Animation of clouds

Erich Crameri

Seminar WS 2002/03
Physically-based methods for
3D games and medical applications

Grundlage

- "A Simple, Efficient Method for Realistic Animation of Clouds„
  - Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, T. Nishita
  - Hiroshima City University / Hiroshima University / University of Tokyo
  - SIGGRAPH 2000

Aufbau

- Einführung
  - Ziel der Arbeit
- Bisherige Arbeiten zum Thema
- Methoden
  - Zweiteiliger Prozess
    - Simulation
    - Rendering
- Fazit
- Beispiele

Einführung

- Anwendungsgebiete für realistisch simulierte Wolken
  - z.B. in Aussenszenen für
    - Architekturstudien
    - Flugsimulatoren
    - Filmszenen / Werbung
- Wolken beschrieben durch Dichteverteilung
  - Definiert über den Raum
    - Farbe und Form ändert je nach Betrachtungs- bzw. Beleuchtungswinkel

Einführung (2)

- Zwei Kategorien für bewegte „Gas"-Simulation
  - Physikalisch basiert
    - Fluiddynamik
    - Vorteil
      - „korrekte" Simulation der Natur
    - Nachteil
      - sehr rechenaufwändig
  - Heuristische Verfahren
    - Vorteil
      - wenig rechenintensiv
    - Nachteil
      - Parameterwerte für realistische Resultate per Trial & Error suchen
      - unhandlich

Ziel der Arbeit

- Einfache Methoden
- Realistische Animation
- Möglichst schnell
  - real-time im Visier
  - Hardware ausnützen
  - Standard-PC als Plattform

Einführung (3)

- Zwei Phasen der Animation (generell)
  - Simulation
    - Generieren der bewegten Wolkendaten
      - Berechnen der Dichteverteilung
  - Rendering
    - Darstellung der Wolkendaten auf dem Bildschirm
      - Von der Dichteverteilung zu fotorealistischen Daten im Framebuffer

Einführung (4)

- Verwendete Methoden
  - Simulation
    - Mit Hilfe eines zellulären Automaten
      - Simuliert Bewegung durch einfache Boolsche Operationen
  - Rendering
    - Volume Rendering aufgrund der Dichteverteilung
    - Projizieren einer 2D Textur
    - Drei Effekte berücksichtigt
      - Wolkenfärbung
        - mit Berücksichtigung einfacher Lichtstreuung
      - Wolkenschatten auf dem Boden
      - Lichtstrahlen durch Wolken
    - Effekte können mit Open-GL realisiert werden
      - Ausnützen der Hardware

Bisherige relevante Arbeiten

- Simulation
  - Kay Nagel et al. 1992
    - Verwendet zellulären Automaten für die Wolkendynamik
    - Liefert noch zu unrealistische Resultate
      - Methode wird erweitert
    - Methode liegt zwischen physikalischem und heuristischem Ansatz
- Rendering
  - Nishita et al. 1996
    - Fasst mehrere Methoden/Effekte zusammen
      - Absorption & Streuung an Partikeln
      - Färbung durch gegebene Skylight-Verhältnisse
      - Lichtstrahlen durch Wolken
      - Schatten auf dem Boden

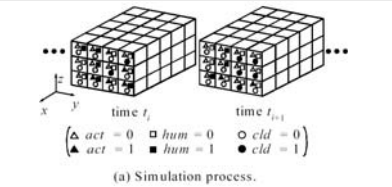
Methoden - Grundidee

- Simulation
  - Raum unterteilt in Voxels
  - Ein Voxel entspricht einer Zelle im Automaten
  - Pro Zelle drei Boolsche Variabeln
    - vapor/humidity (*hum*)
    - clouds (*cld*)
    - phase transition or activation factor (*act*)

Methoden – Grundidee (2)

- Rendering
  - Hier interessiert nur
    - *cld=0* oder *1* pro Voxel
  - Harte binäre Dichteverteilung wird geglättet
    - Werte in [0..1] pro Voxel
  - Volumen Rendering der Wolken
    - 1. Lichtintensität im Zentrum jedes Voxels
      - Textur des Wolkenschatten als „Nebenprodukt"
    - 2. Wolkenbild erzeugt mit einer Splatting Methode
      - Erzeugen & projizieren von 2D Texturen (Billboards)
      - Lichtstrahlen durch Blending der Wolkenschatten auf Schalen um den Betrachter

Methoden – Grundidee (3)



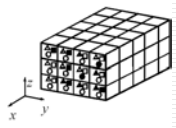
(a) Simulation process.

