Diplomarbeit

Fachgebiet der Diplomarbeit:
Graphische Datenverarbeitung

Thema der Diplomarbeit:

Entwicklung einer interaktiven 3D-Echtzeitapplikation zur Simulation von Wetterflügen

Unternehmen, in dem die Diplomarbeit durchgeführt wurde:
weltenbauer. in Wiesbaden

Diplomand:	René Nold
Referentin:	Prof. Dr.-Ing. Dipl.-Math. Monika Lutz
Korreferentin:	Dipl.-Math. (FH) Sonja Emmel
Betreuer bei weltenbauer:	Robert Mayer

Sommersemester 2005

Fachhochschule Gießen-Friedberg
Bereich Friedberg

Fachbereiche: Informationstechnik-Elektrotechnik-Mechatronik
Mathematik-Naturwissenschaften-Datenverarbeitung
Mathematik-Naturwissenschaften-Informatik

Studiengang Medieninformatik

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbstständig und nur mit den angegebenen Hilfsmitteln und Literaturstellen verfasst zu haben.

Frankfurt/Main, den 04.07.2005

René Nold

Danksagung

Hiermit möchte ich mich bei meiner Professorin Prof. Dr.-Ing. Dipl.-Math. Monika Lutz für die hervorragende Betreuung während der Diplomarbeit bedanken.
Meiner Laboringenieurin Dipl.-Math. (FH) Sonja Emmel möchte ich meinen besonderen Dank aussprechen, weil Sie mich in jeder Phase der Arbeit sachkundig und richtungsweisend begleitete, mich stets ermunterte und viel Geduld mit mir hatte.
Meinem Betreuer Robert Mayer danke ich für den Diplomvorschlag und die vielfältige Unterstützung bei der Anfertigung dieser Arbeit seitens der Firma weltenbauer.
Ich danke auch Klaus Hallasek, der mir eine wertvolle Hilfe bei der Erstellung der praktischen Arbeit dieser Diplomarbeit war.

Folgende Leute haben mich außerdem auf die eine oder andere Weise unterstützt: Jannis Singh (weltenbauer.), Martin Andel (weltenbauer.), Andreas Bekier (hr), Mahin Fischbach (hr), Rüdiger David (hr) sowie Alexander Horst (Lichtblick4D).

Abschließend danke ich meiner Familie und meiner Freundin Inga für so vieles.

Inhaltsverzeichnis

1 Einleitung

Beim Wetterbericht der öffentlich-rechtlichen Rundfunkanstalten ist der Wetterflug, eine kurze Animation, welche die Wetterlage einer festgelegten Region zu einem bestimmten Zeitpunkt verdeutlicht, seit 1995 ein fester Bestandteil des Programms. Dieser Wetterflug sowie alle übrigen Grafiken, die im Wetterbericht von ARD, ZDF und den Dritten Programmen zum Einsatz kommen, werden beim Hessischen Rundfunk (hr) in Frankfurt am Main produziert.

Unbestreitbar ist, dass der Informationsgehalt eines Wetterflugs im Vergleich zu einer Übersichtskarte oder einer Drei-Tages-Vorschau eher gering ist. Zumeist wird in einem Wetterflug eine kurze Strecke zu einem bestimmten Zeitpunkt überflogen. Gezeigt wird also nur das Wetter einer bestimmten Region, ohne zeitliche Veränderung. Das begrenzt die Nutznießer auf die in den überflogenen Gebieten lebenden Menschen, ein relativ geringer Anteil der Zuschauer. Und dennoch hält der hr an seinen Wetterflügen fest.

Worum es bei Wetterflügen im Wesentlichen geht, ist die Schaffung von Stimmungen und Eindrücken beim Zuschauer. Bei einer Übersichtskarte sieht man vorüber ziehende Wolkenbänder, Sturmtiefs oder Regengebiete aus einer sehr distanzierten, satellitenähnlichen Position. Es lässt sich zwar grob einschätzen, ob die Wolken im Tagesverlauf zuziehen oder ob es anfangen wird zu regnen, aber der Zuschauer befindet sich in „sicherer“ Entfernung zu dem Geschehen, so dass er sich nicht wirklich betroffen fühlt. Bei Wetterflügen hingegen bewegt sich die Kamera unterhalb der Wolkendecke. Das Wetter läuft über dem Zuschauer ab und unter sich sieht er die jeweilige Region, auf die das lokale Wetter Auswirkungen haben wird. Durch Einsatz von Lichtfarbe, Wolkenschatten, die Verwendung von Nebel und weiteren visuellen Hilfsmitteln soll eine Wetterstimmung an den Zuschauer vermittelt werden. Er befindet sich mitten im Geschehen und soll das Wetter erleben können. Zudem identifiziert sich der Zuschauer unter Umständen mit dem in dem Wetterflug gezeigten Gebiet, sei es, weil beispielsweise seine Arbeitsstelle in einer Schlechtwetterzone liegt oder der Nachbarort von einem Gewitter heimgesucht wird. Durch die tägliche Änderung der Flugroute sowie den regionalen Bezug des hr auf das Bundesland Hessen ist die Wahrscheinlichkeit für regelmäßige Zuschauer recht hoch innerhalb weniger Sendungen auf einen Flug mit persönlichem Betreff zu stoßen. So wird ein lokaler Bezug des Zuschauers zu dem gezeigten Wetter erreicht.

Ein weiterer Grund für Wetterflüge ist die technische Vorreiterrolle, die der hr auf diesem Gebiet innehat. Diese Form der täglichen Wetterflüge wird in der aktuellen Qualität bei keinem anderen deutschen Fernsehsender produziert. Es gibt zwar zu besonderen Ereignissen (z.B. Formel-1-Rennen) einen in der Machart ähnlichen Flug bei dem

privaten Sender RTL, jedoch ist die grafische und meteorologische Qualität sowie die tägliche Aktualität der hr-Wetterflüge bisher unerreicht.

Für die Produktion eines solchen Wetterflugs werden die aktuellen Wetterdaten, die direkt vom Deutschen Wetter Dienst (DWD) in Offenbach geliefert werden, mit Hilfe eines TriVis-Systems ausgewertet. Das TriVis-System ist eine auf UNIX basierende Software, welche die im ASCII-Format angelieferten Daten zu Wolkenentwicklung, Niederschlag, Temperatur, Wind, Luftfeuchtigkeit und allen anderen Wetterphänomenen graphisch in 2D und 3D darstellen kann. TriVis wurde im Fraunhofer Institut für Graphische Datenverarbeitung in Darmstadt in Zusammenarbeit mit dem Deutschen Wetterdienst entwickelt [1]. Die auf diesem System berechneten Wolkenbilder und Niederschlagskarten werden dann von Grafikern in 3ds max und Adobe After Effects aufwändig visualisiert. Der gesamte Arbeitsablauf, von der Auswertung der Daten bis hin zum fertig berechneten Bild, ist sehr zeitintensiv und verbietet kurzfristige Änderungen. Dieser Mangel an Flexibilität sowie der ungeheure Aufwand von Ressourcen für jede Wetterproduktion sind sehr ineffektiv und beeinträchtigen die Produktivität der Grafikabteilung des hr.

Die Nachteile, die sich aus dem bisherigen Arbeitsablaufs ergeben, führten zu der Idee, die Produktion des Wetterflugs vom vorberechneten Bild bzw. Film in den Echtzeitbereich zu verschieben. Mit der in den letzten Jahren sich sehr schnell entwickelnden Grafikhardware, getrieben durch den boomenden Computerspielemarkt, ist es heute möglich auch sehr aufwändige Abläufe und Modelle mit immenser Polygonzahl in hoher grafischer Qualität in Echtzeit darzustellen.

Benötigt wird also eine Echtzeit-Applikation, die ähnlich einem Flugsimulator eine bestimmte, vorher festgelegte Strecke über das 3D-Modell einer real existierenden Landschaft entlang fliegt. Dabei werden die auftretenden Wetterphänomene an den vorhergesagten Positionen in ihrer korrekten Ausprägung dargestellt.

Die Vorteile eines Echtzeitsystems sind Folgende: Der Flug kann direkt in Ausgabequalität konfiguriert und begutachtet werden. Langwierige Berechnungen einzelner Bilder zur Kontrolle von getätigten Einstellungen entfallen völlig. Dadurch erhöht sich die Produktionsgeschwindigkeit bzw. die Anzahl der möglichen Wetterproduktionen pro Tag. Auch können kurzfristige Änderungen schnell und einfach eingefügt werden. Da die Echtzeit-Applikation bereits in Ausgabequalität arbeitet, die fertigen Bilder also direkt auf der Grafikkarte generiert werden, müssen diese lediglich auf einen Datenträger gespeichert werden, um das finale Material für die Animation zu erhalten. Im Gegensatz dazu ist es bei dem bisherigen Arbeitsablauf nötig, die 3ds-max-Szene nach der Einrichtung zum Berechnen an die Renderfarm zu schicken. Dieser Verbund von 40 überaus leistungsfähigen Rechnern benötigt je nach Komplexität der Szene zwischen 20 und 60 Minuten zur Berechnung des finalen Bildmaterials. Zudem belastet

der stetige Netzwerkverkehr, hervorgerufen durch die Verteilung der Renderjobs, das Hausnetz und beeinträchtigt die Arbeit anderer Grafiker. Der Umstieg auf ein Echtzeitsystem setzt also Hardwareressourcen frei, die dann an anderer Stelle genutzt werden können und reduziert die Last im Netzwerk.

Die Aufgabe dieser Diplomarbeit ist es, eine Echtzeitapplikation zu entwickeln, die den bisherigen Arbeitsablauf zur Erstellung eines Wetterflugs in 3ds max von der Konfiguration der entsprechenden 3D-Szene bis hin zum Erstellen des finalen Bildmaterials ersetzt. Auf Grund der Echtzeitdarstellung in Ausgabequalität wird die Applikation einfacher und schneller zu bedienen sein und deshalb weniger Zeit pro Produktion in Anspruch nehmen.

2 Entwicklung eines Prototyps

In einer Vorlaufphase wurde ein Prototyp entwickelt, um die Möglichkeiten und Vorteile einer 3D-Echtzeit-Applikation aufzuzeigen und die technische Machbarkeit zu testen.

Zunächst wurde für den Prototyp ein Pflichtenheft definiert, das die Anforderungen des hr und auch Vorstellungen und Ideen der Firma Weltenbauer beinhaltete. Das Pflichtenheft wurde relativ knapp und simpel als Fließtext erstellt. Es umfasst die Verwendung der vom hr gestellten Basisdaten (Geometrien, Texturen, Städtedaten) sowie die Maßgaben an Kameraeinstellungen und Modifikationsmöglichkeiten. Von Seiten der Firma Weltenbauer kamen noch weitere Ideen, wie beispielsweise der Einsatz von Geräuschen zur Verdeutlichung der Wetterphänomene sowie die Echtzeitausspielung der Animation mit Hilfe einer Video-Capture-Karte[1] (Blackmagic Decklink Pro), hinzu. Nach erfolgter Definition des Pflichtenhefts wurde mit der eigentlichen technischen Umsetzung des Prototyps begonnen. Umfang und Zweck der implementierten Merkmale werden im Folgenden beschrieben.

Der Prototyp beschränkt sich auf eine einzelne 3D-Szene, in der die Landschaftsgeometrie von Hessen in einer echtzeitoptimierten Form zu sehen ist. Die entsprechenden Texturen wurden in einer leicht geringeren Auflösung auf die Geometrie aufgebracht. Sowohl die Landschaftsgeometrien als auch die Texturen wurden hierfür vom hr zur Verfügung gestellt. Die gesamte Szene wird von einer Wolkenkuppel überspannt, auf die ein Wolkenfilm projiziert wird. Für den Schatten, den die Wolken werfen, wird derselbe Film, allerdings in der Helligkeit invertiert, auf die gesamte Oberfläche der Landschaft projiziert. Ein Dutzend hessische Städte werden durch entsprechende Namens-

[1] Video-Capture-Karte: Eine Video-Capture-Karte ist eine zusätzliche Grafikkarte in einem Rechner, die ein Computergrafiksignal in ein fernsehtaugliches Signal (bspw. PAL oder NTSC) umwandelt.

schilder und eine sich über die Zeit verändernde Temperatur symbolisiert. Große Städte erhalten ein zusätzliches Symbol in Form von Wahrzeichen wie beispielsweise dem Messeturm in Frankfurt oder dem Herkules-Denkmal in Kassel. Diese Wahrzeichen werfen je nach Uhrzeit und Sonnenstand einen entsprechenden Schatten. Für die Darstellung von Regen ist ein einfaches Partikelsystem integriert, das jedoch nicht positioniert werden kann, sondern lediglich direkt vor der Kamera den Regen simuliert. Wird der Regen „eingeschaltet“, werden zusätzlich Gewittergeräusche abgespielt. Eine animierte Textur stellt den Wind in Form vorüber ziehender Pfeilen dar.

Der Prototyp verfügt über insgesamt vier unterschiedliche Kameramodi, welche die Vielseitigkeit der Applikation zeigen sollen: Eine Orbit-Kamera, die mit der Maus steuerbar um einen festen Punkt rotiert und deren Zoom veränderbar ist, eine Totale, die die gesamte Szene mit den vorüber ziehenden Wolken von oben darstellt, eine Flug-Kamera, bei der die Navigation in der Szene wie bei einem Flugzeug mit einem Joystick möglich ist und zu guter Letzt eine animierte Kamera, die einen festgelegten Pfad entlang fliegt. Eine Technik, die bei der gegenwärtigen Erstellung der Wetterflüge mit 3ds max zum Einsatz kommt, ist die Verwendung von Level-of-Detail. Diese Technik, bei der zwei unterschiedlich hoch aufgelöste Versionen eines Objektes abhängig von der Entfernung zur Kamera angezeigt werden, sollte ebenfalls zum Einsatz kommen. Da die Realisierung dieses Merkmals für die gesamte Geometrie des Landes Hessen zu aufwändig geworden wäre, wurde eine zweite, kleine Applikation erstellt, die nur die Verwendung von Level-of-Detail für ein kleines Gebiet sowie das Überblenden von Texturen, eingesetzt für einen Tag-Nacht-Wechsel, demonstrierte. Damit sind die im Pflichtenheft geforderten folgenden Hauptmerkmale für den Wetterflug-Prototyp implementiert worden:

- Implementierung von animierten und frei einstellbaren Kameras
- Darstellung der Landschaftsgeometrie von Hessen
- Darstellung der zugehörigen Texturen
- Darstellung der Städtenamen und Wahrzeichen
- Darstellung von Wolken und Wolkenschatten auf dem Boden
- Darstellung von Regen
- Implementierung von Level-of-Detail
- Darstellung des Tag-Nacht-Wechsels

Der Prototyp diente in erster Linie der Darstellung der im Echtzeitbereich machbaren grafischen Qualität. Allerdings konnten auch wertvolle Erfahrungen in Bezug auf effektivere Logikerstellung, die Verwendung von im Echtzeitbereich unüblichen Textur- und Geometriegrößen sowie den Einsatz von Video-Capture-Karten gemacht werden.

Nach erfolgreicher Produktion und Abnahme des Prototyps fiel die Entscheidung, die Echtzeitapplikation zur Simulation von Wetterflügen zu entwickeln. Bereits zur Erstellung des Prototyps wurde das Projekt mit dem Namen „echtwetter“ tituliert, der im Folgenden auch Verwendung findet. Im nächsten Kapitel werden die Möglichkeiten der für den Prototyp und auch die finale Applikation eingesetzten 3D-Engine Quest3D näher erläutert.

3 Einsatz der 3D-Engine Quest3D

3.1 Einleitung

Da der Hauptteil dieser Arbeit auf der Entwicklung mit der Software Quest3D basiert, werden die Möglichkeiten sowie die Einsatzgebiete näher vorgestellt. Ziel ist dabei, ein gewisses Grundverständnis für die Arbeitsweise von Quest3D sowie einige Grundfunktionen der Echtzeit-3D-Entwicklungsumgebung zu vermitteln um das Verständnis der späteren Abschnitte dieser Arbeit zu erleichtern.

3.2 Merkmale von Quest3D

Quest3D ist eine auf der Programmiersprache C++ basierende 3D-Engine der Firma Act-3D aus Leiden in den Niederlanden. Als 3D-Engine bezeichnet man ein Programm, das für die Darstellung von zumeist realitätsnaher 3D-Computergrafik zuständig ist. „Sie bietet einem Programmierer eine große Palette von grafischen Funktionen und Effekten (geometrische Objektbeschreibung, Oberflächentexturen, Licht und Schatten, Transparenz, Spiegelungen usw.), so dass er für seine spezielle Anwendung diese nicht stets neu programmieren muss“ [2].

Die für die Erstellung des Prototyps sowie der eigentlichen Applikation zur Simulation von Wetterflügen verwendete Version 2.5 von Quest3D verwendet DirectX 8.1. Bei DirectX handelt es sich um einen von Microsoft entwickelten Befehlssatz zur Verwendung in Echtzeitanwendungen, der vor allem in Spielen Einsatz findet. Mit diesem können die Grafikkarte, aber auch andere für Spiele und Echtzeitanwendungen notwendige Hardware und Peripheriegeräte, wie z.B. Soundkarte, Joystick und Netzwerkkarten angesteuert werden [3].

Eine Besonderheit von Quest3D ist die grafische Programmieroberfläche, das heißt, zur Erstellung einer Echtzeitapplikation sind keinerlei Programmierkenntnisse in C++

nötig. Quest3D arbeitet stattdessen mit Logikbausteinen, so genannten Channels, die mittels „Drag-and-Drop“ aus einer Bibliothek verfügbarer Channels auf die Arbeitsoberfläche gezogen werden. Diese werden dann mit der Maus mittels „Point and Click“ miteinander zu einer Logikstruktur verknüpft (Abbildung 1).

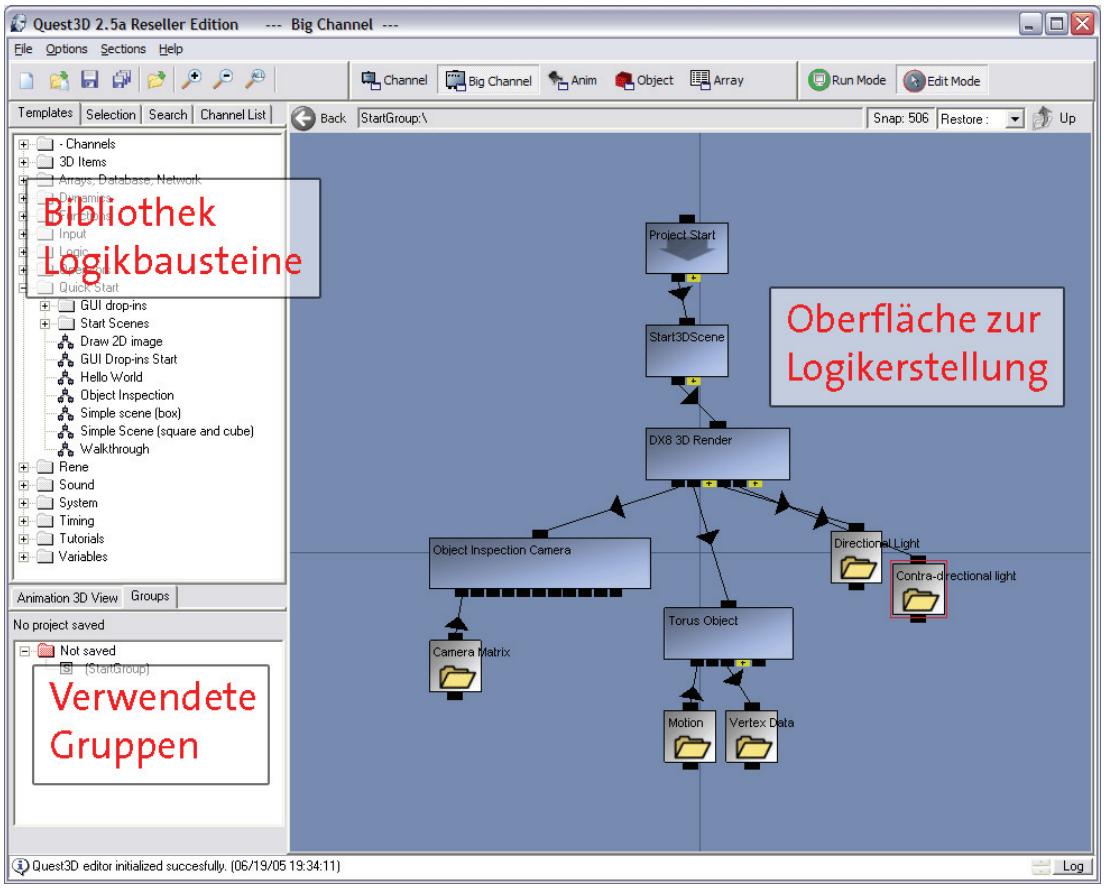


Abbildung 1: Quest3D Arbeitsoberfläche

Die einzelnen Channels sind mit C++-Klassen gleichzusetzen, die je nach Funktionalität unterschiedliche Parameter benötigen und Werte an ihren übergeordneten Channel zurückgeben können. Jeder Logikblock besitzt einen Aufrufer, in Quest3D als Caller bezeichnet, der an der Oberseite eines jeden Channels sitzt. Je nach Art besitzt ein Channel auch die Möglichkeit andere Channels an sich anzuschließen. Diese Verbindungen werden an der Unterseite eines Channels angebracht und Children (Kindknoten) genannt (Abbildung 2).

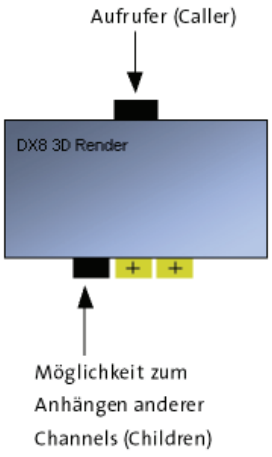


Abbildung 2: Channel

Insgesamt gibt es in Quest3D ca. 200 verschiedene Channels, von der einfachen If-Abfrage bis hin zu komplexeren Channels wie dem Partikel-Emitter. Aus diesen Channels können, je nach Anforderung, mehr oder minder weit verzweigte Logiken erwachsen. Die Logiken können sich auf das Anzeigen einiger simpler Geometrien beschränken oder aber ein komplettes Walkthrough[2] mit Interaktionen des Benutzers darstellen. Die in der Echtzeitapplikation verwendeten Channels werden jeweils bei ihrem ersten Auftreten genauer erklärt.

Der Ablauf einer fertigen Logikstruktur erfolgt nach der Top-Down-Methode, also von oben nach unten. An beliebiger Stelle in einer Logikstruktur wird ein Channel als Startpunkt für die Applikation festgelegt. Ab hier werden die unten angehängten Children von links nach rechts und von oben nach unten abgearbeitet. Ein kompletter Durchlauf der Logikstruktur von der Wurzel an der Spitze bis hin zum letzten unten angehängten Child ergibt ein einzelnes Bild, Frame genannt; vorausgesetzt natürlich, die entsprechenden Renderlogiken sind in der Struktur umgesetzt. Abhängig von der verwendeten Hardware, insbesondere spielt hier die Grafikkarte eine große Rolle, wird der gesamte Logikbaum so oft durchlaufen wie eben möglich. Die dabei generierten Frames ergeben dann die Echtzeit-Applikation, die nach Möglichkeit mit mindestens 25 Bildern pro Sekunde abläuft um einen flüssigen Ablauf der dreidimensional dargestellten Vorgänge anzuzeigen.

3.3 Anwendungsgebiete für Quest3D

Die Anwendungsgebiete von Quest3D sind breit gefächert. Ein stark vertretener Anwendungsbereich ist die Architektur. Hier wird Quest3D zur Vorvisualisierung von Gebäuden und Wohnräumen verwendet. Architekten, zumeist wenig erfahren in der Erstellung von Echtzeitanwendungen, können in kurzer Zeit mit wenigen einfachen Vorlagen unter Verwendung der in einem CAD-System gefertigten Bauzeichnungen und einer 3D-Software einfache Walkthroughs erstellen oder Szenen, in denen interaktiv zwischen mehreren Kameras hin- und hergeschaltet werden kann. Quest3D lässt sich aber auch zur Darstellung von Baufortschritten bei Großprojekten, als Hilfsmittel im Bereich des Facility Managements oder zur Ablaufsimulation bestimmter Vorgänge in einem Gebäude verwenden (Abbildung 3).

[2] Walkthrough: Eine aus der Ego-Perspektive gesehene 3D-Szene, in der man sich beispielsweise mit Maus und/oder Tastatur frei bewegen kann.

Abbildung 3: Einsatz von Quest3D im Bereich Architekturvisualisierung [4]

Ein weiteres Einsatzgebiet für Quest3D bieten die Bereiche Produktvisualisierung und Produktkonfigurator. Neue Produkte und Designentwürfe können nach der Erstellung in einer 3D-Software mit wenigen Klicks in einer Echtzeit-Szene dargestellt werden. Auf diese Weise kann das Produkt vom Anwender leicht von allen Seiten betrachtet werden. Mit ein wenig zusätzlicher Logik lässt sich die Zerlegung in Baugruppen sehr einfach animieren und auf Knopfdruck ein 3D-Objekt in seine Bestandteile zerlegen und wieder zusammensetzen. Details eines Objekts, wie Farben oder Materialien, lassen sich ebenfalls über Logiken austauschbar machen. Auf diese Weise ist es ein Leichtes Konfiguratoren für beispielsweise Autos oder Möbel zu bauen (Abbildung 4).

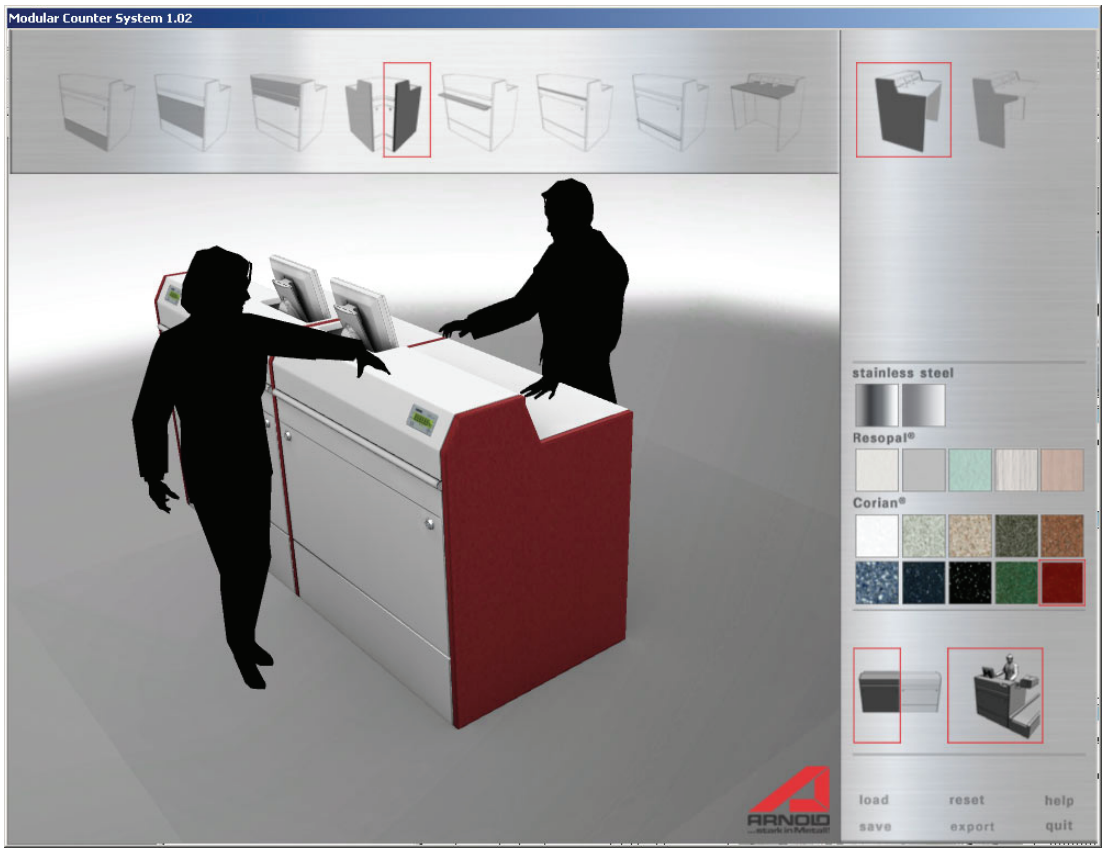


Abbildung 4: Einsatz von Quest3D im Bereich Produktkonfigurator [5]

Ein relativ neuer Einsatzbereich ist die Verwendung von in Quest3D erstellten Echtzeitanwendungen für computerunterstützte Trainingsprogramme (e-Learning). In diesen werden Übungssituationen abgebildet, auf die der lernende Anwender angemessen reagieren muss. Bestes Beispiel sind Fahr- und Flugsimulatoren, die bereits vielerorts Einsatz finden. Hier ist besonders das erst kürzlich in Norwegen eröffnete „Offshore Simulation Center“ zu erwähnen, bei dem Quest3D zur Visualisierung der unterschiedlichen Trainingssituationen wie beispielsweise dem Bekämpfen von Bränden auf Ölbohrplattformen verwendet wurde [6].

Selbstverständlich lässt sich Quest3D, wie jede andere 3D-Engine auch, zur Erstellung von Spielen verwenden. Ob man nun Abenteuer-Spiele, First-Person-Shooter, Fahr- oder Flugsimulatoren betrachtet, alles was sich für einen sinnvollen Zweck verwenden lässt, kann auch in einem Spiel Verwendung finden und umgekehrt.

Neben diesen erwähnten Bereichen, in denen sich Quest3D bereits etabliert hat, ist die Firma Weltenbauer dabei sich ein neues Gebiet der Echtzeitvisualisierung zu erschließen. Dabei geht es vor allem um Echtzeit-Computergrafiken im Bereich Fernsehen. Auch hier bietet Quest3D unzählige Möglichkeiten zum sinnvollen Einsatz; sei es als Quizwand in einer Live-Quizshow, zur Live-Darstellung von Börsen- oder Wahldaten oder zur Ablaufsimulation von Katastrophen wie Flugzeugabstürzen oder Zugunglü-

cken. Vor allem der Echtzeitaspekt spielt hier eine große Rolle. Bisher war es immer nötig, entsprechende Animationen vorher zu erstellen, einen Film zu berechnen und diesen dann auf Knopfdruck abzuspielen. Mit der Interaktivität aus Quest3D ist es möglich solche 3D-Szenen aufzusetzen und dann über entsprechende Logiken live zu ändern, sei es um Wahldaten nach einer weiteren Prognose während einer Sendung zu aktualisieren oder die von einem Kandidaten zufällig ausgewählte Frage auf Knopfdruck des Moderators abzuspielen.

Die 3D-Engine Quest3D findet ob ihrer schnell zu lernenden Programmiermethode immer mehr Anwender aus den unterschiedlichsten Bereichen, was die Bandbreite der Anwendungsgebiete mittlerweile weit über den Bereich der Spieleprogrammierung hin ausgedehnt hat. In den nun folgenden Kapiteln wird der Entwicklungsprozess der „echtwetter“-Applikation unter Verwendung von Quest3D beschrieben.

4 Anforderungsanalyse, Konzipierung und Anpassungsarbeiten für einen Echtzeitwetterflug

4.1 Anforderungsanalyse

4.1.1 Einleitung

Auf Basis der mit dem Prototyp erlangten Erfahrungen und den primären Wünschen der Verantwortlichen beim hr wurde zuerst angefangen, das Projekt so genau wie möglich zu definieren. Die Wünsche des Kunden richteten sich dabei im groben nach den bereits bestehenden Möglichkeiten in dem bis dato aktuellen Arbeitsablauf mit 3ds max. Alles, was in Bezug auf Wetter in dieser 3D-Software möglich ist, sollte so oder ähnlich auch in der Echtzeitapplikation „echtwetter“ machbar sein. Deshalb werden im folgenden Abschnitt der bisherige Arbeitsablauf sowie die vorhandenen Ausgangsdaten genauer analysiert.

4.1.2 Analyse der vorhandenen Arbeitsabläufe, Basisdaten und Skripte

Es war notwendig sich mit den Arbeitsabläufen der Wetterredaktion und der angeschlossenen Grafikabteilung des Fernsehens des Hessischen Rundfunks vertraut zu machen, um die Anforderungen an die Echtzeitapplikation richtig einschätzen zu können.

Die eigentliche Arbeit zur Erstellung eines Wetterflugs beginnt mit der Festlegung einer Flugroute durch einen Redakteur der Wetterredaktion. Zumeist kommen hier bereits erstellte Flüge zum Einsatz, da sich im Laufe der Zeit bestimmte Routen be-

währt haben und die tägliche Neuerstellung eines Flugs zu viel Aufwand bedeutet. Im Anschluss an die Auswahl der Flugstrecke wird eine Beschreibung der zu fliegenden Route inklusive einer Liste aller Städte, die auf der Strecke zu sehen sein sollen, an die Grafikabteilung durchgegeben. Während die Grafikabteilung mit der Erstellung eines neuen Flugs oder der Änderung eines bereits vorhanden Flugs beginnt, wird die Liste mit den Städtenamen an die Meteorologen des Deutschen Wetterdienstes in Offenbach weitergeleitet, die ein vorbereitetes Formular mit den entsprechenden zum Flugzeitpunkt prognostizierten Temperaturen ausfüllen und an den hr zurücksenden. Zeitgleich werden die aktuellen Wetterdaten für Luftdruck, Niederschlag, Wind und Wolken per Datenleitung direkt vom Deutschen Wetterdienst in das in der Grafikabteilung vorhandene TriVis-Softwaresystem geladen. Das TriVis-System berechnet an Hand der übertragenen ASCII-Daten entsprechende Wolkenbilder und Wolkenfilme sowie Niederschlagskarten (Abbildung 5). Diese finden in den meisten Wettersendungen der öffentlich-rechtlichen Sendeanstalten zumeist in der Übersichtskarte für Deutschland Verwendung, werden aber auch für den Wetterflug benötigt.

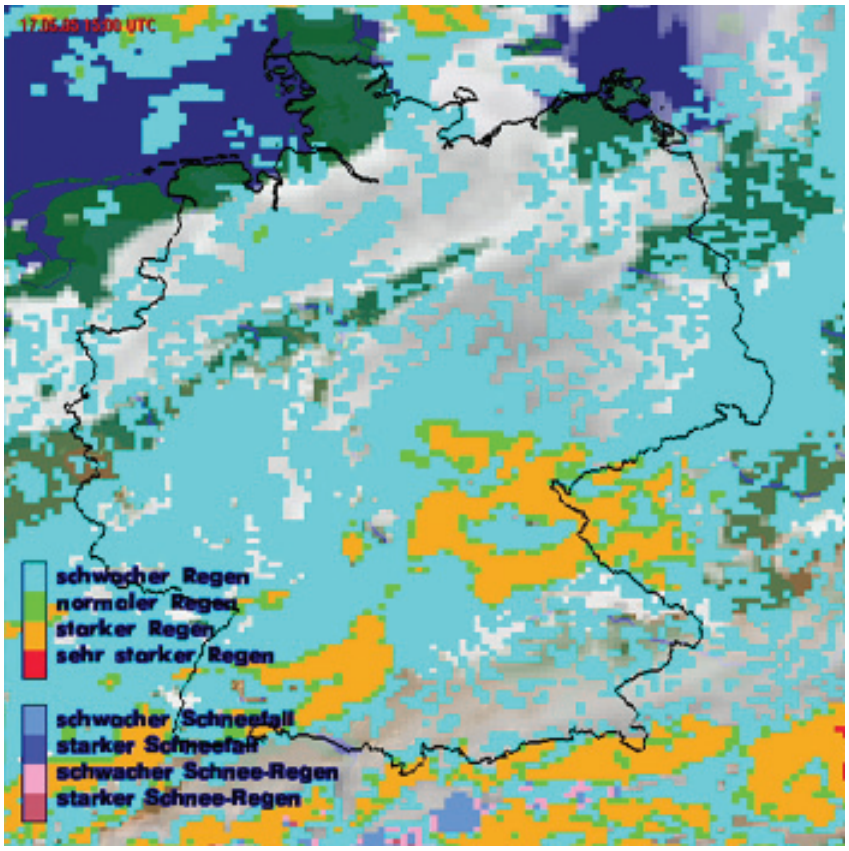


Abbildung 5: TriVis-Niederschlagskarte

Die 3ds max-Szene, die den eigentlichen 3D-Wetterflug abbildet, wird nach Beziehen aller Basisinformationen sowie der Berechnung der Wolkenbilder und Niederschlags-

karten innerhalb einer bis zwei Stunden von einem Grafiker an einer 3D-Workstation erstellt. Durch die Verwendung unterschiedlicher, speziell für den hr entwickelten MAX-Skripts[3] wird die schnellere Bearbeitung einer Szene gewährleistet. Es kommen vier unterschiedliche Skripte zum Einsatz, die zum Laden der Landschaftsgeometrie und der zugehörigen Texturen, der Auswahl der Städte, den Einstellungen zu Lichtstimmung und Atmosphäre sowie zur Einstellung des Niederschlags dienen und deren Verwendung im Folgenden beschrieben wird. Ziel ist es schließlich, die Funktionalität dieser Skripte auch in der Echtzeitanwendung anzubieten.

Das erste verwendete Skript, der „Snow and Season Selector", dient dem Laden der Landschaftsgeometrie sowie der zugehörigen Texturen. Die Landschaftsgeometrie liegt in einzelnen Kacheln im 3ds max Format vor, die jeweils einem realen Gebiet von 50 km^2 entsprechen und das gesamte Gebiet der Bundesrepublik sowie ein wenig der angrenzenden Nachbarländer abdecken. Der Ursprung der Daten ist ein auf Satellitendaten basierendes, extrem fein unterteiltes Gitternetz gewesen, das zur besseren Handhabung in der 3D-Software in kleinere Quadrate zerschnitten wurde. Des Weiteren wurden von diesen Quadraten unterschiedlich aufgelöste Versionen mit reduzierten Polygonanzahlen erstellt. In der Wetterflugszene werden für die weiter entfernten Gebiete, von denen man als Zuschauer noch gerade so eben die Umrisse erkennen kann, diese reduzierten Kacheln verwendet, um die Szene in 3ds max zügiger bearbeiten zu können und die nötige Zeit zur finalen Berechnung der Bilder soweit es geht zu reduzieren.

Das zu ladende Gebiet wird zunächst als Rechteck, durch die Angabe einer Startkachel oben links und einer Endkachel unten rechts, definiert. Der jeweilige Detailgrad der zu ladenden Kacheln wird in einem Optionsfeld angegeben. Zusätzlich wird noch bestimmt, welche der im TIF-Format vorliegenden Texturen geladen werden sollen. Ob und wenn ja wie hoch Schnee liegt, wird in einer Drop-Down-Liste angegeben. Entsprechende Masken werden dann durch das Skript automatisch auf den ausgewählten Texturensatz appliziert. Ergebnis des Skripts ist die geladene Landschaftsgeometrie sowie der korrekte Texturensatz in der gewünschten Auflösung (Abbildung 6).

[3] MAX-Skript: Eine Skriptsprache zur Automatisierung von Abläufen und der Anpassung von Befehlen und Benutzeroberfläche innerhalb von 3ds max; vergleichbar mit MEL in Alias Maya.

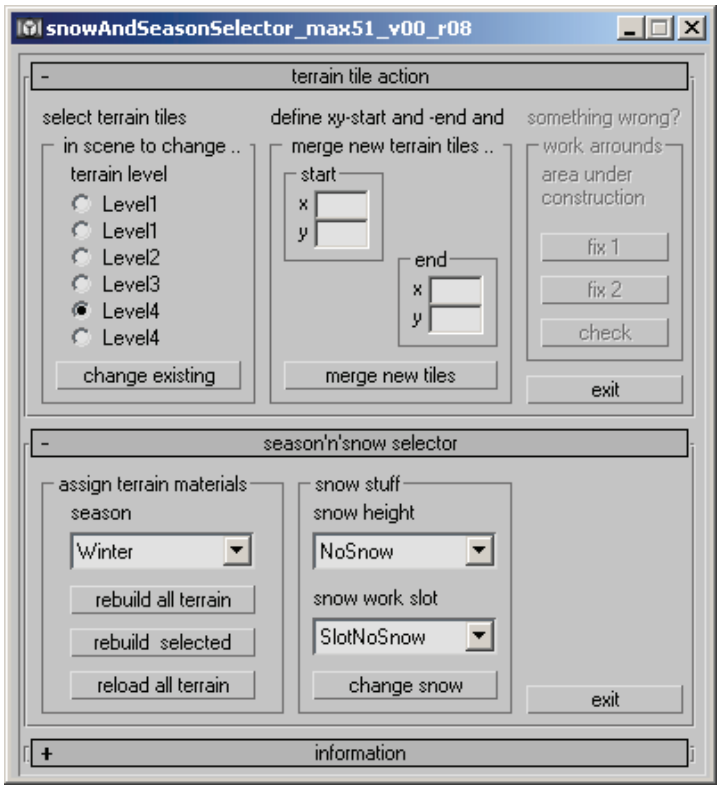


Abbildung 6: MAX-Skript „Snow and Season Selector“

Das nächste Skript befasst sich mit der Auswahl der auf dem Flug anzuzeigenden Städte. Dafür wird die für diesen Zweck vorhandene Access-Städtedatenbank gestartet, die insgesamt 14.235 deutsche Städte umfasst. In einem Auswahlformular innerhalb der Access-Datenbank wählt der Grafiker nun die auf dem Flug zu sehenden Städte aus und startet über einen Knopfdruck ein Makro, das die ausgewählten Städte mit Namen und Positionsangabe in eine Textdatei schreibt, die dann in 3ds max von dem Städteskript ausgelesen wird. Das Skript positioniert nun an Hand der Daten aus der Datenbank ein Städtenamensymbol bestehend aus dem Namen der Stadt sowie der angehängten Temperatur, einem leicht transparenten Hintergrundobjekt um die Lesbarkeit des Textes zu erhöhen und einem kleinen Prisma das auf die entsprechenden Position auf der Landschaft zeigt (Abbildung 7).

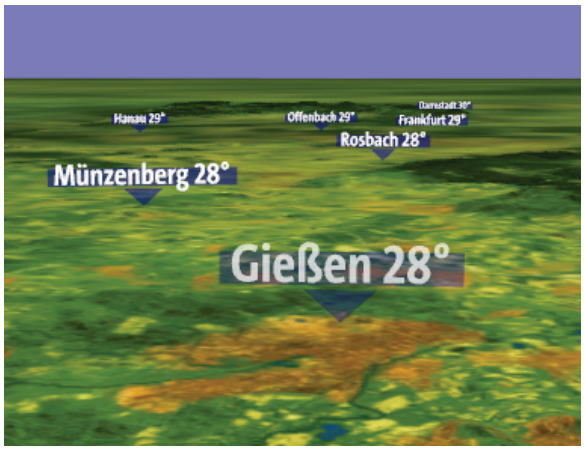


Abbildung 7: Städtenamensymbole

Als nächstes klickt der Grafiker jeweils eine Stadt an, das Prisma dient dabei als Anfasser, und kann dann über ein Eingabefeld im Städteskript die, von den Meteorologen durchgegebene, Temperatur eintragen. Damit sind die Einstellungen im Städteskript abgeschlossen und es geht an die Erstellung der Wetterphänomene (Abbildung 8).

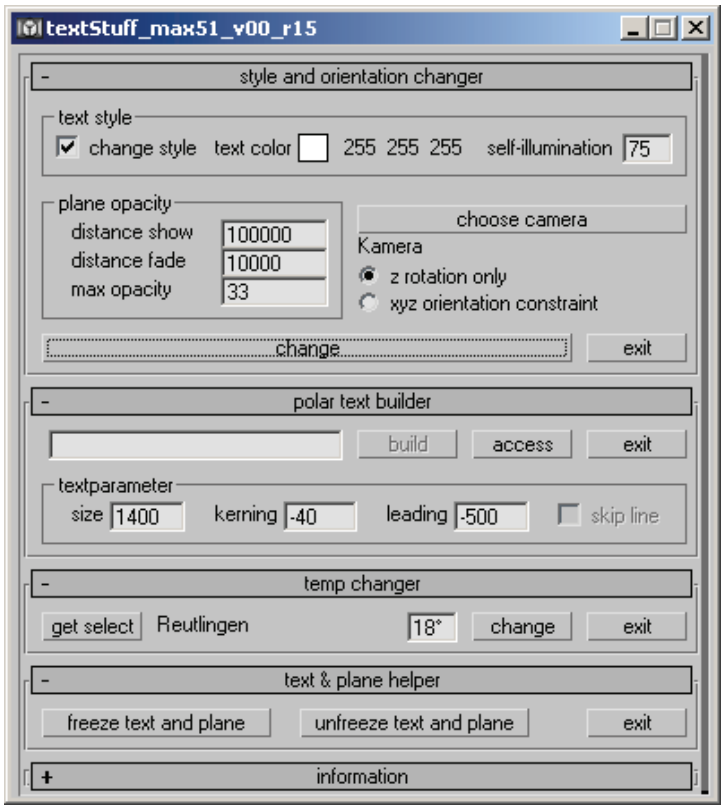


Abbildung 8: MAX-Skript „Städteeinstellungen“

Bei der Erstellung der Wetterphänomene kommt zunächst ein Skript zum Einsatz, das die aktuellen Wolkendaten auf zwei Rechteckflächen aufbringt und diese dann mittels Displacement-Mapping[4] verformt. Eine der beiden Rechteckflächen wird dabei nach oben verformt, die andere nach unten. Dadurch entsteht der Eindruck volumetrischer Wolken (Abbildung 9).

