

Computergrafik

**Vorlesung im Wintersemester 2014/15
Kapitel 8: Prozedurale Modelle,
Content Creation, Texturen & Geometrie**

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie



Prozedurale Landschaften



Bilder: Terragen 3 Gallery²

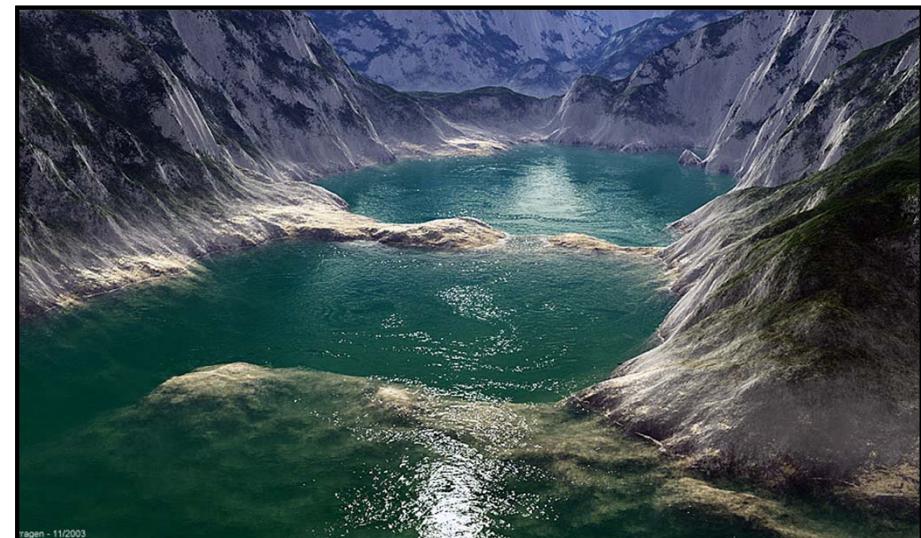
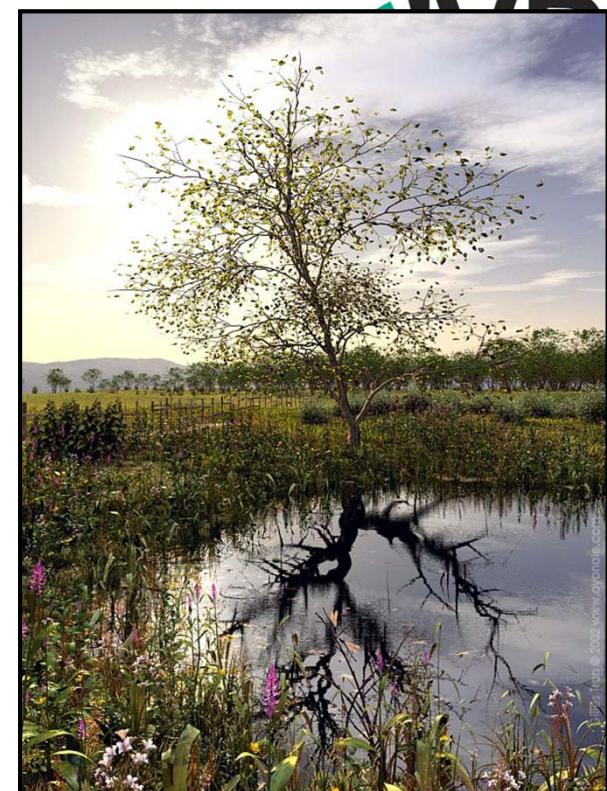
Prozedurale Modellierung

- ▶ diese Landschaft besteht aus ca. 100.000.000 Dreiecken



Überblick

- ▶ Motivation
- ▶ Rauschen, Turbulenz und prozedurale Texturen
- ▶ Textursynthese
- ▶ Hypertextures und Distanzfelder
- ▶ L-Systeme
- ▶ kurzer Ausblick