(b) Rendering process.

Figure 1: Overview of our method.

Simulation - Wachstum

- Entspricht Nagels Methode
- Zellulärer Automat modelliert Luftvolumen
  - Rechteck parallel zu den Koordinatenachsen
  - Anzahl Zellen
    - $n_x \times n_y \times n_z$



- Variabeln
  - *hum=1* : genügend Wasserdampf für Tropfen-/Wolkenbildung
  - *act=1* : Phasenübergang Dampf/Wasser ist bereit
  - *cld=1* : Wassertropfen/Wolken vorhanden

Simulation - Wachstum (2)

- Übergangsfunktionen des ZA

$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \wedge \neg act(i,j,k,t_i),$$
$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \vee act(i,j,k,t_i),$$
$$act(i,j,k,t_{i+1}) = \neg act(i,j,k,t_i) \wedge hum(i,j,k,t_i) \wedge f_{act}(i,j,k),$$

- wobei

$$f_{act}(i,j,k) = act(i+1,j,k,t_i) \vee act(i,j+1,k,t_i) \vee act(i,j,k+1,t_i)$$
$$\vee act(i-1,j,k,t_i) \vee act(i,j-1,k,t_i) \vee act(i,j,k-1,t_i)$$
$$\vee act(i-2,j,k,t_i) \vee act(i,j-2,k,t_i) \vee act(i,j,k-2,t_i)$$
$$\vee act(i+2,j,k,t_i) \vee act(i,j+2,k,t_i)$$

- eine Boolsche Funktion
  - berücksichtigt, dass Wolken tendenziell horizontal und nach oben wachsen

Simulation - Extinktion

- Ergänzend zu Nagel
  - In Natur tritt Wasser/Dampf-Phasenübergang auch auf
- Neue Zustandsvariable für Auflösung
  - 1. Animator setzt Auflösungs-WSK p_{ext}
  - 2. $\forall$ Zellen mit *cld=1*: Zufallszahl $rnd\ (0 \le rnd \le 1)$
  - 3. *cld* ändert Zustand, wenn $rnd < p_{ext}$
- Neues Problem
  - Wolken werden nicht mehr generiert nach Auflösung

Simulation - Extinktion (2)

- Weitere Parameter für *hum* und *act*
  - Dampf-WSK p_{hum}
  - Phasenübergangs-WSK p_{act}
  - Analog $\forall$ Zellen mit *hum=0* bzw. *act=0*
    - Zufallszahl $rnd\ (0 \le rnd \le 1)$
  - *hum/act* ändert Zustand, wenn $rnd < p_{hum} / rnd < p_{act}$
- Zusätzliche Übergangsfunktionen (*IS(e)=1*, falls *e* wahr)

$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \wedge IS(rnd > p_{ext}(i,j,k,t_i)),$$
$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \vee IS(rnd < p_{hum}(i,j,k,t_i)),$$
$$act(i,j,k,t_{i+1}) = act(i,j,k,t_i) \vee IS(rnd < p_{act}(i,j,k,t_i))$$

- Mit p_{ext}, p_{hum} und p_{act} variabel pro Zelle und Zeit
  - Animator kann eine Art Wetter modellieren
  - und die Bewegungen der Wolken kontrollieren

Simulation - Advektion durch Wind

- Weitere Übergangsfunktionen nötig
- Idee
  - Alle Variabeln in Windrichtung verschieben
  - Windgeschwindigkeitsfunktion $v()$
- Bsp. mit Wind entlang x-Achse

$$cld(i,j,k,t_{i+1}) = \begin{cases} cld(i - v(z_k), j, k, t_i) & i - v(z_k) > 0 \\ 0 & sonst \end{cases}$$

$$hum(i,j,k,t_{i+1}) = \begin{cases} hum(i - v(z_k), j, k, t_i) & i - v(z_k) > 0 \\ 0 & sonst \end{cases}$$

$$act(i,j,k,t_{i+1}) = \begin{cases} act(i - v(z_k), j, k, t_i) & i - v(z_k) > 0 \\ 0 & sonst \end{cases}$$