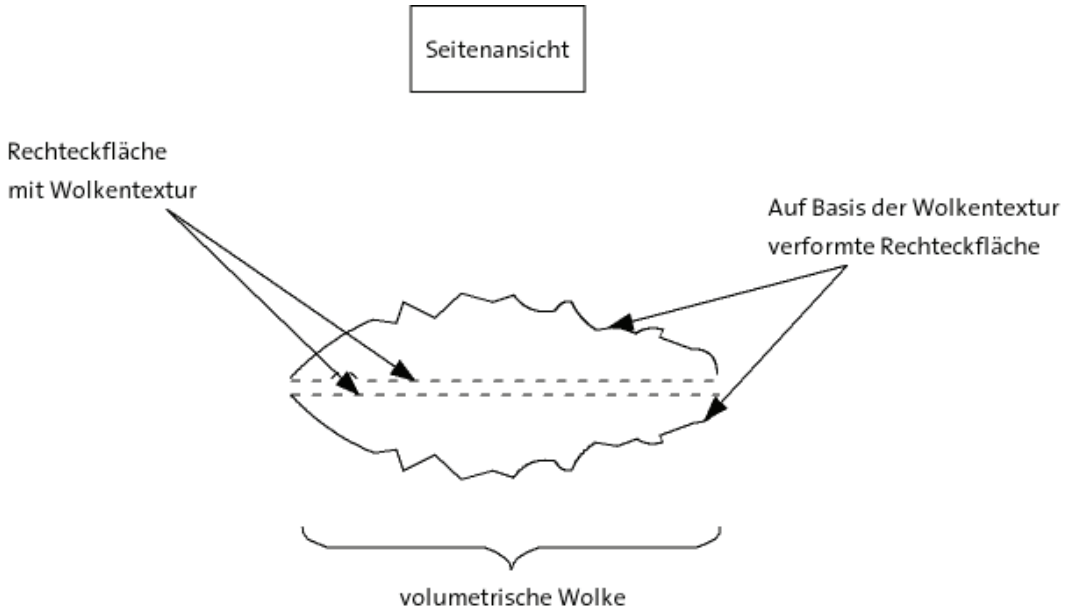


Abbildung 9: Volumetrische Wolke

Es wird außerdem eine Himmelskuppel in die Szene integriert, die wie eine Käseglocke auf die gesamte Szene aufgesetzt wird. Im Folgenden wird über das Environment-Skript die Lichtstimmung in der Szene angepasst. Hier wird neben den Nebeleinstellungen auch die Lichtfarbe und Intensität sowie der passende Sonnenstand, wichtig für korrekte Schattenberechnungen, via Skript gesetzt.

Zu guter Letzt wertet ein Skript die Niederschlagsdaten aus und setzt entsprechend der vorhergesagten Werte Partikelemitter in die Szene. Der Grafiker überprüft die gesetzten Partikelemitter und reduziert wenn nötig die Anzahl, um eine Verdeckung von Informationen durch den herabfallenden Regen oder Schnee zu verhindern.

Während des gesamten Arbeitsablaufs ist es mehrfach nötig kleinere Vorschaubilder zu berechnen um die gemachten Einstellungen zu überprüfen und wenn nötig zu ändern. Das Berechnen eines solchen Vorschaubildes kann an Tagen mit viel Niederschlag, also hoher Anzahl von zu berechnenden Partikeln, bis zu zwei Minuten dauern. Das verlangsamt den Erstellungsprozess enorm. Ist der Grafiker mit seiner Arbeit zu-

[4] Displacement-Mapping nennt man die Technik bei der Vertexpunkte einer Geometrie an Hand eines Bildes, in ihrer Position „verlagert“ werden.

frieden, schickt er die Szene an den Rendermanager der hr-internen Renderfarm. Der Rendermanager ist ein einzelner Computer, der die gesamte Szene, inklusive aller nötigen Texturen, automatisch an die in der Renderfarm bereitstehenden Rechner verteilt. Diese berechnen jeweils einen einzelnen Frame und senden dieses Bild an den Rendermanager zurück, bevor sie mit dem nächsten Bild fortfahren. Der Rendermanager speichert die einzelnen Bilder in einem zentralen Ordner. Das Ergebnis der Berechnung der 3ds-max-Wetterflugszene sind entsprechend der Länge der Animation 375 bis 500 Einzelbilder (übliche Länge eines Wetterflugs), die bereits fernsehgerecht in Halbbildern[5] gerechnet werden. Bis zu diesem Punkt soll die Echtzeitapplikation den eben beschriebenen Arbeitsablauf nach erfolgreicher Entwicklung ablösen.

Nach Abschluss der Analyse des Arbeitsablaufs bei der Wetterproduktion in der Grafikabteilung wurden zusätzlich die Anforderungen, Wünsche und Verbesserungsvorschläge der Grafiker ermittelt und in die in den folgenden Kapiteln aufgeführte Liste der Freiheitsgrade sowie in die Leistungsbeschreibung aufgenommen.

4.2 Konzipierung

4.2.1 Entscheidung: Zwei Module

Aus der Analyse des Arbeitsablaufs resultierte die Anforderung der einfachen Konfiguration eines Wetterflugs durch die Verwendung von entsprechenden Eingabemasken und Auswahllisten.

Aufgrund der wenig geeigneten Funktionalität von Quest3D zur Darstellung von Tabellen, Listen und Formularelementen, wurde die globale Konfiguration eines Flugs in ein so genanntes Authoring-Modul ausgelagert. In diesem können unter Verwendung von HTML und PHP sämtliche Einstellungen getroffen werden, die auf einer Auswahl der in der Datenbank vorhandenen Informationen basieren. Konfigurationsmöglichkeiten, die in das Authoring-Modul ausgelagert wurden, sind die globalen Flugparameter, die Festlegung der Flugroute über Drop-Down-Elemente, die Auswahl der auf dem Flug zu sehenden Städte aus einer Liste sowie die Temperatureingabe für eben diese ausgewählten Städte. Die einzelnen Formulare sind mit ihrer Funktionalität in Kapitel 5.3 näher beschrieben.

Die Darstellung des eigentlichen Wetterflugs geschieht in der Quest3D-Engine, welche das Runtime-Modul der „echtwetter“-Anwendung bildet. Das Runtime-Modul

[5] Halbbilder: Die Halbbildtechnik kommt beim Fernsehen zum Einsatz. Bei einem Halbbild ist nur jede zweite Zeile zu sehen. Dabei bilden jeweils zwei Halbbilder ein Vollbild. Eines enthält die ungeraden Zeilen (1,3,5,..), das andere die geraden Zeilen (2,4,6,...). Das in Europa übliche PAL Format zeigt 50 Halbbilder pro Sekunde an. Durch das Aufteilen von Vollbildern in Halbbilder wird eine höhere Bewegungsauflösung erreicht.

wertet die im Authoring-Modul fest gelegten Parameter aus. Zusätzliche Parameter für den Wetterflug, die ein visuelles Feedback benötigen, zum Beispiel die Lichtstimmung, wurden in dieses integriert (Abbildung 10).

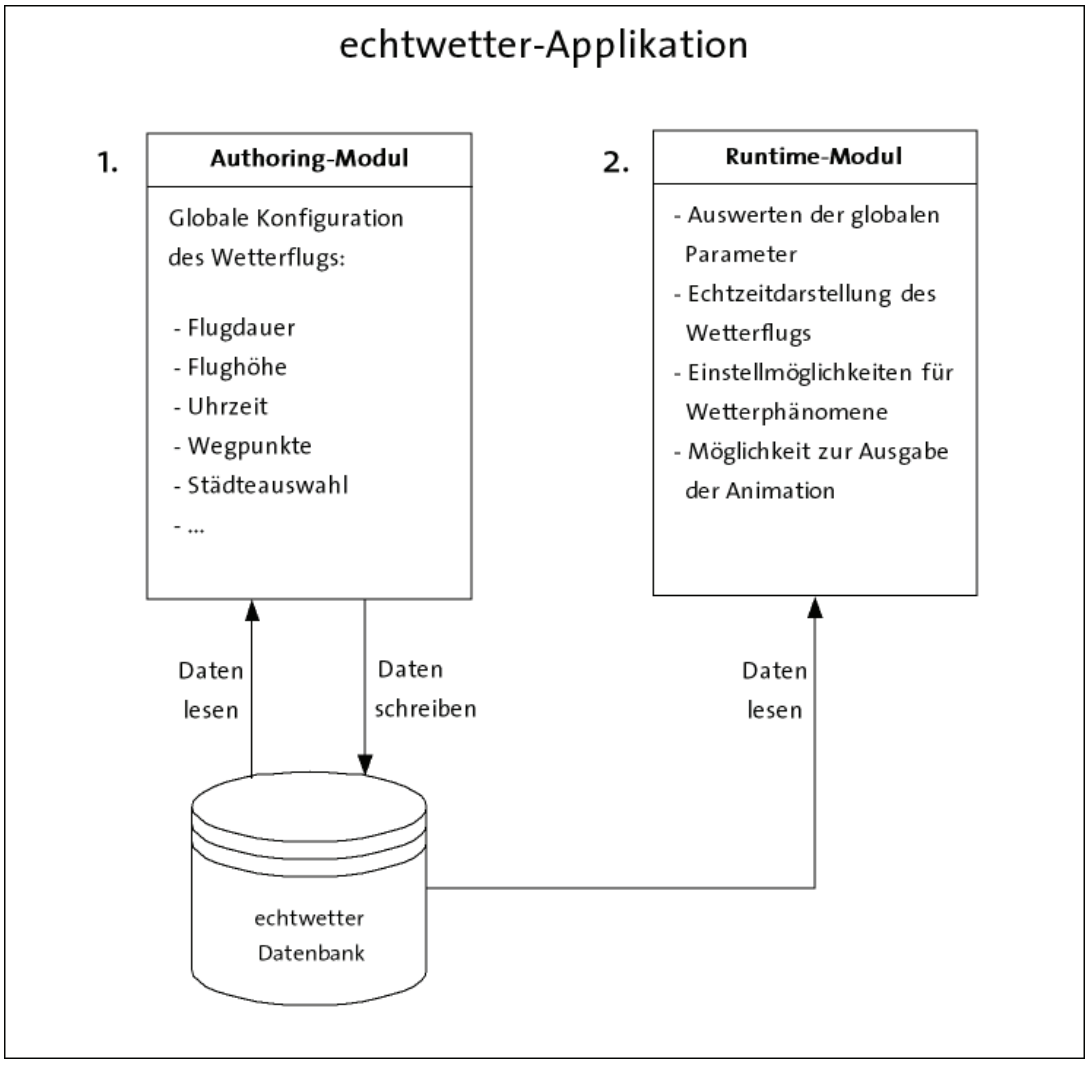


Abbildung 10: Aufbau der echtwetter-Applikation

4.2.2 Bestimmung der Freiheitsgrade

Nachdem die Entscheidung getroffen wurde die Echtzeitapplikation in zwei Module, das Authoring- und das Runtime-Modul, aufzuteilen, wurde zur genaueren Definition des gesamten Projekts eine Liste der Freiheitsgrade erstellt. Diese dokumentiert, wer in der späteren Anwendung was durch welches „Werkzeug“ ändern kann und ob die Integration für die erste Version unbedingt notwendig ist. Es ist nur natürlich, dass sich diese Liste im Verlauf des Projekts noch ändern kann, weil es zum Beispiel sinnvoller ist einen Parameter im Runtime-Modul mit visuellem Feedback zu definieren als in der HTML-Konfigurationsmaske ohne jede Art von Kontrolle. Dennoch ist die Erstel-

lung der Liste ein gutes Hilfsmittel in der Planung der einzelnen Module der „echtwetter“-Anwendung.

#	Einstell-/ Änderungsmöglichkeit	Änderungen vornehmende Partei	Art der Eingabe/Änderungstools	Nötig in Version 1.0 ?
1	Dauer des Flugs in Frames oder Sekunden	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
2	Welche Tageszeit wird in der Ansicht angezeigt?	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
3	Auswahl der anzuzeigenden Städte	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
4	Flugstrecke und Flughöhe	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) Dropdown Menü mit Auswahl der zu Verfügung stehenden Städte => Datenbank	Ja
5	Welche Kacheldaten werden geladen?	Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
6	Einbindung Wolkenbild	Grafiker	Dateiauswahldialog	Ja
7	Temperaturen/Luftfeuchtigkeit/ Niederschlagswahrscheinlichkeit in den Städten	Redakteur/ Grafiker/ automatisiert DWD	HTML Eingabemaske oder automatisierte Übernahme in die Datenbank	Mind. Eingabemaske
8	Regen (Aussehen)	weltenbauer.	Quest3D	Ja
9	Regen (lokale Positionierung/ Ausprägung)	Grafiker/ automatisiert DWD	Editor noch unklar	Mind. manuelle Positionierung

10	Schnee (Aussehen)	weltenbauer.	Quest3D	Ja
11	Schnee (lokale Positionierung/ Ausprägung)	Grafiker/ automatisiert DWD	Editor noch unklar	mind. manuelle Positionierung
12	Schneehöhe auf Boden	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
13	Wind (Richtung/Geschwindigkeit)	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
14	Wahrzeichen ein-/ausschalten	Redakteur/ Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Ja
15	Wahrzeichen austauschbar	Grafiker	gesonderte HTML Eingabemaske	Nein
16	Wahrzeichen erweiterbar	weltenbauer.	Quest3D	-
17	Erstellen unterschiedlicher Texturensätze (Frühling, Sommer, Herbst, Winter)	Grafiker	Photoshop & Quest3D	Nein
18	Auswahl unterschiedlicher Texturensätze (Frühling, Sommer, Herbst, Winter)	Grafiker	HTML (Step-by-Step Abfrage) => Datenbank	Nein
19	Lichtstimmung (Einstellen von Farbe/Helligkeit von Himmel und Texturen)	Grafiker	Editor noch unklar (mit Vorschaufunktion)	Ja
20	Hinzufügen neuer Orte/Städte mit Namen und Position	Grafiker	gesonderte HTML Eingabemaske	Nein

4.2.3 Leistungsbeschreibung

Nach erfolgter Analyse der Basisdaten sowie der Bestimmung der Freiheitsgrade wurden die Ansprüche an die „echtwetter“-Applikation in Form einer Leistungsbeschreibung so genau wie möglich definiert. Auf die konkretere Form eines Pflichtenhefts mit detaillierten Beschreibungen der Umsetzung wurde bewusst verzichtet, da zu Anfang des Projekts, trotz der Erfahrungen mit dem Prototyp, die exakte Umsetzung einiger in dem Projekt anstehender Anforderungen, wie zum Beispiel dem dynamischen Laden von Geometrien, noch nicht klar war.

Die Anforderungen an die Echtzeitapplikation richteten sich im Groben nach den bereits bestehenden Möglichkeiten bei der Erstellung eines Wetterflugs mit 3ds max und werden im Folgenden kurz aufgezählt.

- Möglichkeit zur Darstellung der gesamten Bundesrepublik in Teilstücken.
- Möglichkeit zur Definition des zu ladenden Teilstücks der Landschaftsgeometrie.
- Verwendung von zwei Detailstufen sowohl für Geometrien als auch für Texturen, mit entfernungsabhängiger, dynamischer Umschaltung.
- Möglichkeit zur einfachen Definition eines Kameraflugs an Hand von Eingabemasken.
- Möglichkeit zur einfachen Austauschbarkeit der Texturen.
- Automatisierte Darstellung von Städtenamen inklusive Temperaturen auf Basis der vorhandenen Datenbank. Die Auswahl der Städte erfolgt über eine Eingabemaske.
- Darstellung von Wetterphänomenen (Wolken, Nebel, Niederschlag, Blitze und Wind).
- Einstellmöglichkeiten in Form, Farbe und Ausprägung für die genannten Wetterphänomene.
- Einstellmöglichkeiten für Lichtstimmungen.
- Darstellung von Wahrzeichen mit Echtzeitschattenwurf.
- Möglichkeit zur Einzelbildausgabe der fertig erstellten Animation.

Zur Erfüllung der in der Leistungsbeschreibung definierten Anforderungen müssen zunächst die vorhandenen Basisdaten in Form von Landschafts- und Wahrzeichengeometrien sowie deren Texturen in ein echtzeitgeeignetes Format gebracht werden. Auch die bereits vorliegende Städtedatenbank muss an die „echtwetter“-Applikation

angepasst werden. Diese Konvertierungen werden im folgenden Kapitel genauer erläutert.

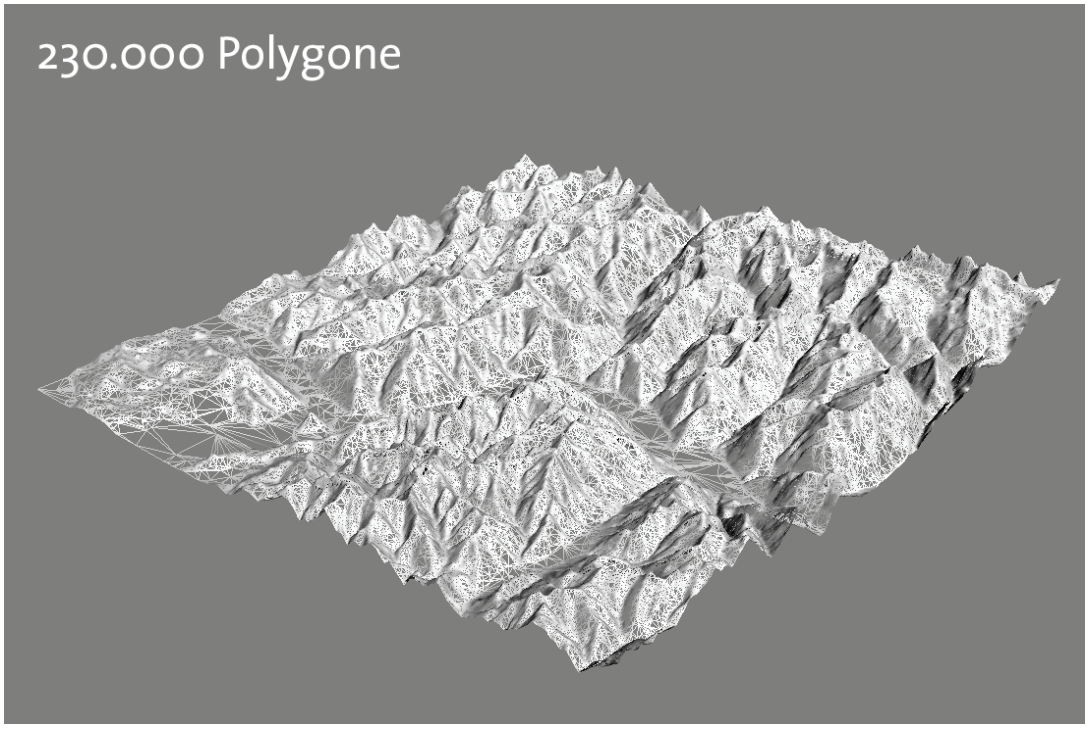
4.3 Konvertierung der vorhandenen Basisdaten

4.3.1 Geometriedaten

Das folgende Kapitel beschäftigt sich mit der Konvertierung der im 3ds-max-Format vorhandenen Geometriedaten in ein in Quest3D verwendbares Datenformat. Quest3D kann mit dem von Microsoft entwickelten DirectX-Format (*.x), das sich speziell für Echtzeitapplikationen eignet, umgehen.

Neben der Konvertierung der Landschaftsgeometrien, die den Schwerpunkt dieses Kapitels bildet, müssen auch die Wahrzeichenobjekte in ein Echtzeitformat umgewandelt werden.

Die Geometriedaten der Bundesrepublik liegen als insgesamt 520 einzelne Kacheln in 3ds-max-Dateien vor. Von jeder dieser Kacheln existieren insgesamt vier Detailstufen. Für die „echtwetter“-Applikation werden jedoch nur die Detailstufen eins (höchste vorhandene Auflösung) und vier (niedrigste vorhandene Auflösung) benötigt (Abbildung 11).



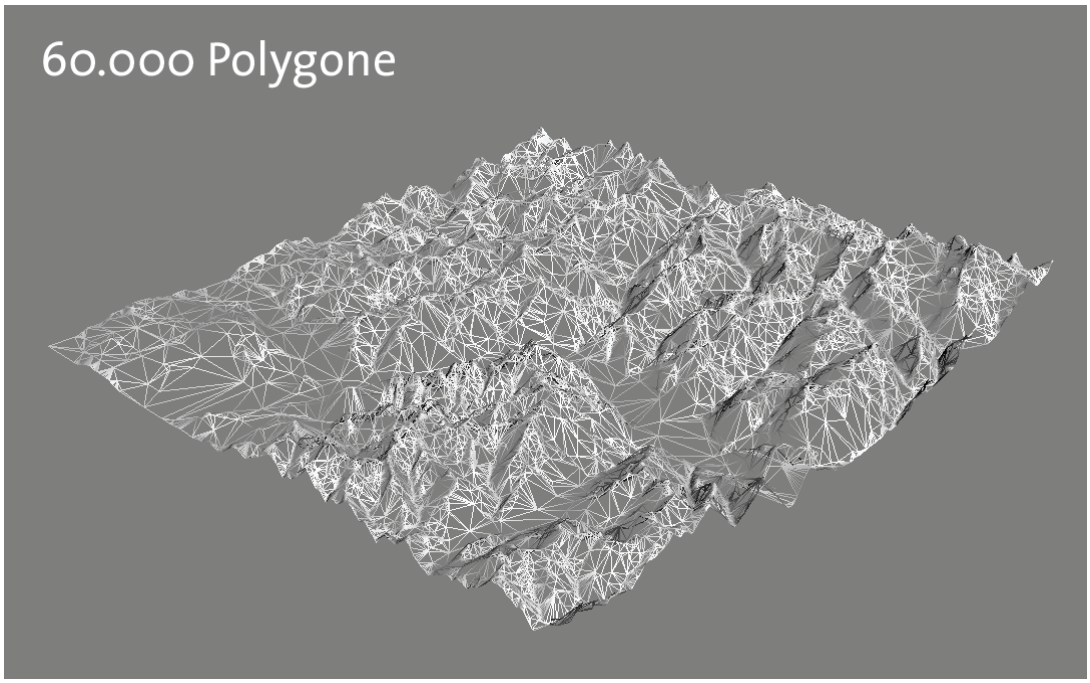


Abbildung 11: Hohe und niedrige Detailstufe einer Landschaftskachel

Das bedeutet, sowohl die niedrig als auch die hoch aufgelöste Version einer Kachel müssen aus 3ds max in das DirectX-Format exportiert werden, insgesamt 1040 Dateien. Doch bevor die Kacheln exportiert werden können, ist eine Repositionierung der Kacheln notwendig. Die in 3ds max vorhandenen Daten sind, ob ihrer Herkunft (Satellitendaten), in einem globalen Zusammenhang positioniert. Das heißt eine einzelne Landschaftskachel befindet sich genau an der Stelle, wo sie im Bezug auf die ganze Welt gesehen hingehört. Da für den Satelliten Deutschland nicht der Nabel der Welt ist, befinden sich die Kacheln sehr weit vom Szenenursprung entfernt, was zu sehr großen Positionswerten führt. Da Quest3D nur mit einer einfachen Floatgenauigkeit arbeitet und es bei sehr großen Positionswerten, welche die vorhandenen Stellen zur Speicherung der Mantisse überschreiten zu Ungenauigkeiten kommt, andererseits eine exakte Positionierung der Kacheln aber unbedingt notwendig ist, um entsprechende Lücken zwischen den Kacheln zu vermeiden, ist eine Verschiebung aller Kacheln in den Szenenursprung sinnvoll. Dies erleichtert die Arbeit in Quest3D sehr, weil die Kacheln direkt im Szenenursprung zu finden sind und nicht irgendwo außerhalb des Sichtbereichs der Kamera positioniert werden. Des Weiteren werden alle Kacheln auf 1 % ihrer ursprünglichen Größe skaliert, da lediglich der Betrag der Größe beim Import in Quest3D interpretiert wird, nicht aber die Maßeinheit.

Neben den eben beschriebenen Transformationen muss außerdem ein zusätzlicher Satz Texturkoordinaten auf die einzelnen Kacheln aufgebracht werden. Der erste schon vorhandene Texturenkoordinatensatz beinhaltet die Ausrichtung der jeweiligen Kacheltextur auf der entsprechenden Kachel. Für die in der „echtwetter“-Applikation

geplante Darstellung des Wolkenschattens muss jedoch ein zweiter Satz Texturkoordinaten angelegt werden, der sich über alle Kacheln erstreckt (Abbildung 12).

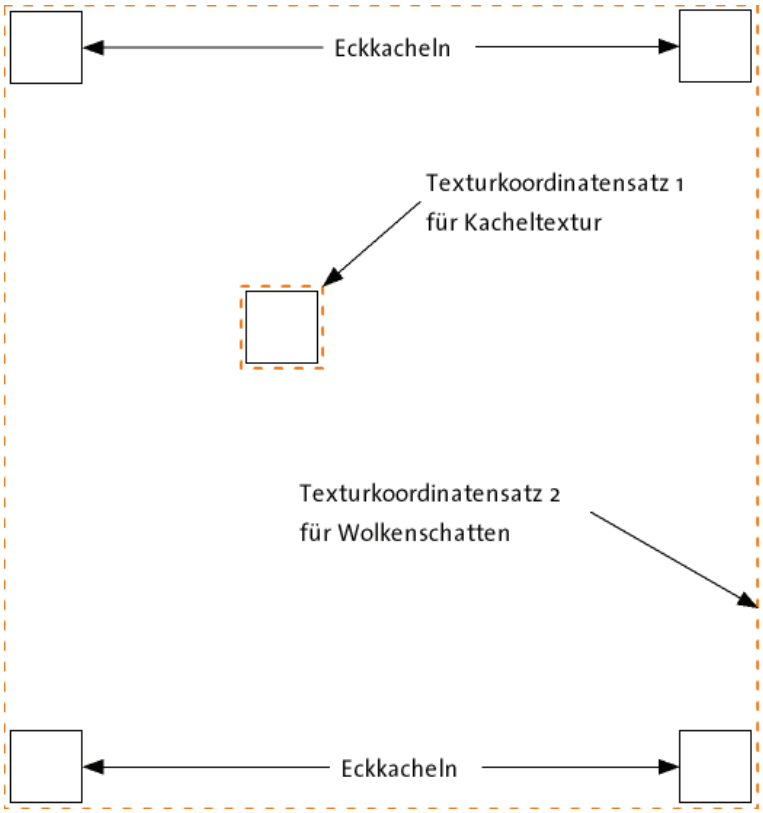


Abbildung 12: Zusätzlicher Satz Texturkoordinaten für den Wolkenschatten

Es empfiehlt sich die Kacheln direkt mit einem weißen, matten Material zu bestücken, da diese Einstellungen mit exportiert werden und später in Quest3D nicht mehr von Hand vorgenommen werden müssen.

Für den Export der Kacheldaten kommt der, als freie Erweiterung im Internet erhältliche, Panda DirectX Exporter zum Einsatz [7]. Dieser Exporter besitzt als einziger die Fähigkeit zwei Sätze Texturkoordinaten herauszuschreiben. Beim Exportieren der Kacheln unter Verwendung des Panda DirectX Exporters, ist darauf zu achten, dass der Export der Geometrie und der Flächen-Normalen (Mesh Normals) sowie der Texturkoordinaten (Mapping Coordinates) aktiviert ist. Die Optimierung der Polygonnetze (Mesh Optimization) muss unter allen Umständen auf die Option „none“ gestellt werden, da es ansonsten zu Unstimmigkeiten in der Geometrie kommen kann. Auf die Animation und die übrigen Optionen sowie die Konvertierung der Texturen kann verzichtet werden.

Nach erfolgreichem Export aller Kacheln, sowohl der niedrig als auch der hoch aufgelösten, liegen diese nun in insgesamt 1040 einzelnen *.x-Dateien vor. Im nächsten

Schritt werden immer die zwei Detailstufen einer Kachel in eine Channel Group in Quest3D importiert und dann unter dem Namen der jeweiligen Kachel (z.B. mona100_pol_x70_y111.cgr) abgespeichert. Es entstehen 520 *.cgr-Dateien, die jeweils die beiden unterschiedlich hoch aufgelösten Geometrien inklusive zweier Sätze Texturkoordinaten beinhalten. Die in der jeweiligen Channel Group weiterhin implementierten Logiken zur Verwendung der beiden Detailstufen wird in Kapitel 6.4.1 ausführlich beschrieben.

Obwohl es, laut Aussage der Quest3D Entwickler, keinerlei Beschränkung in der Anzahl der Polygone importierter Objekte gibt, kommt es beim Import von Kacheln mit mehr als 400.000 Polygonen zu Schwierigkeiten. Die betroffenen Kacheln werden zwar als Channels auf der Logikoberfläche angezeigt, die Darstellung der Kacheln ist jedoch nicht möglich. Von diesem Phänomen betroffene Kacheln der hohen Detailstufe wurden in 3ds max mit dem „Optimize“-Befehl in ihrer Polygonanzahl reduziert und ein zweites Mal exportiert.

Die in 3ds max vorhandenen 3D-Modelle der Wahrzeichen müssen ebenfalls exportiert werden. Diese teilweise aus sehr vielen Einzelteilen bestehenden Modelle werden zunächst mittels des in 3ds max enthaltenen „Collapse“-Befehls auf ein Objekt reduziert. Manche der Modelle mussten komplett neu gebaut werden, da das Gitternetz der bestehenden Objekte teilweise zu hoch aufgelöst war oder nicht zulässige Geometrie enthielt. Im Anschluss wurden die Modelle auf die gleiche Art transformiert wie zuvor die Landschaftskacheln, um den in den 3ds-max-Dateien enthaltenen Bezug zur Landschaft nicht zu verlieren. Zu guter Letzt wurden die Modelle ebenfalls mit dem Panda DirectX Exporter in einzelne Dateien geschrieben und erfolgreich in eine Channel Group in Quest3D importiert. Die Konvertierung der auf den Modellen befindlichen Materialien und Texturen der Wahrzeichen sowie die Aufbereitung der Landschaftstexturen der einzelnen Kacheln werden im folgenden Kapitel näher erläutert.

4.3.2 Texturen

Der folgende Abschnitt befasst sich mit den für die Echtzeitapplikation nötigen Texturen und deren Konvertierung. Die Texturen für Landschaftskacheln müssen in der Größe angepasst und zur Verwendung in der Echtzeitapplikation in ein anderes Format konvertiert werden. Die Texturen der Wahrzeichen werden aus komplexeren Materialien gewonnen und ebenfalls in ein echtzeitoptimiertes Format umgewandelt.

Die Texturen für die einzelnen Landschaftskacheln liegen im TIF-Format (Tagged Image File) in einer Größe von 2500 x 2500 Pixeln vor. Diese Texturen sind in unkomprimierter Form ca. 21 MB groß und somit für den Einsatz im Echtzeitbereich, wo ja nur

ein begrenzter Texturspeicher auf der Grafikkarte zur Verfügung steht, gänzlich ungeeignet. Auch eine Kompression der Texturen durch die Verwendung des JPEG-Formats (Joint Photographic Experts Group) bringt im Echtzeitbereich nichts, da die Grafikkarte die Textur in komprimierter Form nicht verwenden kann, diese erst entkomprimiert werden müsste und so der Vorteil der kleineren Dateigröße wieder verloren geht. Aus diesem Grund kommt das von Microsoft entwickelte und in DirectX 7.0 eingeführte DDS-Bildformat (Direct Draw Surface) zum Einsatz [8].

Texturen können im DDS-Format komprimiert abgespeichert und in dieser Form von der Grafikkarte verwendet werden. Das DDS-Format bietet dabei unterschiedlichste Optionen der Bildspeicherung, zum Beispiel verschiedene Bittiefen für die einzelnen Farbkanäle oder optionale Verwendung von Alpha-Kanälen. Das wichtigste Merkmal an Texturen im DDS-Format ist jedoch das so genannte Mip-Mapping. Dabei werden in der Bilddatei mehrere Auflösungsstufen der Textur abgespeichert. Zwischen den verschiedenen Auflösungsstufen wird abhängig von der Entfernung zur Kamera automatisch hin und her geschaltet, vergleichbar mit der Level-of-Detail-Technik bei Geometrien.

Wird nämlich ein weit entferntes Objekt in einer 3D-Szene mit einer hoch aufgelösten Textur dargestellt, muss der Farbwert für einen einzelnen Pixel aus sehr vielen Texturpunkten gemittelt werden. Die Anzahl der zu mittelnden Texturpunkte kann durch die Verwendung einer geringeren Auflösungsstufe stark reduziert werden. Genau zu diesem Zweck wird Mip-Mapping verwendet.

Texturen können nur in einem komprimierten DDS-Format abgespeichert werden, wenn das Bild quadratische Ausmaße hat und die Pixelanzahl einer Zweierpotenz entspricht, zum Beispiel 256 x 256 oder 2048 x 2048 Pixel.

Die im TIF-Format vorliegenden Texturen der Landschaftskachel mussten in das DDS-Format konvertiert werden. Dazu wurde zunächst die Bildgröße auf ein 2048er-Format gebracht und anschließend mit automatischer Erstellung der Mip-Maps in ein komprimiertes DDS-Format (DXT1) abgespeichert. Zu diesem Zweck wurde in Adobe Photoshop eine entsprechende Aktion erstellt und diese mit Hilfe der Stapelverarbeitung auf alle Bilder angewendet.

Nach der Konvertierung der Texturen in das DDS-Format werden nur noch 2 MB pro Textur im Speicher der Grafikkarte belegt.

Dateiformat	Dateigröße (2048 x 2048/RGB)	Größe im Speicher der Grafikkarte
TIF (Tagged Image File) mit LZW Komprimierung	7,41 MB	21,84 MB
JPEG (Joint Photographic Experts Group) mittlere Kompression (Stufe 6)	0,97 MB	21,84 MB
DDS (Direct Draw Surface) DXT1 Komprimierung	2,00 MB	2,00 MB

Die Materialien für die Wahrzeichen der Städte in 3ds max sind sehr komplex zusammengestellt; teilweise sind einzelne Polygone eines 3D-Objekts mit unterschiedlichen Materialien belegt. Da jedes verwendete Material einzeln aus 3ds max herausexportiert werden müsste, wurde nach einer besseren Lösung gesucht. An dieser Stelle kommt das so genannte Texture-Baking-Verfahren zum Einsatz, bei dem die Materialien eines Objektes in eine einzige Textur „gebacken“ werden. Die gebackene Textur für die Wahrzeichen wurde wie auch die Landschaftstexturen im DDS-Format abgespeichert.

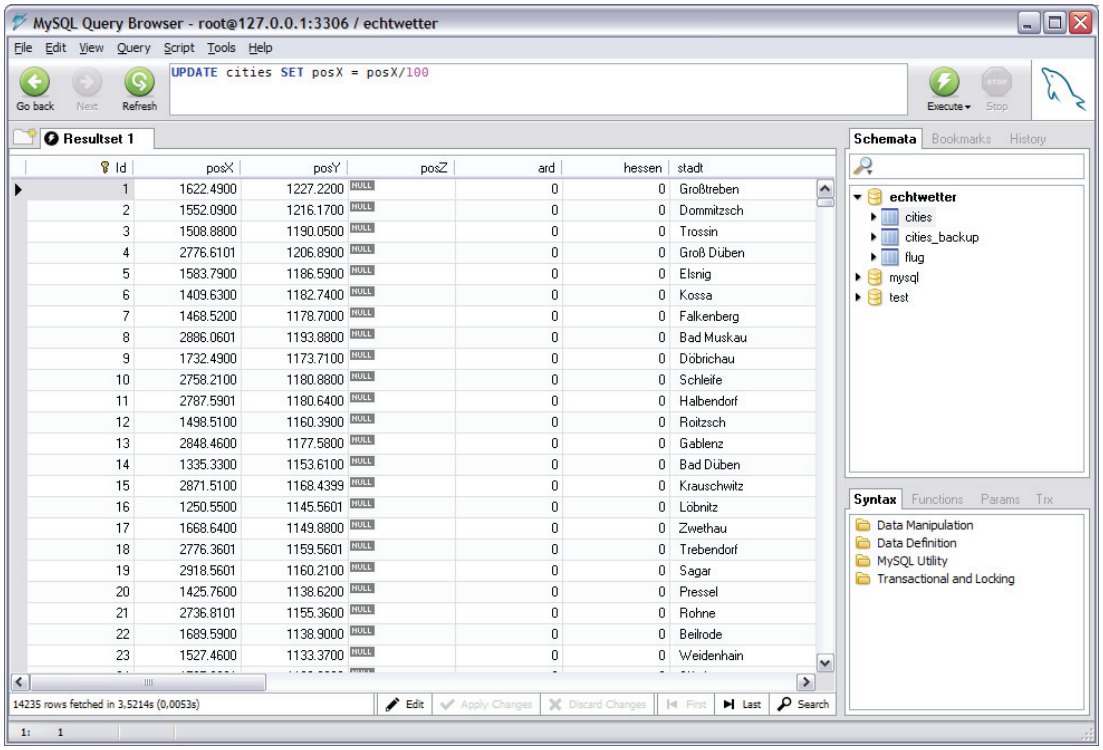
Auch alle anderen in der Echtzeitanwendung verwendeten Texturen – wie beispielsweise Wolken, Menüelemente und Niederschlagstexturen - wurden zwecks Speicherplatzoptimierung im DDS-Format abgespeichert.

4.3.3 Städtedatenbank

Die beim hr bereits vorhandene Städtedatenbank, welche die Daten von insgesamt 14.235 Städten beinhaltet, sollte auch in der Echtzeitanwendung weiterhin Verwendung finden. Da zur Konfiguration der Wetterflüge eine Kombination aus in PHP erstellten Eingabemasken und der Datenbank MySQL zur Speicherung der Eingaben verwendet wird, musste die vorhandene Access-Datenbank in das MySQL-Format konvertiert werden. Hierfür wurde der gesamte Datenbestand der Access-Datenbank in einen durch Tabstopps getrennten Textfile exportiert und anschließend in die MySQL-Datenbank importiert. Lediglich die Spaltenzuweisung muss dabei einmalig neu vorgenommen werden.

Neben der Datenübernahme in die MySQL-Datenbank macht die Repositionierung und Skalierung der Landschaftskacheln in Kapitel 4.3.1 eine Anpassung der Positions-

werte nötig. Die Verschiebung der Kacheln entspricht dabei einer Subtraktion in x-Richtung sowie einer Addition in y-Richtung. Die Skalierung auf 1 % der Originalgröße der Geometrien entspricht einer Division der Positionswerte durch 100. Dabei ist zu beachten, dass die zuvor in Integer-Werten abgespeicherten Positionswerte im Datentyp Float abgespeichert werden, da ansonsten die durch die Division zu Stande kommenden Nachkommstellen verloren gehen und so die exakte Positionierung der Städte nicht mehr möglich ist. Die Veränderungen der Positionswerte wurden mittels des MySQL-Query-Browsers vorgenommen, in dem ein entsprechender Update-Befehl definiert und ausgeführt wurde (Abbildung 13).



Id	posX	posY	posZ	ard	hessen	stadt
1	1622.4900	1227.2200	NULL	0	0	Großtreben
2	1552.0900	1216.1700	NULL	0	0	Dommitzsch
3	1508.8800	1190.0500	NULL	0	0	Trossin
4	2776.6101	1206.8900	NULL	0	0	Groß Düben
5	1583.7900	1186.5900	NULL	0	0	Elsnig
6	1409.6300	1182.7400	NULL	0	0	Kossa
7	1468.5200	1178.7000	NULL	0	0	Falkenberg
8	2886.0601	1193.8800	NULL	0	0	Bad Muskau
9	1732.4900	1173.7100	NULL	0	0	Döbrichau
10	2758.2100	1180.8800	NULL	0	0	Schleife
11	2787.5901	1180.6400	NULL	0	0	Halbendorf
12	1498.5100	1160.3900	NULL	0	0	Roitzsch
13	2848.4600	1177.5800	NULL	0	0	Gablenz
14	1335.3300	1153.6100	NULL	0	0	Bad Düben
15	2871.5100	1168.4399	NULL	0	0	Krauschwitz
16	1250.5500	1145.5601	NULL	0	0	Löbnitz
17	1668.6400	1149.8800	NULL	0	0	Zwethau
18	2776.3601	1159.5601	NULL	0	0	Trebendorf
19	2918.5601	1160.2100	NULL	0	0	Sagar
20	1425.7600	1138.6200	NULL	0	0	Pressel
21	2736.8101	1155.3600	NULL	0	0	Rohne
22	1689.5900	1138.9000	NULL	0	0	Beilrode
23	1527.4600	1133.3700	NULL	0	0	Weidenhain

Abbildung 13: MySQL-Query-Browser

4.4 Zusammenfassung

Zur Verwendung in der zu entwickelnden Wetterflugapplikation wurden die vorhandenen Basisdaten in ein echtzeittaugliches Format gebracht. Die Geometrien wurden repositioniert und skaliert und mit Hilfe des Panda DirectX Exporters in das von Microsoft entwickelte X-Format exportiert, die Texturen mittels eines Photoshop-Automatismus in das DDS-Format gebracht sowie die bereits vorhandene Städtedatenbank an die Repositionierung und die Skalierung der Kacheln angepasst und im MySQL-Format abgespeichert. Damit war die Grundlage geschaffen im Folgenden das

Authoring- und das Runtime-Modul zu entwickeln, aus denen sich die „echtwetter“-Applikation zusammensetzt.

5 Entwicklung eines Authoring-Moduls

5.1 Einleitung

Das Authoring-Modul dient der Festlegung der globalen Parameter eines Wetterflugs. Als Benutzeroberfläche dient dabei eine in HTML und PHP erstellte Eingabemaske, welche die Konfiguration des Wetterflugs in eine Datenbank schreibt, um sie der Echtzeitanwendung zugänglich zu machen. Zu diesem Zweck musste die bestehende Datenbank erweitert werden. An Hand der HTML/PHP-Maske kann ein Benutzer schnell und einfach einen Wetterflug erstellen, der anschließend im Runtime-Modul angezeigt werden kann.

5.2 Erweiterung der Datenbank

Die im MySQL-Format vorhandene „echtwetter“-Datenbank musste, um die Flugkonfigurationen abspeichern zu können, um eine zusätzliche Tabelle erweitert werden. Die schon vorhandene Städtetabelle wurde lediglich um zwei weitere Spalten ergänzt.

In der zusätzlichen Tabelle namens „Flug“ werden die globalen Parameter eines Flugs gespeichert. Neben dem Namen, der Länge, der Uhrzeit und der Flughöhe werden die zu ladenden Landschaftskacheln und die zu verwendenden Wolkenbilder abgelegt. Weiterhin wird die Flugroute, bestehend aus maximal fünf Wegpunkten, abgespeichert. In der letzten Spalte wird jeweils der aktive Flug ausgewählt.

Die bereits vorhandene Städtetabelle, die aus dem Access-Format ins MySQL-Format konvertiert wurde (Kapitel 4.3.3), wird um zwei Spalten erweitert. Eine der beiden Spalten beinhaltet die lokale Temperatur zum Flugzeitpunkt, die andere die Sichtbarkeit der Stadt auf dem Flug (Abbildung 14).

