Echtzeit-Gelände-Triangulierung mittels eines adaptiv-hierarchischen Interpolationsverfahrens

K. Dobler [*], A. Paul [†], G. Sachs [‡], Ch. Zenger [§]
Technische Universität München

Zusammenfassung

Es wird ein Echtzeit-Gelände-Triangulierungs-Algorithmus vorgestellt, der für ein Flugführungssystem mit synthetischer Sicht konzipiert wurde. Dieser Triangulierungs-Algorithmus bietet neben eines frei wählbaren und stetigen Level-Of-Detail-Verlaufs aufbauend auf einer hierarchischen Interpolation der Höhendaten des Geländes die Möglichkeit zur Datenkompression. Der Interpolationsfehler, der sog. hierarchische Überschuß, dient dabei als Kriterium, um redundante Daten zu identifizieren, und kann auch als ein Maß für die Güte der Geländeapproximation genutzt werden. Das System der synthetischen Sicht besteht aus zwei Grundelementen: eine computergenerierte, dreidimensionale Nachbildung des Geländes auf der Basis eines digitalen Geländemodells und integrierten Flugführungsanzeigen.

1 Einleitung

1.1 Hintergrund

Flugmanöver in Bodennähe wie Anflug und Landung stellen für Flugzeugbesatzungen eine anspruchsvolle Aufgabe dar, die sowohl ein entsprechendes Situationsbewußtsein als auch ausreichende Führungshilfen erfordert. Deshalb bedeutet schlechte Außensicht eine große Einschränkung für den bodennahen Flug, vor allem, wenn Flugzeuge nicht für den Allwetterflugbetrieb zugelassen und/oder Flugplätze nicht mit entsprechenden bodenseitigen Landehilfen ausgestattet sind. Daraus ergeben sich auch heute noch, wenn sich die Außensicht verschlechtert, sogar an gut ausgestatteten Flugplätzen Verspätungen und Absagen von Flügen, die augenscheinlich Nachteile für Passagiere und Fluggesellschaften mit sich bringen.

Neue Konzepte, die unter dem Begriff „Synthetische Sicht“ bekannt sind, werden heute als ein Mittel angesehen, die bereits aufgezeigten Auswirkungen von schlechten Sichtbedingungen auf den Flugbetrieb abzumindern oder ganz zu vermeiden [1]. Ein System, das seine Tauglichkeit in mehreren Flugversuchsreihen des Lehrstuhls für Flugmechanik und Flugregelung der Technischen Universität München unter Beweis gestellt hat, basiert auf der Generierung einer synthetischen Sicht mittels eines Computers. Dabei wird dem Piloten ein 3-dimensionales computergeneriertes Abbild der Landschaft mit integrierter Flugführungssymbolik angezeigt(Abb. 1, [2]). Die Flugzeugposition, die zur Darstellung des zur realen Welt korrespondierenden Bildausschnittes nötig ist, wird durch ein integriertes Präzisionsnavigationssystem ermittelt, welches Differential-GPS und Trägheitsnavigation miteinander verbindet [3]. Um die Echtzeitanforderungen bezüglich der Geländedarstellung erfüllen zu können, wird in einem neuen Ansatz, der Gegenstand dieses Beitrags ist, ein hierarchisches Interpolationsverfahren angewandt, das eine adaptive Echtzeittriangulierung des Geländes erlaubt.

Als Geländemodell wird eine zweigeteilte Datenbasis verwendet, die sich in Höhendaten und Oberflächenbeschreibungsdaten gliedert (Abb. 2). Die Höhendaten sind als ein gleichmäßiges Gitter mit einer Gitterweite von $3'' \times 3''$ abgelegt, die in Mitteleuropa einem Abstand von ca. 60×90 m entspricht. Die Oberflächenbeschreibungsdaten gliedern sich in Flächen-, Linien- und Punktobjekte (z.B. Wälder, Flüsse und Türme).

[*] Dipl.-Ing. Klaus Dobler, Lehrstuhl für Flugmechanik und Flugregelung
[†] Dipl.-Inform. Andreas Paul, Lehrstuhl für Ingenieuranwendungen in der Informatik und numerische Programmierung
[‡] Prof. Dr.-Ing. Gottfried Sachs, Lehrstuhl für Flugmechanik und Flugregelung
[§] Prof. Dr. Christoph Zenger, Lehrstuhl für Ingenieuranwendungen in der Informatik und numerische Programmierung