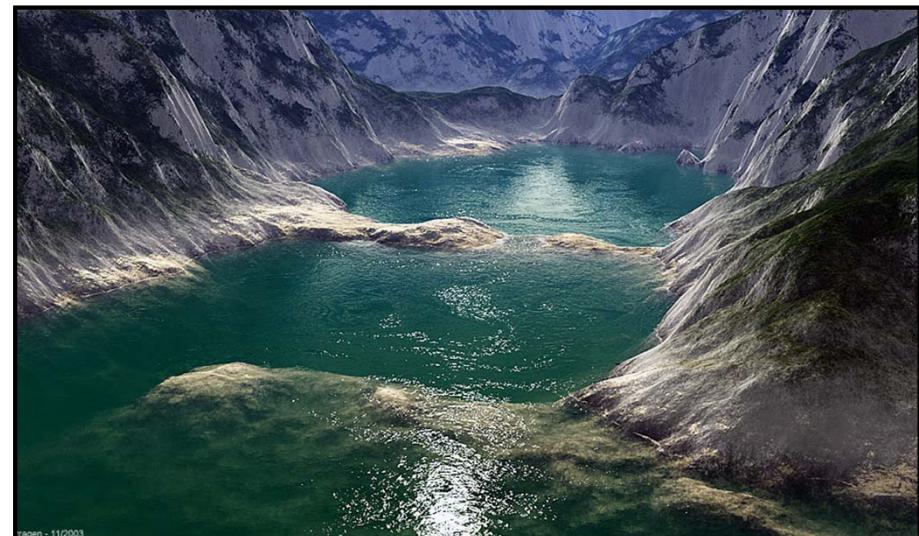
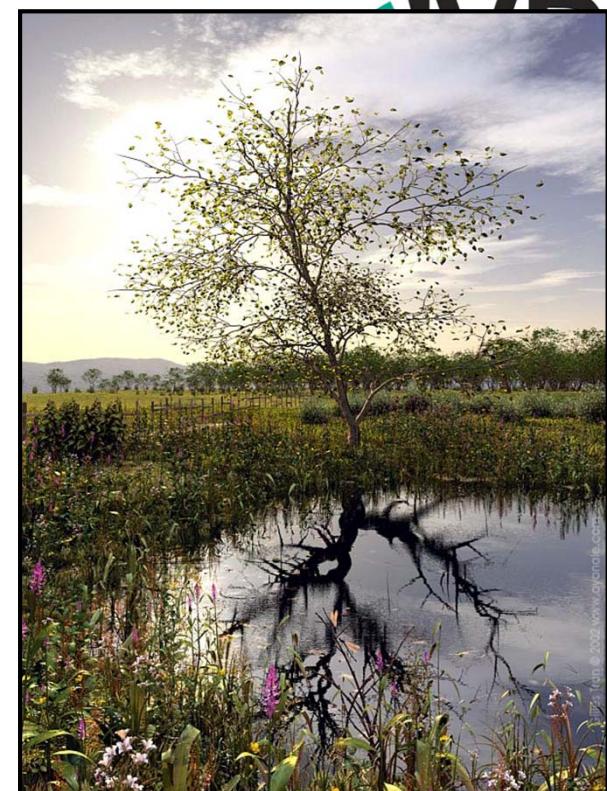


Überblick

- ▶ Motivation

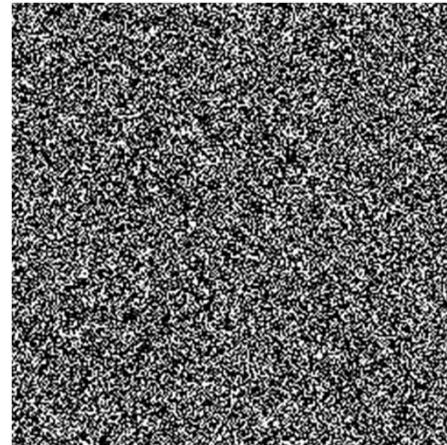
- ▶ kompakte Beschreibung statt Speicherung
- ▶ Erzeugung von großen Szenen/viel Content
- ▶ modellieren von natürlichen Phänomenen, aber Zufall („**Rauschen**“) statt physikalischer Simulation

- ▶ beginnen wir mit Texturen...