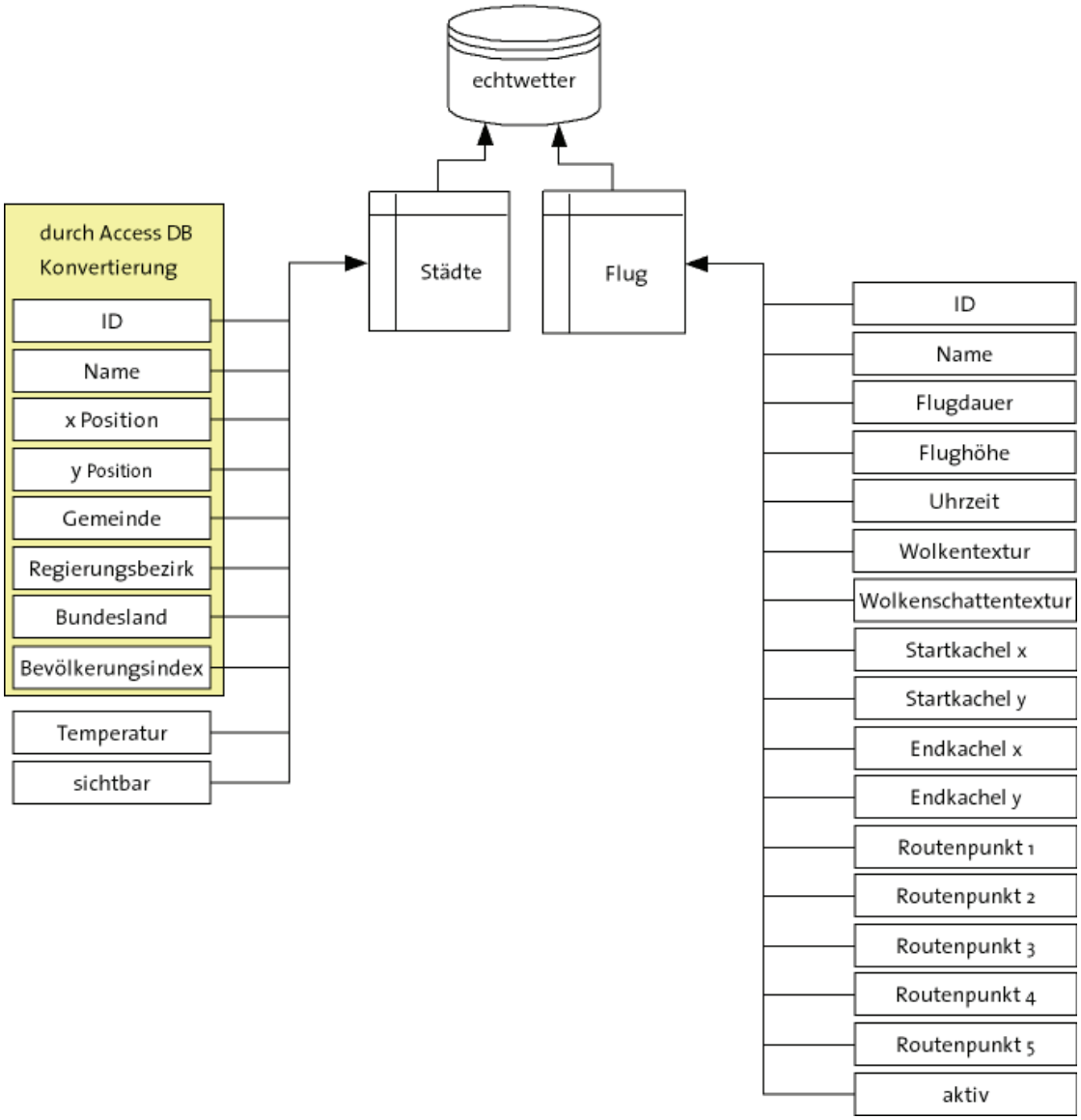


Abbildung 14: Erweiterung der „echtwetter“-Datenbank

5.3 Erstellung der HTML/PHP Masken

5.3.1 Städteauswahl

Der erste nötige Schritt bei der Konfiguration eines Wetterflugs ist die Auswahl der auf dem Flug zu sehenden Städte. Zu diesem Zweck wurde ein Formular entwickelt, das die Auswahl der Städte an Hand einstellbarer Filterregeln möglich macht. Der Benutzer kann beispielsweise im Filterbereich ein bestimmtes Bundesland auswählen und sich alle dort liegenden Städte anzeigen lassen. Die Auswahl kann über das Festlegen der Bevölkerungsgröße weiter eingeschränkt werden. Es ist auch möglich eine Stadt direkt über die Eingabe des Namens in die Vorauswahl zu übernehmen. Die in der Mitte der Seite befindliche Spalte zeigt die getroffene Vorauswahl. Durch Aktivie-

ren der Checkbox und das anschließende Drücken des „Auswahl übernehmen“-Buttons werden die in der Vorauswahl selektierten Städte in die Auswahl der anzuzeigenden Städte in der rechten Spalte übernommen (Abbildung 15).

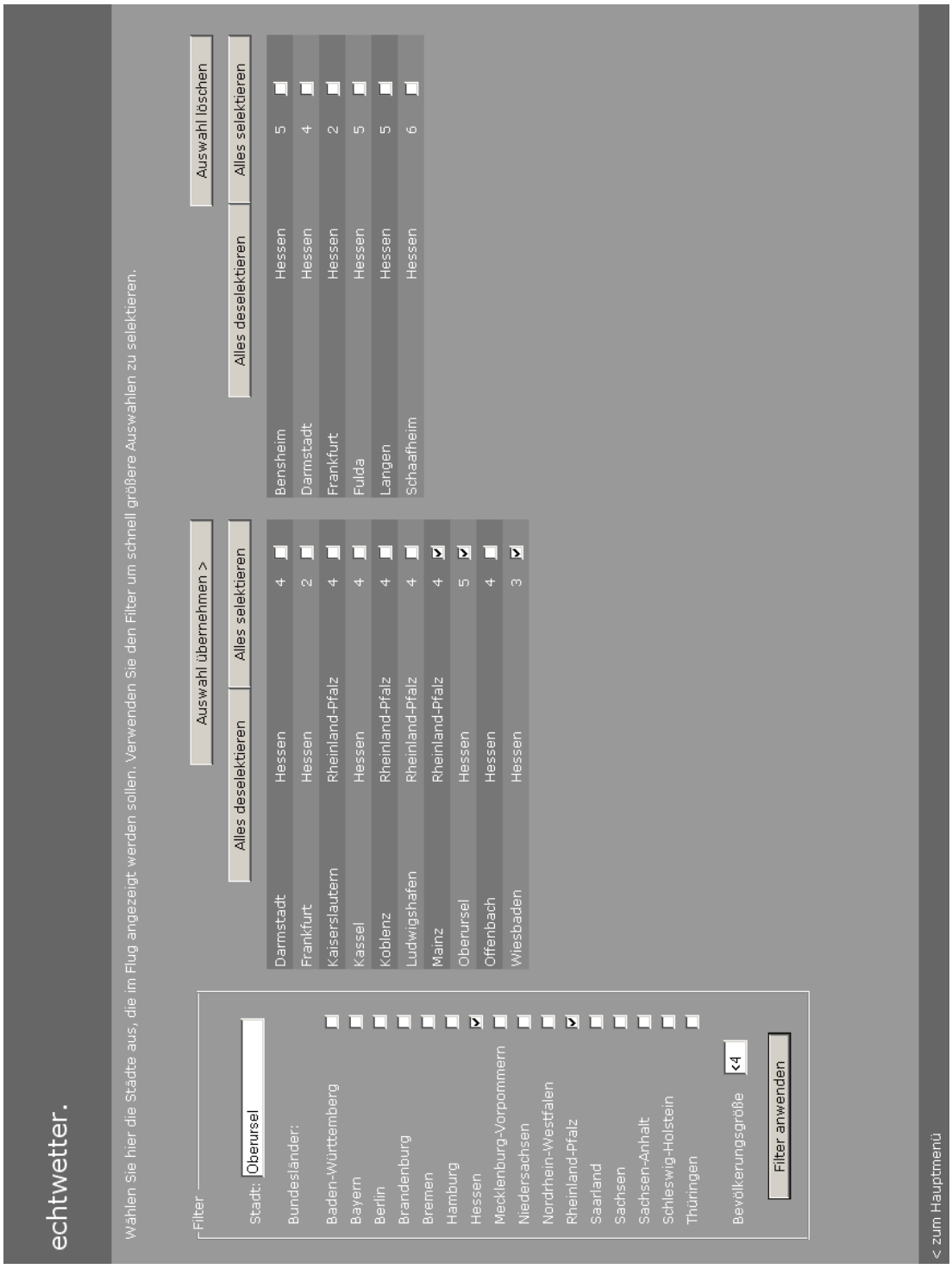


Abbildung 15: Maske zur Städteauswahl

5.3.2 Erstellung eines neuen Flugs

Nach der erfolgten Auswahl der auf dem Flug zu sehenden Städte werden in der folgenden Eingabemaske weitere Parameter des Flugs festgelegt. Zunächst sollte zur einfacheren Identifikation eines Flugs ein eindeutiger Name, der beispielsweise die Flugroute und das Flugdatum beinhalten kann, vergeben werden. Daraufhin werden die Länge des Flugs in Frames, die Uhrzeit und die Flughöhe in Metern eingegeben. Anschließend wird das zu überfliegende Gebiet in Form von Start- und Endkachel eines zu ladenden Rechtecks definiert. Zur einfacheren Bestimmung der benötigten Kacheln existiert eine Deutschlandkarte, auf die das Raster der Kacheldaten aufgetragen ist (Abbildung 16). Die folgenden zwei Eingabefelder werden mit den absoluten Pfaden zur berechneten Wolkentextur und der entsprechenden Wolkenschattentextur gefüllt. Abschließend wird die Flugroute über die Auswahl der maximal fünf Wegpunkte definiert. In den Drop-Down-Listen stehen nur die in der Städteauswahl definierten Städte zur Verfügung (Abbildung 17).

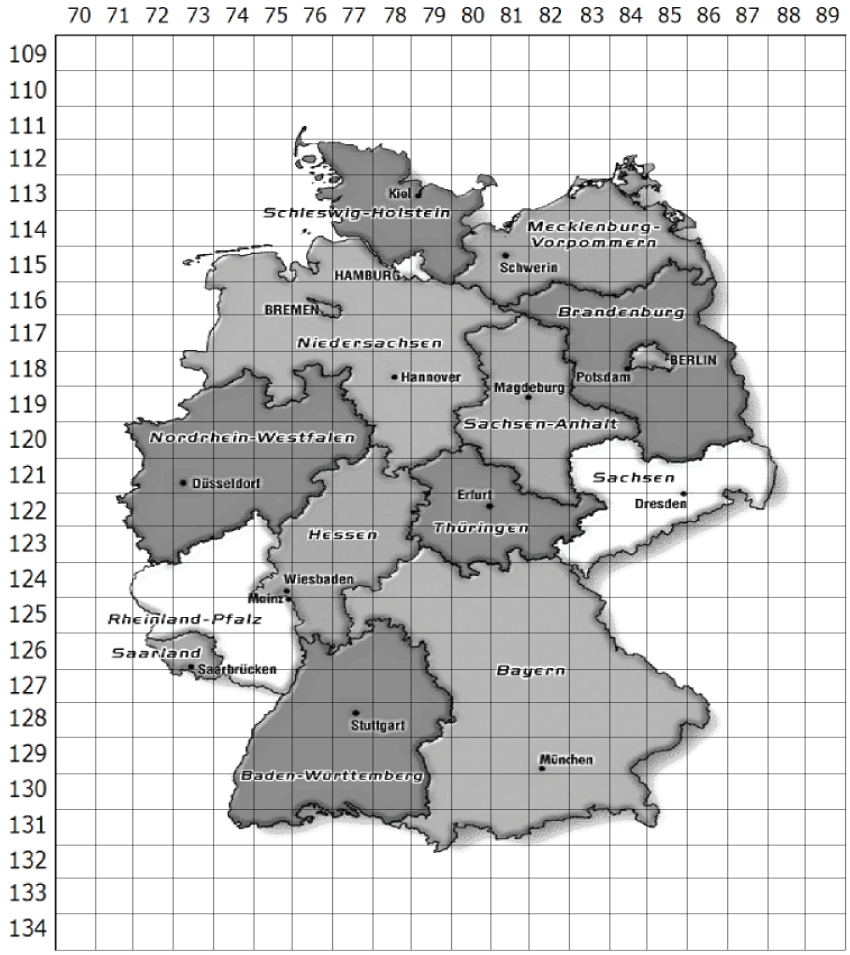


Abbildung 16: Deutschlandkarte mit Kachelraster

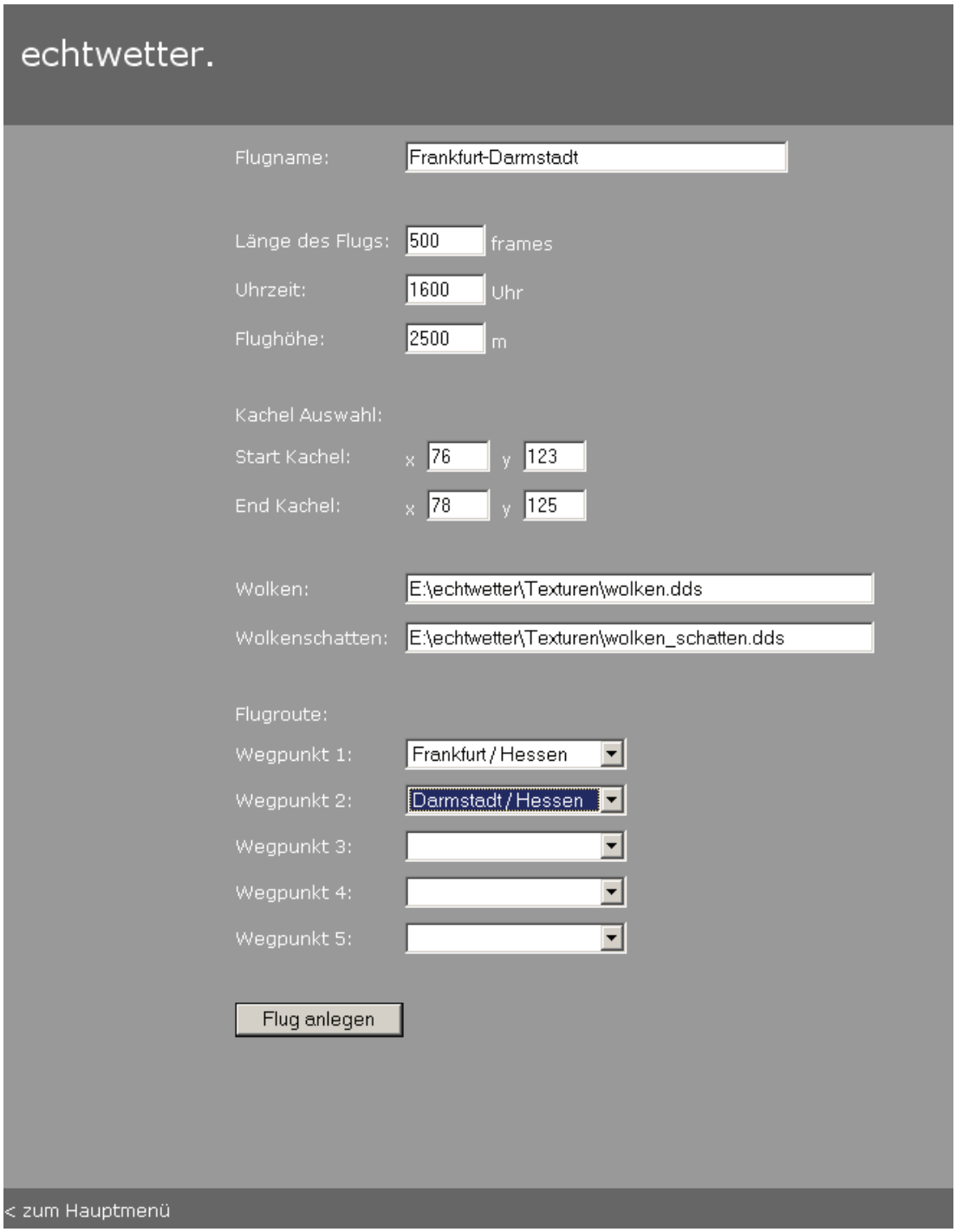


Abbildung 17: Maske zur Flugerstellung

5.3.3 Temperatureingabe

Ein weiterer nötiger Schritt zur Erstellung eines vollständigen Flugs ist die Eingabe der zum Flugzeitpunkt vorherrschenden Temperaturen in den ausgewählten Städten. Zu diesem Zweck gibt es eine zusätzliche Eingabemaske, welche die in der Städteauswahl aktiv gesetzten Städte mit einem Eingabefeld zur Temperaturangabe anzeigt. Die

Temperaturen werden nach Betätigung des Buttons „Temperaturen eintragen“ in die Datenbank geschrieben, von wo sie zur Laufzeit der Applikation ausgelesen werden können (Abbildung 18).

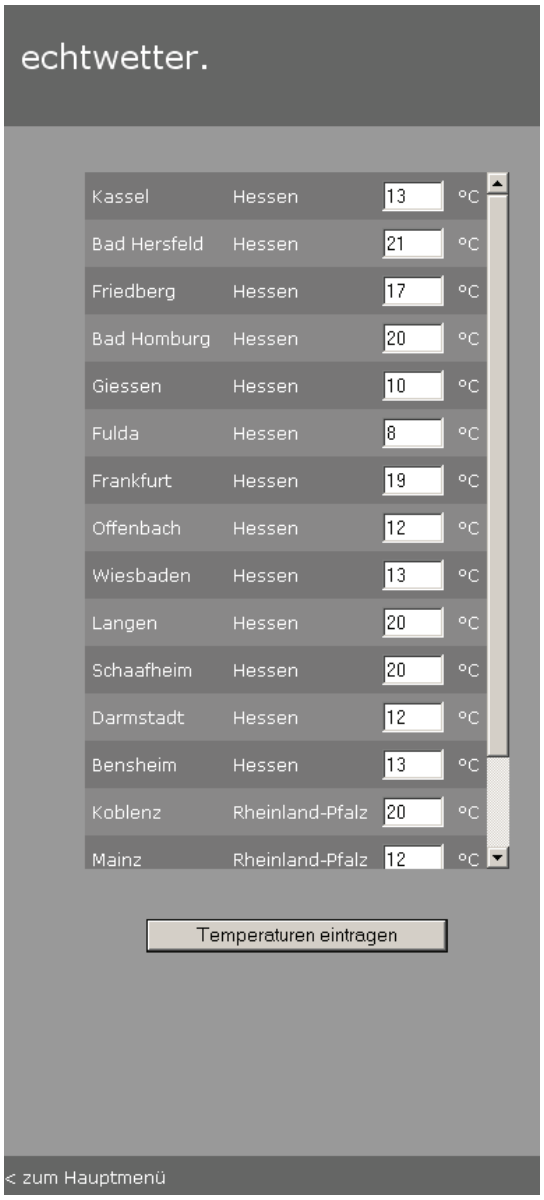


Abbildung 18: Maske zur Temperatureingabe

5.4 Zusammenfassung

Über die oben beschriebenen HTML/PHP-Eingabemasken werden die globalen Parameter eines Wetterflugs eingegeben. Angegeben werden darin neben der Flughöhe, der Flugdauer, der Uhrzeit, der Wegstrecke, den anzuzeigenden Städten und deren Temperaturen auch das in einzelne Kacheln unterteilte, zu ladende Gebiet. Sowohl die Auswahl der Städte, die als Wegpunkte dienen, als auch die Selektion der Städte, die

auf dem Flug zu sehen sind, geschieht dabei an Hand der in der Datenbank vorhandenen Städtetabelle. Die in die Masken eingegeben Daten werden anschließend in die Flugtabelle der MySQL-Datenbank geschrieben, auf die das Runtime-Modul, dessen Entwicklung in den folgenden Kapiteln beschrieben wird, zugreift, die Flugdaten ausliest, auswertet und darstellt.

6 Entwicklung des Runtime-Moduls

6.1 Modulplanung

Die Entwicklung des Runtime-Moduls von „echtwetter“ begann mit einer programminternen Ablaufplanung. Die in Abbildung 19 gezeigte Grundstruktur zur Erstellung einer Echtzeitapplikation wurde entsprechend den dynamischen Merkmalen, die die Anwendung aufweisen sollte, erweitert. Ausgehend von den minimalen Grundanforderungen einer jeden, mit Quest3D erstellten, Anwendung wurde die Struktur der finalen Applikation geplant. Minimale Anforderungen, um eine 3D-Szene in Quest3D darzustellen, sind eine Kamera, das anzuzeigende 3D-Objekt sowie eine Lichtquelle. Im finalen Projekt soll die Kamera einem berechneten Pfad folgen, der an Hand von in der HTML-Maske definierten und in die Datenbank gespeicherten Wegpunkten definiert wird. Das heißt, an dieser Stelle ist eine zusätzliche Logik notwendig um die Flugdaten aus der Datenbank auszulesen, auszuwerten und daraus einen Pfad zu generieren. Auch die Auswahl der in „echtwetter“ angezeigten 3D-Modelle von Landschaft, Städtesymbolen sowie Wahrzeichen erfolgt in der HTML-Maske und hat somit einen dynamischen Charakter. Die ebenfalls in der Datenbank abgelegten Werte zur Größe des zu ladenden Gebiets, der Anzahl der anzuzeigenden Städte und deren Temperaturen sowie der variablen Darstellung der Wahrzeichen erfordern auch zusätzliche Logik. Die einfache Grundstruktur der Echtzeitapplikation wird zudem durch die Verwendung von Partikel-Systemen erweitert, die einen nicht zu verachtenden Teil der „echtwetter“-Applikation ausmachen.

Einen großen Logikkomplex bildet auch die in der Applikation geforderte Variabilität von Lichtintensitäten, Sichtweitenregulierung, Niederschlagseigenschaften und Anzeigequalität, die unter dem Begriff „Einstellmöglichkeiten“ zusammengefasst werden (Abbildung 20).

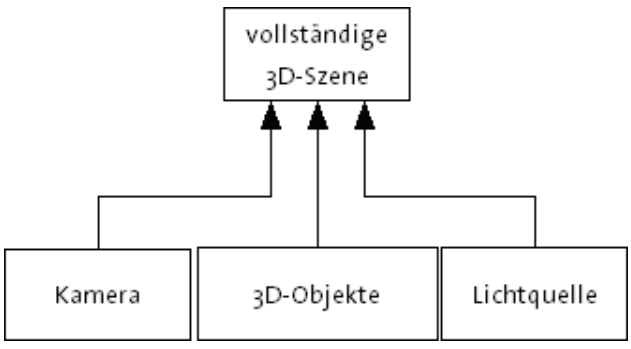


Abbildung 19: Grundstruktur einer 3D-Szene

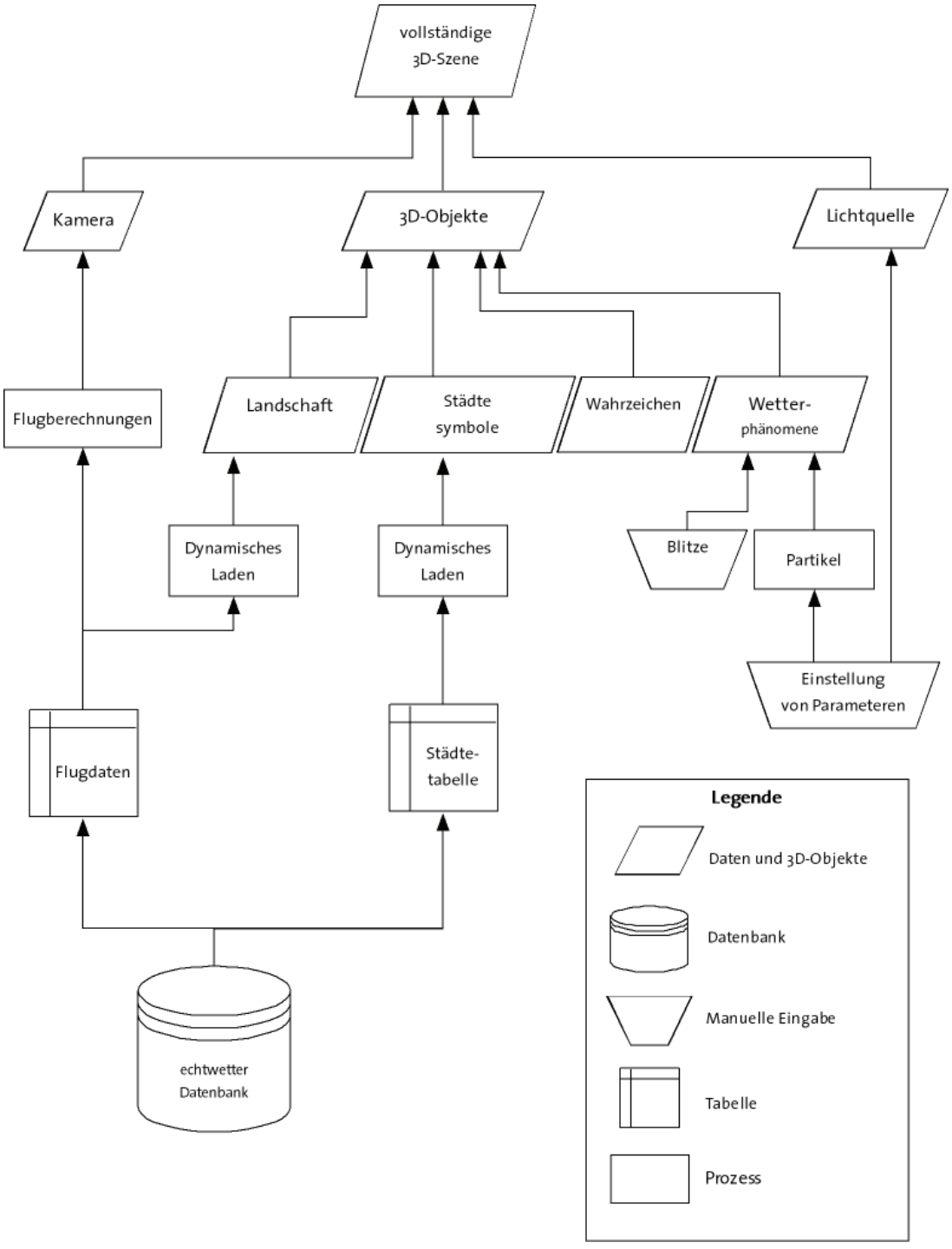


Abbildung 20: Erweiterung der Grundstruktur durch dynamische Elemente

Die finale Applikation wird in einzelnen Modulen erstellt, die getrennt voneinander entwickelt werden können und jeweils in eigenen Dateien abgelegt werden. Diese Dateien werden als Channel Groups bezeichnet. Channel Groups bestehen immer aus zwei zusammengehörenden Dateien, einer CGR-Datei (CGR = Channel GRoup) und einer IGR-Datei (IGR = Information GRoup). Die CGR-Datei enthält die eigentlichen

Logiken, die in Quest3D erstellt werden. Dazu gehören auch die eigentlichen 3D-Modelle sowie die Texturen und alles was sonst nötig ist um eine 3D-Szene darzustellen. In den IGR-Dateien befindet sich lediglich die Information über die Anordnung der Channels und Ordnerstrukturen der in den CGR-Dateien verwendeten Channels. Eine CGR-Datei ohne entsprechende IGR-Datei ist so gut wie nicht verwendbar. Die einzelnen Module, die jeweils in eigene Channel Groups ausgelagert werden, werden für die finale Applikation unter Verwendung von Public Channels und entsprechenden Public Channel Callers zu der dann funktionierenden Anwendung verknüpft. Bei Public Channels, kenntlich gemacht durch die Farbe rot, handelt es sich um Channels, deren Wert oder Funktionalität auch außerhalb der aktuellen Channel Group zur Verfügung gestellt wird. Der Zugriff auf einen Public Channel erfolgt, indem an der Stelle, an dem er zum Einsatz kommen soll, ein vom Typ her gleicher Channel angehängt und dann zu einem Public Channel Caller erklärt wird. Dieser kann so dateiübergreifende Verknüpfungen herstellen, vorausgesetzt der aufzurufende Channel ist an der angegebenen Stelle zu finden. Public Channel Caller werden zum schnelleren Erkennen in komplexen Logiken automatisch dunkelblau eingefärbt. Diese Art der Verknüpfung kann mit globalen Variabeln oder Funktionen verglichen werden, wie sie in jeder gängigen Programmiersprache zum Einsatz kommen (Abbildung 21).

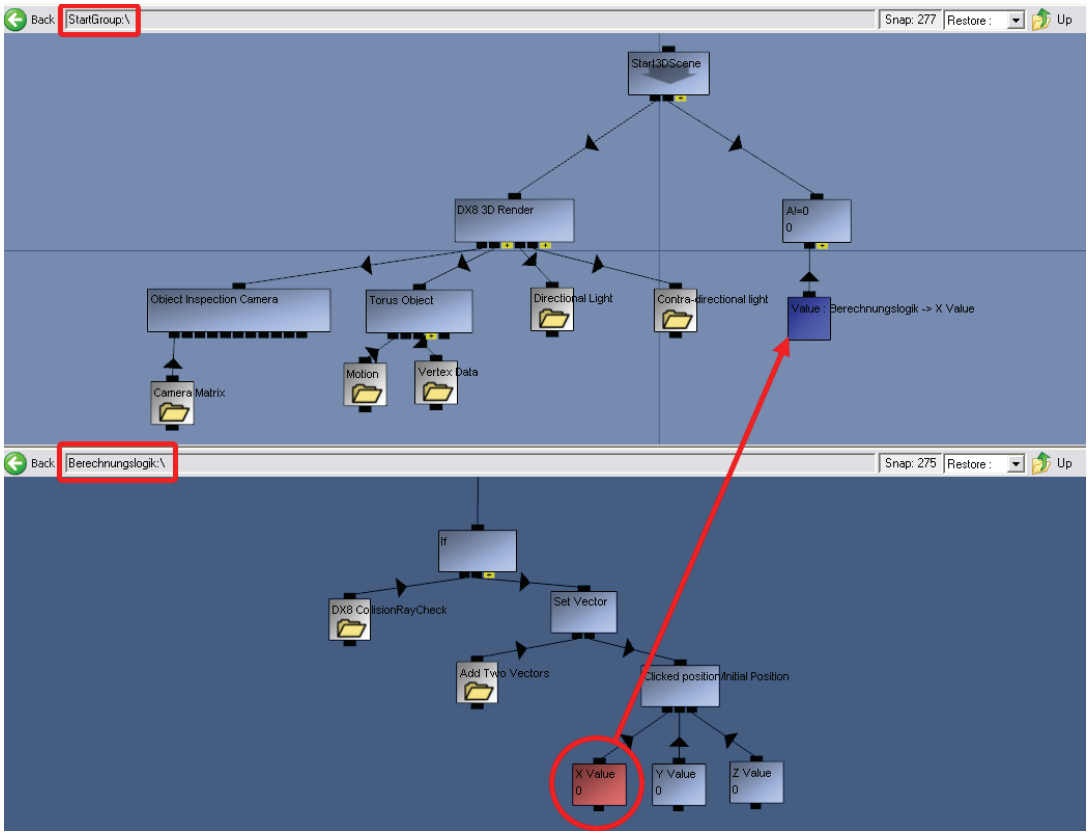


Abbildung 21: Public Channel Caller

Im folgenden Abschnitt wird beispielhaft die zur Flugberechnung und Kameraanimation implementierte Logik detailliert beschrieben um die Arbeitsweise mit Quest3D kennen zu lernen. Die übrigen Module werden lediglich vom Prinzip her erläutert; die exakte Umsetzung beziehungsweise die Abbildungen der Channel-Konstrukte werden nur bei wichtigen Logiken näher erklärt oder befinden sich im Anhang. Das Zusammenfügen der einzelnen Module zur dann vollständigen Echtzeitanwendung wird im Kapitel 6.10 erläutert.

6.2 Flugberechnungen

6.2.1 Flugberechnung auf Basis angegebener Routenpunkte

Einer der grundlegenden Bestandteile einer jeden 3D-Szene ist die Kamera. Ohne sie wäre die Darstellung von Objekten nicht möglich. Für die „echtwetter“-Applikation kommt eine animierte Kamera zum Einsatz, die mit konstanter Geschwindigkeit eine im Authoring-Modul bestimmte Route entlang fährt.

Im folgenden Abschnitt wird die Logik zur Berechnung des animierten Kamerapfades erläutert. Dabei wird zuerst der Verbindungsaufbau zur „echtwetter“-Datenbank genauer beschrieben. Im Anschluss folgen die, mit den aus der Datenbank ausgelesenen Werten, durchgeführten Operationen zur Distanzenberechnung und der Berechnung der zeitlichen Positionierung der zugehörigen Keyframes. Der letzte Teil der Logik befasst sich mit dem Setzen der berechneten Keyframes zur Animation eines Hilfsobjekts.

Die gesamten Kalkulationen zur Erstellung des Flugpfads werden beim Start der Applikation nur ein einziges Mal durchgeführt. Die Verzögerungen, die dabei durch Datenbankanfragen entstehen, können, da sie nur einmal auftreten, in Kauf genommen werden. Die für die Berechnung der Flugrouten nötigen Daten wie Flugdauer, die Flughöhe und die Positionen der einzelnen Wegpunkte werden im Authoring-Modul angegebenen und mittels PHP in die MySQL-Datenbank eingetragen. Der erste nötige Schritt ist also die Verbindung zur MySQL-Datenbank und das Auslesen der entsprechenden Werte. Hierzu wird einmalig mittels eines entsprechenden Datenbank-Channels (DB Source) eine Verbindung zu der „echtwetter“-MySQL-Datenbank geöffnet. Der DB Source Channel arbeitet an dieser Stelle mit einem MySQL-Treiber-Channel zusammen. Es ist mit entsprechenden Treiber-Channels aber auch möglich sich zu anderen Datenbanken wie Oracle oder Access zu verbinden. Angehängt an den MySQL-Treiber-Channel, namens DB DriverMySQL, befinden sich die entsprechenden Parameter um die Verbindung zu öffnen. Übergeben wird zunächst der Hostname, in diesem Fall die Loopback-Adresse 127.0.0.1, da die MySQL-Datenbank auf dem gleichen System läuft wie die Echtzeitapplikation. Des Weiteren wird die Portnummer überge-

ben, die sich im Normalfall nach der betriebenen Datenbank richtet und bei MySQL standardmäßig auf 3306 eingestellt wird. Eine erfolgreiche Verbindung benötigt zudem noch den Namen der Datenbank, zu der verbunden werden soll, sowie einen Benutzernamen und ein zugehöriges Passwort. Die Verbindung zu Datenbank wird nur ein einziges Mal direkt nach Start der Applikation geöffnet; dafür sorgt der One Time Channel am oberen Ende der Substruktur. Durch diesen geöffneten Kanal können dann beliebig viele Anfragen an die Datenbank gestellt werden (Abbildung 22).

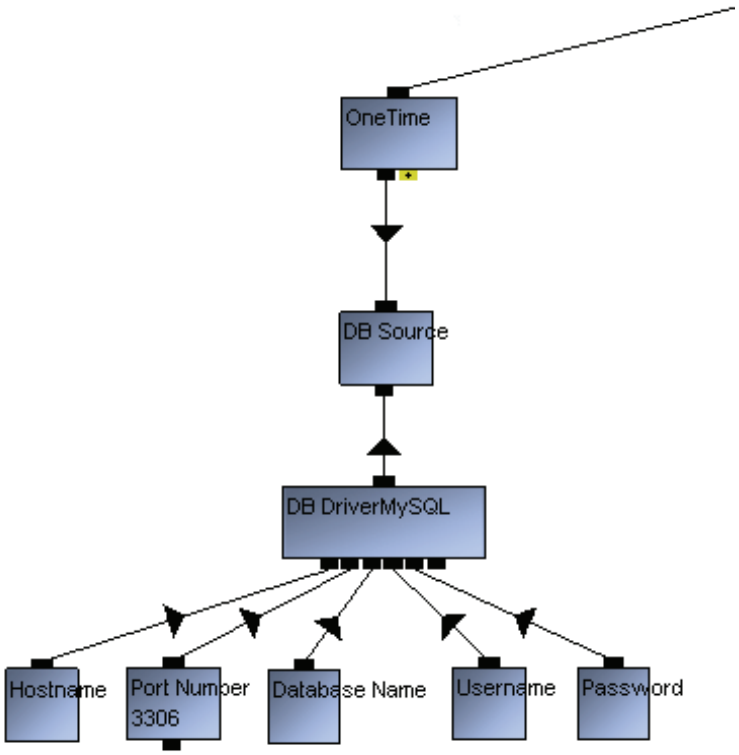


Abbildung 22: Verbindung zur MySQL-Datenbank

Wurde die Verbindung zur Datenbank erfolgreich hergestellt, kann die erste Anfrage gestartet werden. Damit die Anfrage nicht zu früh gestellt wird, wird über einen DB Info Channel der Status der Verbindung abgefragt und erst bei positivem Rückgabewert die Anfrage ausgeführt (Abbildung 23). Die Anfrage „Get Waypoints“ an die Datenbank, genauer gesagt an die Flugtabelle, liest die vollständigen Flugdaten des aktiven Flugs aus (Abbildung 24).

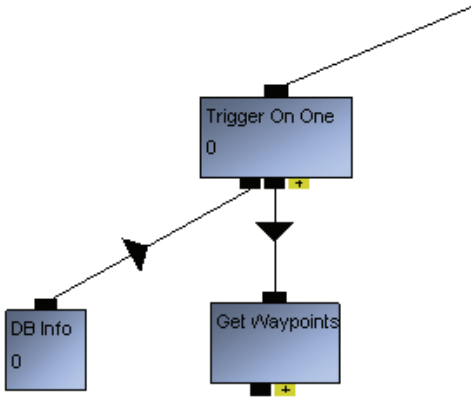


Abbildung 23: Statusabhängige Datenbankanfrage

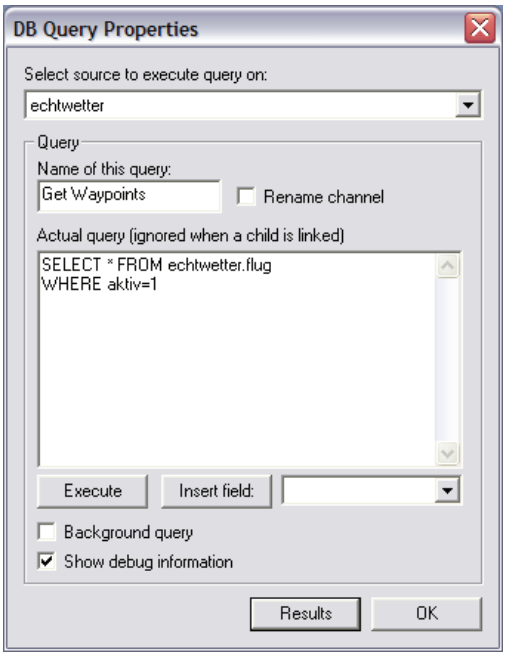


Abbildung 24: MySQL Select Befehl

Die Wegpunkte beziehungsweise die Städte, die eine Flugroute definieren, sind in der Flugdatenbank nur mit ihrem Index gespeichert. Im nächsten Schritt werden aus der Städtetabelle an Hand der Indizes die zu den Städten gehörenden x- und y-Positionen und zur besseren Übersicht und Kontrolle auch der Name der jeweiligen Stadt geladen. Die Anfragen an die Städtedatenbank müssen dabei für jeden der maximal fünf Wegpunkte separat erfolgen, da eine gemeinsame Abfrage der Positionen aller Städte die Reihenfolge der Wegpunkte durcheinander bringen würde. Die Reihenfolge ist aber in jedem Fall einzuhalten, da es durchaus einen Unterschied macht, ob man von Kassel, über Gießen nach Frankfurt fliegt oder aber anders herum. Die Rückgabe einer Datenbankanfrage lässt sich zwar sortieren, jedoch nur nach den Spalten der Ergebnisse, nicht nach den Parametern der Anfrage. Daher ist eine gesonderte Anfrage pro Wegpunkt die einfachste Lösung. Bevor die Anfragen ausgeführt werden, muss allerdings der Status der vorangegangenen Anfrage einen positiven Rückgabewert melden (Abbildung 25).

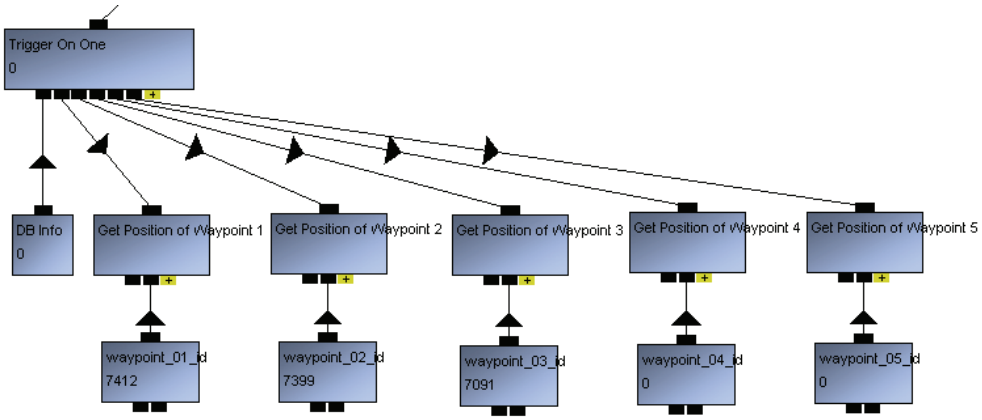


Abbildung 25: Auslesen der Wegpunktpositionen

Die fünf Anfragen liefern für gesetzte Wegpunkte jeweils einen x-Wert, einen y-Wert und den zugehörigen Städtenamen zurück. Für die noch folgenden Berechnungen der Gesamtdistanz sowie der Distanzen zwischen den einzelnen Wegstrecken, empfiehlt es sich die Positionswerte in einen Vektor umzuwandeln. Die Positionsvektoren werden dann zum einfacheren Umgang in ein entsprechendes Vektorarray gefüllt. Auch dieser Vorgang kann erst ausgeführt werden, nachdem alle fünf Positionsanfragen an die Datenbank erfolgreich durchgeführt wurden (Abbildung 26).

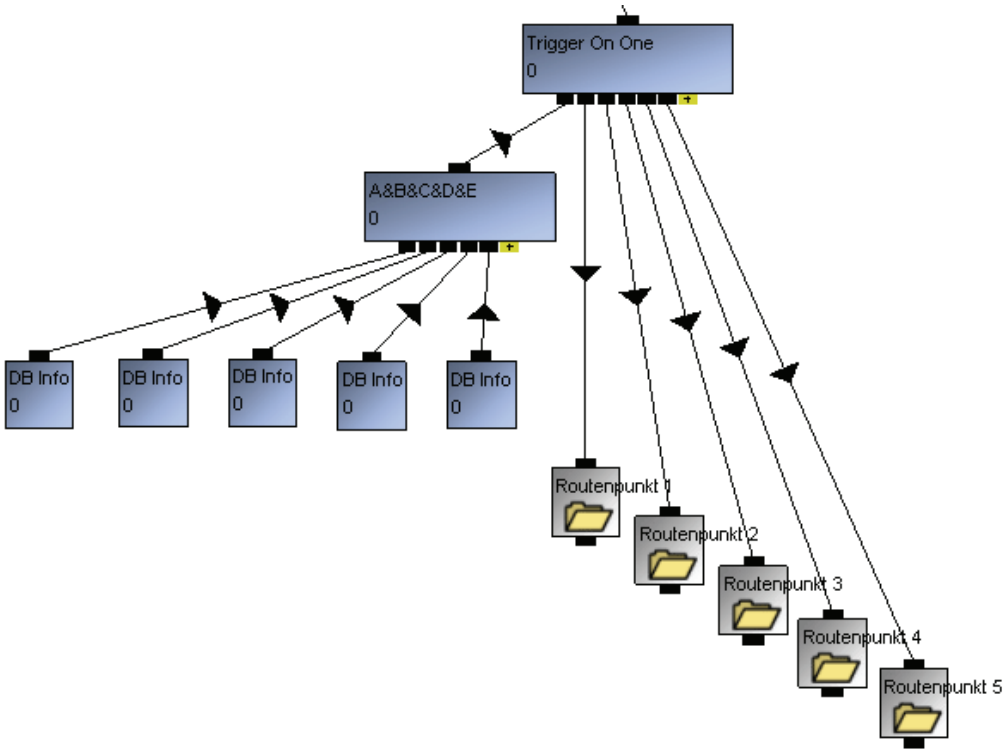


Abbildung 26: Auslösen der Konvertierung in Vektoren

Die fünf Zustände der einzelnen Anfragen werden über einen Expression Value Channel (A&B&C&D&E) durch ein logisches UND miteinander verknüpft. Erst wenn alle fünf Anfragen eins zurückgeben, wird auch der Expression Value Channel eins und die Logik wird fortgesetzt. Ein Expression Value Channel ist eine besondere Art von Channel, ein Multitalent, das oft und in unterschiedlichsten Logiken zum Einsatz kommt. In diesem Channel können mathematische Formeln stehen, Vergleiche zwischen Variablen angestellt, Zufallszahlen generiert oder einfach nur Negationen durchgeführt werden. Angehängte Channels können in die, in das Eigenschaftsfeld des Expression Value Channels, eingebbaren Formeln mit einbezogen werden. Der Buchstabe A steht dabei für den an erster Stelle angehängten Channel, der Buchstabe B für den zweiten Channel und so weiter. Es können dabei beliebig viele Channels angehängt werden (Abbildung 27).