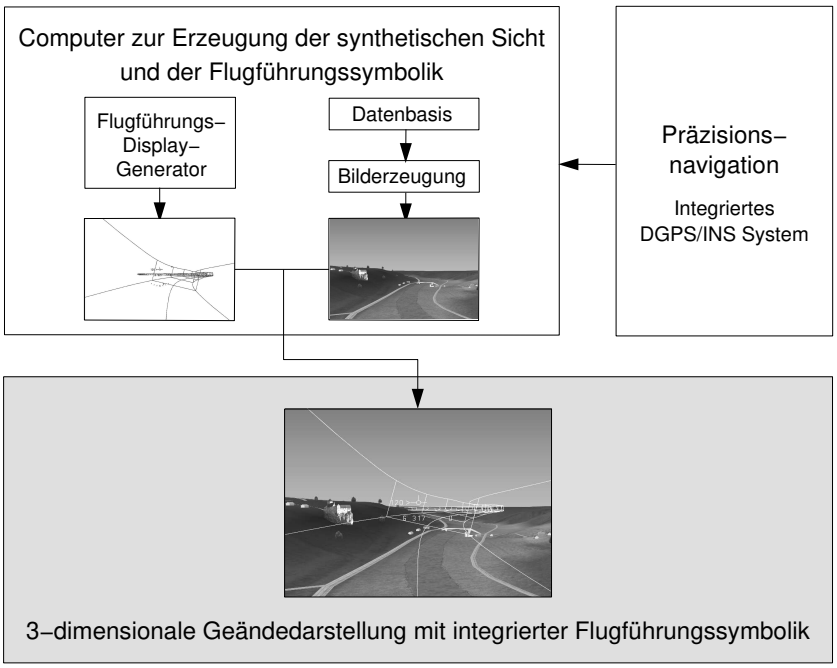

Abbildung 1: Konzept der synthetischen Sicht

1.2 Entwurfskriterien

Für den Einsatzzweck des Systems als neuartige Flugführungsanzeige ergeben sich folgende Entwurfskriterien:

- Da das System in einer Echtzeitumgebung arbeitet, muß eine Bildfrequenz von mindestens 25 Bildern/Sekunde erreicht werden, wobei zusätzlich zum Gelände auch Oberflächenobjekte und Flugführungssymbolik darzustellen sind.
- Das dargestellte Gelände soll so realitätsnah wie möglich sein, um das Situationsbewußsein des Piloten und seine Flugführungsmöglichkeiten zu verbessern.
- Die Sichtweite soll mindestens 25 km betragen. Um dies mit der oben geforderten Bildwiederholfrequenz zu erreichen, ist ein Level-Of-Detail-Verfahren für die Darstellung des Geländes erforderlich, das über eine einstellbare Fehlerschranke in Abhängigkeit von der Entfernung zum Beobachter kontrollierbar ist.
- Das System soll sowohl die Möglichkeit zur Datenkompression enthalten als auch mit komprimierten Daten arbeiten können, um den erforderlichen Festplattenplatz und Arbeitsspeicher zu reduzieren und somit die Kosten des Gesamtsystems zu senken.

2 Geländetriangulierung

Um diese Vorgaben erfüllen zu können, insbesondere Datenkompression und einfach zu kontrollierender Genauigkeitsgrad, wurde eine hierarchische Interpolationstechnik für die Geländetriangulierung gewählt. Diese Technik wurde zunächst im Rahmen einer Diplomarbeit ([4]) auf ihre Tauglichkeit im Einsatz zur Generierung einer synthetischen Sicht hin untersucht.

Das hier vorgestellte System wurde parallel zu dem von Lindstrom in [5] vorgestellten Verfahren entwickelt. Das Vorgehen der beiden Verfahren unterscheidet sich zwar im Detail, die erzeugte Triangulierung ist jedoch identisch. Lindstrom zeigt die Echtzeittauglichkeit seines Systems, indem er ein Gelände als Drahtgittermodell auf einem Zweiprozessor-Rechner vom Typ SGI Onyx RealityEngine^2 (150 MHz) darstellt. Das hier vorgestellte System wurde mit volltexturierten Polygonen bei zusätzlicher Belastung der Graphik durch weitere zu zeichnende Objekte (Gebäude, Head-Up-Display etc.) auf seine Echtzeittauglichkeit hin untersucht. Die verwendete Hardwareplattform war dabei ein Einprozessor-Rechner SGI Indigo^2 Workstation (195 MHz) mit MXI-Graphikkarte.

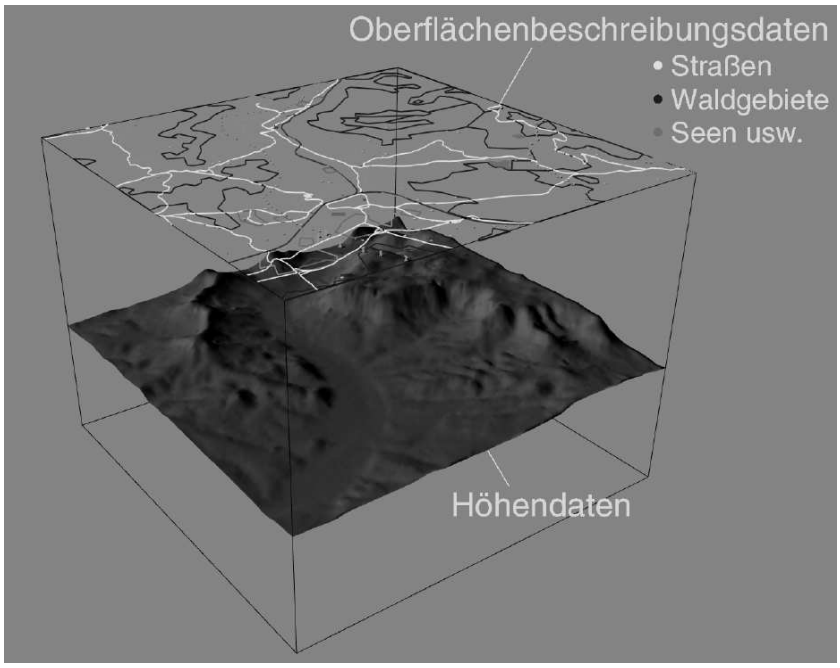

Abbildung 2: Digitales Geländemodell

Aufwendigere Techniken zur Geländetriangulierung, von denen eine z.B. in [6] vorgestellt wird, sind in den meisten Fällen weniger geeignet für den Echtzeiteinsatz.

