Frameraten. Sie wirken für die meisten Anwendungen ausreichend plausibel und lassen durch die sparsame Verwendung von Rechenleistung genügend Raum für komplexere Szenen, Effekte oder sonstige Berechnungen.

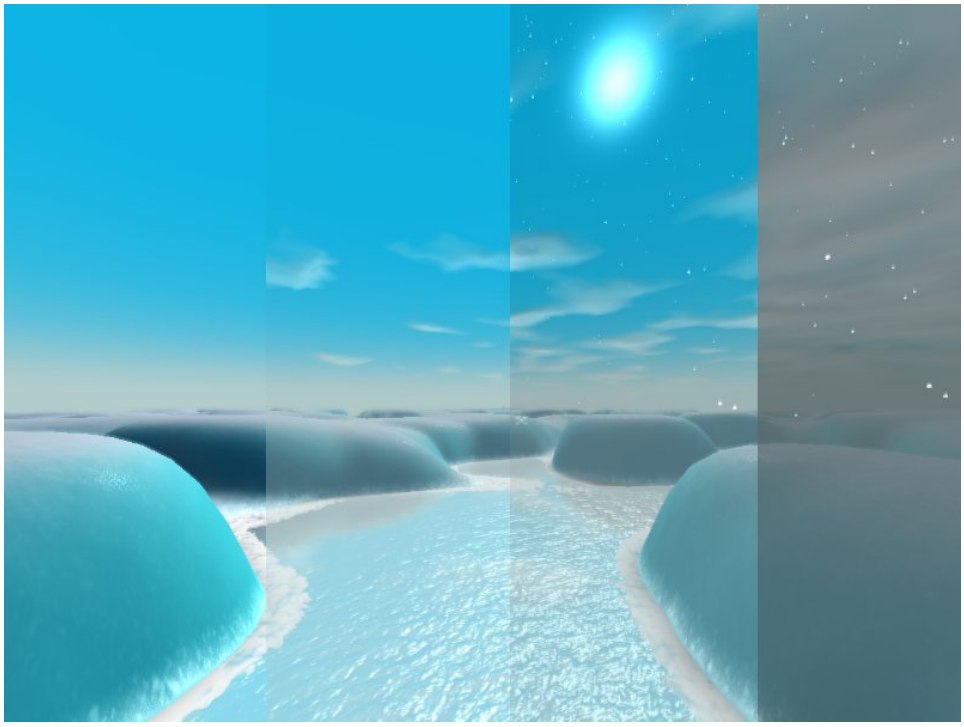

Abbildung 9: Vergleich zwischen unterschiedlich starker Bewölkung

12 Ergebnis-Bilder

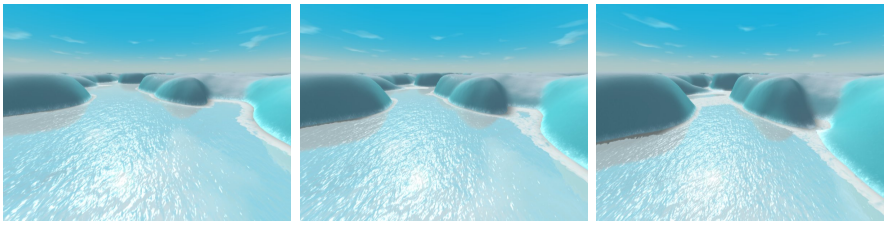

Abbildung 10: Eisschollen bei verschiedenen Gewässerhöhen

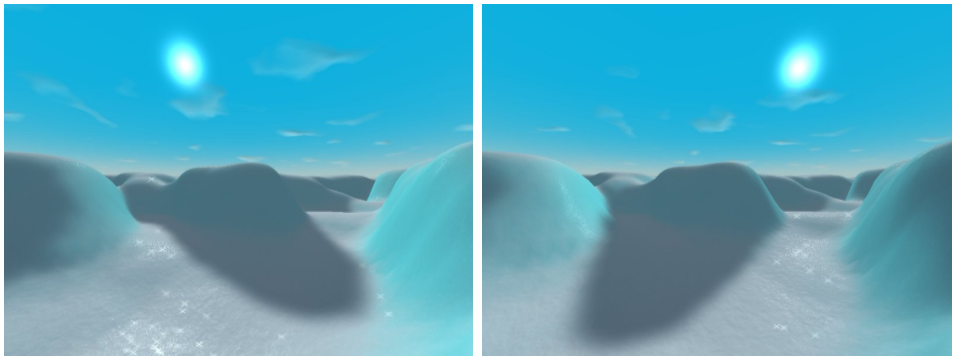

Abbildung 11: Schatten

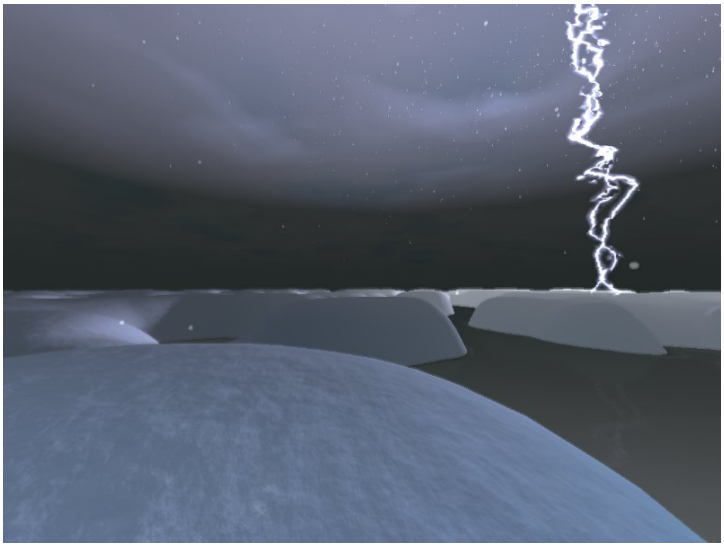

Abbildung 12: Ein Blitz aus der Entfernung

13 Literaturverzeichnis

1 Virtual Terrain Project : „http://www.vterrain.org“

2 Hugues Hoppe: „Geometry Clipmaps: Terrain Rendering using Nested Regular Grids“, Online erhältlich über [1]

3 Duchaineau et al.: „ROAMing Terrain: Real-time Optimally Adapting Meshes“, Online erhältlich über [1]

4 Hugo Elias: „Cloud Cover“, Online erhältlich auf: http://freespace.virgin.net/hugo.elias/models/m_clouds.htm

5 Marco Spörl: „A Practial Analytic Model for Daylight with Shaders“, ShaderX 3, edited by W.Engel, Charles River Media, 2005