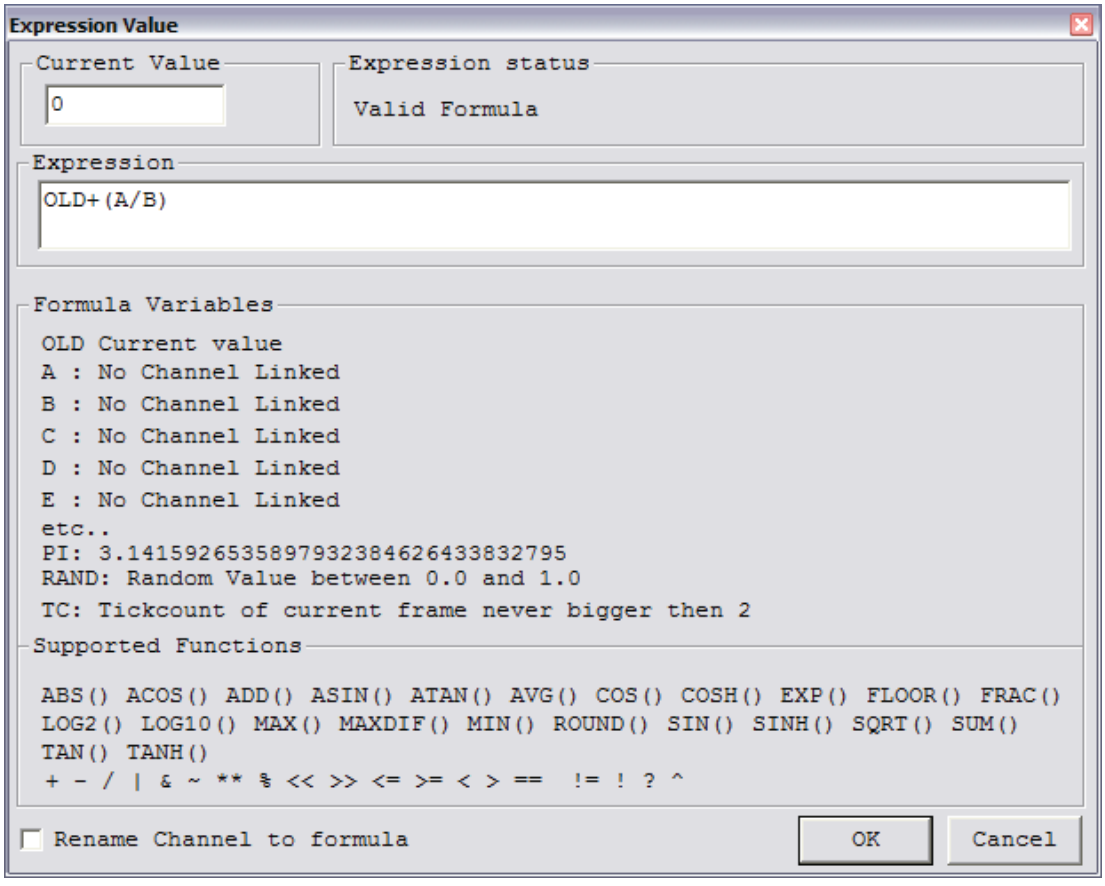


Abbildung 27: Eigenschaften des Expression Value Channel

Bei den in Abbildung 26 an den Trigger Channel angehängten Objekten handelt es sich um Unterordner. Eine aus Channels erbaute Logik lässt sich in diesen Unterordnern zusammenfassen und so besser und übersichtlicher strukturieren.

In den Unterordnern wird aus der x- und y-Position und der in der HTML-Maske angegebenen Flughöhe ein Vektor generiert, der anschließend in ein Vektor-Array abgelegt wird. Da Quest3D im Gegensatz zu 3ds max mit einem Koordinatensystem arbeitet, bei dem y die Höhe bezeichnet, wird die y-Position, die in 3ds max eine der beiden Achsen auf der Landschaftsebene ist, an dieser Stelle in die z-Position eingetragen. Der Name der Stadt, der ebenfalls in der Anfrage an die Datenbank ermittelt wurde, wird in ein Textarray gefüllt. Der Value Channel, der an beiden Arrays angehängt ist, gibt die Position im Array an, in die dann mittels Set Vector Channel beziehungsweise mittels Set Text Channel der Positionsvektor sowie der Städtename eingetragen wird (Abbildung 28). Diese Logik wird für alle fünf Wegpunkte separat durchgeführt.

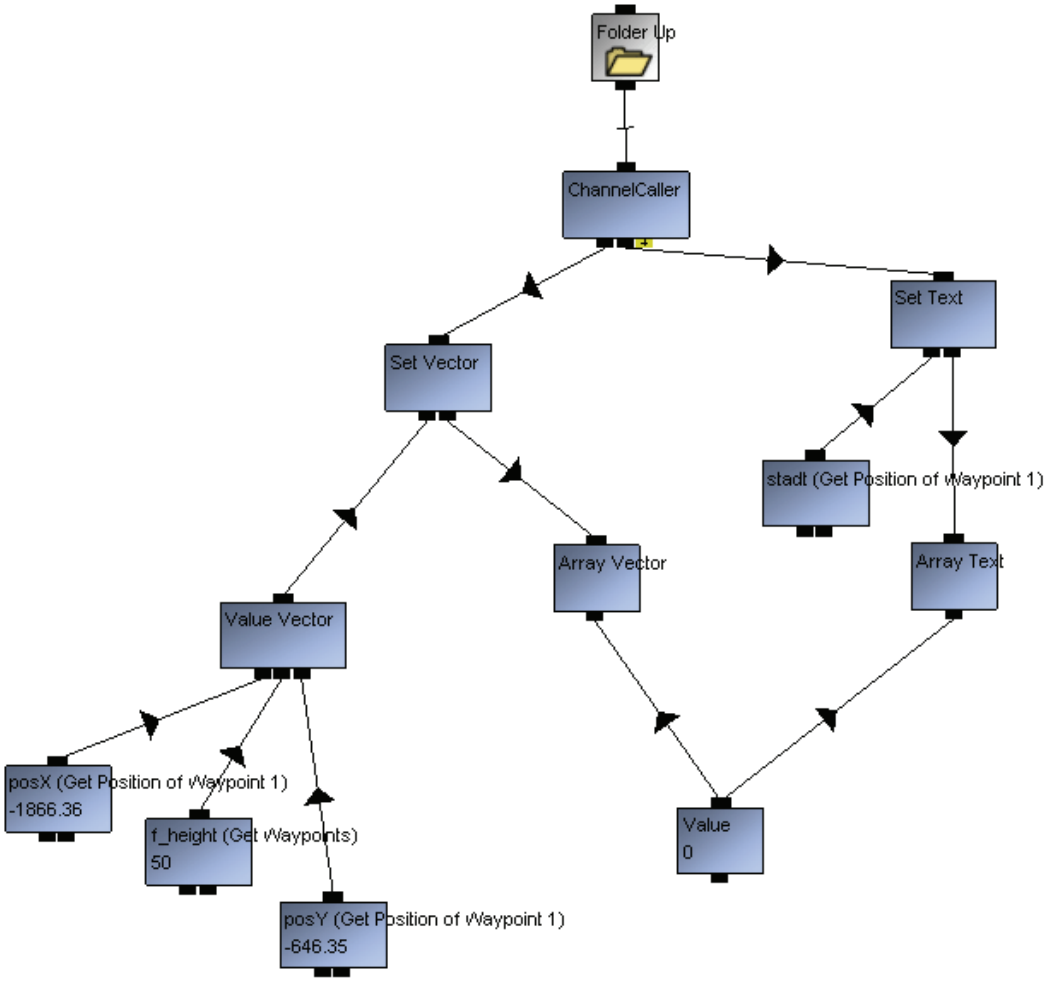


Abbildung 28: Umwandeln der Positionswerte in Vektoren

Im nächsten Schritt wird in einer Schleife die jeweilige Distanz zwischen den einzelnen Wegpunkten, die jetzt als Vektoren vorliegen, errechnet und die Ergebnisse in ein entsprechendes Wertearray gefüllt. Außerdem wird zusätzlich die Gesamtdistanz be-

rechnet. Zur einfacheren Steuerung dieser Schleife wird ein kleines LUA-Skript verwendet.

Bei LUA handelt es sich um eine Skriptsprache, die vom Funktionsumfang und Syntax Javascript sehr ähnlich ist. In Quest3D ist LUA, allerdings mit eingeschränktem Funktionsumfang, erst kürzlich implementiert worden, um komplexe Logiken, für die sehr aufwändige Channel-Strukturen nötig sind, effektiver in wenigen Zeilen Code zu realisieren. Zurzeit sind lediglich die Basisfunktionen sowie die Funktionen für mathematische Berechnungen und die Textoperatoren verfügbar.

Das zur Berechnung der zwischen den Routenpunkten liegenden Distanzen verwendete LUA-Skript beschränkt sich auf das viermalige Aufrufen der an den LUA Skript Channel angehängten Channels in der richtigen Reihenfolge und das Setzen der „ready“-Variablen nach erfolgter Berechnung (Abbildung 29).

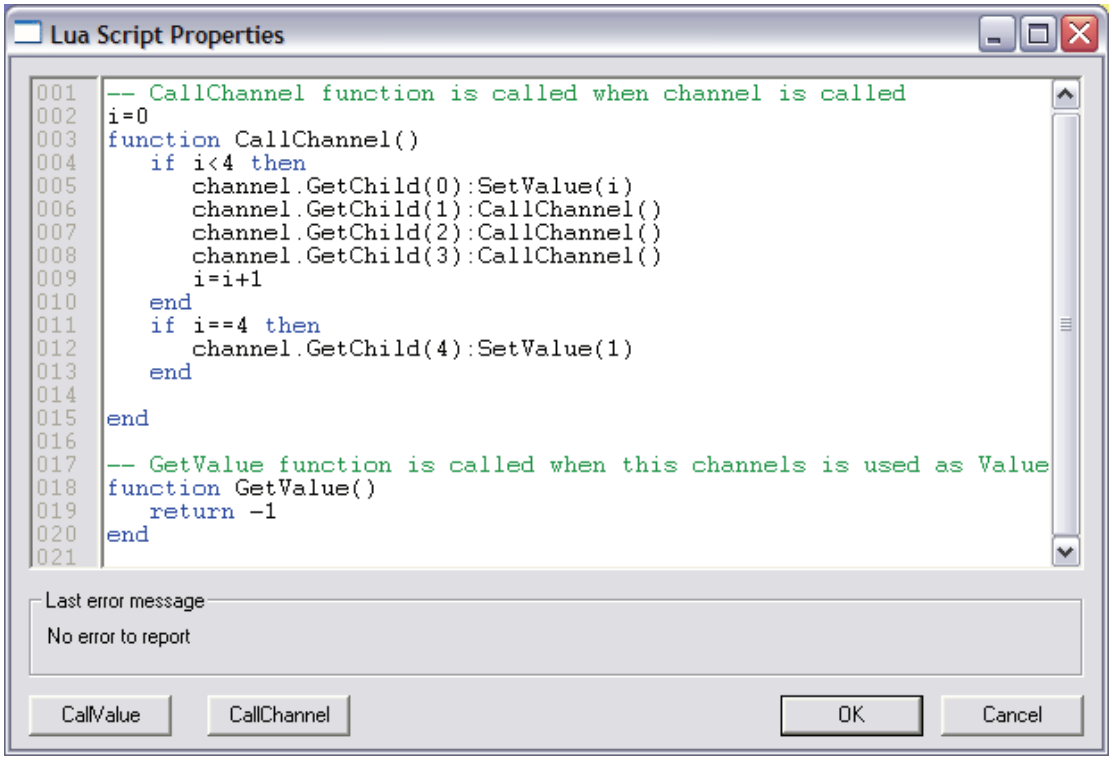


Abbildung 29: LUA-Skript zur Steuerung einer Schleife

Das LUA-Skript setzt zunächst den Zähler i, in der Logik als Counter bezeichnet, der die Positionen im Vektor-Array bestimmt. Die x-Positionen zweier im Array aufeinander folgenden Vektoren werden über einen Expression Value Channel überprüft. So wird gewährleistet, dass auch wirklich zwei Positionen vorhanden sind, zwischen denen sich eine Distanz berechnen lässt. Sind beide Werte ungleich null, d.h. es sind zwei Wegpunkte vorhanden, gibt der Expression Value Channel eins zurück. Der Rückga-

bewert des Expression Value Channels wählt über einen Channel Switch aus, welcher Wert in das bereitstehende Werte-Array eingefüllt wird. Hat der Expression Value den Wert null, ist also unwahr, nimmt der Channel Switch den Wert des an zweiter Position angehängten Value Channels null an. Ist die Bedingung des Expression Value Channels erfüllt, wird dieser eins und der Channel Switch nimmt den Wert der im Get Vector Distance Channel berechneten Distanz an. Wenn entschieden ist, welcher Wert in das Werte-Array übergeben werden soll, wird über einen vom LUA Skript Channel aufgerufenen Set Value Channel dieser Wert in das Distance Vector Array eingetragen. Anschließend werden über einen weiteren Expression Value die jeweiligen Distanzen zur Gesamtdistanz aufaddiert und in die Variable „f_totalLength“ gefüllt. Sind die Distanzen zwischen den Wegpunkten errechnet und in das Array gefüllt sowie die Gesamtflugstrecke berechnet, setzt der LUA Channel die „ready“-Variable auf eins und der nächste Schritt der Logik wird angestoßen (Abbildung 30 & Abbildung 31).

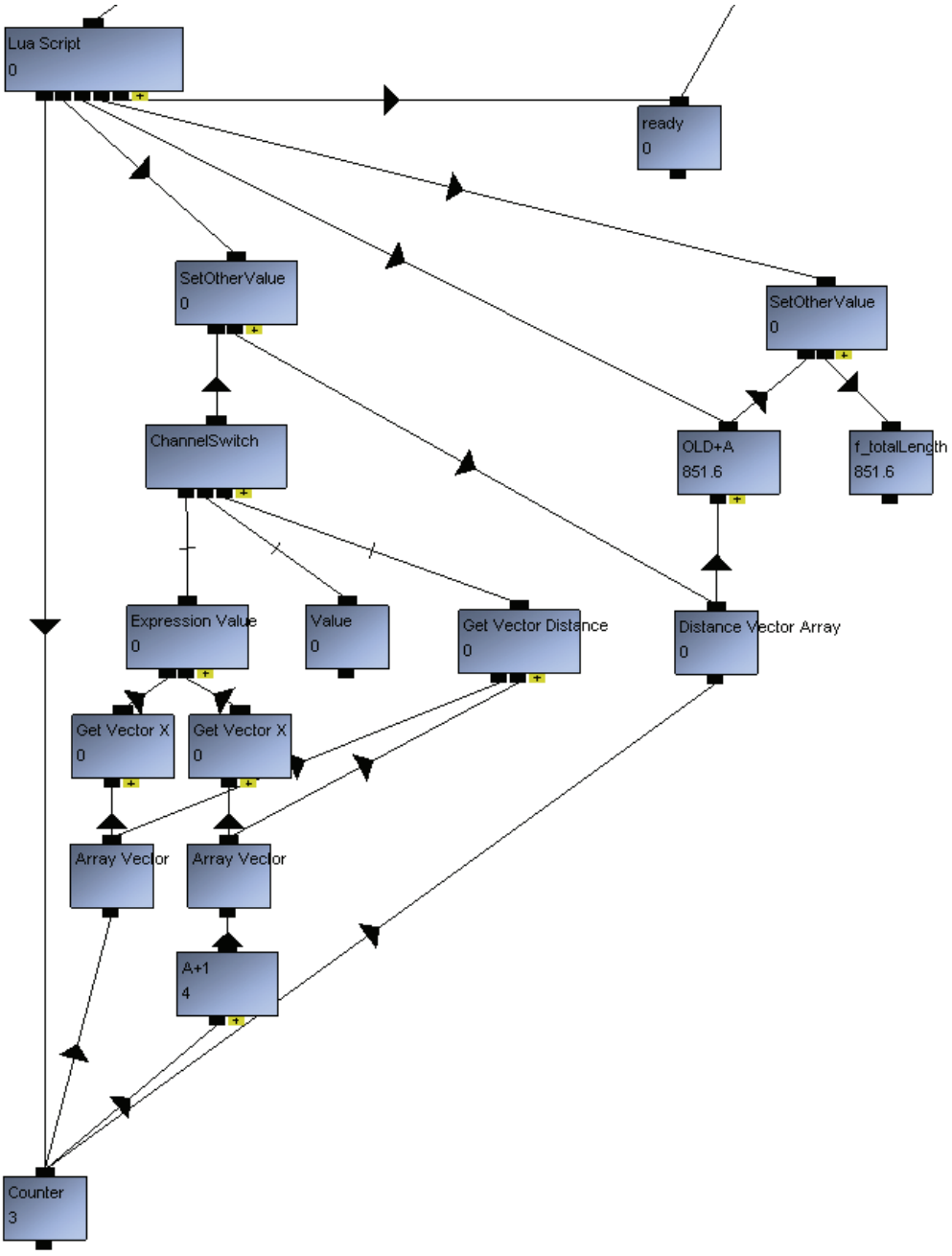


Abbildung 30: Berechnen der Distanzen zwischen den Wegpunkten

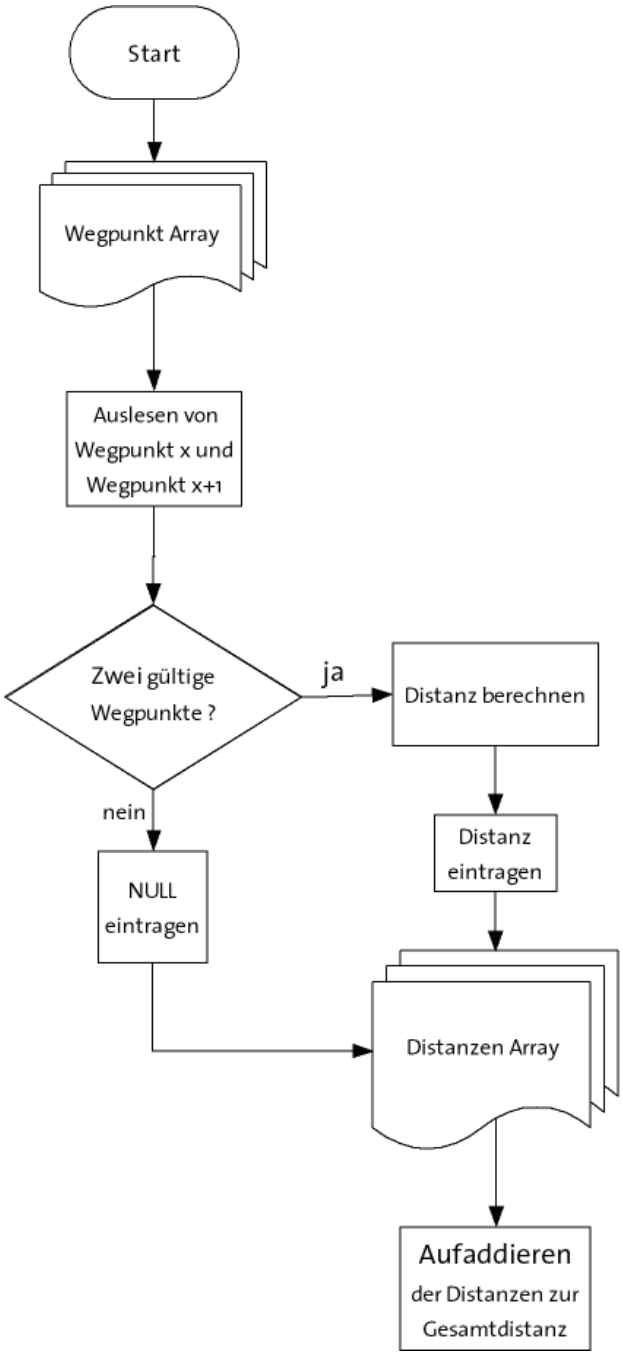


Abbildung 31: Flussdiagramm zur Berechnung der Distanzen zwischen den Wegpunkten

In einer weiteren Logiksubstruktur werden aus der zu Anfang aus der Datenbank ausgelesenen Gesamtdauer des Flugs und den Distanzen zwischen den Wegpunkten die Anzahl der Frames berechnet, die zwischen den einzelnen Wegpunkten liegen und einen Flug mit konstanter Geschwindigkeit ermöglichen. Wie zuvor, wird wieder ein LUA Skript Channel zur Steuerung der Schleife verwendet. Das LUA-Skript setzt zunächst den Zähler, der die Position im Distance Vector Array bestimmt. Im Anschluss wird in einem Expression Value Channel der ausgewählte Distanzvektor mit der Ge-

samtdauer des Flugs multipliziert und das Produkt durch die im Schritt zuvor ermittelte Gesamtlänge des Flugs dividiert (A*B/C). Der Wert der Gesamtlänge wird als so genannter Shortcut, vergleichbar mit einer Verknüpfung bei Windows, an den Expression Value Channel angebunden. Die Ergebnisse dieser Berechnung entsprechen der Anzahl der Frames, nach denen ein Schlüsselbild, im Folgenden auch Keyframe genannt, gesetzt werden muss. Die Einzelergebnisse werden aufaddiert und nacheinander in das Keyframearray gefüllt. Nachdem für die maximal vier vorhandenen Distanzen die Keyframes errechnet wurden, setzt der LUA Skript Channel die „ready"-Variable auf eins um die Logik fortzusetzen (Abbildung 32 & Abbildung 33).

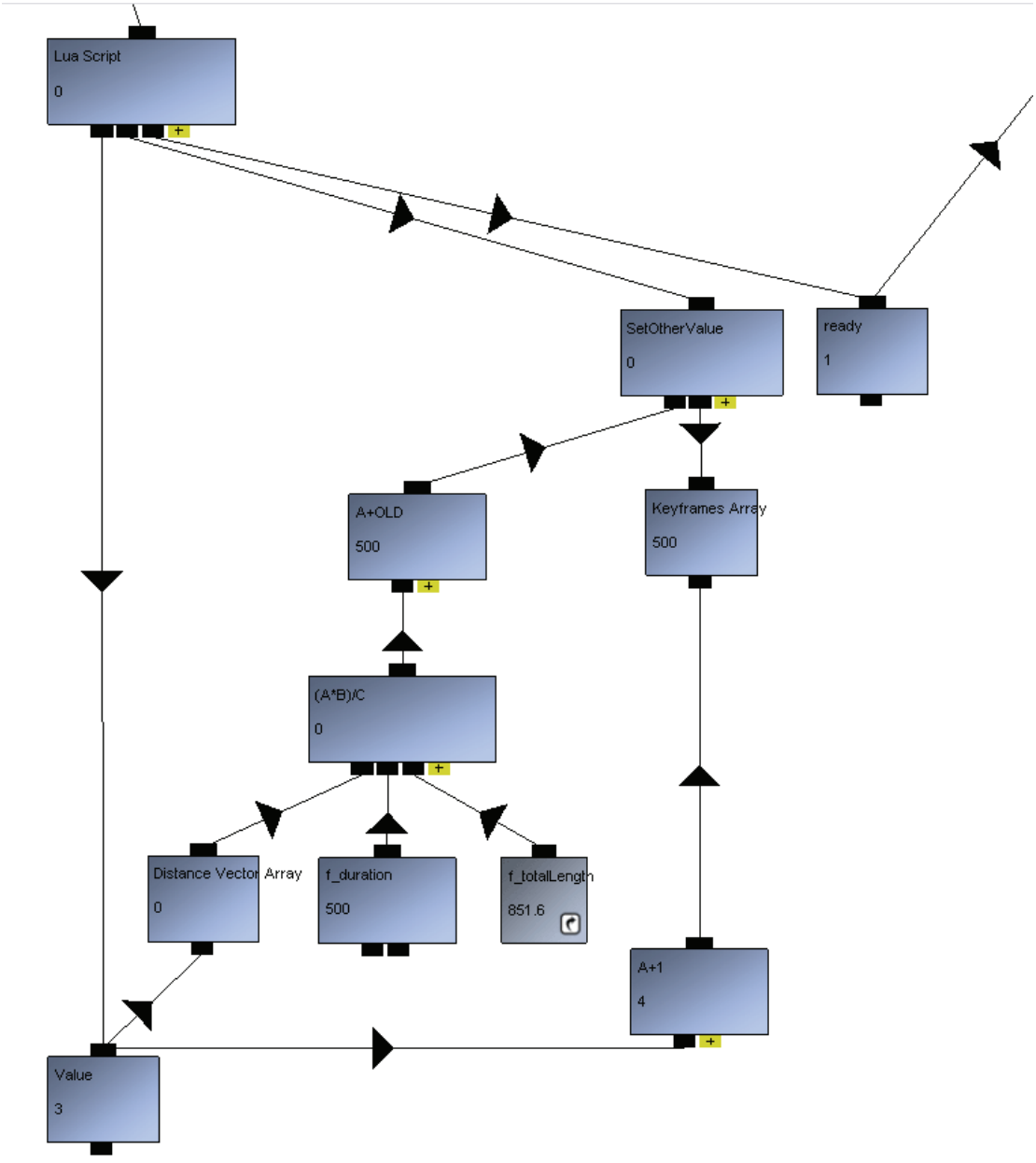


Abbildung 32: Berechnung der Keyframes

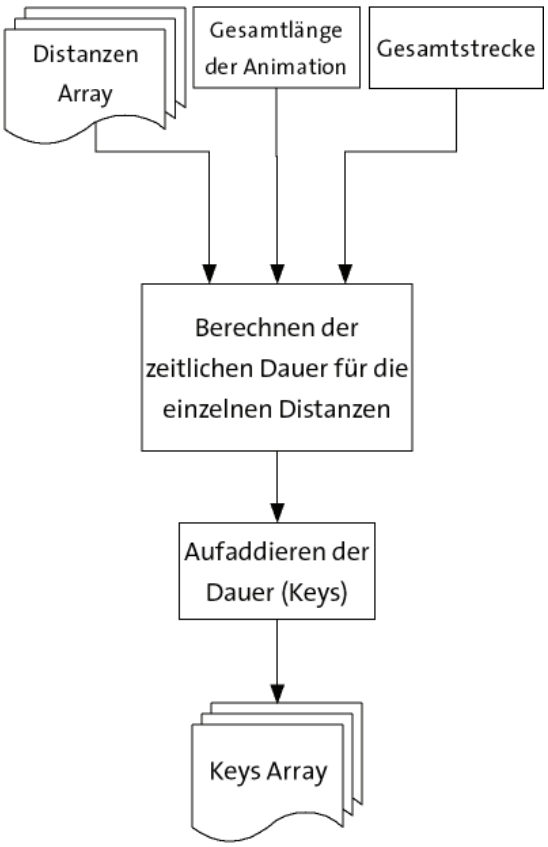


Abbildung 33: Flussdiagramm zur Berechnung der Keyframes

Der nächste Schritt in der Logik beinhaltet das Setzen der Keyframes an den errechneten Positionen. Auch hier wird wieder ein LUA-Skript zur Steuerung der Schleife verwendet, welche die Keyframes nacheinander setzt. Der Wert Counter steuert wieder die aktuelle Position der in dieser Logiksubstruktur verwendeten Arrays. Der LUA Skript Channel überprüft als nächstes den x-Wert des aktuell ausgewählten Wegpunkts im Vektorarray (Positionen). Ist dieser Wert ungleich null, das heißt es ist ein Wegpunkt vorhanden, ruft der LUA Skript Channel den an dritter Position angehängten Channel Caller auf. Der Channel Caller ruft seinerseits die angehängten Channels von links ausgehend nacheinander auf. Dabei wird zunächst der für die Animation der Kamera zuständige Zeitgeber (TimerValue), der hier als Public Channel Caller angebunden ist, auf den im Abschnitt zuvor berechneten Frame gesetzt. Anschließend wird die x-, y- und z-Position des aktuellen Wegpunkts in die korrespondierenden Envelope Channels eines Hilfsobjekts (NULL Object) eingefügt, die ebenfalls als Public Channel Callers angebunden sind.

Bei Envelope Channels handelt es sich um Graphen, die für jeden Eingangswert einen Ausgangswert definieren. Sie werden in Quest3D vor allem zur Kontrolle von Animationen verwendet. In Fall von Abbildung 34 ist auf der x-Achse die Zeit des links unten angegebenen Zeitgebers aufgetragen und auf der y-Achse die x-Position der Kamera.

Die einzelnen gesetzten Keyframes, dargestellt durch kleine Kreuze, welche die Animation bilden, sind über eine interpolierte Kurve miteinander verbunden. Die Interpolationsart wird links im Drop-Down-Menü „Interpolation“ eingestellt. Neben der hier verwendeten TCB-Interpolation (T = tension = Spannung; C = continuity = Stetigkeit; B = bias = Tendenz), die eine sanfte Kurve ergibt, gibt es noch die Step-Interpolation, bei der eine Rechteckkurve entsteht, und die lineare Interpolation, welche die Keys über gerade Linien miteinander verbindet.

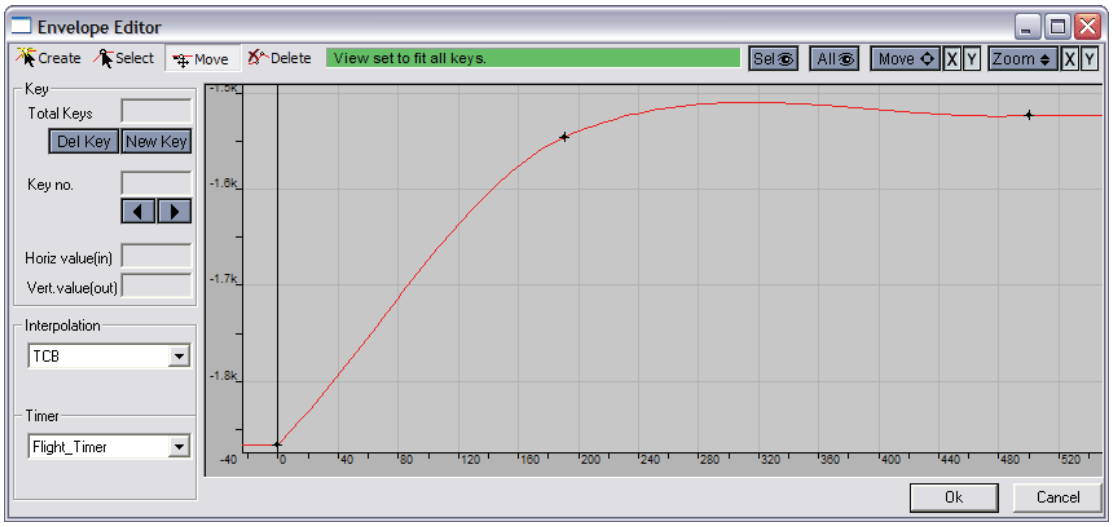


Abbildung 34: Envelope Channel x-Position

Die Keyframes für die Positionsanimation der Kamera beziehungsweise des Hilfsobjekts in der „echtwetter“-Applikation werden durch die beschriebene Logik automatisch gesetzt. Es ist aber auch möglich zu anderen Zwecken die Keyframes direkt in dem Envelope Channel zu setzen oder zu verändern.

Sind alle Keyframes für die Positionsanimation an den entsprechenden Zeitpunkten gesetzt, wird die „ready“-Variable auf eins gesetzt (Abbildung 35).

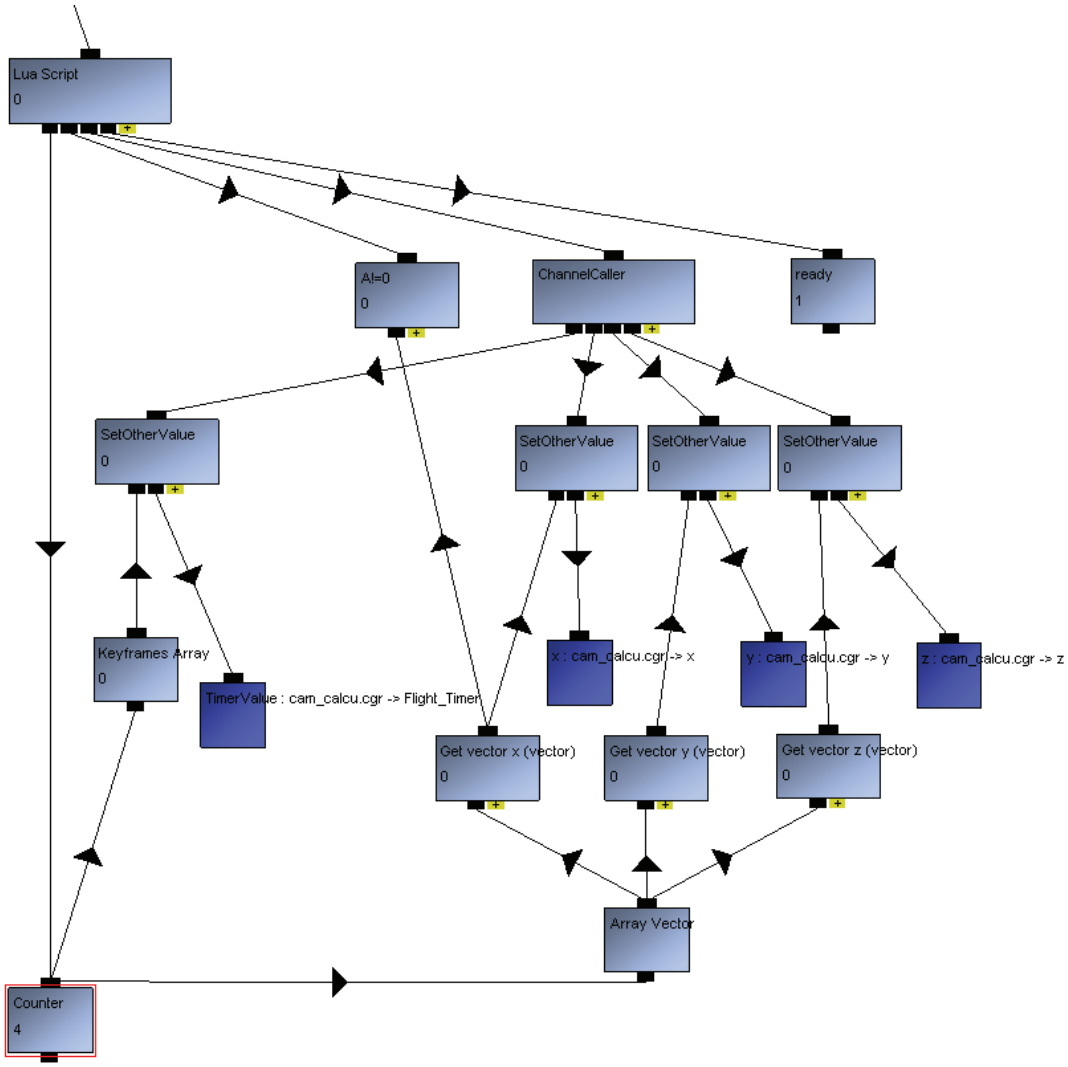


Abbildung 35: Setzen der Keyframes

Das Setzen des Zeitgebers auf den jeweiligen Frame und das anschließende Setzen der Positionswerte fügt den entsprechenden Envelope Channels des Hilfsobjekts jeweils einen neuen Keyframe zu dem angegebenen Zeitpunkt mit dem angegebenen Wert hinzu.

Der Positionsvektor, der gemeinsam mit dem Rotationsvektor und dem Größenvektor die Transformationsmatrix des Hilfsobjekts bildet, wird mit einem Damping Channel an die Matrix gebunden. Der Damping Channel „dämpft“ die Animation des Positionsvektors um den an zweiter Stelle angehängten Wert. So wird der Ablauf der Animation vor allem am Anfang und am Ende weicher (Abbildung 36).

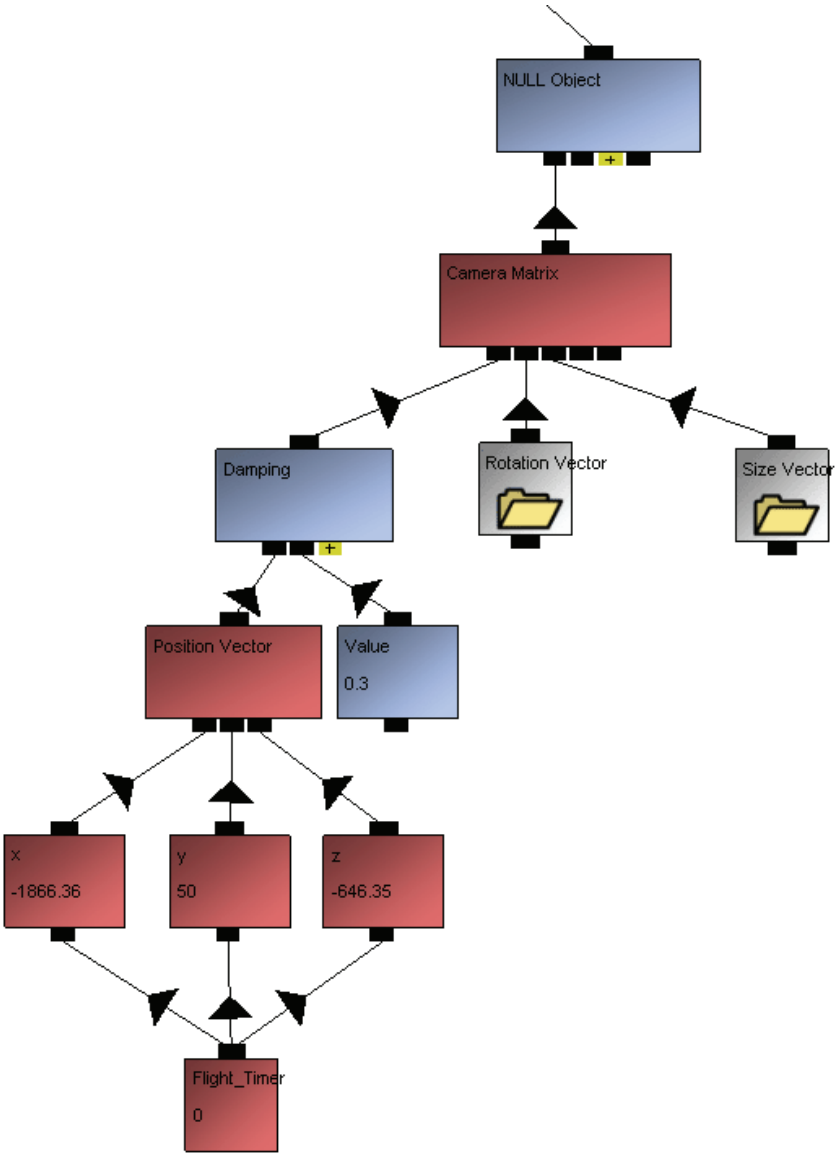


Abbildung 36: Hilfsobjekt

Das Ergebnis der bis zu diesem Punkt durchgeführten Kalkulationen ist ein Hilfsobjekt (NULL Object), das sich genau auf dem im Authoring-Moduls bestimmten Pfad mit konstanter Geschwindigkeit entlang bewegt.

6.2.2 Automatische Blickrichtung versus Freie Blickrichtung

Der kommende Abschnitt behandelt zwei unterschiedliche Kameramodelle, die beide, ob der offenen Leistungsbeschreibung implementiert wurden, von denen sich zuletzt aber nur eines durchsetzte. Die zwei unterschiedlichen Methoden in Bezug auf die Blickrichtung der Kamera werden zunächst vom Prinzip her erläutert. Im Anschluss

wird die technische Umsetzung der zum Einsatz kommenden Methode im Detail beschrieben.

Die zuerst implementierte Methode war die automatische Ausrichtung der Blickrichtung. Hierfür wurden die im vorangegangenen Kapitel durchgeführten Berechnungen zum Setzen von Keyframes nicht auf ein Hilfsobjekt angewandt, sondern direkt auf die Kamera. Ergebnis war eine dem Pfad folgende Kamera mit stetiger Ausrichtung in eine definierte Richtung. Für die automatische Ausrichtung der Blickrichtung der Kamera wurde ein Hilfsobjekt verwendet, das auf dem gleichen berechneten Pfad mit einigen Frames Vorsprung entlang fuhr. Die Kamera wurde mit dem Hilfsobjekt als Lookat-Constraint verknüpft. Auf diese Weise richtete sich die Rotation der Kamera immer nach dem vorausfahrenden Hilfsobjekt. Der zeitliche Abstand zwischen der Kamera und dem zur Ausrichtung vorausfahrenden Hilfsobjekt konnte über eine Variable leicht geändert werden (Abbildung 37).

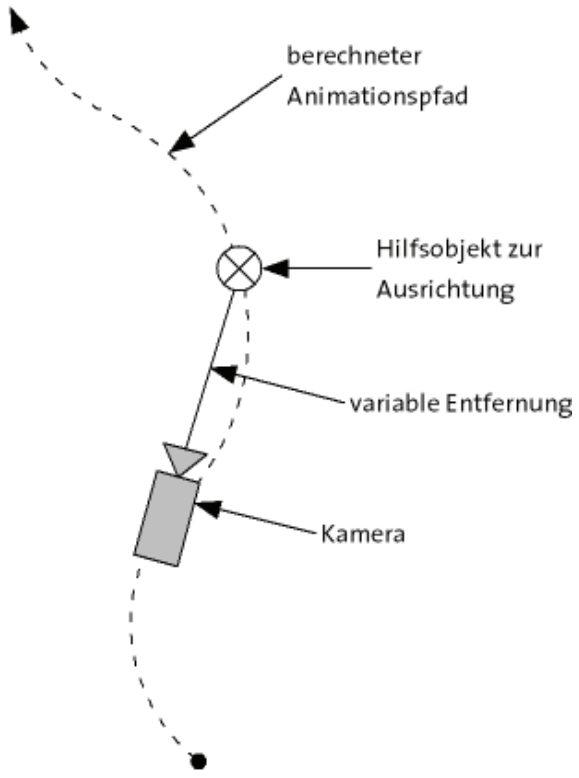


Abbildung 37: Automatische Kameraausrichtung

Die zuvor beschriebene Methode zur Ausrichtung der Kamera wurde bei einer der zahlreichen Zwischenabnahmen von verantwortlichen Grafikern des hr als zu inflexibel erachtet. Die Blickrichtung der Kamera sollte frei einstellbar und über den Verlauf der Pfadanimation auch änderbar sein. Dies führte zur Implementierung der zweiten

Methode, einer über die Maus in ihrer Blickrichtung und ihrem Betrachtungswinkel einstellbaren Kamera, die als Center-of-Interest (CoI) das auf dem Pfad animierte Hilfsobjekt verwendet. Als Basis für diese Kamera wurde ein in Quest3D standardmäßig vorhandener Kameratyp verwendet, die so genannte Orbit Kamera. Sie ermöglicht die freie Rotation um einen festen Bezugspunkt sowie die Änderung des Betrachtungswinkels (Abbildung 38).

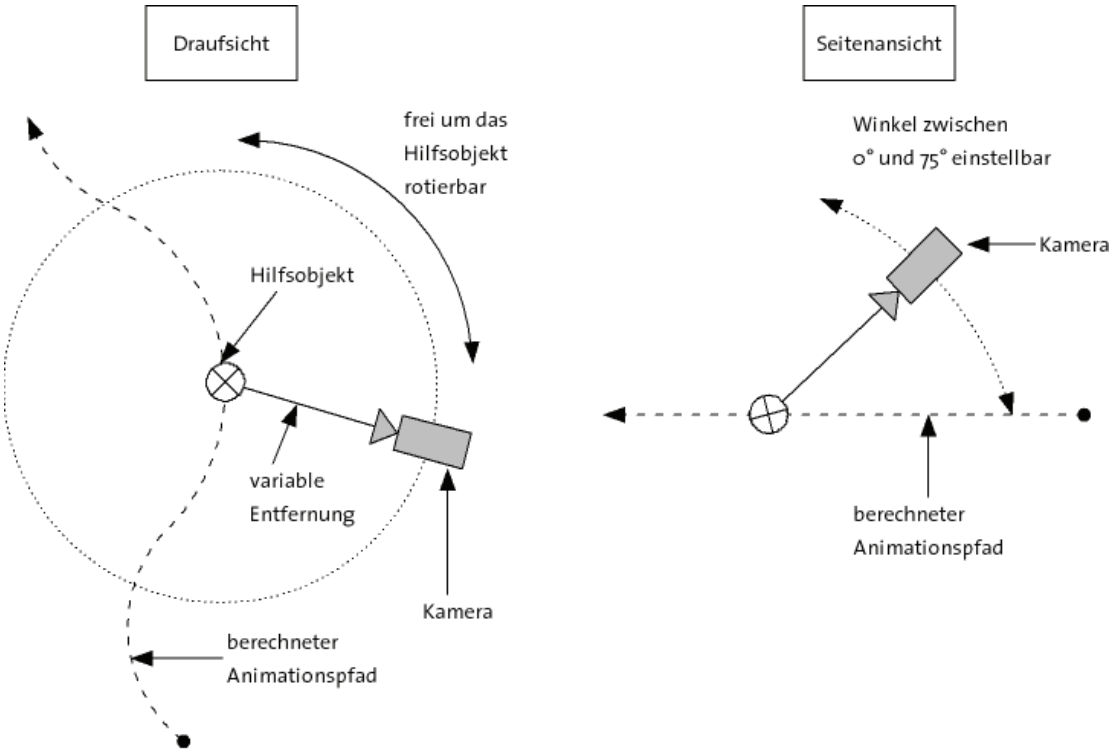


Abbildung 38: Orbit Kamera mit animiertem Hilfsobjekt

In dem verwendeten Channel-Konstrukt der Orbit Kamera, der in Quest3D als Vorlage vorhanden ist, sind bereits die Steuerung der Kamera über die Maus sowie der Bezug auf einen statischen Blickpunkt implementiert. Dieser muss gegen das animierte Hilfsobjekt ausgetauscht werden. Des Weiteren ist es nötig eine Logik zu integrieren, welche die Änderungen der Rotation sowie des Betrachtungswinkels über die Zeit aufzeichnet, um so schnell und einfach Flüge mit ansprechenden Kameraeinstellungen erstellen zu können. Die Kameralogik sowie die nötigen Eingriffe beziehungsweise Erweiterungen werden im Folgenden genauer erklärt.