Weitere Informationen über Datenkompression mit Hilfe hierarchischer Interpolation findet sich in den Arbeiten [7], [8] oder auch [9].

2.1 Hierarchische Interpolation

Um die als reguläres Gitter vorgegebenen Höhendaten hierarchisch interpolieren zu können, müssen die Höhendaten zunächst in rechteckige Gebiete mit $(2^n+1) \times (2^n+1)$ Gitterpunkten unterteilt werden. Der nächste Schritt besteht darin, diese $(2^n+1) \times (2^n+1)$ Gitter (implizit) in hierarchische Level zu unterteilen. Abb. 3 zeigt die Level eines 5×5 Gitters. Hierarchische Vorgänger sind grau markiert.

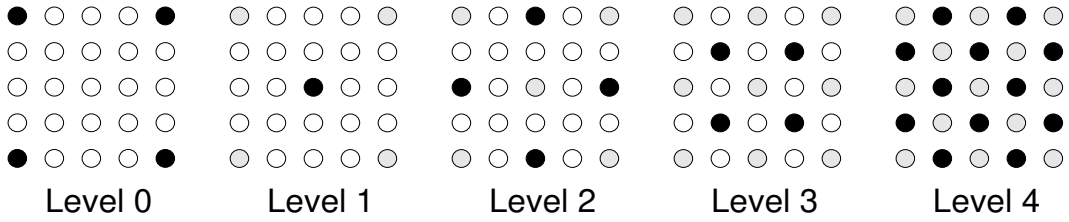

Abbildung 3: hierarchische Level eines 5×5 Gitters

Es ist dabei nicht nötig, die Daten tatsächlich in ihre Level zu trennen. Der hierarchische Zugriff kann allein über den Algorithmus implementiert werden, der auf den in einem Array gespeicherten Höhendaten arbeitet. Dies vermeidet die Probleme, die sich oft beim Arbeiten mit komplizierten und (programmier-) fehleranfälligen Datenstrukturen wie Listen oder Bäumen ergeben. Ein weiterer Vorteil ist, daß aufgrund der kompakt im Speicher liegenden Höhendaten sog. *cache-prefetch*-Mechanismen, wie sie von modernen Prozessoren eingesetzt werden, immer noch die Chance haben, die Zugriffsgeschwindigkeit auf die Daten zu verbessern. Bei Bäumen und Listen ist von vornherein nicht oder nur schwer abschätzbar, wo und in welcher Reihenfolge die Daten schließlich im Speicher abgelegt werden.

Selbstverständlich führt der hierarchische Zugriff auf ein zweidimensionales Array zu großen Sprüngen innerhalb des linearen Speichersegmentes. Solche Sprünge machen in der Regel die Vorteile jeder *cache-prefetch*-Strategie zunichte. Es ist jedoch möglich, die Daten innerhalb des Arrays so umzuorganisieren,

daß ein hierarchischer Datenzugriff lediglich zu Sprüngen nach vorne führt. Auf diese Weise können *cache-prefetch*-Mechanismen wieder greifen.

Um nun einen Anhaltspunkt für die Wichtigkeit eines jeden Höhenwertes zu gewinnen, wird er linear aus seinen hierarchischen Vorgängern interpoliert. Abb. 4 zeigt, welche Punkte verwendet werden, um die Höhenwerte auf jedem Level zu interpolieren.

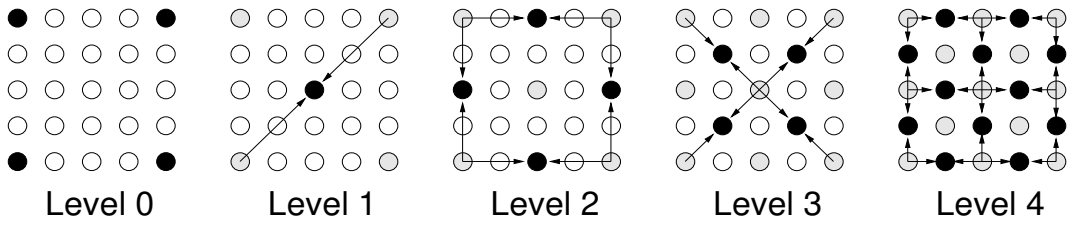

Abbildung 4: Vorgänger jedes Punktes, aus denen er interpoliert wird

Die Differenz zwischen interpoliertem und tatsächlichem Wert, der hierarchische Überschuß, wird für jeden Punkt in einem zweiten Array gespeichert. Dieses Array von Interpolationsfehlern wird dann hierarchisch *bottom-up* Level für Level durchlaufen, wobei jeweils das betragsmäßige Maximum des gerade untersuchten Wertes und seiner unmittelbaren hierarchischen Vorgänger für jeden Gitterpunkt gespeichert wird. Danach findet sich der höchste hierarchische Überschuß, der sich bei der Interpolation ergeben hat, im zentralen Knoten des Gitter wieder.