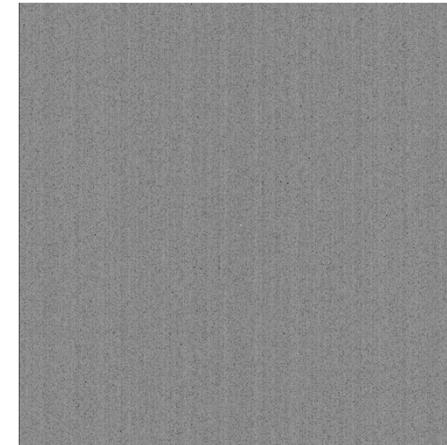


Rauschen? Zufallszahlen!

- ▶ weißes Rauschen: Energie in allen Frequenzbereichen verteilt

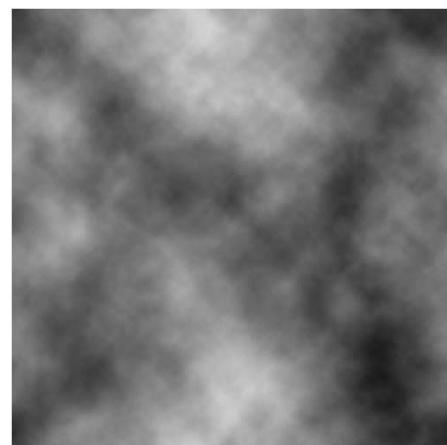


Ortsraum

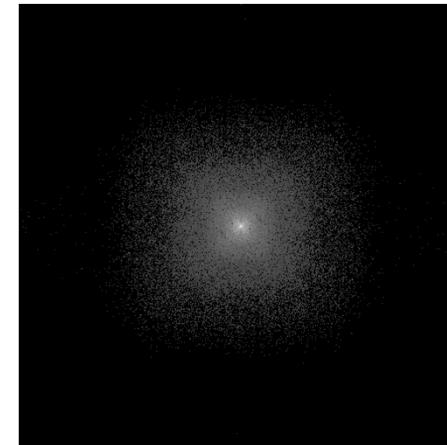


Frequenzraum

- ▶ rotes Rauschen: Energie v.a. in niedrigen Frequenzen → Kohärenz!