Die in Quest3D vorhandene Kameravorlage verwendet einen so genannten Parent-Constraint, um die unterschiedlichen Einstellmöglichkeiten, welche die Orbit Kamera bietet, zur Verfügung zu stellen. Ein Parent-Constraint bewirkt, dass jede Änderung an der Transformationsmatrix eines übergeordneten 3D-Objekts auch auf das angehängte, untergeordnete 3D-Objekt angewendet wird. Änderungen an Position, Rotation

und Skalierung des Elternobjekts werden auch an das angehängte Kindobjekt übergeben.

Auch in Quest3D besitzt jedes 3D-Objekt eine Transformationsmatrix in Form eines DX8 Motion Channels. An diesem Channel können außer den Vektoren für Position, Rotation und Skalierung zwei weitere Matrizen angebracht werden. Die an vierter Position anhängbare Matrix ist die Lookat-Matrix. Ist an dieser Stelle eine Matrix angebracht, dann richtet sich das 3D-Objekt immer zu dieser Matrix hin aus, das heißt, die Rotation des Objektes wird durch die Lookat-Matrix übernommen. Die an fünfter Position angehängte Matrix ist die, für die Orbit Kamera wichtige, Parent-Matrix. Befindet sich an dieser Stelle eine Matrix, werden sowohl die Position, die Rotation als auch die Skalierung aus dieser Matrix übernommen. Die in den eigentlichen Transformationsvektoren angegebenen Werte geben dann nicht mehr die globale Position, Ausrichtung und Größe an, sondern lediglich die Abweichungen zur Parent-Matrix (Abbildung 39).

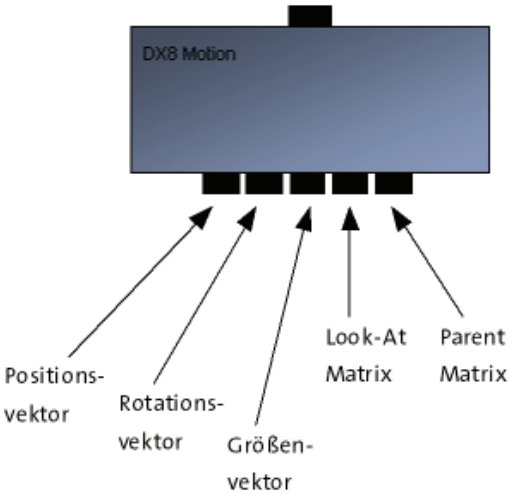


Abbildung 39: DX8 Motion Channel

Die Kameravorlage aus Quest3D funktioniert nach folgendem Prinzip: An die Matrix der Kamera (Camera Matrix) ist lediglich ein Positionsvektor angehängt (Abbildung 40 -> grüner Bereich). Dieser definiert die Abweichung in der Position zur ebenfalls an die Kameramatrix angehängten Parent-Matrix, DX8 Motion, (Abbildung 40 -> gelber Bereich) und die darin definierte Position. Der an der Kameramatrix angehängte Positionsvektor kann innerhalb definierter Grenzen über die rechte Maustaste und gleichzeitige Mausbewegung nach oben oder unten oder das Mausrad im z-Wert verändert werden. Diese Abweichung in z-Richtung zur Position der Parent-Matrix entspricht der Entfernung der Kamera zum Bezugspunkt. Die Rotation der Kamera um den Bezugspunkt wird in der konstruierten Parent-Matrix namens DX8 Motion gesteuert. Der an

den DX8 Motion Channel angehängte Positionsvektor definiert den Bezugspunkt für die Kamera, in diesem Fall den Ursprung der 3D-Szene (0,0,0). Der Rotationsvektor besteht hier lediglich aus zwei Komponenten, da eine Drehung des Hilfsobjektes um die Längsachse (z-Achse) und somit eine Seitenneigung der abhängigen Kamera nicht gewünscht ist. Die x- und y-Achse wird jeweils über Mauseingaben gesteuert. Die Neigung der Kamera ist dabei über das Klicken der linken Maustaste und das gleichzeitige Verschieben der Maus nach oben oder unten steuerbar. Ein Limit Value Channel sorgt dafür, dass der Kameraneigungswinkel innerhalb definierbarer Grenzen bleibt. So wird verhindert, dass die Kamera auch unter die Landschaft geneigt werden kann. Um die Blickrichtung zu ändern, muss die Maus bei gedrückter linker Maustaste seitwärts bewegt werden. Die aus den Mausbewegungen berechneten Rotationswerte werden anschließend noch über Damping Channels gedämpft um die Bewegung der Kamera weicher darzustellen.

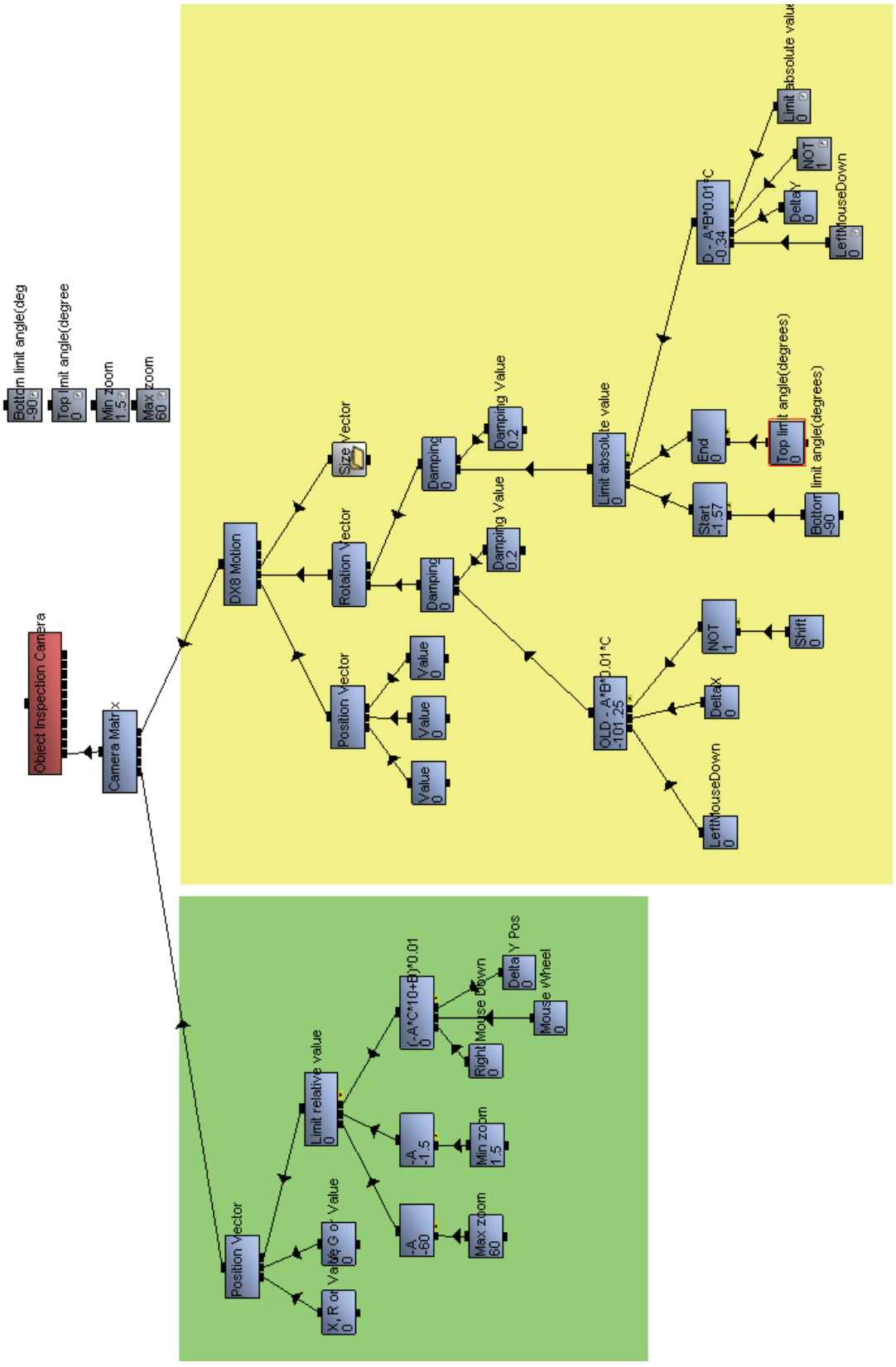


Abbildung 40: Orbit Kamera Vorlage aus Quest3D

Diese Kameravorlage muss nun an die Anforderungen der Echtzeitapplikation angepasst werden. Zunächst wird der statische Positionsvektor in der Parent-Matrix gegen den, in Kapitel 6.2.1 berechneten, animierten Positionsvektor des Hilfsobjekts ausgetauscht. Zur Verknüpfung des in einer anderen Substruktur befindlichen Positionsvektors des Hilfsobjekts wird ein Public Channel Caller verwendet, der den Wert an die Parent-Matrix übergibt. Das Ergebnis ist ein animierter Bezugspunkt, um den die Kamera innerhalb der gesetzten Grenzen rotieren kann.

Die Notwendigkeit, die Kameraeinstellung in Bezug auf Blickwinkel und Ausrichtung über den Zeitraum der Animation zu verändern und diese Veränderungen abzuspeichern, erfordert die Einführung von zusätzlichen Animationskontrollen in Form von Envelope Channels. Dabei wird die Vorgehensweise von 3D-Software wie Alias Maya oder 3ds max kopiert. Durch das Setzen von mehreren Schlüsselbildern (Keyframes) zu unterschiedlichen Zeitpunkten und der Interpolation der Werte zwischen den Schlüsselbildern wird eine flüssige Animation möglich gemacht. Nach demselben Prinzip wird nun die Animation von Blickwinkel und Ausrichtung der Kamera in der Echtzeitapplikation vorgenommen.

Durch Betätigung der Taste K (Abbildung 41 -> blauer Bereich) werden die zu diesem Zeitpunkt aktuellen Werte für Rotation und Zoom in zusätzliche, bereitstehende Envelope Channels gefüllt (Abbildung 41 -> grüner Bereich). Um eine zeitliche Zuordnung der gesetzten Werte zu bekommen wird an die Envelope Channels der für die Animation der Kamera zuständige, gerundete Zeitgeber angeschlossen. Mit den Nummerntasten „1“ und „2“ kann zwischen dem freien Bewegungsmodus und der aufgezeichneten Bewegung hin- und hergeschaltet werden. Zu diesem Zweck wird ein Selector Channel verwendet, der jeweils die Position des zuletzt aktiv gewesenen angehängten Channels an den Channel Switch übergibt. Dieser nimmt dann seinerseits den Wert des an der entsprechenden Position angehängten Channels an und übergibt ihn an den übergeordneten Channel. In diesem Fall werden zwei Channel Switches verwendet, einer für den z-Wert der Kamera und einer für die Auswahl des entsprechenden Rotationsvektors, die aber beide vom selben Selector Channel abhängig sind. An den Channel Switch, der die Auswahl des Rotationsvektors trifft, ist der Selector als Shortcut verknüpft (Abbildung 41 -> gelber Bereich).

Selbstverständlich muss es möglich sein, die aufgezeichneten Bewegungen der Kamera auch wieder zu löschen, um eine neue Animation zu kreieren. Die Logik zum Löschen aller vorhandenen Schlüsselwerte wird ausgelöst, wenn die Tasten „K“ und „Entfernen“ gleichzeitig betätigt werden. Bevor die Schlüsselwerte gelöscht werden, wird noch eine Sicherheitsabfrage gestellt, um ein versehentliches Löschen zu verhindern. Wird die Sicherheitsabfrage positiv beantwortet, werden sowohl die Envelope Channels für die Rotation, als auch der Envelope Channel für den z-Wert zurückge-

setzt. Das Löschen der Schlüsselwerte des z-Werts ist dabei via Shortcut verknüpft (Abbildung 41 -> roter Bereich).

Das Ergebnis dieses gesamten Logikmoduls ist eine Kamera, die um einen, auf einem Pfad animierten, Bezugspunkt frei in x-Richtung rotiert werden kann, in ihrem Neigungswinkel innerhalb gegebener Grenzen anpassbar ist und einen einstellbaren Zoom hat. Die Einstellungen der Kamera können über den zeitlichen Verlauf der Animation mittels Tastendruck aufgezeichnet und anschließend wiedergegeben werden. Weiterhin ist die Aufzeichnung durch eine Tastenkombination wieder löschbar.

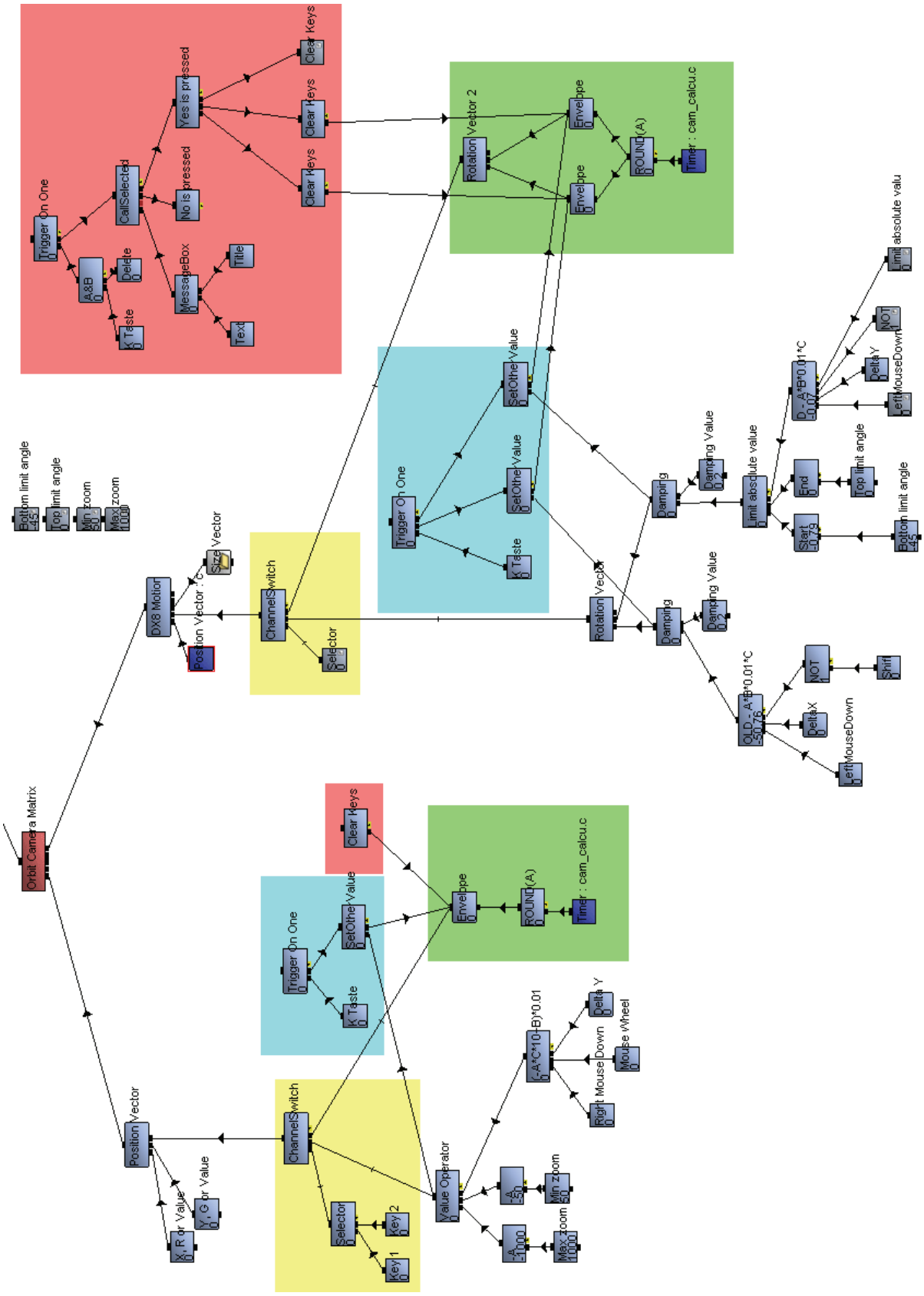


Abbildung 41: Angepasste Orbit Kamera

6.2.3 Steuerung des Zeitgebers

Der kommende Abschnitt beschäftigt sich kurz mit der Steuerung des Zeitgebers, einer Logik, die ebenfalls im Kameraberechnungsmodul enthalten ist. Der Zeitgeber, der die Animation der Kamera kontrolliert, ist komplett über die Tastatur steuerbar. Die Bedienung richtet sich im groben nach den im Videoschnitt üblichen Tastenbelegungen. Über die Leertaste wird die Animation gestartet und kann auch wieder gestoppt werden.

Das Zurückspulen der Animation zu jedem beliebigen Zeitpunkt wird durch Betätigen der Backspace Taste ausgelöst.

Die Animation der Kamera wird bei Erreichen der Gesamtdauer automatisch gestoppt. Dazu wird in einem Expression Value Channel überprüft, ob der aktuelle Frame den gleichen Wert hat wie die aus der Datenbank ausgelesene Gesamtdauer des aktuellen Flugs. Ist diese Bedingung erfüllt, wird der Befehl zum Stoppen der Animation gegeben.

Zuletzt wird noch eine Logik zur Bewegung auf der Zeitleiste benötigt, sprich die Möglichkeit innerhalb der definierten Animation vor und zurück zu spulen. Es werden zwei unterschiedliche Modi implementiert: Langsames Bild-für-Bild vor- oder zurückgehen und das schnelle Spulen, um bestimmte Animationsabschnitte flott zu erreichen.

Die entsprechenden Logiken sind im Anhang A.1 als Abbildungen zu finden.

6.3 Ladelogiken

6.3.1 Logik zum Laden der Landschaftsgeometrie

Im folgenden Kapitel geht es um das dynamische Laden der Landschaftsgeometrien. Dabei wird die notwendige Logik zum automatischen Laden der in einzelnen Channel Groups befindlichen Kacheln beschrieben. Die in den jeweiligen Channel Groups der einzelnen Kacheln implementierten Logiken zur Anzeige und zur Verwendung von Texturen und Level-of-Detail wird in Kapitel 6.4 thematisiert.

Nachdem die Datenbankanfrage zur Ermittlung der Start- und Endkachel erfolgreich durchgeführt wurde, werden die einzelnen Kacheln durch ein LUA-Skript nacheinander in ein Array geladen (Abbildung 42).

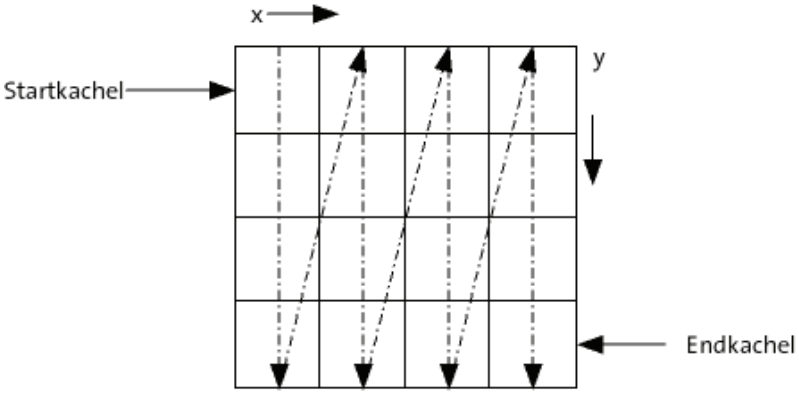


Abbildung 42: Ladereihenfolge der Kacheln

Die Anzeige der Landschaftskacheln erfolgt nach erfolgreichem Laden in das Runtime-Modul, indem pro Baumdurchlauf alle im Array befindlichen Kacheln einmal durch das LUA-Skript aufgerufen werden. Die für diesen Vorgang notwendige Logik sowie der Quellcode des LUA-Skripts finden sich im Detail im Anhang A.2 und A.3.

6.3.2 Logik zum Laden der Landschaftstexturen

Im folgenden Kapitel wird die zum Laden der Landschaftstexturen nötige Logik näher beschrieben. Auf Grund der Anforderung die Texturen der Landschaftsgeometrie leicht austauschen zu können, muss ein dynamisches Laden der Texturen aus einem lokal auf der Festplatte vorhandenen Dateiordner in die Echtzeitanwendung implementiert werden. Auf diese Weise können beispielsweise die blasseren Wintertexturen leicht gegen in der Farbe kräftigere Sommertexturen ausgetauscht werden. Dafür müssen lediglich die Texturen in den jeweiligen Ordnern gegen andere ausgetauscht beziehungsweise der Texturquellpfad entsprechend geändert werden.

Im Normalfall werden in Quest3D importierte 3D-Objekte mit entsprechender Textur komplett in einer Channel Group abgespeichert, das heißt, die Textur ist fest in die Anwendung mit eingebunden. Im Fall von „echtwetter“ ist es jedoch nötig bei jedem Start der Applikation die Texturen erneut zu laden, da sich teilweise tägliche Änderungen an den Texturen ergeben können. Bestes Beispiel ist die variierende Höhe der Schneegrenze im Winter. Für jede Schneehöhe gibt es einen Satz Texturen, der dann entsprechend der vorherrschenden Schneefallgrenze ausgewählt und geladen wird.

Für diesen Ladeprozess, der, ähnlich dem Laden der Kachelgeometrien, nur einmalig nach dem Start durchgeführt werden soll, war ursprünglich ebenfalls ein LUA-Skript vorgesehen, das den Ladeprozess steuern und nach und nach ein Buffer Array mit den entsprechenden Texturen füllen sollte. Nach dem erfolgreichen Laden der Texturen in

das Buffer Array, wären diese nach abgeschlossenem Ladevorgang der Geometrie in die Texture Channels der passenden Kacheln kopiert worden.

Die eben beschriebene Vorgehensweise endete in dem Versuch die Texturen nacheinander zu laden.

Die erste Idee benötigte einen Group Loader Channel, der eine beliebige Datei, deren Pfad in einem an den Group Loader Channel angehängten Text Channel definiert wird, in einen Buffer lädt. Der Status des Ladevorgangs kann über einen Group Loader Status Channel abgefragt werden. Zeigt der Status Channel an, dass ein Ladeprozess abgeschlossen ist oder noch keiner gestartet wurde, wird der erste beziehungsweise der nächste Ladeprozess im Group Loader Channel angestoßen. Ist der Ladeprozess einer Textur abgeschlossen, gibt der Status Channel den Wert eins zurück, die Buffer Array Position wird um eins erhöht, der Pfad zur nächsten zu ladenden Textur wird gesetzt und der Group Loader Channel wird erneut angestoßen. Soweit der Plan. Leider „verschluckt“ sich der Ladeprozess beim Durchlaufen der Schleife, da nach dem Laden großer Texturen diese aus dem Zwischenspeicher in das Buffer Array geschrieben werden müssen, zeitgleich aber schon die Arrayposition um eins inkrementiert wird, weil der Status Channel den Prozess bereits für abgeschlossen hält. Dies führt zu einem nicht korrekt gefüllten Array, bei dem es zu leeren Arraypositionen kommen kann. Eine Rücksprache mit den Entwicklern von Quest3D führte zu der Gewissheit, dass es sich bei diesem Verhalten um einen Bug im Group Loader Status Channel handelt, der in der mittlerweile herausgekommenen Quest3D Version 3.0 behoben wurde. Da diese Version bei der Erstellung der Ladelogik der Texturen noch nicht vorhanden war, musste eine andere Lösung zum Laden der Texturen gefunden werden.

Die einzige andere Möglichkeit, die sich in der Kürze der Zeit realisieren ließ, war ein Auslagern des Ladeprozesses in die jeweilige Kachel Channel Group. Es wurde also eine Logik entwickelt, welche die nötigen Texturen für eine Kachel innerhalb der Kachel Channel Group lädt.

Nachdem die in Kapitel 6.3.1 beschriebene Logik die Geometrie der Landschaftskacheln fertig geladen hat, lädt die Texturenladelogik die zu den Kacheln passenden Texturen für Tag und Nacht jeweils für die hoch und die niedrig aufgelöste Version der Kachel und verknüpft außerdem die zentral geladene Abbildung des Wolkenschattens. Die Tag- und Nachttexturen werden dabei bei jedem Start des Runtime-Moduls aus einem an zentraler Stelle definierten, externen Ordner neu geladen. Im Anhang A.4 befindet sich die gesamte Logikstruktur zum Laden der Texturen innerhalb einer Kachel Channel Group.

6.3.3 Logik zum Laden der Städtesymbole

Der folgende Abschnitt beschreibt die zum Laden der Städtesymbole nötige Logik. Das Städtesymbol selbst ist eine eigene Channel Group, namens „city_names“, in der ein Textobjekt, ein Hintergrundobjekt und ein kleines Prisma sowie ein wenig Logik abgespeichert sind. Die Logik der Channel Group „city_names“ wird in Kapitel 6.5 genauer beschrieben. Die Channel Group „city_names“ wird von einem Logikkonstrukt entsprechend der in der Eingabemaske ausgewählten Anzahl der Städte mehrfach instanziert und in ein Array geladen. Das variable Textobjekt, bestehend aus Stadtnamen und Temperaturangabe sowie die für jede Stadt unterschiedliche Position werden direkt aus der Datenbank über ein LUA-Skript ausgelesen und beim Initialisieren der Objekte angewendet (Abbildung 43). Nach erfolgreicher Initialisierung der Stadtobjekte werden diese durch das LUA-Skript pro Baumdurchlauf einmal aufgerufen, um sie in der 3D-Szene anzuzeigen. Die Logik zum Laden der Städtesymbole sowie das darin enthaltene LUA-Skript finden sich im Anhang A.5 und A.6.

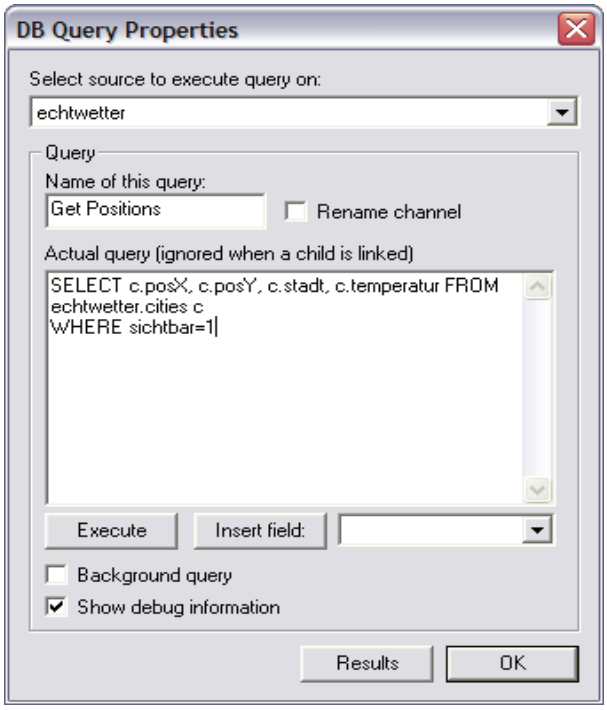


Abbildung 43: Datenbankanfrage Städtepositionen

6.4 Logiken zur Anzeige der Landschaftsgeometrien

6.4.1 Verwendung von Level-of-Detail

Der folgende Abschnitt beschreibt die Verwendung von Level-of-Detail, einer in Echtzeitapplikationen oft verwendeten Technik zur Performancesteigerung, die auch in

der „echtwetter“-Anwendung zum Einsatz kommt. Die Idee, die hinter Level-of-Detail steckt, ist folgende: Wenn ein Anwender einer Echtzeitapplikation ein 3D-Modell aus der Nähe betrachtet, soll jedes Detail gut zu erkennen sein. Dafür ist es notwendig, dass das 3D-Objekt entsprechend hoch aufgelöst ist. Entfernt sich der Anwender von dem 3D-Objekt, so dass es nur noch schemenhaft wahrnehmbar ist und Details nicht mehr zu erkennen sind, genügt es eine in der Auflösung reduzierte Form des 3D-Modells darzustellen. Diese entfernungsabhängige Darstellung von unterschiedlich hoch aufgelösten 3D-Modellen wird als Level-of-Detail (LOD) bezeichnet.

In Quest3D wird zwischen zwei unterschiedlichen Arten von Level-of-Detail unterschieden. Die erste Technik, die auch bei der „echtwetter“-Applikation zum Einsatz kommt, trägt die Bezeichnung Static Level-of-Detail. Zur Realisierung dieser Technik ist es notwendig, das darzustellenden 3D-Modell in mindestens zwei unterschiedlich hoch aufgelösten Versionen vorliegen zu haben. Je nach Entfernung des 3D-Modells zur Kamera wird dann die entsprechende Version angezeigt. Befindet sich der Betrachter sehr nah am 3D-Modell, wird die am höchsten aufgelöste Geometrie angezeigt, bei sehr großer Entfernung die niedrigste dargestellt. Je nach Anzahl der unterschiedlichen vorhandenen Detailstufen erfolgt das Umschalten zwischen den Modellen vom Anwender unbemerkt. Liegen jedoch zu wenige Abstufungen des Modells vor, kommt es zum so genannten „Popping“-Effekt, dem plötzlichen und leider gut zu sehenden Umschalten zwischen zwei Auflösungsgraden.

Die zweite Technik nennt sich Dynamic Level-of-Detail. Der Vorteil dieser Technik ist, dass man für die Implementierung nur ein sehr hoch aufgelöste 3D-Modell benötigt. Die Reduktion der Geometrie wird automatisch durch die CPU berechnet, abhängig von der Entfernung des Modells zur betrachtenden Kamera. Durch das sanfte und stetige Reduzieren der Geometrieauflösung wird der oben erwähnte „Popping“-Effekt vermieden. Diese Technik birgt jedoch zwei andere, alles entscheidende Nachteile. Zum einen ist die dynamische Reduktion der Geometriedaten sehr rechenaufwändig, zum anderen ist es nicht möglich eine Textur auf ein, sich in der Anzahl der Vertexpunkte veränderndes 3D-Modell aufzubringen. Dies ist auch der Grund, warum in den meisten Computerspielen und auch bei „echtwetter“ die Static Level-of-Detail Technik zum Einsatz kommt [9].

Im Folgenden wird die Implementierung des Static Level-of-Detail in der „echtwetter“-Anwendung beschrieben. Als Ausgangsbasis für die Implementierung der Level-of-Detail-Technik dienen die in Kapitel 4.3.1 erstellten Channel Groups, die bereits die beiden unterschiedlich hoch aufgelösten Versionen der jeweiligen Kachel enthalten (Abbildung 44).

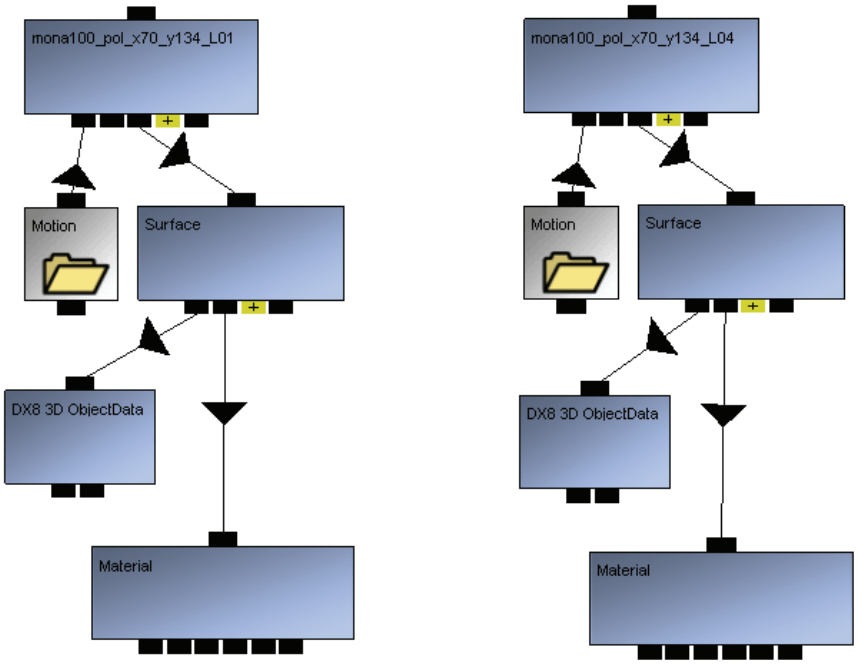


Abbildung 44: Kachel Channel Group Ausgangsbasis

Jedes in Quest3D importierte 3D-Modell wird nach erfolgreichem Import wie in Abbildung 44 zu sehen dargestellt. Bei dem obersten Channel handelt es sich um den DX8 3D Object Channel. Er trägt den Namen des Objekts, das er darstellt. An ihn ist an erste Stelle die Transformationsmatrix, hier zusammengefasst in einen Ordner „Motion“, angehängt. Sie beschreibt die Position, Rotation und die Ausdehnung des 3D-Objekts. An den DX8 3D Object Channel angehängt befinden sich die Vertexinformation und die Materialeigenschaften eines 3D-Objekts.

Zur Implementierung der LOD-Technik wird der in Quest3D vorhandene DX8 Static LOD Channel verwendet. Dieser erwartet eine Matrix, welche die Position, die Rotation und die Skalierung des 3D-Objekts definiert und zur Berechnung der Entfernung zwischen 3D-Objekt und Kamera dient.

Im Normalfall genügt es hier eine Verknüpfung zu der Transformationsmatrix eines der zum Einsatz kommenden 3D-Objekte anzugeben, da diese von der Position her dem zumeist in der Mitte des Objekts sitzenden Pivotpunkts entspricht. Die hier zum Einsatz kommenden Kacheln tragen ihren Pivotpunkt allerdings nicht in der Mitte der Geometrie, sondern immer in der oberen linken Ecke. Da es versäumt wurde eine entsprechende Korrektur des Pivotpunkts direkt in 3ds max vorzunehmen, ist eine Translation des Berechnungspunkts nötig, um die für das Umschalten der einzelnen Detailstufen nötige Entfernung, jeweils zwischen dem Mittelpunkt einer Kachel und der Kamera zu berechnen. Hierfür wird die Translation aus der Transformationsmatrix von einer der beiden 3D-Modelle extrahiert und anschließend mit einem Vektor ad-

diert. Dieser Vektor verschiebt den Berechnungspunkt einer 500 Einheiten langen und breiten Kachel um 250 Einheiten nach rechts und um 250 Einheiten nach unten um so den Mittelpunkt der Kachel zu definieren.

Der LOD Channel benötigt einen Parameter in Form eines Envelope Channels, der an Hand der berechneten Entfernung zwischen Kachel und Kamera entscheidet, welche Detailstufe gerade angezeigt wird. Der Schwellwert zum Umschalten zwischen den einzelnen Detailstufen lässt sich global regeln. Weiterhin benötigt der LOD Channel die unterschiedlich hoch aufgelösten 3D-Modelle der Kachel, angefangen mit dem detailreichsten. Die komplette Level-of-Detail-Logik wird in Kapitel A.7 dargestellt.

6.4.2 Darstellung der Kacheltexturen

Im folgenden Kapitel wird erläutert, wie die Landschaft bei Tag und Nacht dargestellt wird. Nachdem in Kapitel 6.3.2 beschriebenen Ladevorgang besitzt jede Kachel drei Texturen; die Tagtextur, die Nachttextur sowie den Wolkenschatten.

Jede Landschaftskachel verfügt über eine Tag- und Nachttextur, die bei Bedarf ineinander übergeblendet werden können. Die Überblendung kann für alle geladenen Kacheln anhand eines globalen Parameters eingestellt werden. In Quest3D besitzt jedes 3D-Objekt die Fähigkeit zwischen zwei Texturen sanft zu überblenden. Zu diesem Zweck hat jedes 3D-Objekt einen Parameter, der beim Erhöhen die Transparenz der ersten Textur verringert und gleichzeitig die Transparenz der zweiten Textur um den gleichen Betrag erhöht. Der Texturüberblendungsparameter der einzelnen Landschaftskacheln ist mit dem globalen Parameter zur Überblendung verknüpft, so dass über eine Animation des globalen Parameters ein sanfter Wechsel zwischen Tag und Nacht möglich ist.

Der Wolkenschatten besteht aus einer Textur mit RGB- und Alphakanal. Zunächst werden die Farbkanäle von der Helligkeit her reduziert und mittels Modulation auf die Landschaftskacheln aufgebracht. Bei der Modulation werden die Farbwerte der Textur der Landschaftskacheln sowie die entsprechenden Farbwerte der Wolkenschatten-Textur miteinander multipliziert. Der Alpha-Kanal der Wolkenschatten-Textur entscheidet nur, an welcher Stelle Schatten sichtbar ist oder nicht (Abbildung 45 & Abbildung 46 & Abbildung 47).

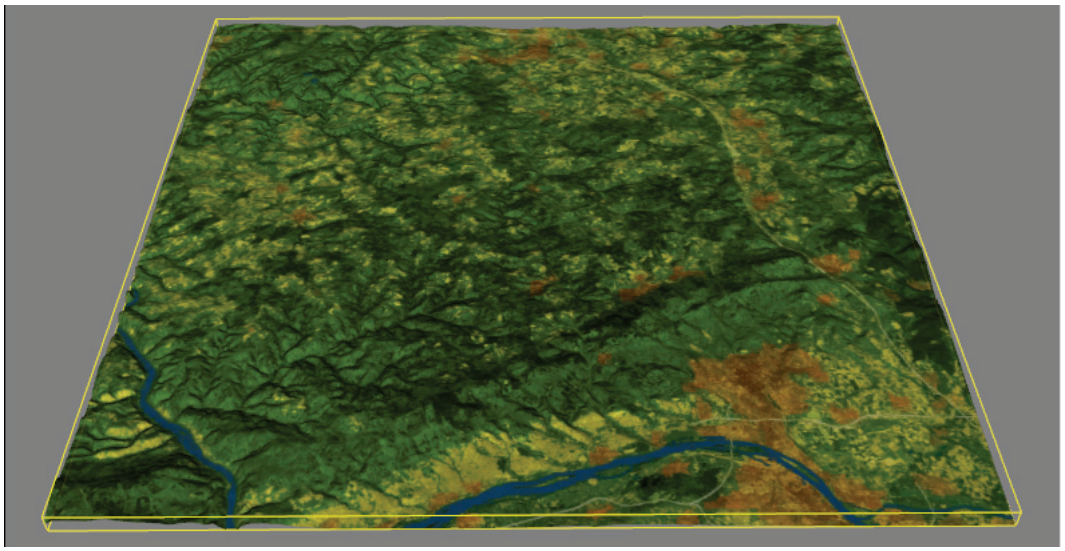


Abbildung 45: Kachel mit Tagtextur

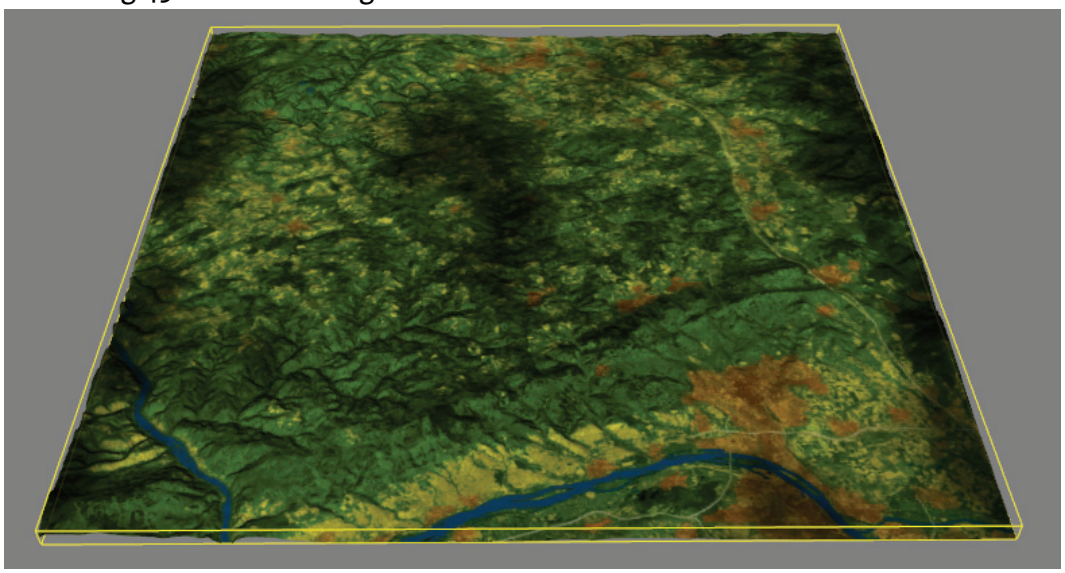


Abbildung 46: Kachel mit Tagtextur und Wolkenschatten

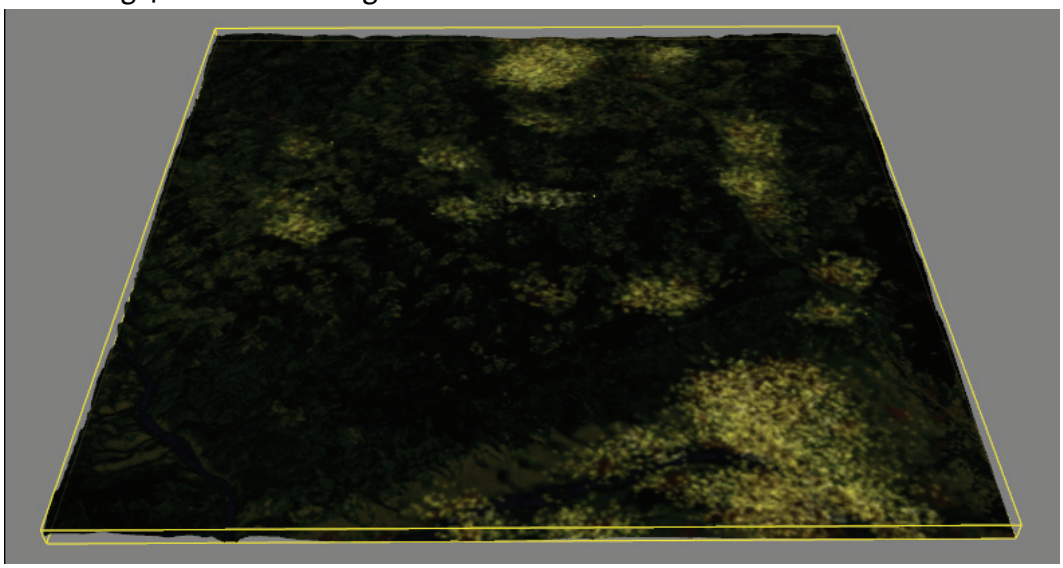


Abbildung 47: Kachel mit Nachttextur

6.5 Logiken zur Anzeige der Städtesymbole

Das folgende Kapitel beschreibt die Channel Group „city_names“, welche die Logik zur Anzeige der Städtesymbole enthält. Diese wird in der Anwendung von dem bereits in Kapitel 6.3.3 beschriebenen LUA Skript Channel entsprechend der Anzahl der in der Eingabemaske ausgewählten Städte instanziert. Innerhalb der Channel Group befinden sich die Logiken zum Darstellen des Städtenamens und der Temperatur, die natürlich bei jeder Stadt unterschiedlich sind, ein Hintergrundobjekt zur besseren Lesbarkeit sowie ein kleines Dreieck, Prisma genannt, das die exakte Position der Stadt auf der Landschaft anzeigt. Dieses wird bei Städten mit Wahrzeichen weggelassen.

Die Position des Städtesymbols wird durch das in Kapitel 6.3.3 beschriebene LUA-Skript gesetzt. Die y-Position, gleichzusetzen mit der Höhe des Stadtsymbols, kann über einen Schieberegler im Bildschirmmenü der Echtzeitapplikation verändert werden. So lässt sich die Höhe für alle Stadtobjekte über einen globalen Wert schnell und einfach ändern.