Die Interpolation und die anschließende Suche nach den Maxima der hierarchischen Überschüsse kann in einem Durchgang während der Initialisierung erfolgen, sie sind also nur eingeschränkt zeitkritisch.

2.2 Rekursive Triangulierung

Ein Vorteil und gleichzeitig der wichtigste Grund für die Wahl der beschriebenen hierarchischen Interpolationstechnik ist die Tatsache, daß sie eine einfache, naheliegende und adaptive Triangulierung erlaubt, die auf dem klassischen *divide&conquer*-Prinzip beruht. Der rekursive Algorithmus, der für die Triangulierung benutzt wird, ist im wesentlichen der folgende:

```
untersuche_dreieck(a,b,c)
{
  if(Interpolationsfehler am Knoten (b+c)/2 < ε)
    zeichne_dreieck(a,b,c);
  else {
    untersuche_dreieck( (b+c)/2, a, b);
    untersuche_dreieck( (b+c)/2, c, a);
  }
}
```

Abb. 5 zeigt eine Folge von Dreiecken, die während der adaptiven Triangulierung eines 5×5-Gitters untersucht werden. Die Prozedur `untersuche_dreieck()` wird zweimal aufgerufen, um das gesammte Gitter zu triangulieren. Einmal für das Dreieck, das aus der nordwestlichen, der nordöstlichen und der südöstlichen Ecke des Arrays gebildet wird, und einmal für das Dreieck, das durch die südöstliche, die südwestliche und die nordöstliche Ecke begrenzt wird. Dabei ist der Interpolationsfehler an den schwarzen Gitterpunkten höher als eine vorgegebene Fehlerschranke ε. Der hierarchische Überschuß an weißen Gitterpunkten ist kleiner als ε. Punkte, die in einem bestimmten Level gerade untersucht werden, sind durch einen Kreis markiert. Abb. 6 zeigt die sich ergebende Triangulierung des gesammten Gitters. In diesem Beispiel hängt die Fehlerschranke ε nicht vom Abstand zum Betrachter ab und ist für alle Punkte gleich.

Um dieses Triangulierungsschema im Rahmen des Konzepts der synthetischen Sicht einzusetzen, ist es sinnvoll, die Prozedur `untersuche_dreieck()` mit einer Terminierungsregel zu versehen, die überprüft, ob das gerade untersuchte Dreieck für den Betrachter sichtbar ist bzw. sein könnte. So kann überflüssige Arbeit vermieden werden, und die meisten der rekursiven Prozeduraufrufe können bereits zu einem sehr frühen Zeitpunkt abgebrochen werden.

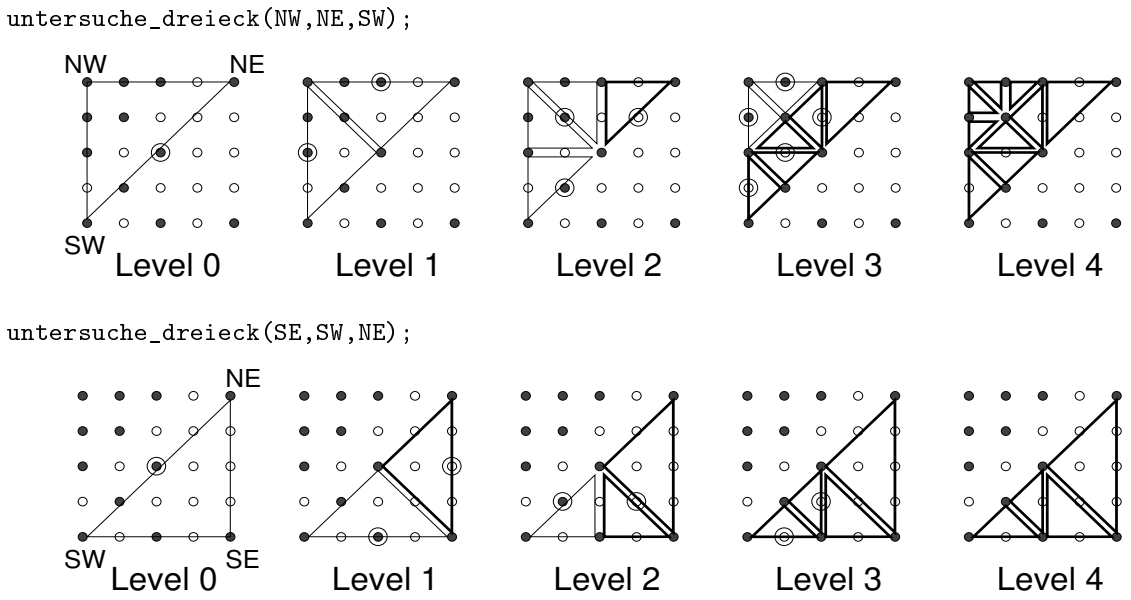

Abbildung 5: Folge von Dreiecken, die während einer Triangulierung untersucht werden.