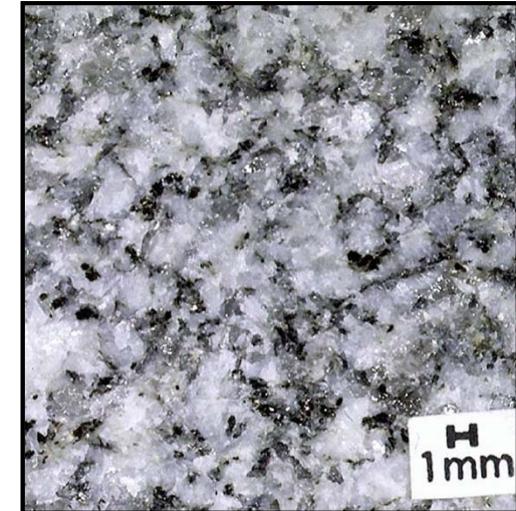
Ortsraum



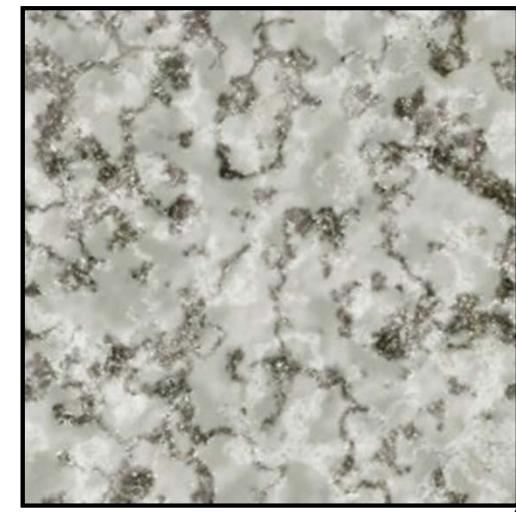
Frequenzraum

Rauschen und Rauschtexturen

- ▶ natürliche Objekte zeigen stochastische Variationen (auf unterschiedlichen Längenskalen) bei räumlicher Kohärenz
 - ▶ wir können zufällige Strukturen aus Rauschtexturen berechnen
- ▶ Anforderungen an Rausch-/Noise-Funktion
 $n(\mathbf{x}) = \mathbf{c}$ mit $\mathbf{x} \in \mathbb{R}^n$, \mathbf{c} ist Skalar
 - ▶ reproduzierbar: $n(\mathbf{x})$ liefert für jede Auswertung bei gleichem \mathbf{x} dasselbe Resultat
 - ▶ keine Periodizität (zumindest nicht sichtbar)
 - ▶ begrenzter Wertebereich (Abb. auf Farben etc.)
 - ▶ definierte Frequenzverteilung, bandlimitiert (Aliasing reduzieren)
 - ▶ räumliche Korrelation: $n(\mathbf{x}) \approx n(\mathbf{x} + \epsilon)$ (hängt mit Bandbegrenzung zusammen)



Granit
Marmor

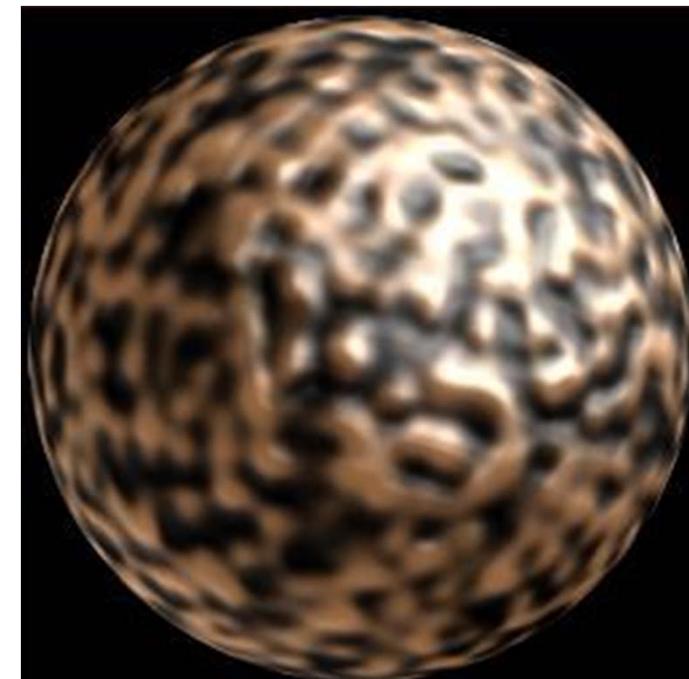
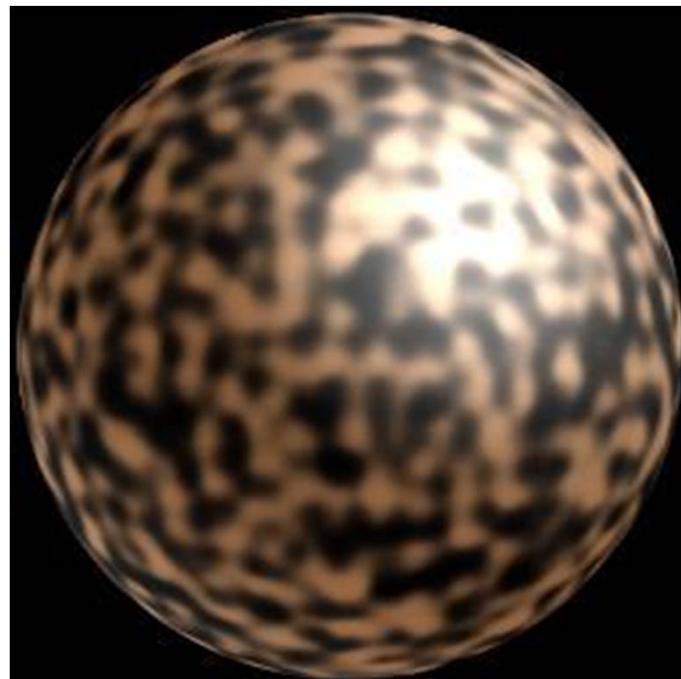