Die Städtesymbole wenden sich in ihrer horizontalen Ausrichtung stets der Kamera zu. Dazu wird an den Lookat-Anschluss der Städtesymbole eine konstruierte Matrix gehängt, welche die Höhe der Kamera unberücksichtigt lässt. Die Höhe, zu der sich das Textobjekt ausrichtet, ist jeweils die eigene Höhe des Objekts, so dass es immer orthogonal zum Boden steht.

Zur Darstellung des Textes, dem Städtenamen sowie der Temperatur wird ein DX8 3D Text From Texture Channel verwendet. Dieser Channel wird an den, die Oberfläche des 3D-Objekts beschreibenden, Surface Channel gehängt. Der DX8 3D Text From Texture Channel stellt Text nicht als wirkliche Geometrie dar, sondern verwendet eine Textur, auf der alle verfügbaren Zeichen entsprechend definiert sind und bildet diese auf eine passende Rechteckfläche ab (Abbildung 48). Der Städtename und die Temperatur werden durch das LUA-Skript aus Kapitel 6.3.3 dynamisch eingefüllt und entsprechend dargestellt.

Abbildung 48: DX8 3D Text From Texture Channel

Außer der eben beschriebenen Logik zur Anzeige und Ausrichtung des Textobjekts gibt es zusätzlich eine Logik zum entfernungsabhängigen Ein- und Ausblenden der Städtesymbole. Je nach Entfernung des Städtesymbols zur Kamera wird dieses über einen Envelope Channel automatisch in der Transparenz verändert. So wird verhindert, dass die Kamera durch einen Städtenamen durchfährt und das ganze Bild verdeckt wird. Der die Transparenz steuernde Envelope Channel definiert allerdings nicht nur eine nahe Entfernung, bei der das Stadtsymbol ausgeblendet wird, sondern auch eine weite Entfernung, ab der das Stadtsymbol überhaupt erst zu sehen ist (Abbildung 49 & Abbildung 50).

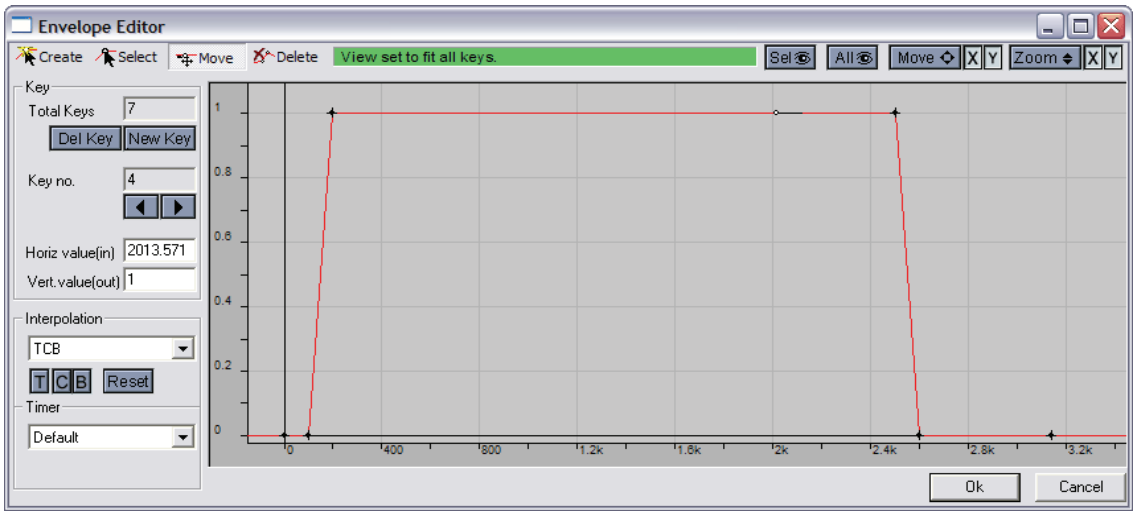
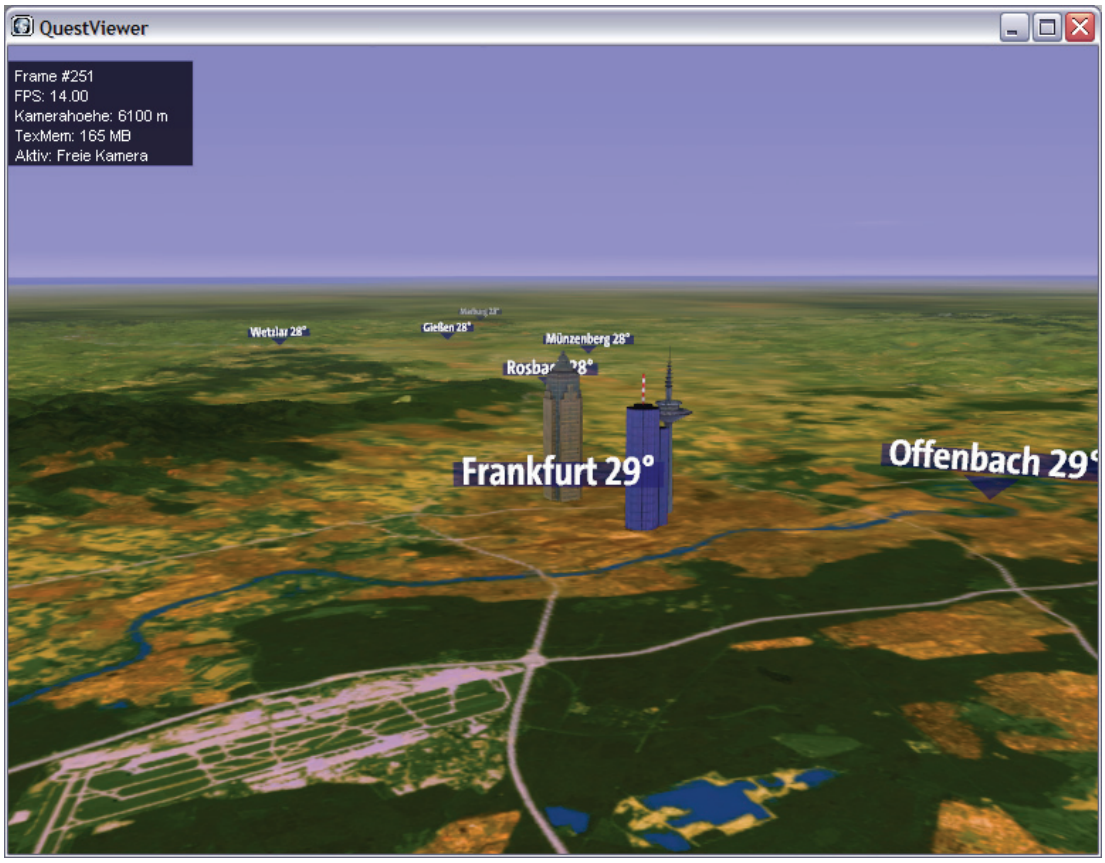


Abbildung 49: Envelope Channel zur entfernungsabhängigen Steuerung der Sichtbarkeit



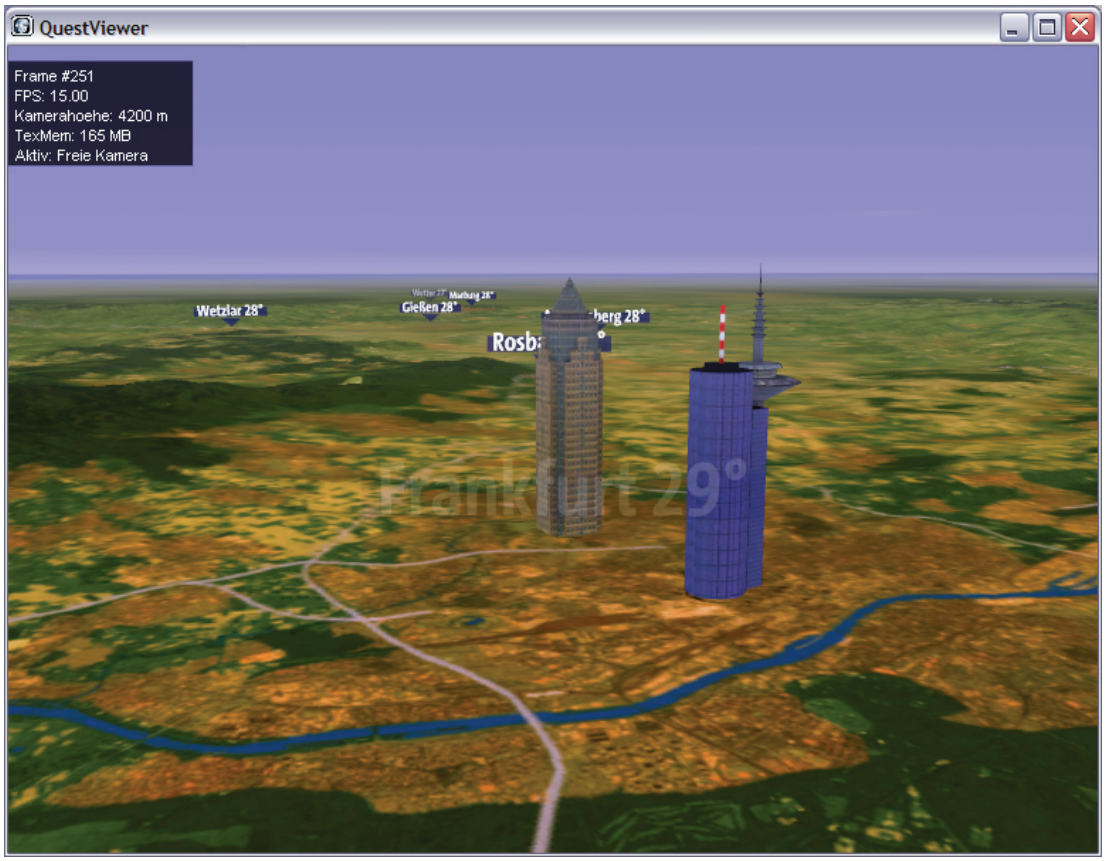


Abbildung 50: Entfernungsabhängiges Ausblenden des Städtenamens

Bei dem Hintergrundobjekt handelt es sich um eine einfache Box. Das einzig besondere ist die dynamische Anpassung der Größe an die Textlänge des Städtenamens und der Temperatur. Zu diesem Zweck werden an Hand der Bounding Box die wirklichen Ausmaße des Städtenamens ermittelt und in die Ausdehnung des Hintergrundobjekts eingetragen. Auf diese Weise richtet sich die Größe des Hintergrunds immer nach der wirklichen Breite des jeweiligen Textes. Eine Berechnung auf Basis der Zeichenlänge ist auf Grund der unterschiedlichen Buchstabenbreiten nicht genau genug, da beispielsweise ein „i“ wesentlich weniger Platz braucht als ein „w“ und Städtenamen wie „Eltville“ mit mehreren sehr schmalen Buchstaben so einen sehr großen überstehenden Rand bekommen würden (Abbildung 51).

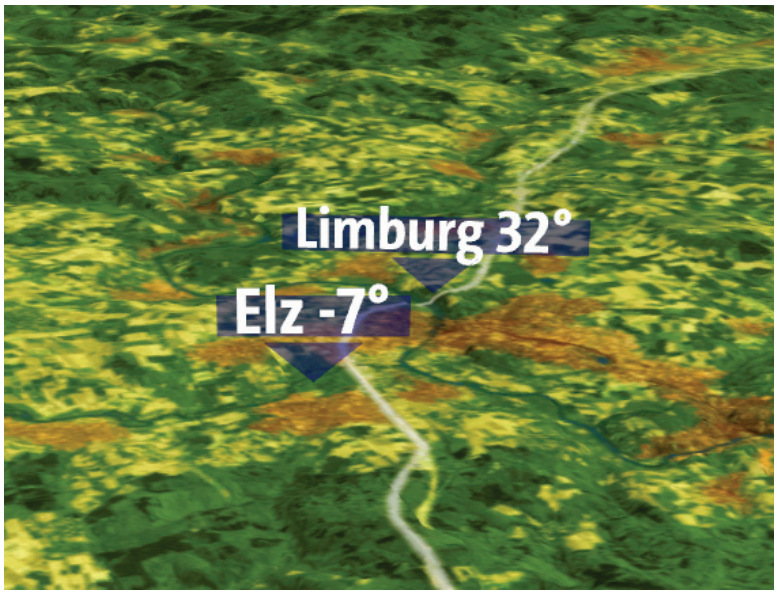


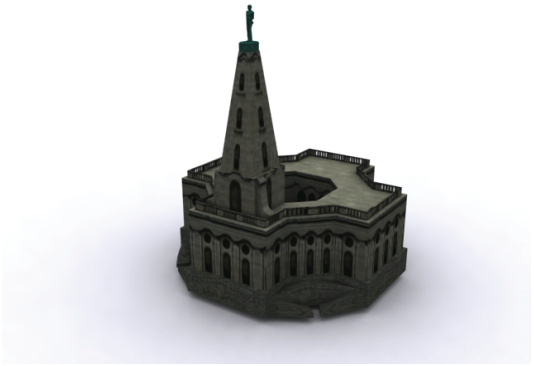
Abbildung 51: Automatische Länge des Hintergrundobjekts

Sowohl das Hintergrundobjekt als auch das kleine Prisma, das die Position der Stadt auf der Landschaft genauer anzeigt, sind über einen Parent-Constraint an das durch das LUA-Skript gesetzte Textobjekt gebunden, so dass sie die Position und die Ausrichtung von diesem übernehmen.

Die für die Anzeige der Städtenamensymbole notwendigen Logiken befinden sich im Anhang A.1.

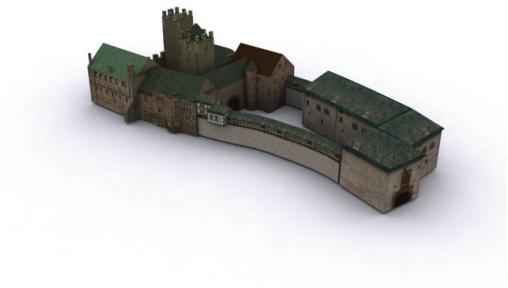
6.6 Wahrzeichen

In der Echtzeitanwendung werden die meisten Städte nur durch das im vorangegangenen Kapitel beschriebene Stadtsymbol, bestehend aus Name und Temperatur, dargestellt. Einige größere Städte verfügen jedoch über zusätzliche, modellierte Wahrzeichen, wie zum Beispiel dem Herkules in Kassel oder dem Biebricher Schloss in Wiesbaden. Diese Wahrzeichen dienen der leichteren Identifikation der Städte und verbessern die Orientierungsmöglichkeiten beim Zuschauer.

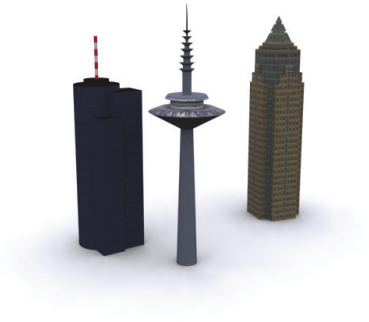


Herkulesdenkmal in Kassel

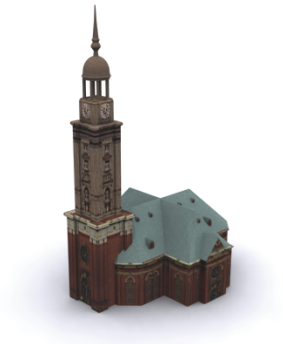
Brandenburger Tor in Berlin



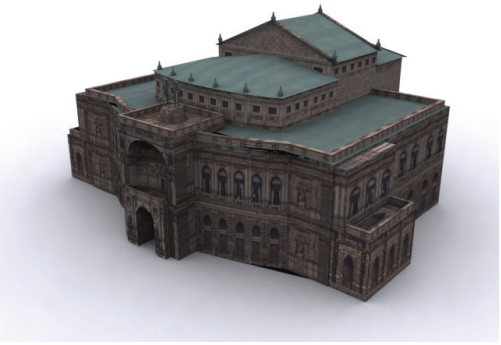
Wartburg in Eisenach



Commerzbank, Messe- und Fernsehturm in Frankfurt



St. Michael in Hamburg



Semperoper in Dresden

Abbildung 52: Wahrzeichen einiger Städte

Die einzelnen 3D-Modelle der Wahrzeichen kommen bereits in den aktuellen Wetterflügen zum Einsatz und liegen als Dateien im 3ds-max-Format vor. Die Konvertierung der Dateien wurde bereits in Kapitel 4.3.1 beschrieben.

An Hand einer Konfigurationsdatei im Textformat, in der alle Städte aufgelistet sind, die ein Wahrzeichen haben, kann bestimmt werden, welche davon angezeigt werden. Nicht anzuzeigende Wahrzeichen werden einfach auskommentiert.

Zur realitätsnahen Darstellung werfen die Wahrzeichen einen Schatten auf die Landschaft. Die Richtung des Schattenwurfs ist von der in der Konfigurationsmaske angegebenen Uhrzeit des Wetterflugs abhängig.

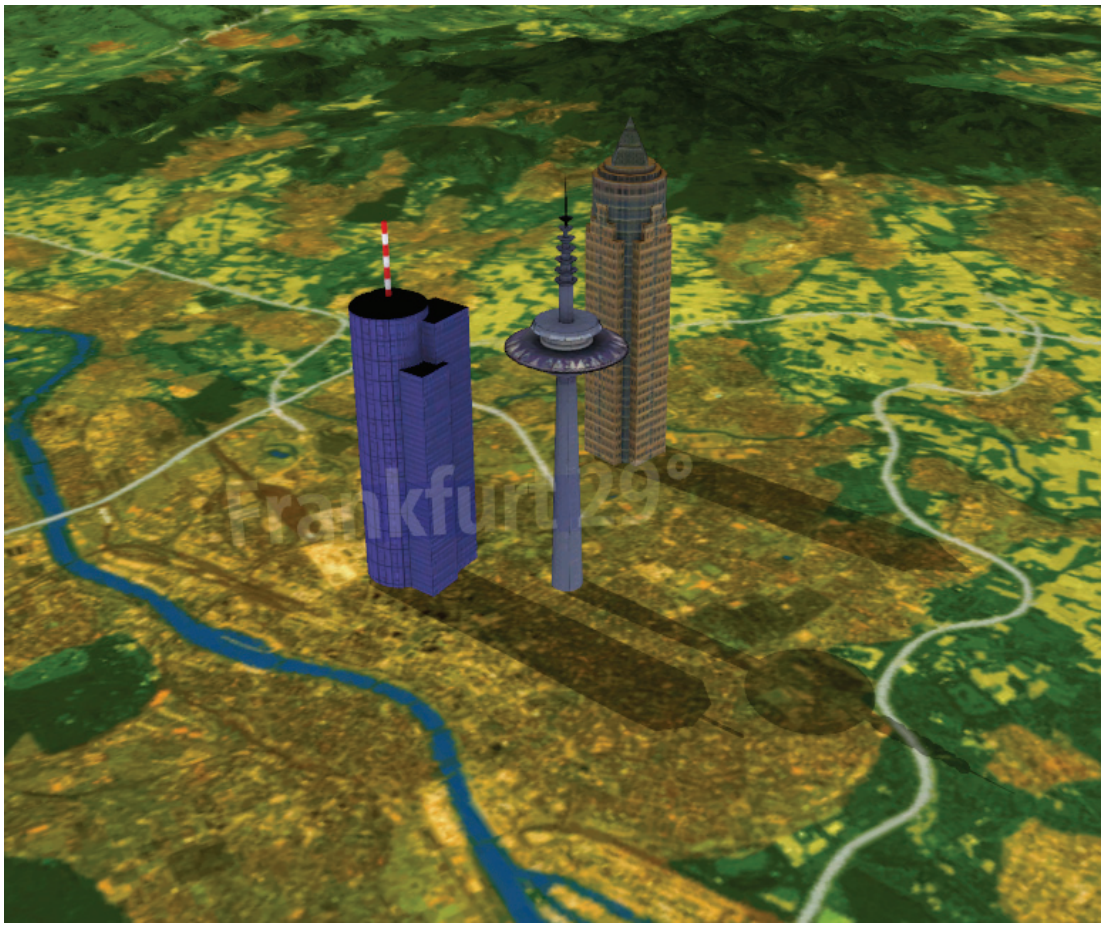


Abbildung 53: Echtzeitschattenwurf der Wahrzeichen

Dieser dynamische Schatten wird durch eine Echtzeitschattenlogik auf Basis des Stencil-Shadow-Prinzips implementiert. Der dabei verwendete Stencil-Buffer ist ein Teil des Z-Buffers, der für jeden Pixel einen zusätzlichen Wert speichern kann. Der Stencil-Shadow wird nun wie folgt erzeugt. Zunächst wird der Stencil-Buffer auf null gesetzt und nachfolgend an Hand der Silhouette des Wahrzeichens und der Position des Lichts ein Schattenvolumen berechnet. Anschließend werden die Seiten des Schattenvolu-

mens auf ihre Ausrichtung zur Kamera hin untersucht und für jede Vorderseite der Wert im Stencil-Buffer um eins erhöht und für jede Rückseite um eins verringert. Bei Überlappungen von Vorder- und Rückseiten steht im Stencil-Buffer für die entsprechenden Pixel wieder der Wert null drin. Nur an den Stellen, wo der Schatten später sichtbar sein soll, hat der Stencil-Buffer einen Wert ungleich null. Die so ausgewählten Pixel werden über einen voreingestellten Wert abgedunkelt und stellen so den Schatten dar (Abbildung 54) [10].

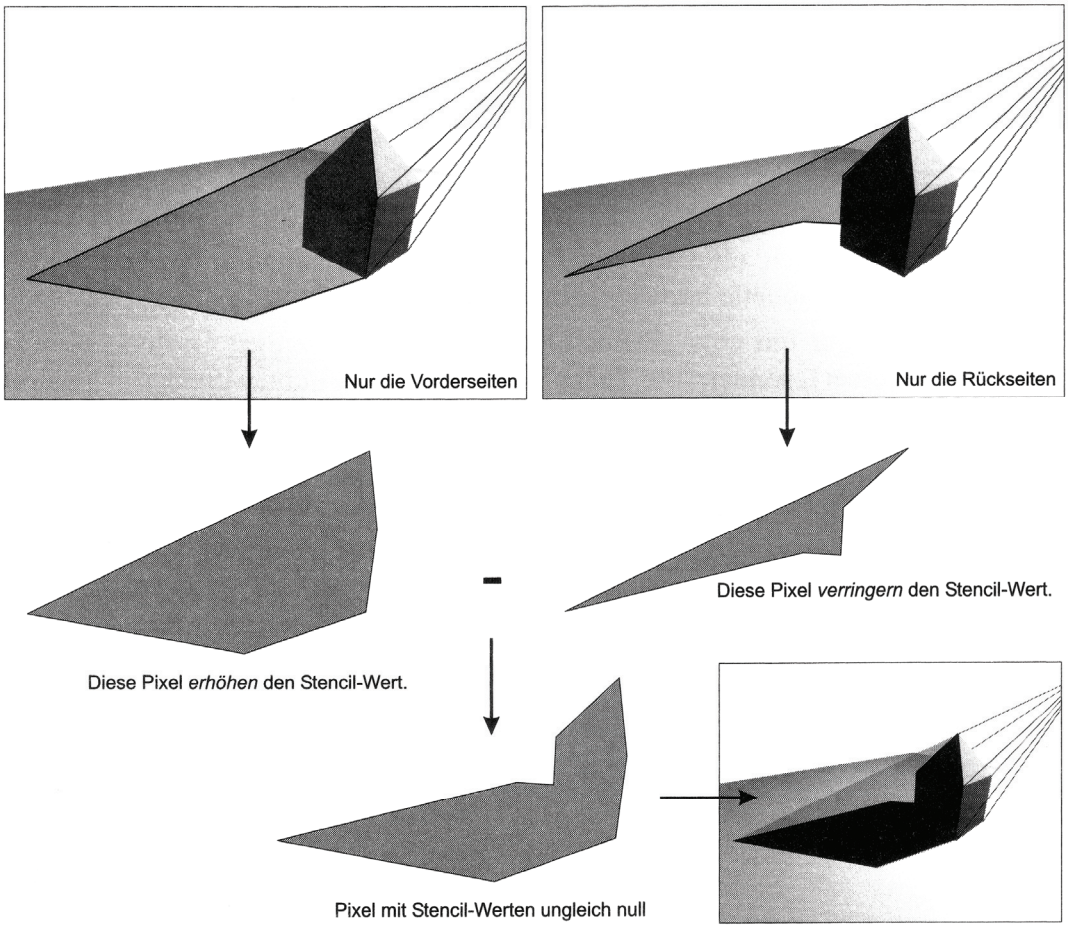


Abbildung 54: Stencil-Shadow-Prinzip [10]

Für die Berechnung des Schattenvolumens musste in 80% aller Fälle ein Schattendummyobjekt gebaut werden, da die bereits vorhandenen Wahrzeichenmodelle entweder zu komplex gebaut waren oder unzulässige Geometrien enthielten (Abbildung 55).

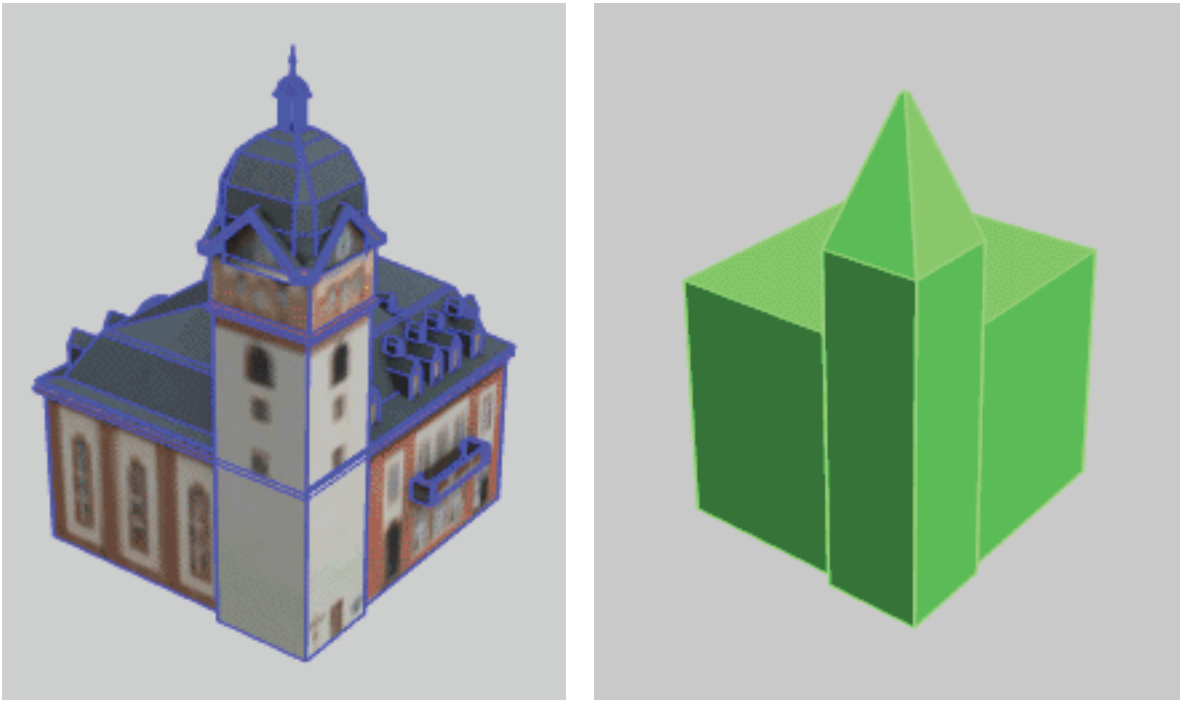


Abbildung 55: Wahrzeichen und zugehöriges Schattendummyobjekt

6.7 Wetterphänomene

6.7.1 Niederschlag

Der folgende Abschnitt befasst sich mit der zur Darstellung von Niederschlag nötigen Logik. Sowohl Regen wie auch Schnee müssen in der „echtwetter“-Anwendung in ihrer Ausprägung, an der prognostizierten Position und in der richtigen Ausdehnung korrekt darstellbar sein. Zu diesem Zweck kommt das in Quest3D vorhandene Advanced Particle System (APS) zum Einsatz. Das APS ermöglicht über eine Vielzahl von Parametern das Aussehen und Verhalten der emittierten Partikel im Detail zu ändern und so die gewünschten Ausprägungen der Niederschlagsarten darzustellen (Abbildung 56).

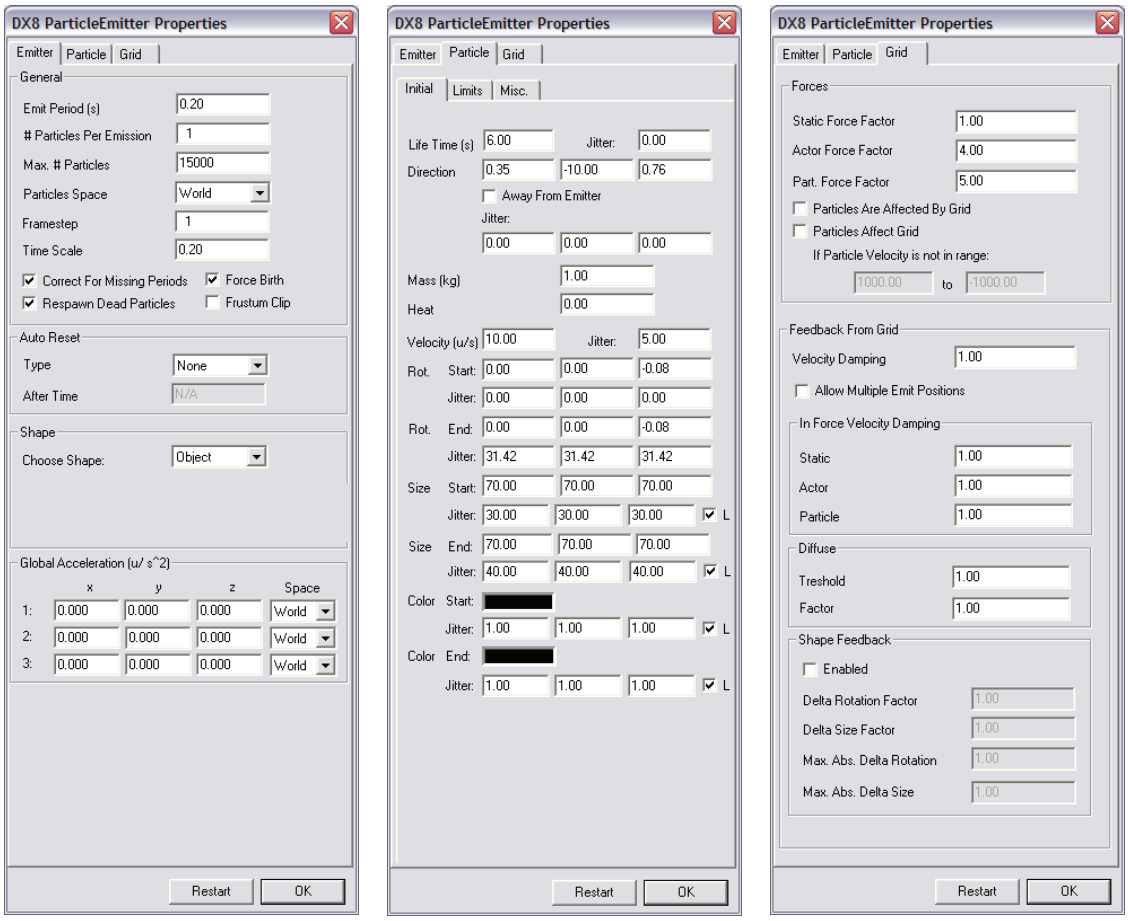


Abbildung 56: Einstellmöglichkeiten APS

Voraussetzung für die Verwendung des APS ist eine Emittergeometrie. Jedes beliebige 3D-Objekt kann als Emittergeometrie verwendet werden. Dabei entspricht jeder Vertexpunkt der Geometrie genau einem Emitter.

In einer 3ds-max-Referenzszene wird die vom TriVis-System aktuell berechnete Niederschlagskarte als Textur auf ein passendes und korrekt positioniertes Rechteck gelegt. Im Anschluss muss der Grafiker an Hand der zu sehenden Niederschlagsgebiete manuell eine Emittergeometrie erstellen, die er anschließend aus 3ds max exportiert und in Quest3D importiert. Diese wird dort in die bestehende Partikellogik eingefügt. Der Grafiker muss also in bestehende Quest3D-Logik eingreifen, was zum momentanen Zeitpunkt auf Grund der fehlenden Flexibilität von Quest3D in Bezug auf automatisierten Import von Geometrie die einzige Möglichkeit darstellt.

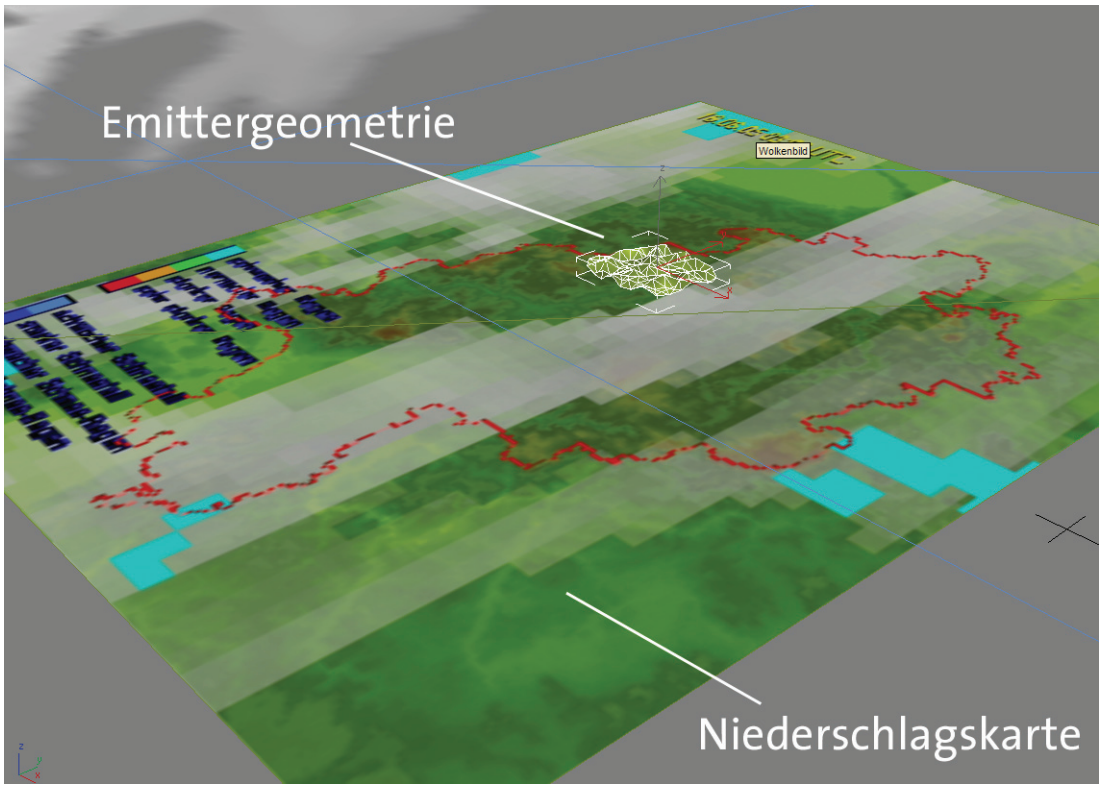


Abbildung 57: 3ds-max-Referenzszene zur Erstellung der Emittergeometrie

Die eigentliche Logik zur Darstellung der Partikel funktioniert wie in Abbildung 59 dargestellt. Die im Ordner „Einstellungen“ zusammengefassten Einstellmöglichkeiten das Partikelsystem betreffend werden in Kapitel 6.8 näher beschrieben.

Der 3D Object Channel namens „Partikel1“ verkörpert die in der Szene zu sehenden Partikel. Daran angehängt findet sich die Transformationsmatrix des nebenstehenden 3D-Objekts „Line01“. Bei „Line01“ handelt es sich um eine, wie oben beschrieben, in 3ds max konstruierte Emittergeometrie, die in Quest3D importiert wurde und deren Position, Ausrichtung und Skalierung jetzt an das Partikelsystem übergeben wird. An den Surface Channel des Partikelobjekts wird der ParticleSpriteRenderer angehängt, der die Darstellungsart der Partikel festlegt. In diesem Fall werden Sprites, kleine Bildchen zur Darstellung von Regen oder Schnee, verwendet, die sich automatisch immer der Kamera zuwenden. Der ParticleSpriteRenderer ist seinerseits mit dem Particle-Emitter Channel verbunden, der das Herzstück des Advanced Particle Systems bildet. Hier stehen die zahlreichen Parameter zur Einstellung des Partikelverhaltens, den Emittereigenschaften und der Beeinflussung der Partikel durch Kräfte zur Verfügung. Des Weiteren werden die Geometriedaten des importierten, in 3ds max erstellten Emitterobjekts an das Partikelsystem übergeben. So werden die darin gespeicherten Vertexpunkte vom Partikelsystem als einzelne Emitter verwendet. Welche Art von Niederschlag in der Echtzeitanwendung zu sehen ist, richtet sich nach dem Aussehen der verwendeten Textur. Zurzeit existieren vier unterschiedliche Texturen, jeweils

zwei für Regen und Schnee. Es ist jedoch ohne größeren Aufwand möglich weitere Texturen hier anzuhängen und so als Sprites für die Partikel zu verwenden (Abbildung 58 & Abbildung 59).

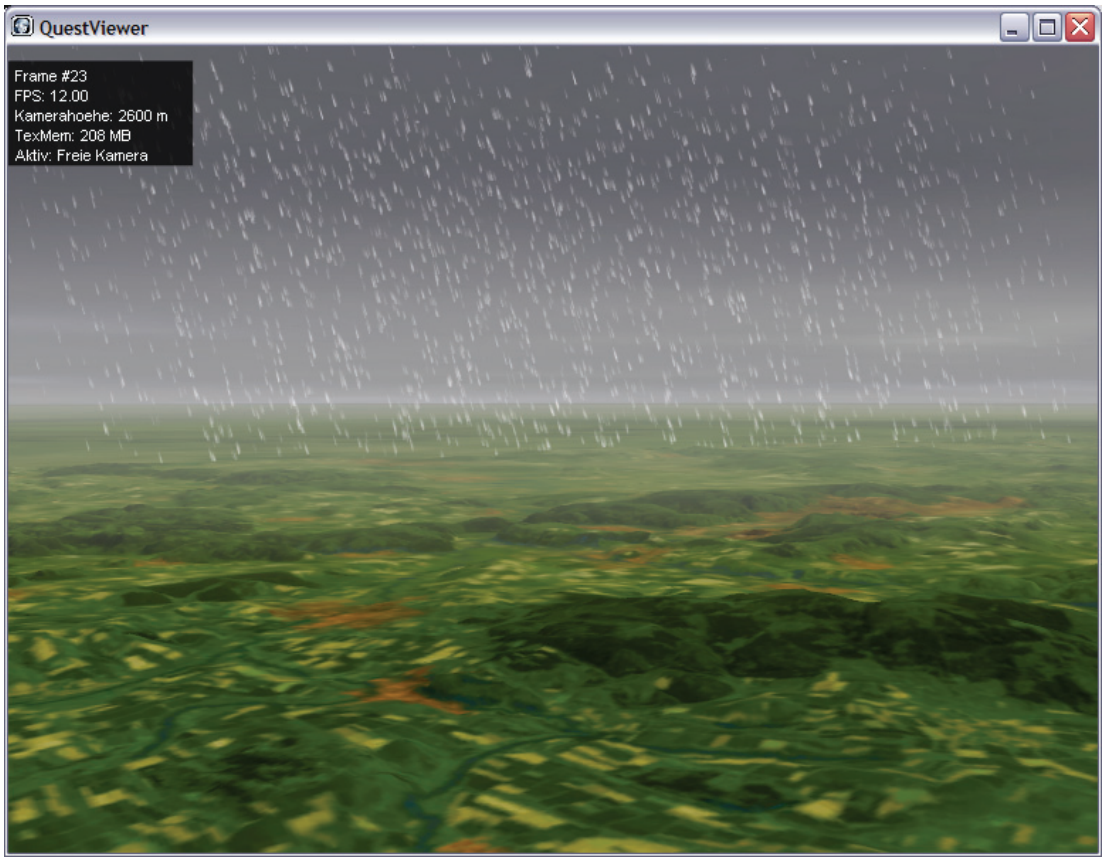


Abbildung 58: Niederschlag

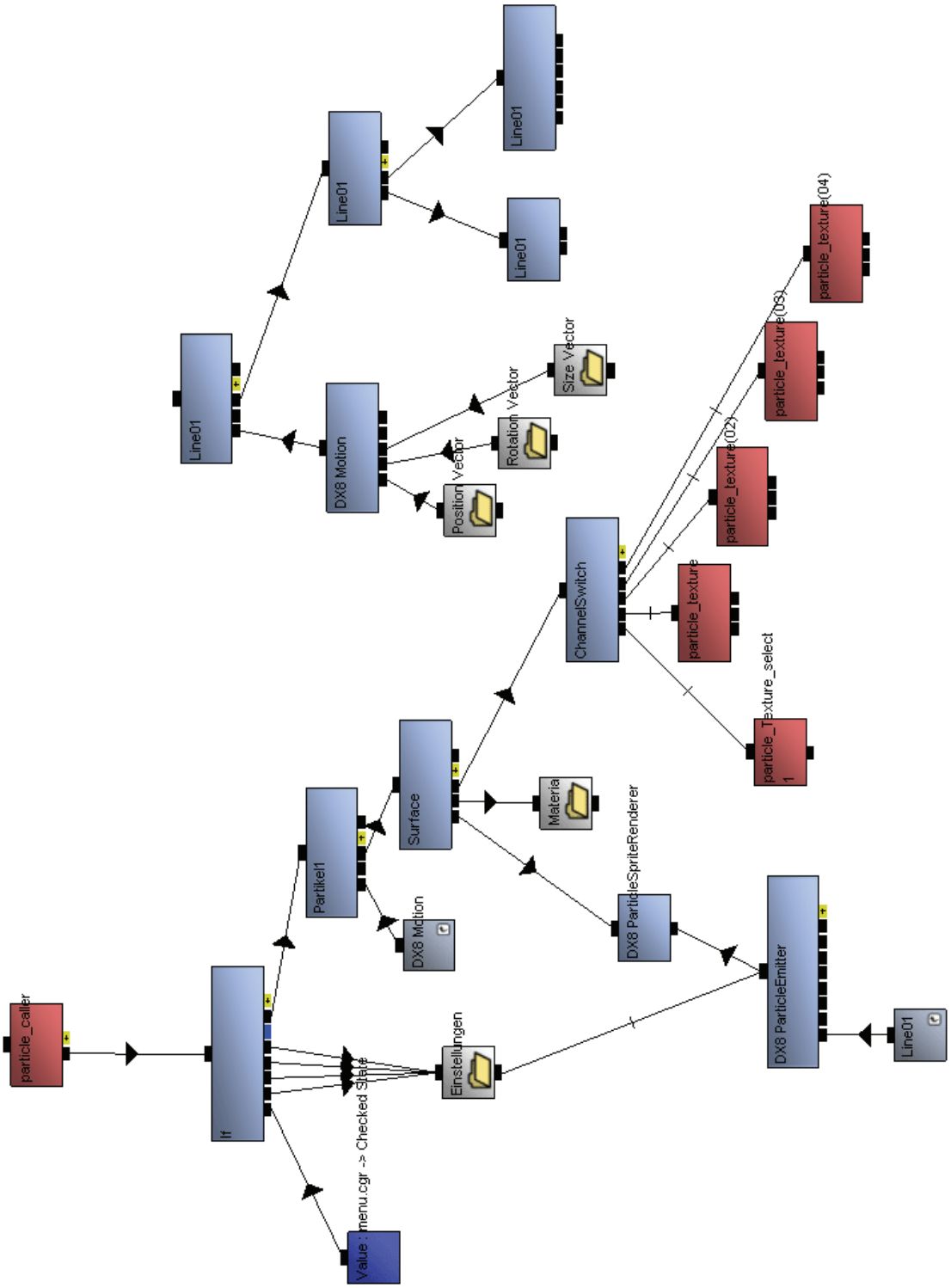


Abbildung 59: Partikellogik

6.7.2 Nebel und Entfernungsdunst

Im folgenden Abschnitt wird die Verwendung von Nebel und Entfernungsdunst näher beschrieben.