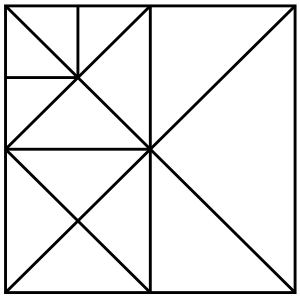

Abbildung 6: Die sich ergebende Triangulierung des Gitters

Das Verfahren, das zur Ermittlung der Sichtbarkeit eines Dreiecks eingesetzt wird, ist von einer 3D-Version des bekannten Cohen-Sutherland-line-clipping-Algorithmus ([10]) abgeleitet. Der wesentliche Unterschied ist, daß es auf Dreiecke anstelle von Linien angewandt wird. Da es aufgrund des eingesetzten Verfahrens nicht genügt festzustellen, ob ein Punkt sichtbar ist oder nicht, ist für jeden während einer Triangulierung untersuchten Punkt eine 4×4-Matrix-Vektor-Multiplikation notwendig. Damit der Cohen-Sutherland-Algorithmus zum Einsatz kommen kann, muß für den Fall, daß ein Punkt nicht sichtbar ist, bestimmt werden, wo er außerhalb des sichtbaren Bereiches liegt (davor, dahinter, links, rechts, darüber oder darunter). Die so nötigen Matrix-Vektor-Multiplikationen stellen den Hauptberechnungsaufwand zur Laufzeit dar. Die Alternative, nicht auf (potentielle) Sichtbarkeit zu prüfen, würde zu einem wesentlich höheren Arbeitsaufwand bei der Untersuchung von unsichtbaren Dreiecken führen.

Der nächste Schritt, der für die synthetische Sicht gemacht werden muß, ist, daß die Fehlerschranke ε von dem Abstand zwischen Betrachter und gerade untersuchtem Dreieck abhängig gemacht wird. Dazu wird ε lediglich durch eine Funktion `eps()` ersetzt, die geeignet konstruiert wurde.

Die `if`-Anweisung in `untersuche_dreieck()` ändert sich dann folgendermaßen:

```
if(Fehler am Punkt (b+c)/2 < eps(d(a,b,c)))
```

Die Funktion `d()` liefert dabei den Abstand zwischen dem Betrachter und dem als Parameter übergebenen Dreieck.

Die beabsichtigte Konsequenz dieser Änderung ist, daß unterschiedliche Fehlerschranken auf die Dreiecke angewendet werden, die während der rekursiven Aufrufe von `untersuche_dreieck()` untersucht werden. Das bedeutet, daß es jetzt möglich ist, daß bei benachbarten Dreiecken eines tiefer rekursiv verfeinert wird als das andere. Diese Situation kann zu T-Kreuzungen in der Triangulierung führen, was sich schließlich als sog. *vertikale Löcher* im Gelände bemerkbar macht.

Eine einfache Methode zur Verhinderung des genannten Problems besteht darin, die Knoten jedes untersuchten und gezeichneten Dreiecks in einem geeigneten Bitfeld zu markieren und die `if`-Anweisung in `untersuche_dreieck()` folgendermaßen zu ergänzen:

```
if(Fehler am Knoten (b+c)/2 < eps(d(a,b,c))
   && Knoten (b+c)/2 nicht im Bitfeld markiert)
```

Damit dieser Ansatz zu dem gewünschten Ergebnis führt, muß das Dreieck, das zu einer tieferen Ebene rekursiv verfeinert wird, beim rekursiven Aufruf zuerst untersucht werden. Das führt zu dem neuen Problem zu entscheiden, welches der Dreiecke `((b+c)/2, a, b)` und `((b+c)/2, c, a)` zuerst untersucht werden soll.

Bis jetzt wurde noch keine einfache und sichere Methode gefunden, mit der abgeschätzt werden kann, wie tief ein Dreieck rekursiv verfeinert werden wird. Verschiedene Ansätze wurden untersucht, aber alle führten früher oder später zu vertikalen Löchern in der Geländedarstellung.

Um das Problem der vertikalen Löcher völlig zu vermeiden, wurde eine zwei-pass-Triangulierung implementiert. Im ersten Durchgang werden Dreiecke in einem Bitfeld markiert, anstatt sie sofort zu zeichnen. Das Zeichnen erfolgt in einem zweiten Durchgang, in dem nur noch das Bitfeld untersucht wird. Die zwei-pass-Version von `untersuche_dreieck()`, die auch die Sichtbarkeitsüberprüfung enthält, folgt:

```
untersuche_und_markiere_dreieck(a,b,c)
{
  if(dreieck_nicht_sichtbar(a,b,c)) return();
  if(Fehler am Knoten (b+c)/2 < eps(d(a,b,c)))
    markiere_knoten(a,b,c);
  else {
    untersuche_und_markiere_dreieck( (b+c)/2, a, b);
    untersuche_und_markiere_dreieck( (b+c)/2, c, a);
    }
}
untersuche_und_zeichne_dreieck(a,b,c)
{
  if(Knoten (b+c)/2 ist nicht markiert)
    zeichne_dreieck(a,b,c);
  else {
    untersuche_und_zeichne_dreieck( (b+c)/2, a, b);
    untersuche_und_zeichne_dreieck( (b+c)/2, c, a);
    }
}
```