Rauschen und Rauschtexturen



Noise-Funktionen nach Ken Perlin (1985)

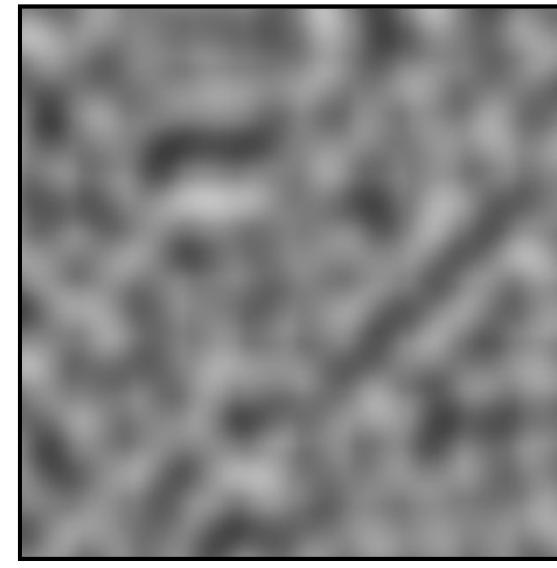
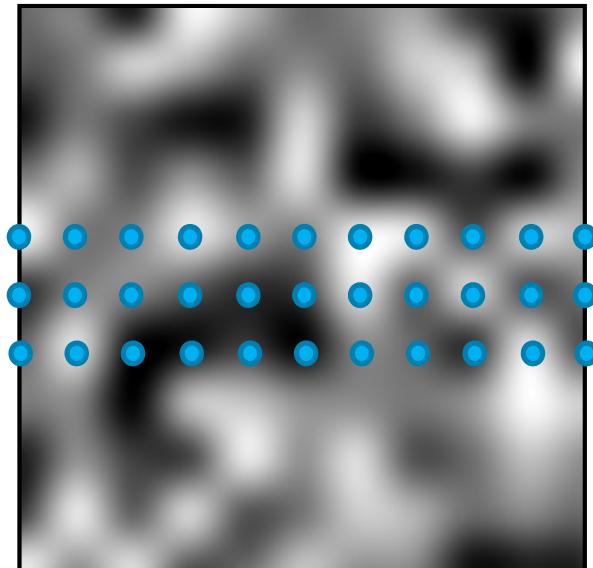
- ▶ die Grundlage für stochastische Modellierung/prozedurale Texturen
- ▶ die Basis bilden Pseudo-Zufallszahlen aus denen wir Bilder mit den geforderten Eigenschaften erzeugen
- ▶ Perlin erhielt einen „Academy Award for Technical Achievement“ der „Academy of Motion Picture Arts and Sciences“ (1997) für seine Arbeit über prozeduraler Texturierung mit Noise- und Turbulenz-Funktionen



Rauschen und Rauschtexturen

Noise-Funktionen nach Ken Perlin (1985)

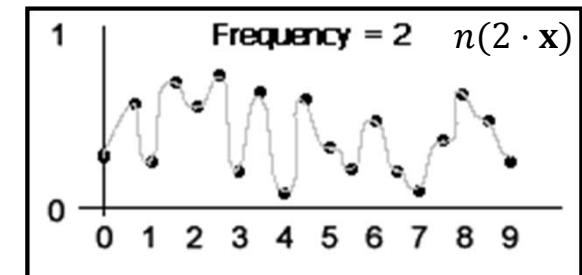
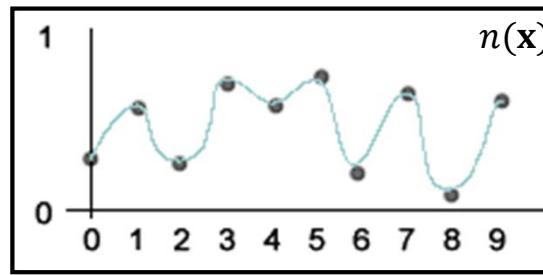
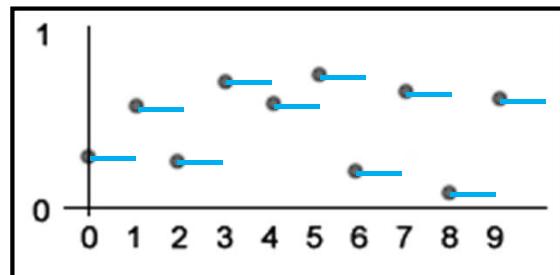
- ▶ die Grundlage für stochastische Modellierung/prozedurale Texturen
- ▶ die Basis bilden Pseudo-Zufallszahlen aus denen wir Bilder mit den geforderten Eigenschaften erzeugen
- ▶ es gibt zwei häufig verwendete Klassen von Noise-Funktionen:
 - ▶ Lattice Value Noise (lattice value = Werte auf einem Gitter)
 - ▶ Lattice Gradient Noise (nicht in der Vorlesung)