Da sich in der Echtzeitapplikation auf Grund der begrenzten Hardwareressourcen, insbesondere der Texturspeicher der Grafikkarte spielt dabei ein große Rolle, immer nur ein Teil der Landschaftsgeometrie laden lässt, wird zur Darstellung der übrigen, weit entfernten Landschaft ein Dummyobjekt verwendet. Das Dummyobjekt besteht lediglich aus einer Rechteckfläche, die den Bereich der Landschaftskacheln komplett abdeckt und mit einer sehr grob aufgelösten Textur der Bundesrepublik Deutschland ausgestattet ist. Auf diese Weise wird verhindert, dass die Kamera bei einem Blick in die Ferne ins Nichts schaut. Das Bodendummyobjekt ist von der Position her ein wenig unterhalb der Kacheln angeordnet. Der Übergang zwischen den geladenen Landschaftskacheln und dem Bodendummyobjekt wird dabei durch die Verwendung von Nebel und Entfernungsdunst unkenntlich gemacht. Der Entfernungsdunst hilft außerdem den in Kapitel 6.4.1 beschriebenen „Popping“-Effekt, der durch die Verwendung der LoD-Technik auftritt, zu verdecken und dem Zuschauer einen gewissen Tiefeneindruck zu vermitteln.

Der erste Versuch Nebel in die 3D-Szene zu integrieren, verwendete einen auf dem Kopf stehenden, stumpfen Kegel, der mit einer Alphatextur belegt wurde und sich mit der Kamera mitbewegte. Die Transparenz sowie die Höhe und der Durchmesser ließen sich dabei einstellen. Der so über die Landschaft gleitende Nebel verursachte allerdings eine Art „Vorhangeffekt“. Die Landschaftsgeometrie und 3D-Objekte tauchten nicht langsam aus dem Nebel auf, sondern wurden von jetzt auf gleich klar dargestellt. Dieser Effekt wurde als zu störend empfunden und die eben beschriebene Methode als nicht praktikabel verworfen (Abbildung 60).

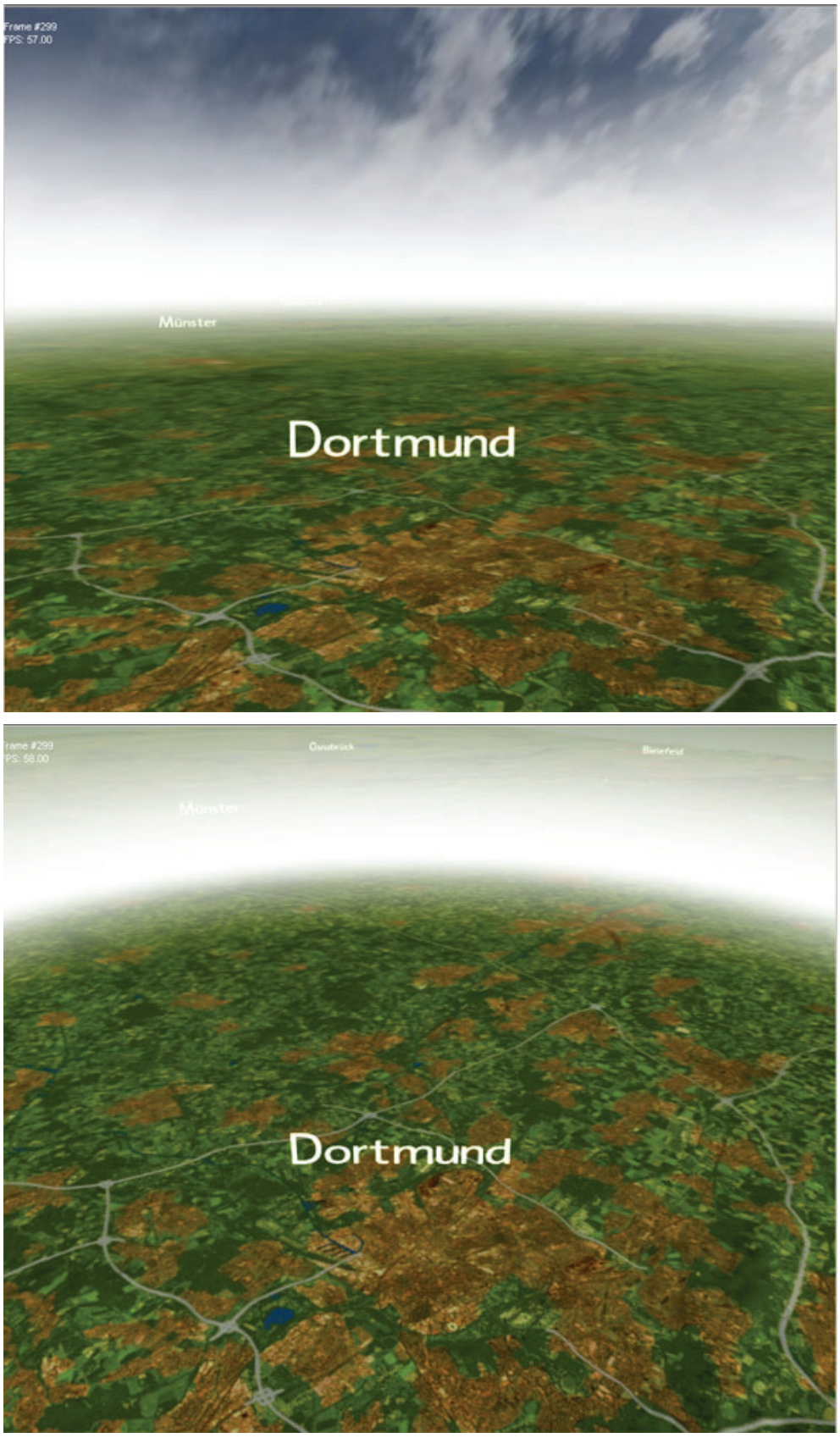


Abbildung 60: Vorhangeffekt beim Nebel

Der zweite und vorerst finale Ansatz verwendet eine Kombination aus einem einstellbaren Entfernungsdunst und einem in die Kamera einblendbaren Nebelrechteck. Der Entfernungsdunst ist als 3D-Objekt in die Szene integriert. Es handelt sich dabei um ein in die Länge gezogenes zylindrisches Objekt, das grob den Ausmaßen der Bundesrepublik entspricht und eine feste Position besitzt. Lediglich die Höhe, die Transparenz sowie die Farbe des Entfernungsdunstes sind einstellbar. Der Entfernungsdunst dient lediglich zur Verdeckung des Horizonts und des Übergangs zwischen Landschaftskacheln und Bodendummy. Für die Darstellung von richtigem Nebel wird eine Rechteckfläche mit Alphatextur, einstellbarer Transparenz und Größe direkt in die Kamera eingeblendet. Dieses Rechteck wird über eine entsprechende Verschiebung in y-Richtung automatisch auf der Höhe des Horizonts gehalten und „vernebelt“ so das gesamte Bild ohne Rücksicht auf Tiefen und Überdeckungen (Abbildung 61).

Abbildung 61: Nebel

Die zur Implementierung nötigen Logiken beschränken sich bei dem Entfernungsdunst auf das Anzeigen des Objekts sowie die Steuerbarkeit der Transparenz, Farbe und Größe. Die Logiken für die Anzeige des Nebelobjekts sind die gleichen, lediglich die Logik zur Ausrichtung des Nebelrechtecks an Hand des Kamerawinkels ist hinzuzu-

fügen. Die Vorgehensweise, um Objekte direkt in die Kameraansicht abzubilden, wird in Kapitel 6.8 näher beschrieben.

6.7.3 Wolkenabbildung

Der folgende Abschnitt behandelt die Darstellung der Wolken in der Echtzeitapplikation.

Die gesamte Landschaftsgeometrie wird von einer Wolkenkuppel überspannt, auf die das im TriVis-System berechnete Wolkenbild als Textur aufgebracht wird. Dieses Wolkenbild wird jedoch zuvor durch einen Adobe Photoshop-Automatismus auf die Ausmaße der Landschaft beschnittenen und im DDS-Format abgespeichert. Neben den RGB-Werten für die Farbigkeit der Wolken enthält diese Textur auch einen Alphakanal, der die Transparenz der Wolken regelt. Die Wolkentextur wird bei jedem Start der Applikation aus dem in der Eingabemaske der Flugkonfiguration definierten Pfad neu geladen. Auf diese Weise kann das Wolkenbild schnell und einfach gegen ein anderes ausgetauscht werden (Abbildung 62 & Abbildung 63).

Die gleiche Wolkentextur kommt in abgedunkelter Form als Wolkenschatten auf dem Boden zum Einsatz. Das Reduzieren der Helligkeit geschieht ebenfalls durch den oben bereits erwähnten Adobe Photoshop-Automatismus, der das abgedunkelte Originalbild anschließend unter einem anderen Namen als Schattentextur ebenfalls im DDS-Format mit Alphakanal abspeichert. Dieser Wolkenschatten wird dann wie in Kapitel 6.4.2 auf die Landschaftskacheln sowie das Bodendummyobjekt aufgebracht.

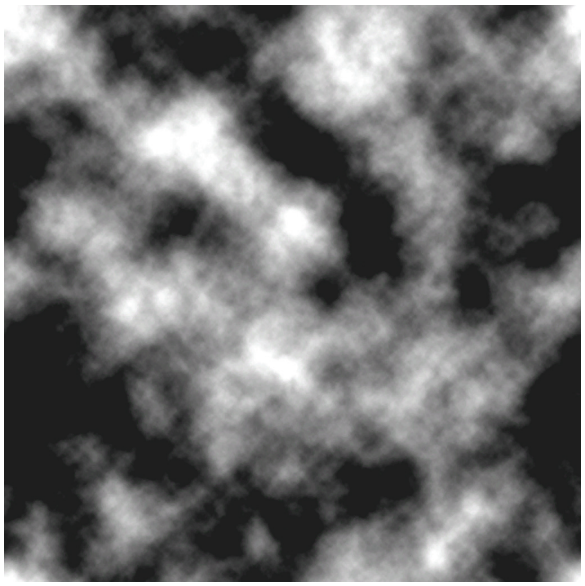


Abbildung 62: Wolkentextur

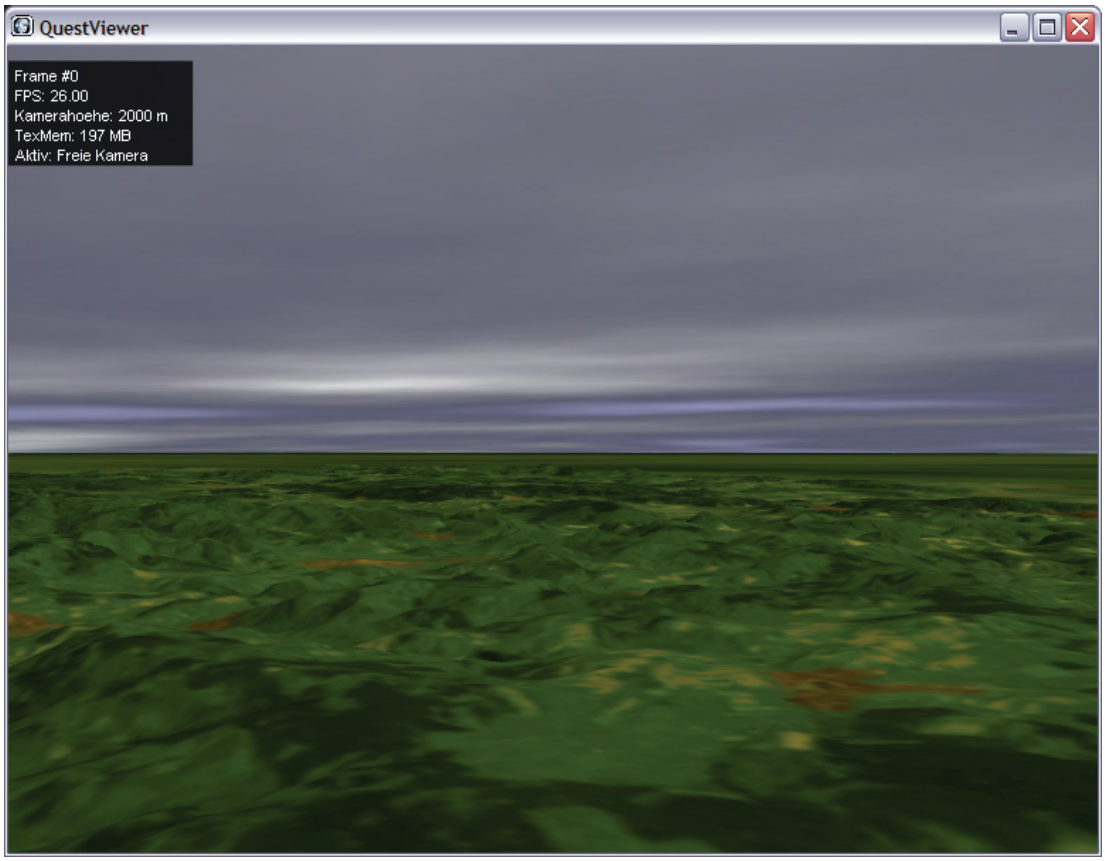


Abbildung 63: Wolkentextur auf der Himmelskuppel

6.7.4 Blitze

Nachfolgend wird die Implementierung der Blitze beschrieben.

Bei dem eigentlichen Blitz handelt es sich um eine einfache Rechteckfläche, die mit einer von fünf unterschiedlichen, jeweils zufällig ausgewählten, animierten Texturen belegt ist und sich in seiner Ausrichtung stets der Kamera zuwendet (Billboard-Effekt). In der Applikation werden die Blitze vom Anwender durch Betätigen der Taste B und das gleichzeitige Klicken mit der linken Maustaste auf die Landschaft, an der Stelle an der der Blitz erscheinen soll, positioniert. Dazu wurde das Bodendummyobjekt mit einer Logik versehen, die beim Anklicken die Koordinaten des ausgewählten Punkts zurückliefert. An Hand dieser Koordinaten wird die Rechteckfläche positioniert und die animierte Textur wird abgespielt. Der Zeitpunkt in der Animation, zu dem der Blitz auf der Landschaft positioniert wird, wird in eine Animationskontrolle (Envelope Channel) abgespeichert. Werden zu einem späteren Zeitpunkt der Animation weitere Blitze eingefügt, dann wird deren Position ebenfalls in der Animationskontrolle abgespeichert und zwischen den einzelnen Positionen wird eine Rechteckkurve interpoliert. Die gesetzten Blitze können über die Tastenkombination B und Entfernen wieder gelöscht werden (Abbildung 64).

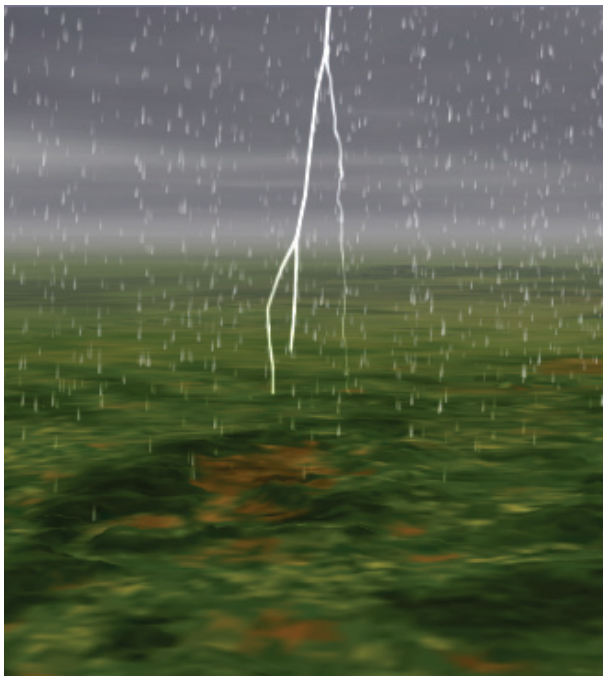


Abbildung 64: Blitz

6.7.5 Wind

Im Folgenden wir die Darstellung des Winds innerhalb von „echtwetter“ beschrieben.

Zur Darstellung des Winds werden kleine bewegte Pfeile verwendet, die den Eindruck erwecken als ob sie über die Landschaft gleiten. Dieser Effekt ist über eine Rechteckfläche realisiert, in die mittels Verformung in 3ds max eine leichte Wellenform modelliert wurde. Auf diese aus 3ds max importierte Geometrie wird dann in Quest3D eine sich wiederholende, animierte Textur mit Alphakanal aufgebracht, welche die Windpfeile darstellt. Diese wandern über die, durch den Alphakanal ansonsten unsichtbare, gewellte Geometrie und erwecken den Anschein, dass sie über die Bodenwellen der Landschaftsgeometrie fliegen. Die gewellte Geometrie bewegt sich dabei immer auf Bodenebene mit der Kamera mit. Die animierte Textur der Windpfeile auf der Geometrie lässt sich durch die Taste W ein- und ausblenden sowie durch die benachbarten Tasten Q und E nach links beziehungsweise rechts drehen. (Abbildung 65)

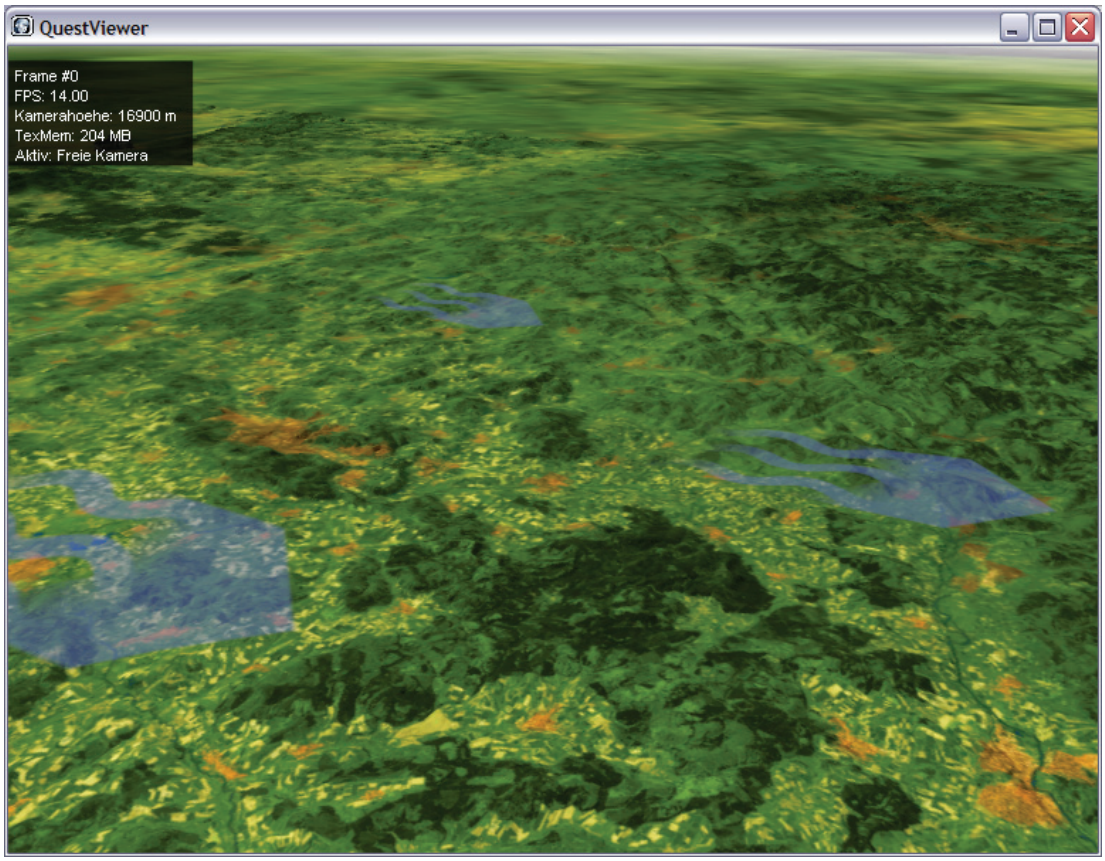


Abbildung 65: Windpfeile

6.8 Logiken für Einstellmöglichkeiten innerhalb des Runtime-Moduls

Das folgende Kapitel befasst sich mit den im Runtime-Modul implementierten Einstellmöglichkeiten für Licht, Niederschlag, Nebel und alle anderen Merkmale, für die ein visuelles Feedback unverzichtbar ist. Zunächst wird jedoch die Logik für das Einblenden der Menüs in die aktive Kameraansicht kurz erläutert.

Durch die Integration einer zusätzlichen orthographischen Kamera wird auf das Bild der bereits bestehenden 3D-Szene eine zusätzliche Ebene appliziert. Die orthographische Kamera rendert lediglich die Menüoberflächen und Bedienelemente, die zur Einstellung der Wetterparameter dienen. Durch Drücken der Tasten F9, F11 und F12 werden die jeweiligen Menüoberflächen zu Beleuchtung, Niederschlagseinstellungen und generellen Darstellungsoptionen in das Sichtfeld der orthographischen Kamera eingeblendet.

Das Beleuchtungsmenü erlaubt eine Anpassung der Lichtstimmung für die Landschaftskacheln sowie die Wahrzeichen, indem die Farbe der Lichtquelle über drei Schieberegler in den RGB-Werten verändert werden kann. Die Farben des Himmels und der Wolkenabbildung sowie der beiden Nebelobjekte für den Entfernungsdunst und den Wetternebel werden im Gegensatz zur Landschaft nicht durch ein Licht be-

einflusst, sondern es wird über Schieberegler die Farbe der Selbstbeleuchtung (emissive) gesteuert. Die Einstellungen für den Nebel umfassen zusätzlich die Möglichkeiten die Transparenz sowie die Ausdehnung anzupassen. Zudem können beide Nebelobjekte jeweils über eine Checkbox ein- und ausgeschaltet werden.

Das Menü zur Steuerung des Niederschlags umfasst vier Schaltflächen zur Selektion der Partikel-Sprites, mit denen die Art des Niederschlags festgelegt werden kann. Des Weiteren gibt es sechs Schieberegler zur Modifikation des Niederschlags, wie zum Beispiel die Fallgeschwindigkeit und die Größe. Eine Checkbox dient zum An- und Ausschalten des gesamten Partikelsystems, eine Windrose zur Bestimmung der vorherrschenden Windrichtung. Durch Einstellen der Windrichtung an der Windrose und die Angabe der Windstärke mittels des entsprechenden Schiebereglers werden die Niederschlagspartikel in die gewünschte Richtung abgelenkt. Der Schieberegler für die Varianz beeinflusst die zufällige Rotation von Partikeln auf ihrem Weg vom Himmel zur Erde und eignet sich hervorragend zur Darstellung von Schneetreiben oder Schneeverwirbelungen. Beim Schieberegler zur Einstellung der Rotation hingegen handelt sich um eine exakte Drehung der Partikel, um die Querverschiebung der Partikel unter Beeinflussung von Wind in der Textur zu berücksichtigen (Abbildung 66).

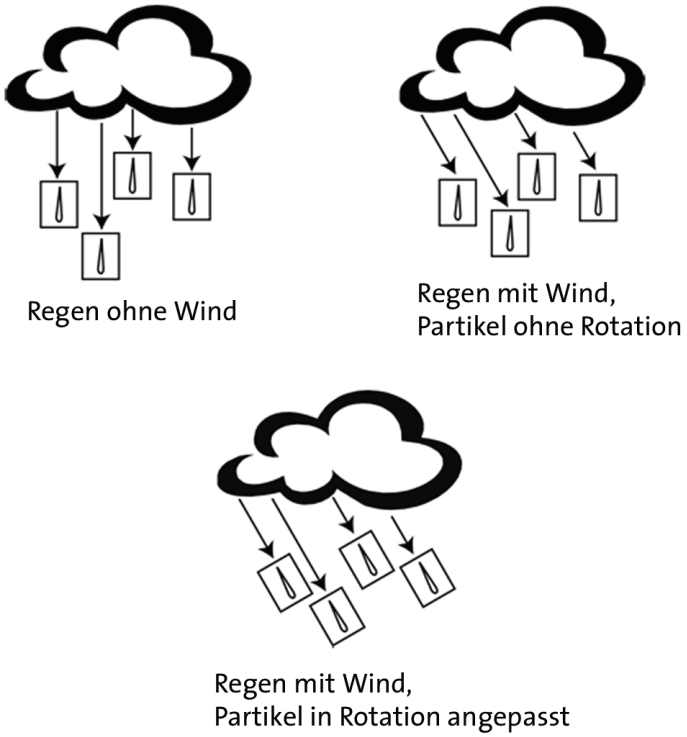


Abbildung 66: Auswirkungen des Parameters „Rotation“

Das Menü, in dem die generellen Darstellungsoptionen angepasst werden können, beinhaltet einen Schieberegler zur Einstellung des Schwellwerts für das Level-of-Detail. Über diesen kann die Entfernung eingestellt werden, bis zu der die hohe Auflösung der Landschaftskacheln dargestellt wird. Weiterhin finden sich zwei Schieberegler zur Definition der Sichtbarkeitsgrenzen der Städtesymbole. Je nach eingestellter Entfernung zur Kamera werden die Städtesymbole früher oder später ausgeblendet. Über einen letzten Schieberegler kann die Höhe der Städtesymbole über der Landschaft global angepasst werden.

Die Abbildungen der in diesem Abschnitt beschriebenen Menüs befinden sich in der Bedienungsanleitung in Kapitel 7.

6.9 Erstellen der Einzelbilder

Bereits kurze Zeit nach Anfang der Entwicklung kristallisierte es sich heraus, dass ein Ausspielen der Animation über eine Video-Capture-Karte nicht zu überzeugenden Ergebnissen führen würde. Auf Grund der durch die Anzahl der darzustellenden Polygone sowie der ständig variierenden Menge der zu sehenden Texturen permanent schwankenden Framerate kommt es zu plötzlichen Geschwindigkeitsänderungen innerhalb der Animation, die eine pumpenartige Kamerabewegung bewirken. Das ist für die live Ausspielung der Animation ungeeignet. Deswegen wurde bereits zu Beginn der Entwicklung die Möglichkeit getestet, die auf der Grafikkarte berechneten Bilder aus dem Framebuffer der Grafikkarte auf die Festplatte zu schreiben. Auf diese Weise ist es möglich selbst aufwändige Wetterdarstellungen, die nicht mehr mit den optimalen 25 FPS laufen, als Einzelbilder auszugeben und zu einer flüssigen Animation zusammenzusetzen.

Genau für diesen Zweck existiert ein entsprechender Channel im Funktionsumfang von Quest3D. Dieser speichert das auf der Grafikkarte erzeugte Bild an einen beliebig definierbaren Platz, zum Beispiel auf einem lokalen Datenträger. Zur Steuerung der Renderlogik kommt ein kleines LUA-Skript zum Einsatz, das beim Betätigen der Taste R einen „Datei speichern“-Dialog aufruft, in dem der Pfad für das Ausgangsmaterial definiert wird, dann die gesamte Animation automatisch zurückspult und anschließend Frame für Frame die gesamte Länge des konfigurierten Wetterflugs an die angegebene Stelle abspeichert. Die Nummerierung der Einzelbilder wird dabei ebenfalls durch das LUA-Skript an Hand der Framezahl der Animation durchgeführt.

Bei diesem Prozess werden Vollbilder generiert, die für das Fernsehformat ungeeignet sind. Dies macht eine Nachbearbeitung der Bildsequenz mit einer Konvertierung des Bildmaterials in Halbbilder in der Postproduktion nötig.

6.10 Integration der Logikbausteine zum Runtime-Modul

Das folgende Kapitel beschreibt das Zusammenfügen der in den vorangegangenen Abschnitten beschriebenen einzelnen Logikmodule zum vollständigen Runtime-Modul.

In der Start-Channel-Group werden die einzelnen, in separaten Channel Groups erstellten Logiken nacheinander aufgerufen. Als erstes werden die Kameraberechnungen durchgeführt und das die gesamte Szene beleuchtende gerichtete Licht gesetzt. Im Anschluss wird der Ladeprozess der Landschaftsgeometrien aufgerufen, der nach erfolgter Initialisierung die einzelnen Kacheln bei jedem Baumdurchlauf aufruft um sie anzuzeigen. Die Städtenamen werden geladen und auf der Landschaft positioniert. Als nächstes werden die Wahrzeichen und die Umgebung bestehend aus Wolkenkuppel, Bodendummy und dem Entfernungsdunstobjekt aufgerufen und dargestellt. Es folgen Wind, Niederschlag und Blitze. Bis zu diesem Zeitpunkt ist die Reihenfolge, in der die Elemente aufgerufen wurden, ohne Belang.

Nach dem nun die 3D-Szene fertig dargestellt ist, kommt an nächster Stelle die Integration der Logik zum Erstellen der Einzelbildsequenz und erst im Anschluss die gesamte Menülogik mit all ihren einblendbaren Schiebereglern und Knöpfen. Dies hat den Vorteil, dass selbst wenn vor dem Abspeichern der Bilder das Menü nicht ausgeblendet wurde, das Menü trotzdem nicht mit gerendert wird, weil es im Bildaufbau erst nach der Einzelbilderstellung aufgerufen wird.

Zusätzlich zu der gesamten, nacheinander aufgerufenen Logik, die bei jedem Baumdurchlauf einmal komplett aufgerufen wird, gibt es noch eine kleine zusätzliche Logik, die nur einmalig direkt beim Start des Runtime-Moduls das Fenster der Applikation auf die PAL-Ausgabegröße von 768 x 576 Pixel bringt.

6.11 Zusammenfassung

In den vorangegangenen Abschnitten wurde die Erstellung der Komponenten, aus denen das Runtime-Modul besteht, genauer beschrieben. Zunächst wurden die Logiken zur Auswertung der im Authoring-Modul angegebenen Daten und die zur Erstellung der 3D-Szene notwendigen Ladevorgänge für Geometrien, Städtesymbole und Texturen implementiert, die direkt nach dem Start des Runtime-Moduls ausgeführt werden. Die automatische Berechnung des Animationspfads der Kamera erfolgt ebenfalls in der Startphase der Applikation. Weitere Bestandteile des Runtime-Moduls sind die Festlegung der Kameraausrichtung über den zeitlichen Verlauf der Animation sowie die Logiken zu anwenderseitigen Einstellmöglichkeiten der Wetterphänomene in ihrer Ausprägung. Die Funktionalität des Runtime-Moduls endet mit der Logik zur Erstellung des finalen Bildmaterials.

7 Bedienungsanleitung

Im Folgenden wird die Erstellung eines Wetterflugs an einem Beispiel erklärt. Zur Konfiguration eines Wetterflugs müssen die Informationen über die Flugstrecke, die Temperaturen, Wolkeninformationen in Form von Bildern, die Windrichtung und den vorhandenen Niederschlag zur Verfügung stehen.

Zuerst wird das auf dem TriVis-System berechnete Wolkenbild über einen Photoshop-Automatismus in das Echtzeitformat DDS konvertiert und abgespeichert. Anschließend wird es in abgedunkelter Form in eine separate Datei als Wolkenschattenbild gespeichert.

Nun werden die globalen Parameter des Flugs im Authoring-Modul der „echtwetter"-Applikation festgelegt. Die genauen Eingabemöglichkeiten wurden bereits in Kapitel 5 beschrieben. Falls auf dem zu erstellenden Flug Niederschlag zu sehen ist, muss noch eine entsprechende Emittergeometrie für das Partikelsystem in 3ds max erzeugt und in Quest3D importiert werden.

Nachfolgend wird das Runtime-Modul gestartet. Nach erfolgreichem Start des Runtime-Moduls muss zuerst das Licht im Beleuchtungsmenü (Taste F9) durch Drücken der „Anwenden"-Schaltfläche aktiviert werden. Nach dem Laden befindet sich die Kamera am Ende der aus den Wegpunkten berechneten Flugstrecke im letzten Frame der Animation, so dass die Animation zunächst mit der Backspace-Taste zurückgespult werden muss. Im Folgenden kann die Animation durch Betätigung der Leertaste beliebig gestartet oder gestoppt werden, bei Erreichen des letzten Frames stoppt die Animation automatisch.

Nach dem Start der Applikation befindet sich der Anwender im „Freie Kamera"-Modus, was an dem Informationsfeld in der linken oberen Bildschirmecke zu erkennen ist (Abbildung 71). In diesem Modus kann der Anwender mit der Maus den Kamerblick frei einstellen. Mit Hilfe der freien Kamera wird nun im ersten Frame der Animation ein ansprechendes Startbild festgelegt und mit der K-Taste ein Keyframe gesetzt. Die Animation wird abgespielt und zu beliebigen Zeitpunkten können zusätzliche Kameraeinstellungen durch Setzen weiterer Keys abgespeichert werden. Die Ausrichtung der Kamera wird zwischen den gesetzten Key weich interpoliert. Eine gute Animation kommt mit möglichst wenigen Keys aus. Zumeist benötigt ein linearer Flug über zwei Wegpunkte auch nur zwei Keys. Der fertige Flug kann im „Animierte Kamera"-Modus (Taste 2) begutachtet werden. Bei Nichtgefallen können alle Keys mit der Tastenkombination K + Entf gelöscht werden und eine neue Animation mit der freien Kamera (Taste 1) erstellt werden.

Im Folgenden werden im Beleuchtungsmenü (Taste F9) das Licht für die Kacheln, die Farben der Wolken, des Himmels, des Nebels sowie die Ausprägungen des Nebels eingestellt (Abbildung 67).

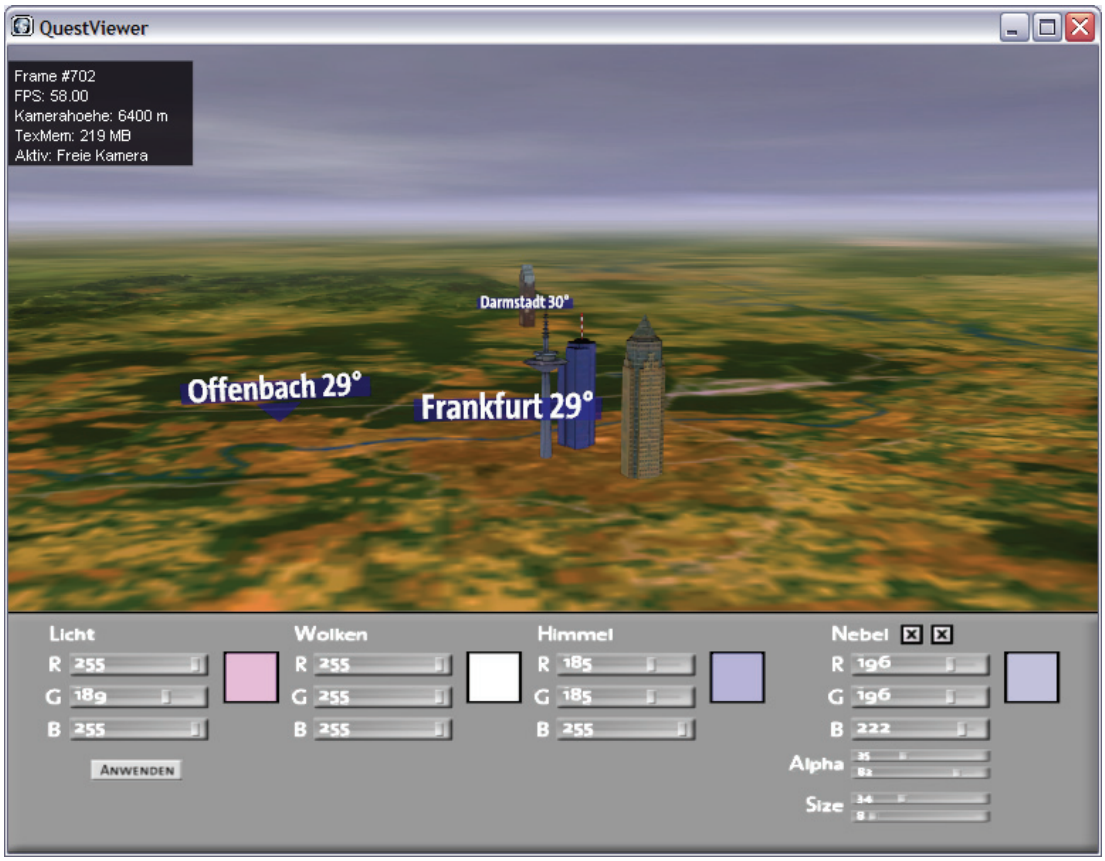


Abbildung 67: Beleuchtungsmenü

Die Taste F11 ruft das Menü für den Niederschlag auf. Sofern ein Partikelemitter vorhanden ist, kann durch Anklicken der Checkbox „Partikel“ der Niederschlag eingeschaltet werden. Über die Textur lässt sich die Art des Niederschlags, ob Regen oder Schnee, und die Dichte bestimmen sowie deren Ausprägung über die Schieberegler genauer definieren. Die Windrose dient zur Bestimmung der Ablenkungsrichtung der Niederschlagspartikel; es ist dabei darauf zu achten, dass der Schieberegler für die Windstärke nicht auf null steht, da sonst keinerlei Auswirkungen zu sehen ist. Oft geht Niederschlag mit einem Gewitter einher, die in der „echtwetter“-Applikation durch auf die Landschaft niedergehende Blitze dargestellt werden. Blitze können im Verlauf der Animation durch Drücken der Taste B und das Klicken mit der Maus auf den Boden zeitlich und örtlich exakt positioniert werden. Die gesetzten Blitze lassen sich mit der Tastenkombination B + Entf wieder löschen (Abbildung 68).

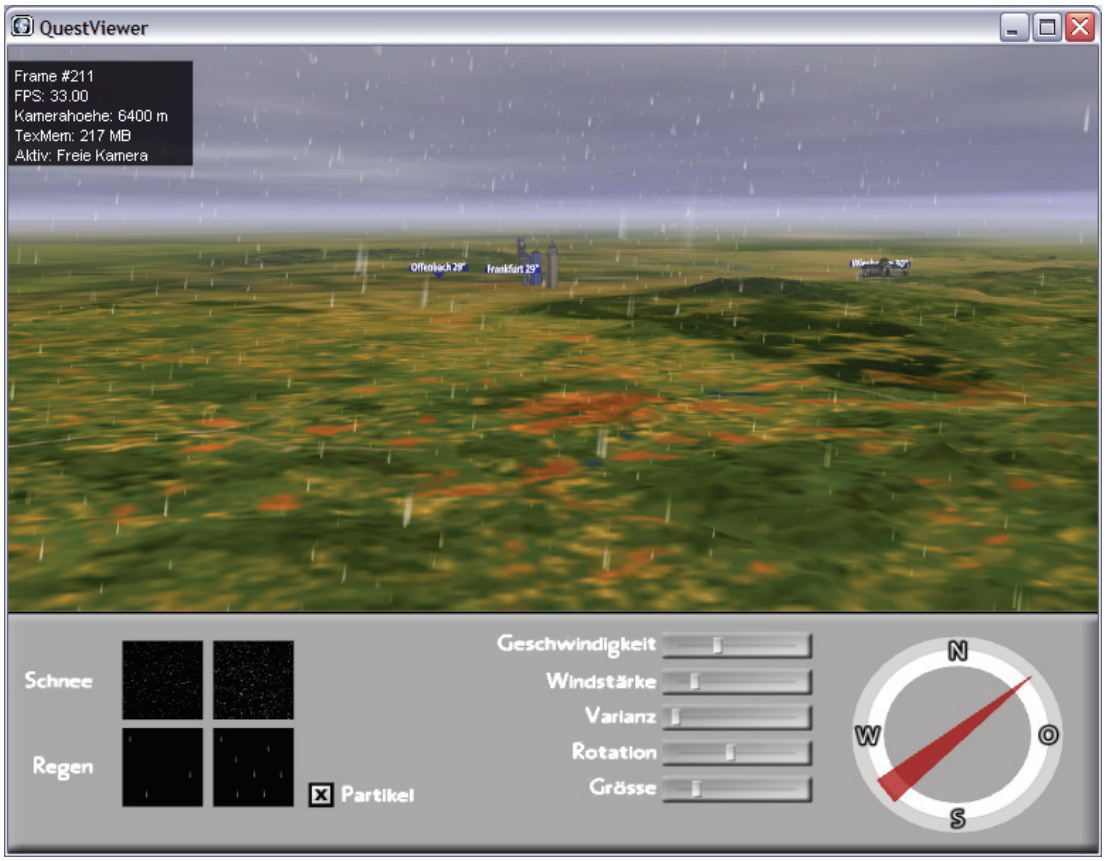


Abbildung 68: Menü Niederschlag

Zum Abschluss wird die Darstellungsqualität der Landschaftsgeometrien über den Regler „LoD“ im Menü „Darstellungsqualität“ (Taste F12) so lange erhöht, bis das Umschalten zwischen den Detailstufen der einzelnen Kacheln nicht mehr wahrnehmbar ist. Damit ist die Konfiguration des Wetterflugs abgeschlossen und der Flug kann ausgegeben werden. Dazu wird durch Betätigung der Taste R ein „Speichern unter“-Dialog aufgerufen, in dem der Speicherort sowie die Dateinamen der Einzelbilder festgelegt werden (Abbildung 69).

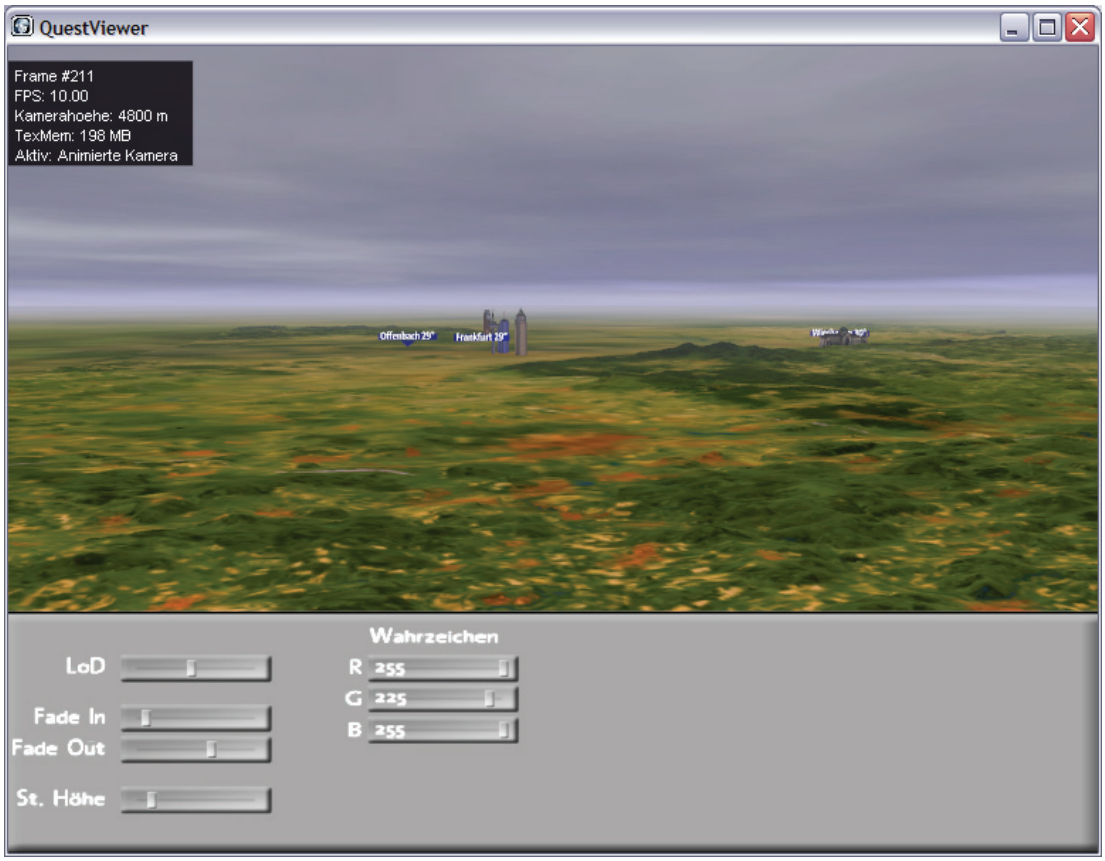


Abbildung 69: Menü Darstellungsqualität

Da die „echtwetter“-Applikation Vollbilder ausgibt, die nicht zur Darstellung auf dem Fernseher geeignet sind, ist eine Bearbeitung der Bilder in einer Postproduktionssoftware nötig. Dabei werden die entstanden Vollbilder in fernsehtaugliche Halbbilder umgewandelt (Abbildung 70).

Abbildung 70: Halbbild

8 Testphase

8.1 Performancetests

Der folgende Abschnitt befasst sich mit Performancetests, die während der Entwicklung durchgeführt wurden. Zur Bewertung der Leistungsfähigkeit der „echtwetter“-Anwendung wurden zwei Werte herangezogen, die Bildrate (Frames pro Sekunde [FPS]) und der verfügbare Grafikkartenspeicher.

Die Bildrate entspricht der Anzahl der Durchläufe durch den gesamten Logikbaum der Applikation inklusive aller angeschlossenen, dynamisch hinzu geladenen Channel Groups. Dieser Wert wird über den DX8 Sysinfo Channel direkt in der Anwendung ausgegeben (Abbildung 71).