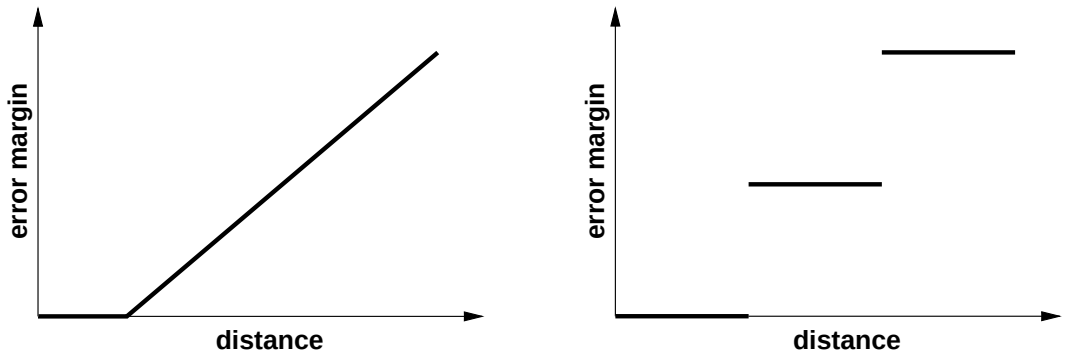

Abbildung 7: Beispiele für sinnvolle `eps()`-Funktionen

Einige Beispiele, wie eine brauchbare `eps()`-Funktion aussehen könnte, werden in Abb. 7 gezeigt. Die linke Funktion ist die, die zur Zeit eingesetzt wird. Der zulässige Fehler ist hier Null innerhalb eines Abstands von 1-5 km vom Betrachter. Dies entspricht einer exakten Darstellung des Geländes innerhalb dieses Radius. Danach steigt die Fehlerschranke linear auf einen Wert von 600 m in einer Entfernung von 25 km.

Die rechte Funktion könnte eingesetzt werden, um drei separate Level-Of-Detail-Bereiche zu simulieren. Vorteil dieser `eps()`-Funktion ist es, daß sich die Triangulierung des Geländes nur in der Nähe der Sprungstellen der Funktion ändert.

Mit diesen Beispielen sollte gezeigt werden, daß der Level-Of-Detail leicht kontrollierbar ist und daß es keine Einschränkungen für die Form der `eps()`-Funktion gibt.