6 Preetham et al.: „A Practical Analytic Model for Daylight“, Computer Graphics Annual Conference Series, 1999, Seite 91-100

7 Michael Horsch: „OpenGL Water Tutorial“, erhältlich Online auf: www.BonzaiSoftware.com/water_tut.html

8 Stefano Lanza: „Animation and Display of Water“,ShaderX 3, edited by W.Engel, Charles River Media, 2005

9 Tomoyuki Nishita et. al.: „Efficient Rendering of Lightning Taking into Account Scattering Effects due to Cloud and Atmospheric Particles“, Online erhältlich auf: http://nis-lab.is.s.u-tokyo.ac.jp/nis/abs_eg.html

10 Planetside Software: TerraGen v0.9, online erhältlich auf: http://www.planetside.co.uk/terragen/

11 Natalya Tatarchuck : „Rendering Multiple Layers of Rain with a Post-processing Composite Effect“, ShaderX 5, edited by Wofgang Engel. Charles River Media, 2006: pp. 303-313. Teilweise online erhältlich auf http://ati.amd.com/

12 Richard Williams: „The Animators Survival Kit - A Manual of Methods, Principles and Formulas for Classical, Computer, Games, Stop

Motion and Internet Animators by the Man who framed 'Roger Rabbit'“, Faber and Faber, 2001

13 libNoise: „http://libnoise.sourceforge.net/“