Simulation - Bitfeld Manipulationen

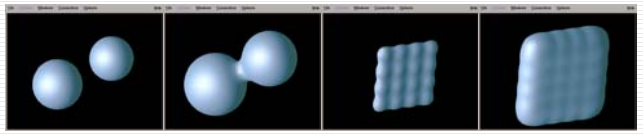
- 1 Bit pro Variable
- 3 Bits pro Zelle/Voxel
- Integer Datentyp repräsentiert mit m Bits
  - Bitoperationen mit Integers
  - Übergange von m Zellen parallel pro Bitoperation berechnet
- Problem
  - Zufallszahl für Auflösungsberechnung
- Lösung
  - Look-Up Table
    - zufällige Bitsequenzen

Simulation - Bewegungskontrolle

- Design der Wolkenbewegung über
  - Dampf-WSK
  - Phasenübergangs-WSK
  - Auflösungs-WSK
- Ellipsoide modellieren Luftpakete
  - Dampf-WSK innen > als aussen
  - Phasenübergans-WSK innen > als aussen
  - Auflösungs-WSK innen < als aussen
- Design der Wolkenart
  - Ellipsoidparameter
    - Ausprägung, Anzahl, Ort

Rendering – Dichteverteilung

- Vor Bilderzeugung
  - Zweistufige Filterung von
    - Raumdiskrete, binäre (!) Dichteverteilung
  - nach
    - Stetige Dichteverteilung mit Werten in [0..1]
- „Interpolation" zwischen den Zellen
  - Metaball Methode
    - Feldfunktion innerhalb Kugel mit Radius R
    - Isoflächen



Rendering – Dichteverteilung (2)

- Stufe 1

$$q(i,j,k,t_i)=\frac{1}{(2t_0+1)(2k_0+1)(2j_0+1)(2i_0+1)}\sum_{t'=-t_0}^{t_0}\sum_{k'=-k_0}^{k_0}\sum_{j'=-j_0}^{j_0}\sum_{i'=-i_0}^{i_0}w(i',j',k',t')\,cld(i+i',j+j',k+k',t_i+t')$$

  - Raumdiskrete Glättung über Raum/Zeit
    - Dichte in $[0..max(w())]$ abgebildet
  - Gewichtung
    - $w() > 0$
  - Glättungsausdehnung
    - i_0, j_0, k_0, t_0

Rendering – Dichteverteilung (3)

- Stufe 2

$$\rho(\underline{x},t_i)=\sum_{i,j,k\in\Omega(\underline{x},R)}^{N}q(i,j,k,t_i)\,f\left(\left|\underline{x}-\underline{x}_{i,j,k}\right|\right)$$

  - Metaball Feldfunktion als Basis

$$f(r)=\begin{cases}-\frac{4}{9}a^6+\frac{17}{9}a^4-\frac{22}{9}a^2+1 & (r\le R)\\ 0 & (r>R)\end{cases},\quad a=\frac{r}{R}$$

  - Effektradius des Metaballs
    - R
  - Menge der Zellen innerhalb Metaball um $\underline{x}$ mit Radius R
    - $\Omega(\underline{x}, R)$
  - Zentrum der Zelle (i,j,k)
    - $\underline{x}_{i,j,k}$

Rendering – Wolken

- Wolkenfarbe als Summe
  - aus attenuiertem Hintergrundlicht
  - und gestreutem Sonnenlicht

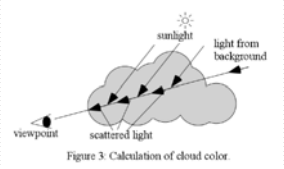
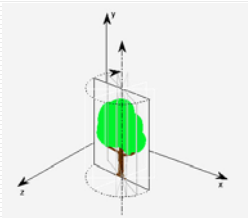


Figure 3: Calculation of cloud color.

- Rendering in drei Schritten...

Rendering – Wolken (2)

- Schritt 0:
  - Trick
    - Billboards ∀ Dichten von Metaball-/Voxelzentren vorberechnen
      - Textur = kumulierte Attenuation im Metaball
      - Dichten zu n=64 Varianten diskretisieren
        - Speichersparend
    - Spart intensives real-time Berechnen



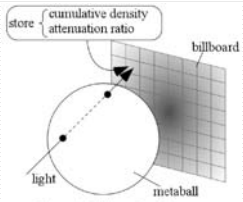
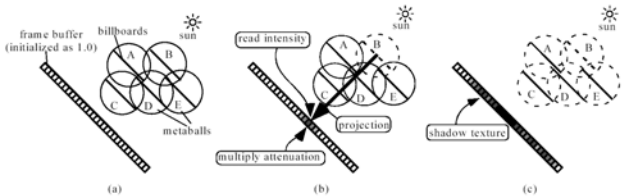


Figure 4: Billboard and its texture.