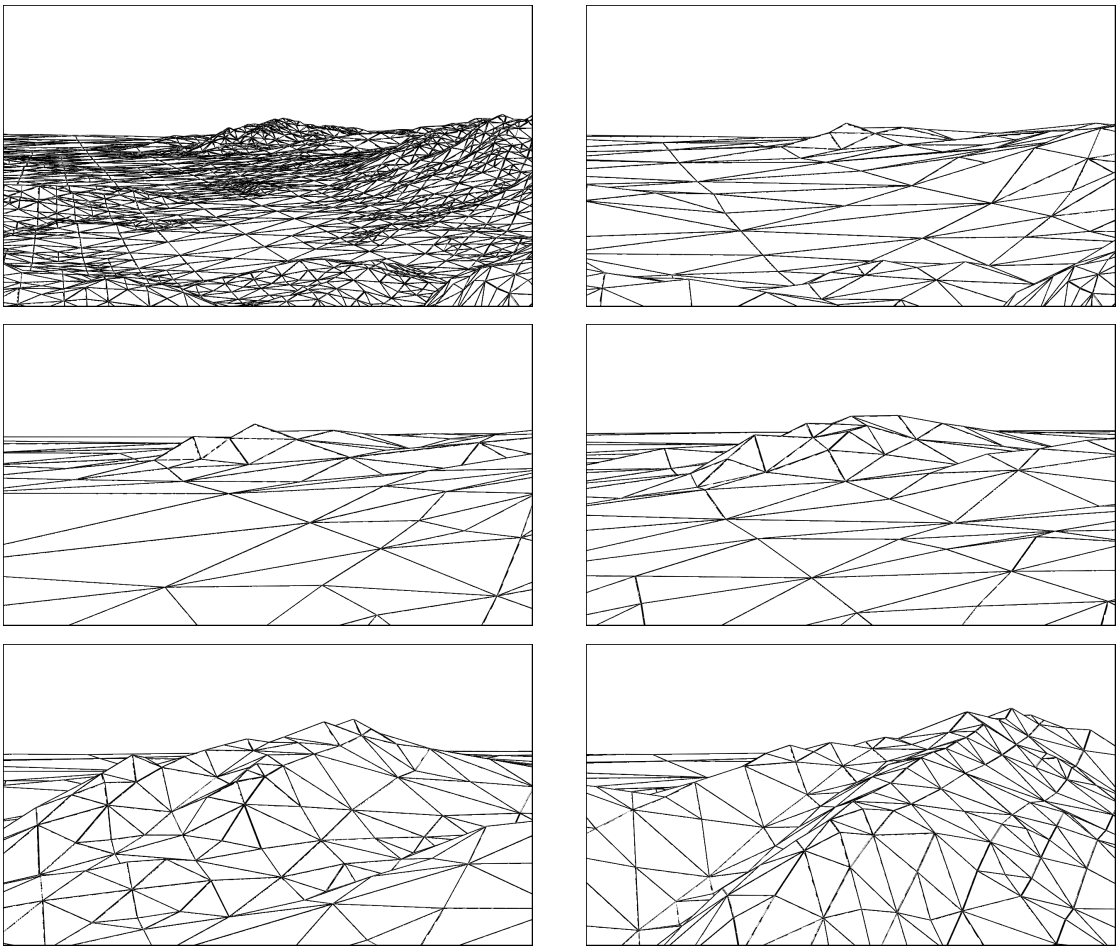

Abbildung 8: Annäherung an einen Berg

Abb. 8 zeigt eine Folge von Bildern, wie sie sich ergeben, wenn sich der Beobachter einem Berg nähert. Das erste und das zweite Bild wurden von derselben Position aus erzeugt, wobei das erste Bild mit einer deutlich geringeren Fehlerschranke berechnet wurde, um zu zeigen, wie das Gelände bei exakter Wiedergabe aussehen würde. Der Rest der Sequenz zeigt, daß die Triangulierung umso detaillierter wird, je näher der Betrachter an den Berg kommt.

Eine Nachbildung des Geländes mit texturierten Polygonen ist in den Abb. 9 und 10 gezeigt. Hierbei wurden einmal ca. 6500 Dreiecke verwendet (Abb. 9) und einmal lediglich ca. 600 Dreiecke (Abb. 10).

3 Ergebnisse und Ausblick

Das hier beschriebene System wurde für den Einsatz auf einer Workstation vom Typ Indigo^2 Silicon Graphics entwickelt, die mit einer R10000 CPU, 128MB Hauptspeicher und einer Graphikkarte Maximum IMPACT mit 4MB Texturspeicher ausgesrüstet ist. Die Designvorgaben von 25 km Sicht bei einer Bildrate von 25 Bildern pro Sekunde werden auf diesem relativ kostspieligen System erreicht. Da das Programm jedoch in C++ unter Verwendung der Bibliotheken OpenGL und Motif geschrieben wurde, ist es weitgehend plattformunabhängig. Es ist absehbar, daß innerhalb weniger Jahre erheblich kostengünstigere Systeme verfügbar sein werden, die auf einem Mehrprozessorboard und einer für den PC-Markt entwickelten Graphikkarte basieren, die OpenGL auf Hardwareebene unterstützen. Dies eröffnet die Möglichkeit zu einem Low-Cost-System.