Auch der noch zur Verfügung stehende Grafikkartenspeicher wird über einen solchen DX8 Sysinfo Channel ausgewertet. Der verfügbare Speicher variiert je nach verwendeter Grafikkarte und dem darauf befindlichen Angebot an Speicher (auf aktuellen Consumer-Grafikkarten finden sich zwischen 64 und 256 MB Speicher) sowie der in der Anwendung jeweils zu sehenden Geometrie und den zugehörigen Texturen, die in eben diesem Speicher Platz finden müssen.

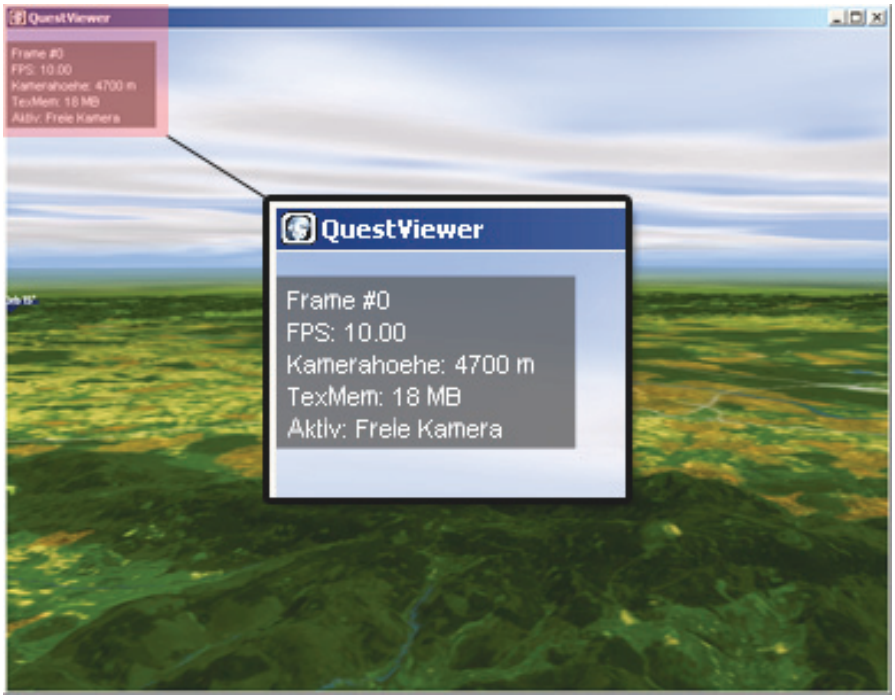


Abbildung 71: Anzeige von Framerate und verfügbarem Grafikkartenspeicher

Zur Ermittlung der Leistungsfähigkeit der Applikation in Bezug auf die Bildwiederholrate, wurde eine Testflugstrecke eingerichtet, die in den Performancetests immer wieder als Referenz diente. Dabei wurde eine Strecke über bergiges Gelände ausgewählt, da dort die Landschaftsgeometrien sehr hoch aufgelöst sind und dort deshalb extrem viele Polygone gleichzeitig dargestellt werden müssen. Die ausgewählte Flugbahn erstreckte sich von Wiesbaden nach Frankfurt über den Taunus hinweg nach Friedberg. Diese Flugroute macht das Laden von insgesamt neun Landschaftskacheln nötig und entspricht somit in den Ausmaßen einem real zu produzierendem Wetterflug für Hessen. In den einzelnen Abschnitten der Entwicklung wurde dieser eingerichtete Flug immer wieder abgeflogen, um die Auswirkungen der zuletzt vorgenommen Veränderungen auf die Bildrate zu ermitteln.

Die Bildrate ist für das Erstellen des finalen Bildmaterials nicht ausschlaggebend, da ein von der Framerate unabhängiger Prozess implementiert wurde, der die Einzelbilder auf die Festplatte schreibt. Jedoch wird auf diese Weise die Bedienbarkeit der Applikation gewährleistet, die bei zu niedrigen Bildwiederholraten nicht mehr zuverlässig funktioniert, da Knöpfe und Regler nicht mehr richtig reagieren und auch das Setzen von Blitzen sowie die Einstellungen zum Partikelsystem schwierig werden.

Der zur Verfügung stehende Speicher der Grafikkarte begrenzt die Größe der im Runtime-Modul zu ladenden Szene. Sämtliche sichtbare Geometrie sowie die darzustellenden Texturen müssen zur Laufzeit in diesem Speicher gehalten werden. Neben den

Texturen für die Wolkendarstellung, die Sprites des Niederschlags und die Texturen der Wahrzeichen nehmen vor allem die Texturen der Landschaftsgeometrie mit ihrer sehr hohen Auflösung einen großen Teil des verfügbaren Speichers ein. Ein noch größerer Anteil des Speichers wird durch die sehr hoch aufgelösten Kacheln der Landschaft mit bis zu 250.000 Polygonen pro Kacheln in Beschlag genommen.

Um die Produktionstauglichkeit der Applikation zu gewährleisten, wurden immer wieder in der Strecke variierende Testflüge durchgeführt und der jeweilige Bedarf an Speicher gemessen. Durch diese Testflüge wurde dann auch die maximale Größe des zu ladenden Gebiets ermittelt, die bei der gegenwärtig verwendeten Hardware 16 Landschaftskacheln beträgt. Dieser Wert kann nach oben hin variieren, abhängig von der Auflösung des Geländes, das überflogen wird. Die Landschaftsgeometrie im Norden Deutschlands besteht sogar in der hohen Auflösung zumeist aus nicht mehr als 20.000 Polygonen. Hier können dann weitaus mehr als nur 16 Kacheln geladen werden.

8.2 Produktionstest

Neben den im vorangegangenen Kapitel beschriebenen Performancetests, die iterativ während des gesamten Entwicklungsprozesses durchgeführt wurden, um die Bedienbarkeit und Produktionstauglichkeit der Applikation zu gewährleisten, wurden zusätzlich Produktionstests durchgeführt, die die grafische Qualität der „echtwetter"-Anwendung sowie den eigentlichen Produktionsablauf auf die Probe stellten.

Die grafische Qualität des Wetterflugs wurde ständig durch einen zur Seite gestellten Grafiker überwacht. Vor allem die Ausprägungen der Wetterphänomene wurden immer wieder auf ihre Tauglichkeit hin überprüft.

Da es sich bei der „echtwetter"-Applikation um eine Computeranwendung handelt und die Konvertierung von computergeneriertem Bildmaterial ins Fernsehformat nicht unproblematisch ist, wurde auch die in der Postproduktion nötige Bildaufbereitung mehrfach getestet und in ihren Einstellungen optimiert. Zum Beispiel wird ein Effekt-Filter auf das ins Halbbildformat konvertierte Ausgangsmaterial angewandt, der das bei Halbbildern auftretende Texturflimmern reduzieren soll. Das Endergebnis wurde auf einem Testmonitor ausgegeben um die beim Zuschauer ankommende Qualität realistisch einschätzen zu können.

Der endgültige Produktionstest, bei dem die für das „max"-Wetter zuständigen Grafiker des hr den zuvor erlernten, kompletten Arbeitsablauf zur Erstellung eines Wetterflugs in Quest3D anwenden mussten, wurde während des Hessentags in Weilburg vom 17.06. bis zum 26.06.2005 durchgeführt. Der mit Quest3D erstellte Wetterflug wurde während dieser Zeit im hr-Fernsehen täglich gesendet. Während dieser Produk-

tionstests wurde der zeitliche Aufwand einer Wetterflugproduktion mit der „echtwetter“-Anwendung getestet um einen Vergleichswert zu der herkömmlichen Erstellung eines Wetterflugs zu bekommen.

8.3 Zusammenfassung

Die während der Entwicklung der Applikation durchgeführten Performancetests, die die Produktionstauglichkeit sicherstellen sollten, führten zu dem Ergebnis, dass die Applikation in der entwickelten Form einsatzfähig ist. Der am Ende der Entwicklungszeit durchgeführte Produktionstest unter den Bedingungen einer realen Fernsehproduktion bestätigte dieses Ergebnis.

9 Darstellung der Ergebnisse und Ausblick

Die Entwicklung einer interaktiven 3D-Echtzeitanwendung zur Simulation von Wetterflügen begann mit der erfolgreichen Erstellung eines Prototyps. Anhand des Prototyps konnten ansatzweise die grafische Qualität sowie die technischen Möglichkeiten einer in Quest3D erstellten Anwendung aufgezeigt werden, so dass auf dieser Basis die eigentliche Anwendung zur Simulation von Wetterflügen implementiert werden konnte.

Nach erfolgter Anforderungsanalyse und der daraus resultierenden Definition des Projekts wurde die Anwendung in zwei getrennten Modulen entwickelt. Zur globalen Konfiguration eines Wetterflugs wird dabei das in PHP umgesetzte Authoring-Modul verwendet. Dieses schreibt die Parameter eines Flugs in eine Datenbank, welche dann vom Runtime-Modul, das zur Darstellung des eigentlichen Wetterflugs dient, ausgewertet werden. Die für das Runtime-Modul nötigen Logikkomponenten zum dynamischen Laden und Anzeigen der 3D-Objekte, zur Darstellung der Wetterphänomene, und die nötigen Logiken zur Anpassung der Ausprägungen der Licht- und Wettereinstellungen wurden in einzelnen Bausteinen entwickelt, die dann zu einer funktionsfähigen Anwendung zusammengefügt wurden.

Die für die Erstellung des Bildmaterials nötige Renderlogik gibt die im Runtime-Modul konfigurierte Wetteranimation zum Schluss als Einzelbilder aus, die anschließend noch durch einen Postproduktionsprozess in ein fernsehgerechtes Format konvertiert werden müssen.

Bereits während der Entwicklung der „echtwetter“-Anwendung entstanden Möglichkeiten zur weiterführenden Optimierung in den Bereichen Benutzerfreundlichkeit, Darstellungsqualität, automatisierte Einbindung von Daten und Ausgabemöglichkeiten.

Einer der vordringlichen Verbesserungsansätze in Bezug auf die Benutzerfreundlichkeit ist die zurzeit umständliche Erstellung einer Emittergeometrie zur Darstellung von Niederschlag in 3ds max und dem nötigen Import dieser Geometrie nach Quest3D. Dieser Punkt geht einher mit der besseren, automatischen Integration von Wetterdaten. Es wäre zum Beispiel denkbar, einen Wetterdatensatz, der direkt vom DWD im ASCII-Format angeliefert wird, in Quest3D einzulesen, auszuwerten und daraus entsprechende Partikelemitter zu generieren. Dies würde auch den Zwischenschritt der Berechnung einer Niederschlagskarte auf dem TriVis-System überflüssig machen.

Eine in der Implementierung schon angedachte Verbesserung ist die bereits zurzeit der Produktionstests von den Anwendern geforderte Möglichkeit Kamerakeys in eine externe Datei abzuspeichern und auch wieder laden zu können.

Im Bereich der automatischen Integration von Wetterdaten gibt es zwei Verbesserungsansätze. Zum einen ist die automatische Übernahme der Temperaturdaten für die Städte geplant, welche die manuelle Eingabe über eine Eingabemaske des Authoring-Moduls ersetzen würde. Des Weiteren wäre ein Verfahren zur automatischen Auswertung der wolkenbezogenen DWD-Daten und der gleichzeitig volumetrischen Darstellung der Wolken ein mit Nachdruck geforderter Verbesserungspunkt, der jedoch in der Umsetzung extrem aufwändig sein wird.

Im Bereich der Ausgabemöglichkeiten wird ein verbessertes Renderverfahren angestrebt, das es ermöglicht direkt im Halbbildverfahren Bilder zu speichern, um den bis dato nötigen Schritt in der Postproduktion gänzlich zu ersetzen.

Der mit der „echtwetter“-Applikation erstellte Wetterflug wurde in einer Produktionstestphase im laufenden Programm des Hessischen Rundfunks über einen Zeitraum von zehn Tagen erfolgreich getestet. Der Zeitaufwand von drei bis vier Stunden zur Erstellung eines Wetterflugs mit 3ds max wurde von der „echtwetter“-Applikation mit einer Erstellungszeit des gleichen Wetterflugs durch einen erfahrenen Anwender innerhalb einer Stunde bei weitem unterboten. Es ist zu erwarten, dass in absehbarer Zeit die Produktion der Wetterflüge in Hessen mit der in dieser Diplomarbeit erstellten Echtzeitapplikation vorgenommen wird.

10 Literatur- und Quellenverzeichnis

[1] Helmut Haase, Markus Bock, Elke Hergenröther, Christian Knöpfle, Hans-Joachim Koppert, Florian Schröder, Andrzej Trembilski, Jens Weidenhausen: Meteorology Meets Computer Graphics - a Look at a Wide Range of Weather Visualisations for Diverse Audiences.
In: Computers & Graphics 24 (2000), 3, S. 391-397

[2] Freie Online Enzyklopädie “Wikipedia“

URL: http://de.wikipedia.org/wiki/Grafik-Engine

Stand: 05.05.2005

[3] Microsoft DirectX Homepage

URL: http://www.microsoft.com/windows/directx/default.aspx

Stand: 29.05.2005

[4] Pixvertex Homepage

URL: http://www.pixvertex.com/index.html

Stand: 29.06.2005

[5] Arnold AG Homepage

URL: http://www.arnold.de/index.php?id=96,206,0,0,1,0

Stand: 29.06.2005

[6] Quest3D Homepage

URL: http://www.quest3d.com/index.php?id=174

Stand: 10.05.2005

[7] Andy Tather

Pandasoft Homepage

URL: http://www.andytather.co.uk/Panda/directxmax.aspx

Stand: 29.6.2005

[8] Microsoft DirectX Homepage

URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/ddsfilereference/ddsfileformat.asp

Stand: 21.05.2005

[9] Michael Bender, Manfred Brill

Computergrafik – Ein anwendungsorientiertes Lehrbuch

Carl Hanser Verlag München Wien, 2003

[10] David Scherfgen

3D-Spieleprogrammierung – Modernes Game Design mit DirectX9 und C++

Carl Hanser Verlag München Wien, 2003

A Anhang

A.1 Zeitliche Steuerung der Kameraanimation

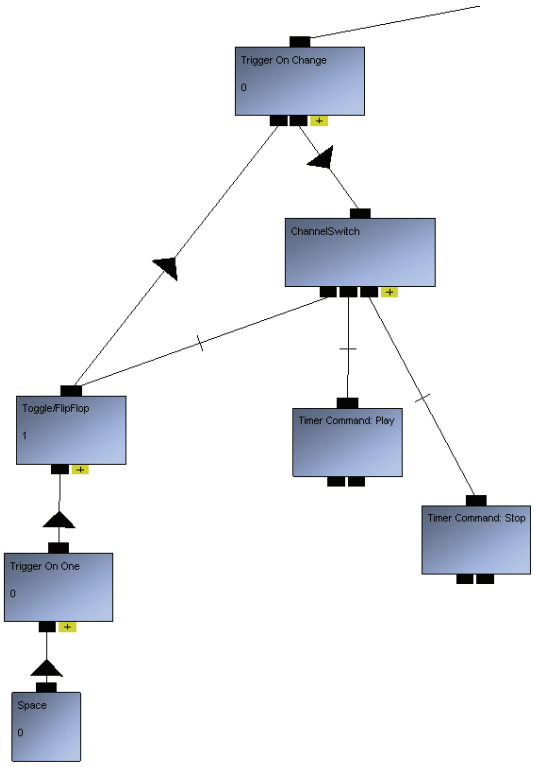


Abbildung 72: Logik zum Abspielen und Stoppen

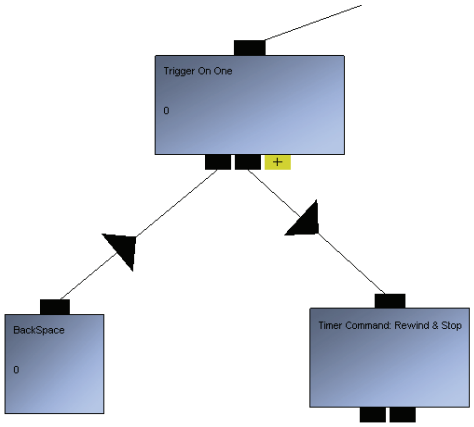


Abbildung 73: Logik zum Zurückspulen

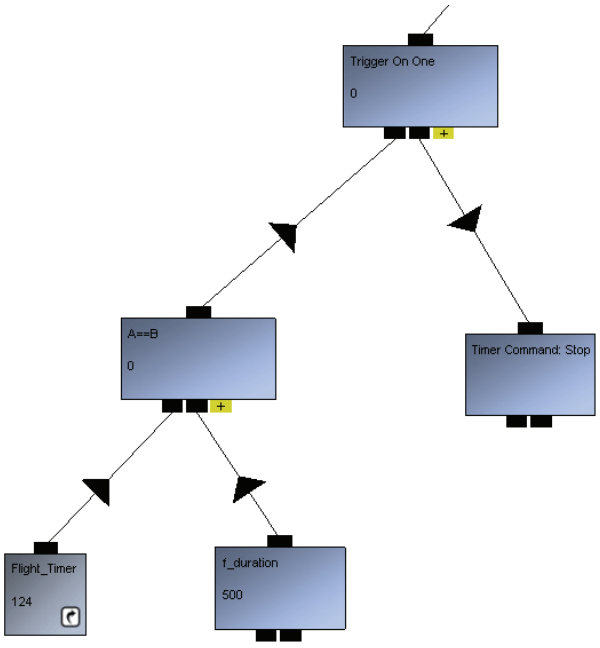


Abbildung 74: Logik zum automatischen Stoppen

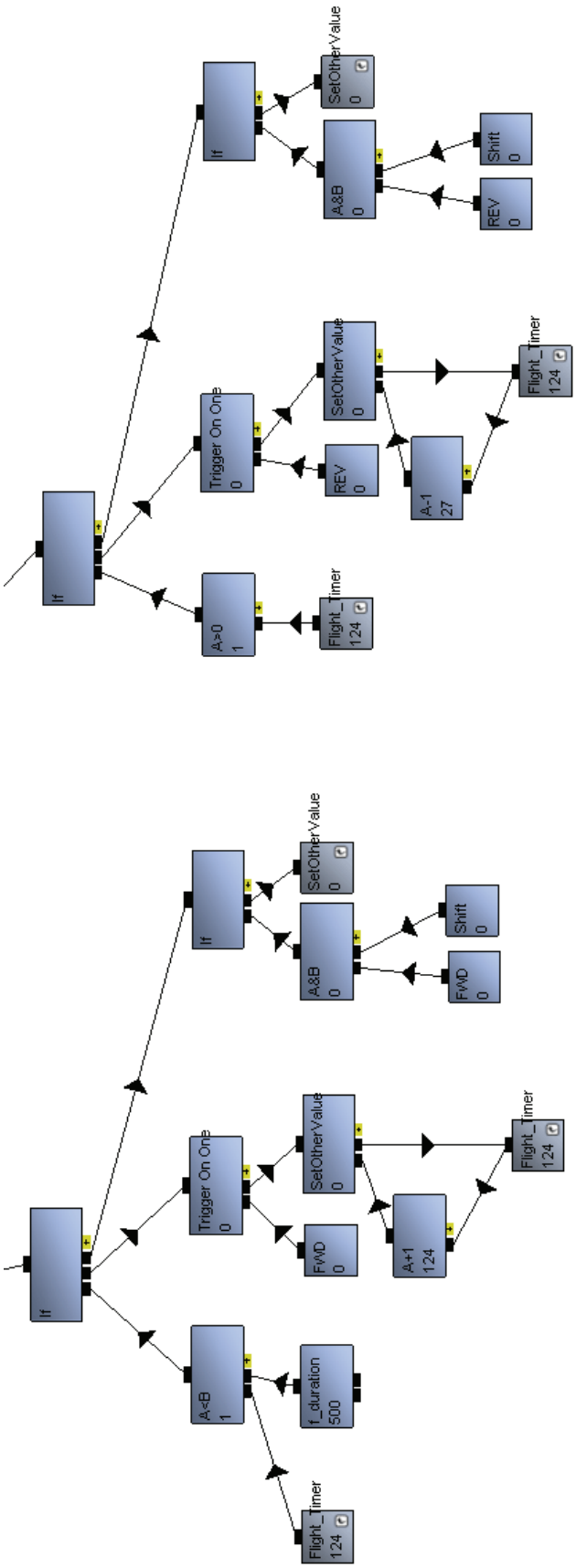


Abbildung 75: Logiken zum Vor- und Zurückspulen der Animation

A.2 Logik zum Laden der Kacheldaten

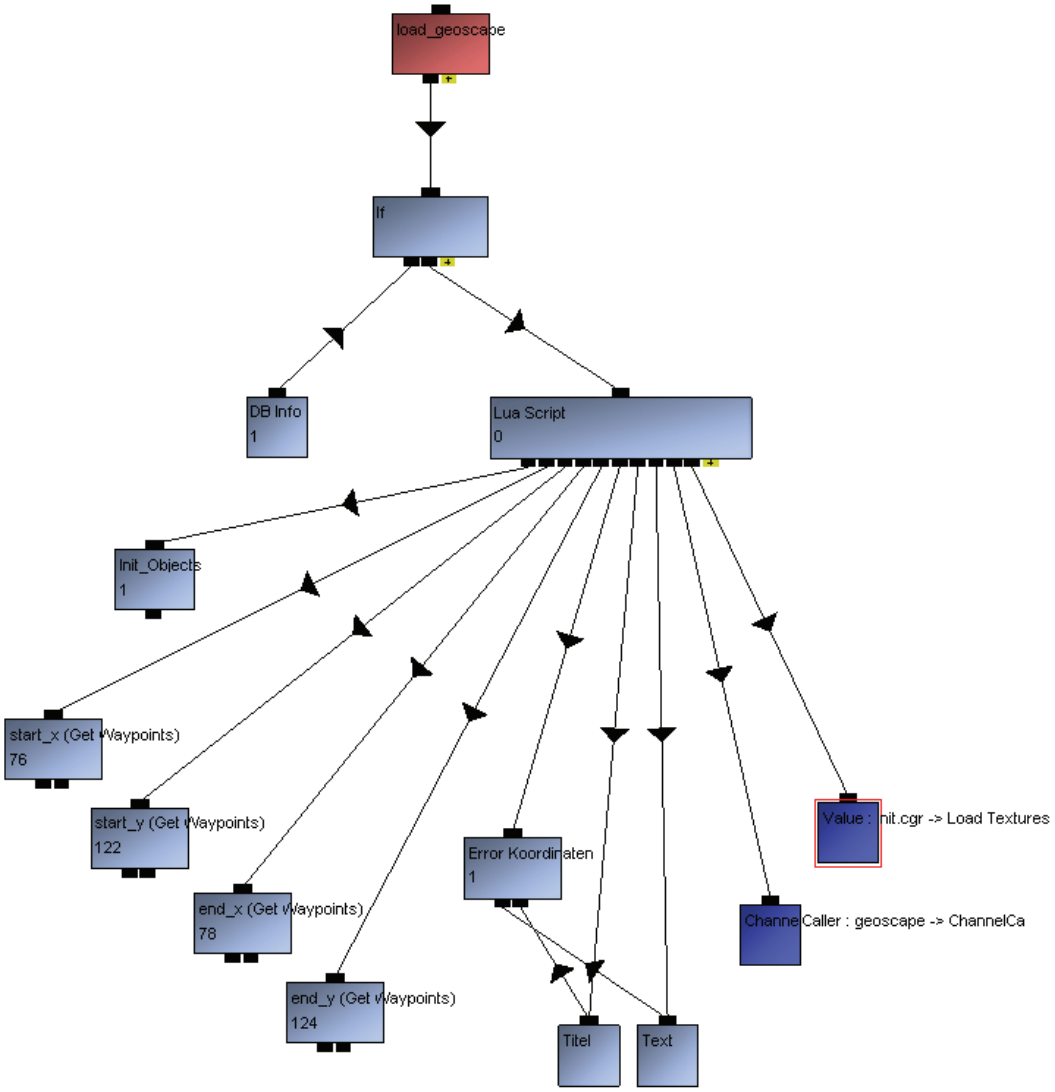


Abbildung 76: Logik zum Laden der Landschaftsgeometrie

A.3 Skript zum Laden der Kacheldaten

```
-- Skript zum Laden der Kacheldaten unter Angabe der Start- und End-
kachel
-- Version 1.3 erstellt am 04.02.2005 von René Nold
-- Letzte Änderung 05.04.2005

local i=0

-- CallChannel function is called when channel is called

function CallChannel()

   local initObjectsHandle=channel.GetChild(0)
   local initObjects=initObjectsHandle:GetValue()

-- Objekte schon geladen?

   if initObjects==0 then

-- Auslesen der Start- und Endkoordinaten

      local start_x=channel.GetChild(1):GetValue()
      local start_y=channel.GetChild(2):GetValue()
      local end_x=channel.GetChild(3):GetValue()
      local end_y=channel.GetChild(4):GetValue()

      local msgBoxHandle=channel.GetChild(5)
      local msgBoxTitle=channel.GetChild(6)
      local msgBoxText=channel.GetChild(7)

-- Fehlercheck: Startkoordinate größer als Endkoordinate?

      if start_x > end_x or start_y > end_y then
         msgBoxTitle:SetText("Fehler!")
         msgBoxText:SetText("Die Start Koordinate ist größer als die
         End Koordinate")
         msgBoxHandle:CallChannel()
      end

-- Zwischenspeichern der start_y Koordinate zur Wiederverwendung in
der inneren while Schleife

      local start_y_temp=start_y

-- Zwei verschachtelte while Schleifen zum Laden der einzelnen Ka-
cheln

      while start_x<=end_x do

         while start_y<=end_y do

-- Laden der Kachel Channel Groups in das Array geoscape an die Po-
sition i

            q.LoadChannelGroup("D:\\echtwetter\\CGR_Kacheln\\
            mona100_pol_x"..start_x.."_y"..start_y..".cgr",
            "geoscape",i)

            start_y=start_y+1
            i=i+1
         end
         start_y=start_y_temp
         start_x=start_x+1
      end
      msgBoxTitle:SetText("Geladen!")
      msgBoxText:SetText("Alle Kacheln wurde erfolgreich geladen!")
```

```
      msgBoxHandle:CallChannel()

-- Geometrien geladen, initObjects gleich 1 setzen.

      initObjectsHandle:SetValue(1)

-- Alle Kacheln geladen, starte das Laden der Texturen in den Chan-
nel Groups

      channel.GetChild(9):SetValue(1)

   end

-- Aufruf der geladenen Kacheln (wird bei jedem Baumdurchlauf durch-
geführt)

   local j=0

   while j<i do

      local kachel=channel.GetPublic(8,j)
      kachel:CallChannel()
      j=j+1

   end

end
```

A.4 Texturladelogik

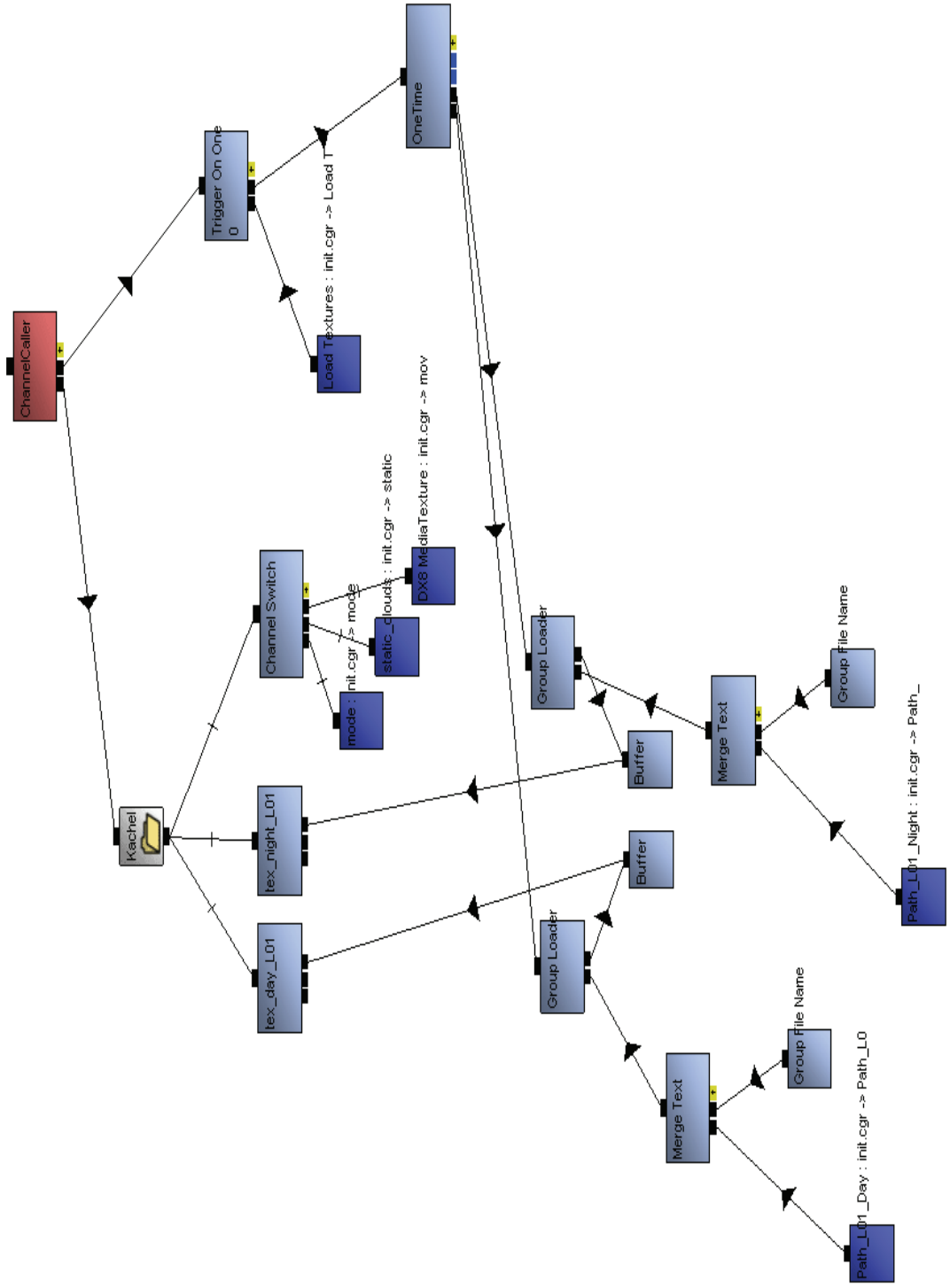


Abbildung 77: In Channel Group integrierte Texturenladelogik

A.5 Logik zum Laden der Städtesymbole

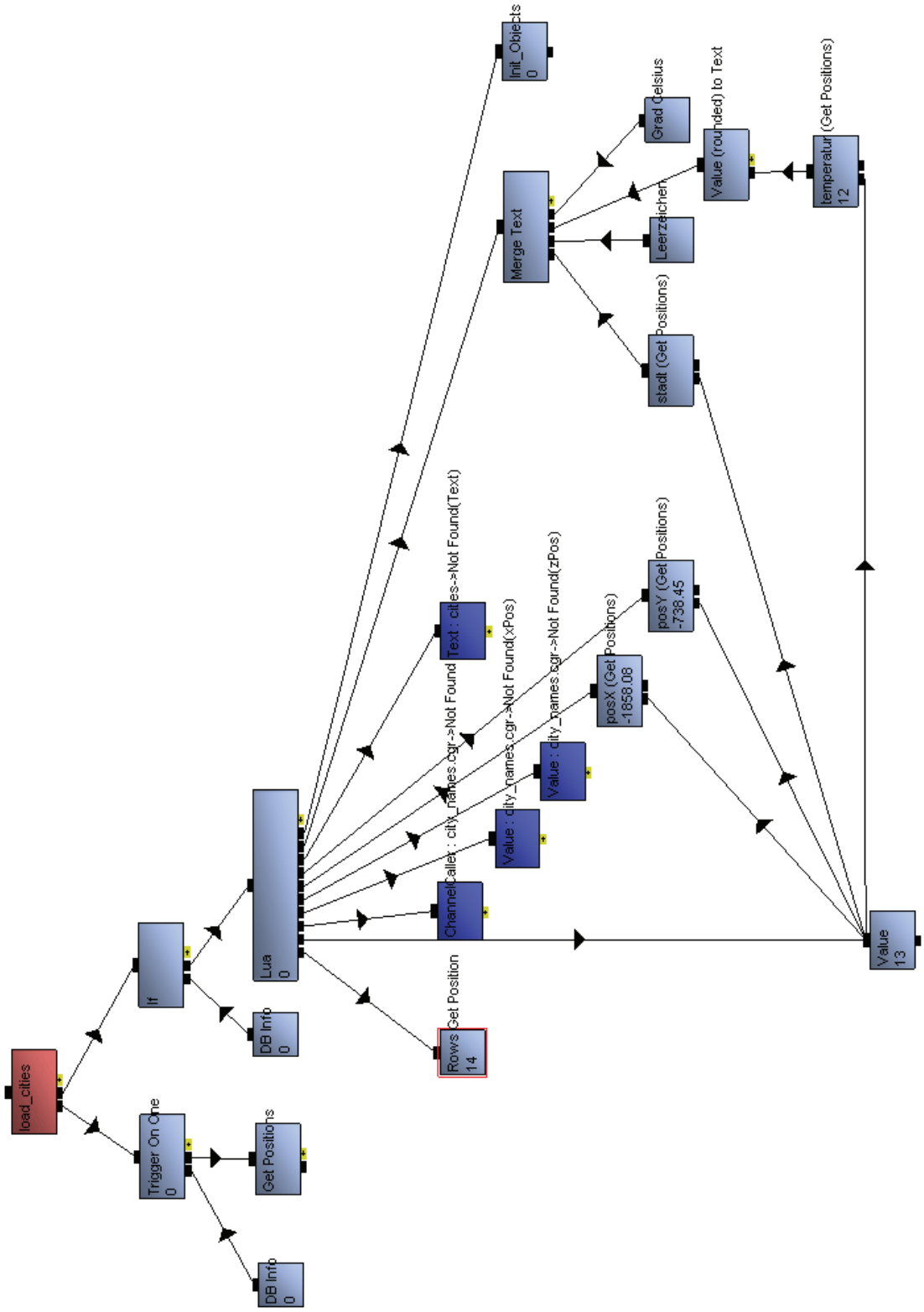


Abbildung 78: Logik zum Laden der Städtesymbole

A.6 Skript zum Laden der Städtesymbole

```
-- Skript zum Laden der Städtesymbole
-- Version 1.1 erstellt am 07.03.2005 von René Nold
-- Letzte Änderung 21.03.2005


function CallChannel()
--Initialisieren der Variablen initObjects und totalRows

   local initObjectsHandle=channel.GetChild(9)
   local initObjects=initObjectsHandle:GetValue()
   local totalRowsHandle=channel.GetChild(0)
   local totalRows=totalRowsHandle:GetValue()

--Wenn die Objekte noch nicht geladen wurden, dann mach, ansonsten
nicht
   if initObjects==0 then

      local i=0
      local j=0
      local x=0
      local z=0
      local name=""

--Laden der Stadtobjekte
      while i<=totalRows do
         q.LoadChannelGroup(".\\city_names.cgr", "cities", i)
         x=channel.GetPublic(3,i)
         z=channel.GetPublic(4,i)
         name=channel.GetPublic(7,i)

         x:SetValue(channel.GetChild(5):GetValue())
         z:SetValue(channel.GetChild(6):GetValue())
         name:SetText(channel.GetChild(8):GetText())
         j=channel.GetChild(1)
         j:SetValue(i)

         i=i+1
      end
      initObjectsHandle:SetValue(1)
   end

--Wenn Objekte geladen, dann ruf Sie bei jedem Baumdurchlauf auf
      local i=0
      while i<=totalRows do
         local object=channel.GetPublic(2,i)
         object:CallChannel()
         i=i+1
      end
end
```

A.7 Level-of-Detail-Logik

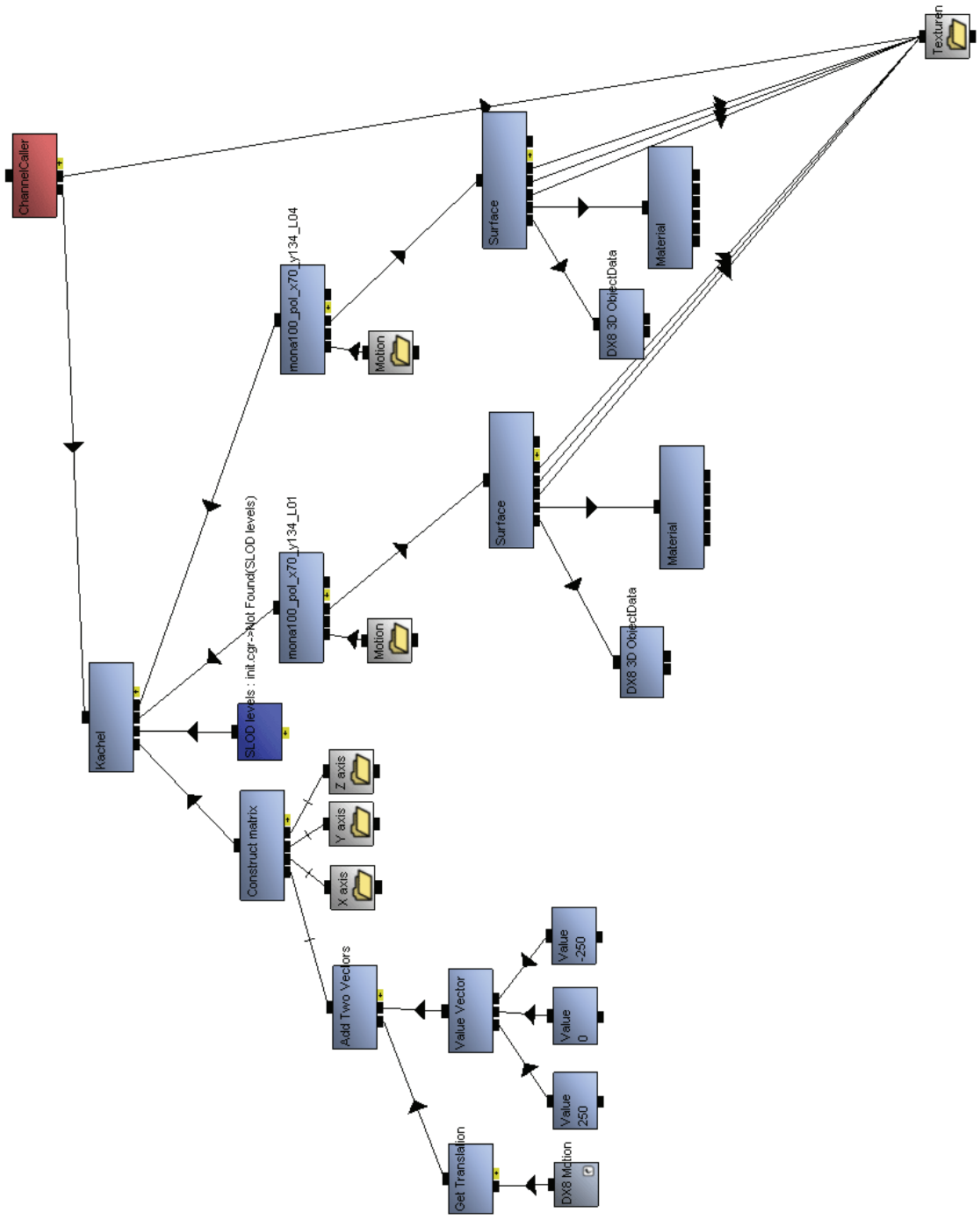


Abbildung 79: Implementiertes LOD

A.8 Logiken zur Anzeige des Städtenamensymbols

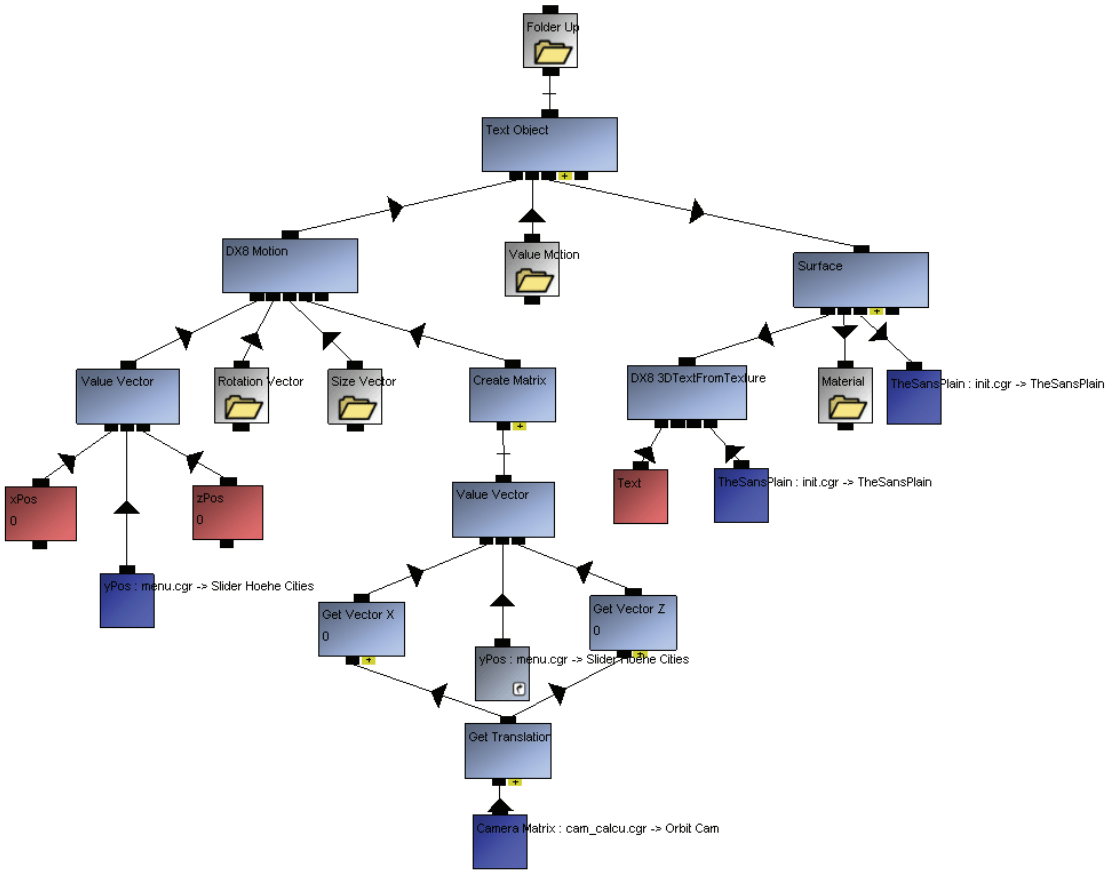


Abbildung 80: Logik zur Anzeige und Ausrichtung des Städtenamens

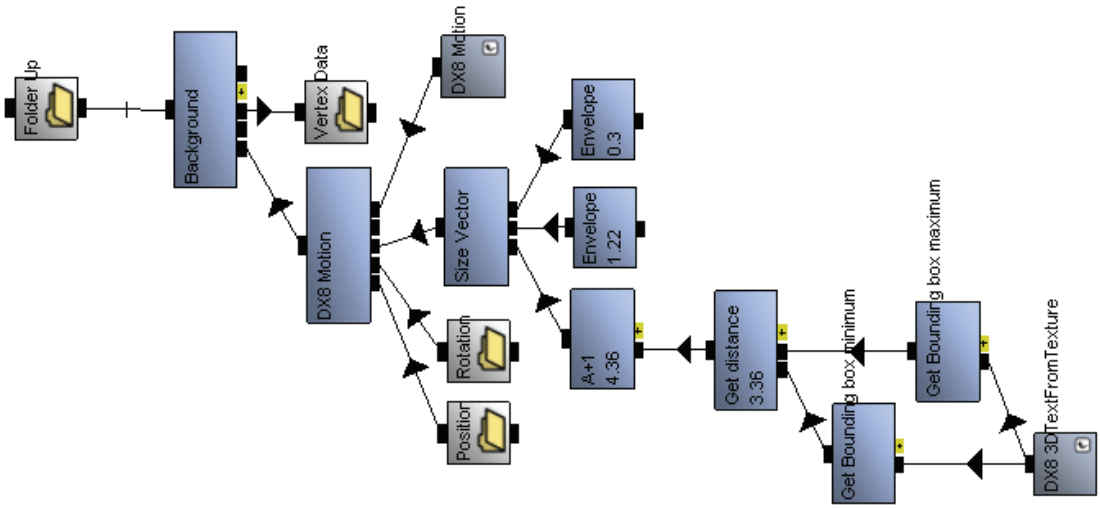


Abbildung 81: Berechnung der Größe des Texthintergrundobjekts

B Anhang CD

B.1 „echtwetter“ Quest3D Source Code Runtime-Modul

B.2 „echtwetter“ PHP/HTML Source Code Authoring-Modul

B.3 Quest3D 2.5a Demo Version

B.4 Quest3D Demo Projekte

B.5 Wetterflugfilme

B.6 Bildmaterial

B.7 Zusätzliche Quellen