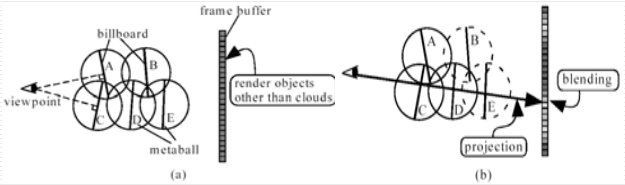
Rendering – Wolken (3)

- Schritt 1:
  - ∀ Metaballs Intensität des Sonnenlichts bestimmen
    - geordnetes „blending“ aller Billboard-Attenuationen
      - Nacheinander in einen Framebuffer
    - Pixel „unter“ Metaballzentrum mult. mit Farbe des Sonnenlichts
      - = Farbe des Metaballs
      - gespeichert in Billboard-Textur
  - Resultierender Framebuffer
    - Schatten-Textur

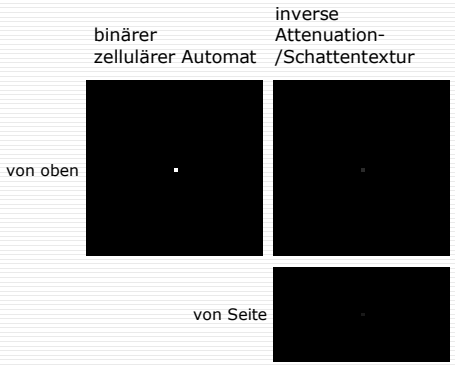


Rendering – Wolken (4)

- Schritt 2:
  - Szene bis auf Wolken rendern
  - Billboards auf Betrachter gerichtet
  - ∀ Metaballs von hinten nach vorne
    - Projizieren der Billboards
      - „blending“ der Billboard-Texturen mit dem Framebuffer



Rendering – Wolken (5)



Rendering – Lichtstrahlen

- Idee
  - Schattentextur auf Schalen „blenden“

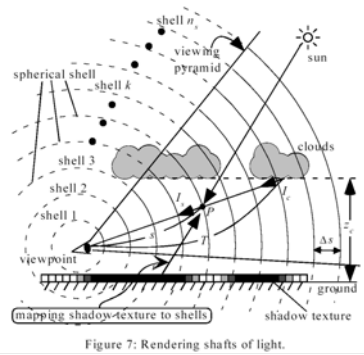


Figure 7: Rendering shafts of light.

Rendering – Lichtstrahlen (2)

- Diskretisierte Intensität des Lichts

$$I = I_c \beta(T) + \sum_{k=0}^{n_s} \gamma(k\Delta s) I_s(k\Delta s) \beta(k\Delta s) \Delta s$$

  - n_s: Anzahl Schalen
  - Δs: Abstand der Schalen
  - I_c: Farbe der Wolke
  - $\beta()$: Dämpfung durch Luft
  - I_s: Intensität des gestreuten Sonnenlichts
  - $\gamma()$: Dämpfung durch Wolke
  - T: Distanz Metaball - Betrachter

Rendering – Lichtstrahlen (3)

- Berechnen der einzelnen Terme
  - $\beta(T)$: analytisch aufgrund der Distanz
    - → 1. Term $I_c \beta(T)$
  - Schalen durch Polygone in Viewing Pyramid approximiert
    - OpenGL
    - ∀ Ecken
      - Berechne Intensität des gestreuten Lichts $I_s()$
      - Berechne Dämpfung zwischen Ecke – Betrachter analytisch
      - Speichere $I_s(k\Delta s)\beta(k\Delta s)\Delta s$ als Polygonfarbe
    - ∀ Polygone
      - Texture-Mapping des Schattens
        - ergibt Multiplikation der Polygonfarbe mit $\gamma()$
    - Schalenpolygone und Wolken nach Reissverschlussprinzip gerendert
      - Schale, Wolke bis nächste Schale, nächste Schale ...
      - da beides transparente Objekte

Fazit

- Vorteile der Methode
  - Realistische Resultate
    - der Dynamik
    - Farbe, Schatten, Lichtstrahlen
  - Schnelle Simulation durch Bitfeldoperationen
  - Wenig Speicheraufwand
  - Grafikhardware ausgenutzt
  - Parameter lassen Design der Szene zu
- Verbesserungen
  - Vektorfelder für Windsimulation
  - Level Of Detail
    - Voxels weit weg könnten grösser sein
      - weniger Metaballs
        - Splatting-Prozess am Zeitintensivsten!