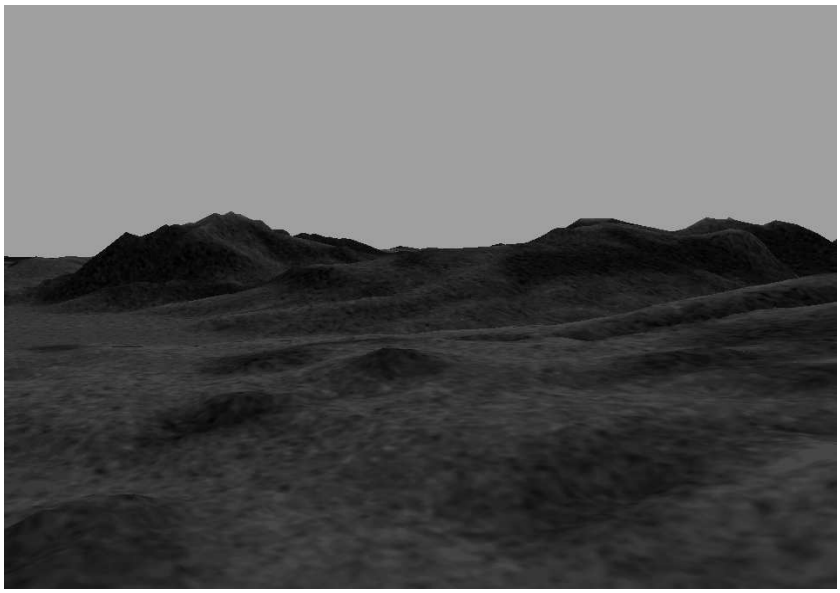

Abbildung 9: Geländenachbildung auf der Basis von ca. 6500 Dreiecken

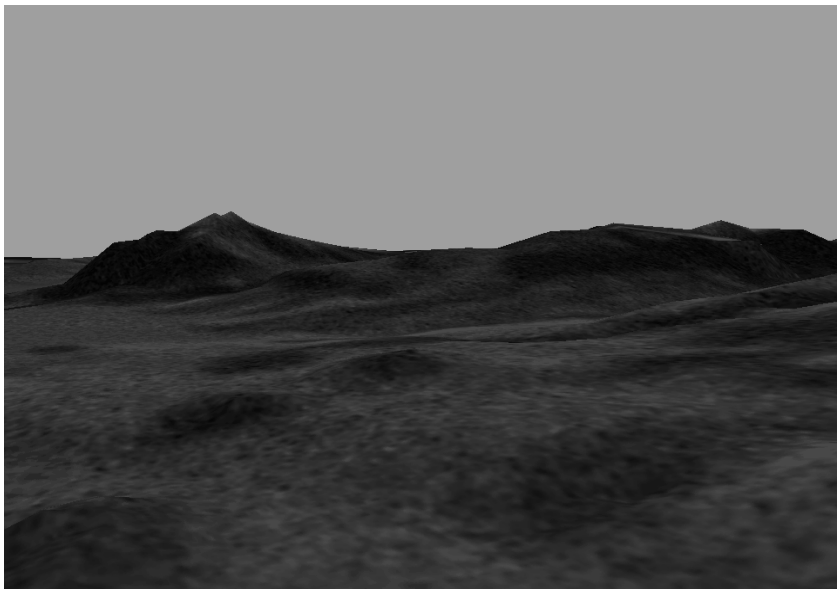

Abbildung 10: Geländenachbildung auf der Basis von ca. 600 Dreiecken

Neben der bereits erwähnten hierarchischen Reorganisation der Daten innerhalb des Arrays werden noch weitere Konzeptelemente implementiert, die die Leistungsfähigkeit des Systems und die Bildqualität steigern sollen.

Dies betrifft z.B. die Markierung von lokalen Maxima im Höhendatenarray als Punkte, die während der Triangulierung nicht übersprungen werden dürfen. Auf diese Weise würde sich die bildliche Darstellung von Bergen nicht verändern („springen“), so wie es bei der jetzigen Lösung der Fall ist, sondern lediglich die Hänge würden mit zunehmender Annäherung detailreicher in Erscheinung treten.

Um die Leistungsfähigkeit der Triangulierung zu steigern, ist außerdem eine inkrementelle Sichtbarkeitsüberprüfung denkbar, die deutlich weniger rechenaufwendig ist als die derzeit eingesetzte 3D-Version des Cohen-Sutherland-Algorithmus.

Ein weiteres, bis jetzt noch nicht behandeltes Problem betrifft das Nachladen von Geländedaten während des Flugs, wenn sich das Flugzeug dem Rand des Gebietes nähert, das datenmäßig im Hauptspeicher erfaßt ist. Hierbei darf das Nachladen von Geländedaten nicht zu einer signifikanten Verzögerung bei der Geländedarstellung führen.

4 Zusammenfassung

Computergenerierte synthetische Sicht wird als ein neuartiges Konzept zur Verbesserung der Flugführung bei schlechten Außensichtverhältnissen vorgestellt. Dabei wird dem Piloten ein 3-dimensionales Abbild des Geländes mit integrierten Flugführungsinformationen auf einem Display dargestellt. Um die zur Darstellung des Geländes benötigte Graphikleistung zu reduzieren, wird ein hierarchisches Interpolationsverfahren eingesetzt. Erste Ergebnisse auf Desktop-Rechnern zeigen, daß unter Benutzung des adaptiven Geländetriangulierungsalgorithmus eine für Zwecke der Flugführung ausreichend genaue Abbildung des Geländes in Echtzeit erzeugt werden kann. Zukünftig kann auch eine Anwendung auf Personalcomputern aufgrund der prognostizierten Leistungssteigerungen erwartet werden.

Literatur

[1] U. Rathmann: Künstliche Sicht. In Carl-Cranz-Gesellschaft, editor, *CCG-Kurs LR 4.05 Moderne Unterstützungssysteme für Piloten*. Oberpfaffenhofen, 1992.

[2] G. Sachs, K. Dobler, G. Schänzer, and M. Dieroff: Precision Navigation and Synthetic Vision for Poor Visibility Guidance. In *AGARD Flight Vehicle Integration Panel*, Lissabon, September 1996.

[3] S. Vieweg and G. Schänzer: Präzise Flugnavigation durch Integration von Satelliten-Navigationssystemen mit Inertialsensoren. In *DGLR-Jahrbuch 1992*, pages 171–177, 1992.

[4] T. Gerstner: Ein adaptives hierarchisches Verfahren zur Approximation und effizienten Visualisierung von Funktionen und seine Anwendung auf digitale 3–D Höhenmodelle. Master's thesis, Institut für Informatik, TU München, 1995.

[5] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner: Real–Time, Continuous Level of Detail Rendering of Height Fields. *Computer Graphics (SIGGRAPH '96 Proceedings)*, 1996.

[6] M.H. Gross, O.G. Staadt, and R. Gatti: Efficient Triangular Surface Approximations using Wavelets and Quadtree Data Structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2), June 1996.

[7] K. Hiller: Datenkompression mit dem Dünn-Gitter-Verfahren. Master's thesis, Institut für Informatik, TU München, 1993.

[8] A. Frank: Hierarchische Polynombasen zum Einsatz in der Datenkompression mit Anwendung auf Audiodaten. Master's thesis, Institut für Informatik, TU München, 1995.

[9] A. Paul: Kompression von Bildfolgen mit hierarchischen Basen. Master's thesis, Institut für Informatik, TU München, 1995.

[10] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes: *Computer Graphics – Principle and Practice*. Addison-Wesley Publishing Company, 2. Edition, 1992.