Beispiel

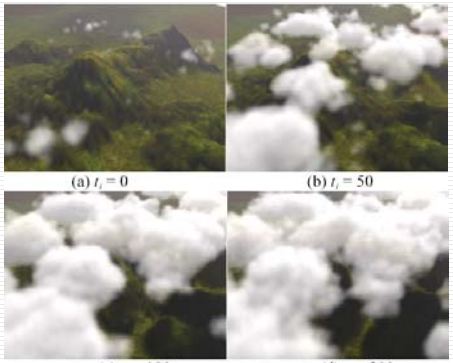


Figure 8: Simulation of cloud formation.

Auflösung:
640x480
#Zellen:
256x128x20
Speicher:
80KB
Zeit:
0.3s/Schritt
<10s/Bild
(Dual PIII 500MHz)
Ellipsoide:
zufällig
Im Zentrum
p_{hum}=0.1
p_{act}=0.001
p_{ext}=0.1

Beispiel (2)

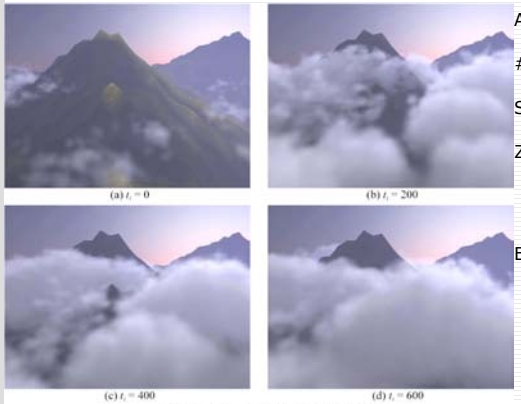


Figure 9: Cloud formation around mountains.

Auflösung:
640x480
#Zellen:
256x128x20
Speicher:
80KB
Zeit:
0.3s/Schritt
<10s/Bild
(Dual PIII
500MHz)
Ellipsoide:
manuell
Im Zentrum
p_{hum}=0.1
p_{act}=0.001
p_{ext}=0.1

Beispiel (3)

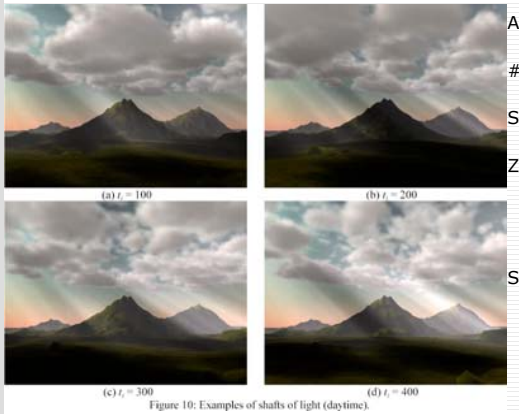


Figure 10: Examples of shafts of light (daytime).

Auflösung:
640x480
#Zellen:
256x256x20
Speicher:
160KB
Zeit:
0.5s/Schritt
20-30s/Bild
(Dual PIII
500MHz)
Schalen:
40

Beispiel (4)

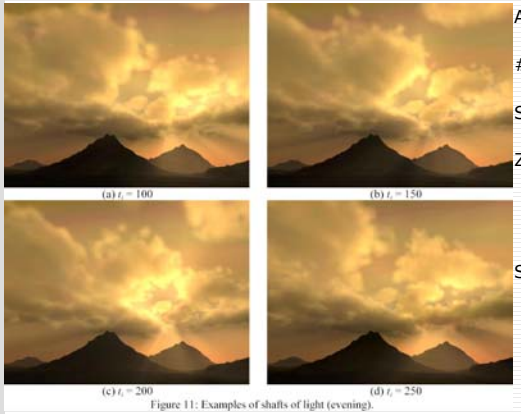


Figure 11: Examples of shafts of light (evening).

Auflösung:
640x480
#Zellen:
256x256x20
Speicher:
160KB
Zeit:
0.5s/Schritt
20-30s/Bild
(Dual PIII
500MHz)
Schalen:
40