Lattice Value Noise

(Pseudo-)Zufallszahlen auf einem Gitter

- ▶ Abbildung $\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \mapsto z \in [-1; 1]$ mit
 $Z(\mathbf{x}) := \text{random}(\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor)$, d.h. Komponenten von \mathbf{x} abgerundet
 - ▶ nicht glatt sondern stückweise-konstant (unendlich hohe Frequenzen)
- ▶ räumliche Korrelation und Bandbegrenzung durch **Interpolation**
 - ▶ Noise-Funktion $n(\mathbf{x})$: Interpolation zwischen den Werten von $Z(\mathbf{x})$
 - ▶ Bsp. in 1D: $n(x) = Z(x) \cdot (1 - f_x) + Z(x + 1) \cdot f_x$ mit $f_x = x - \lfloor x \rfloor$
 - ▶ z.B. trilineare Interpolation aus $Z(\mathbf{x})$, $Z(\mathbf{x} + (1,0,0))$, $Z(\mathbf{x} + (0,1,0))$, ... und den Gewichten aus den Nachkommastellen von \mathbf{x} (also f_x, f_y, f_z)
- ▶ unterschiedlicher Abstand liefert anderes Frequenzspektrum
 - ▶ z.B. $n(2 \cdot \mathbf{x})$ hat doppelt so hohe Frequenzen wie $n(\mathbf{x})$



Lattice Value Noise



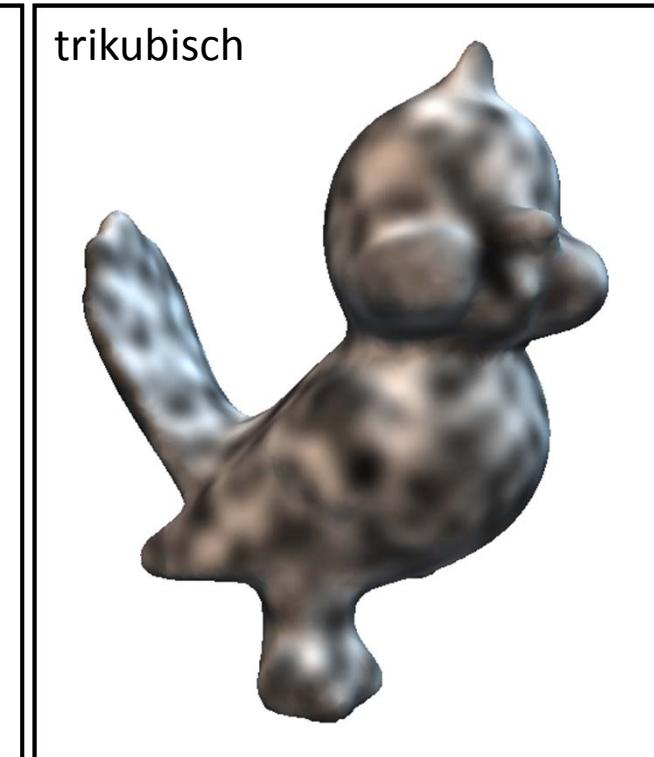
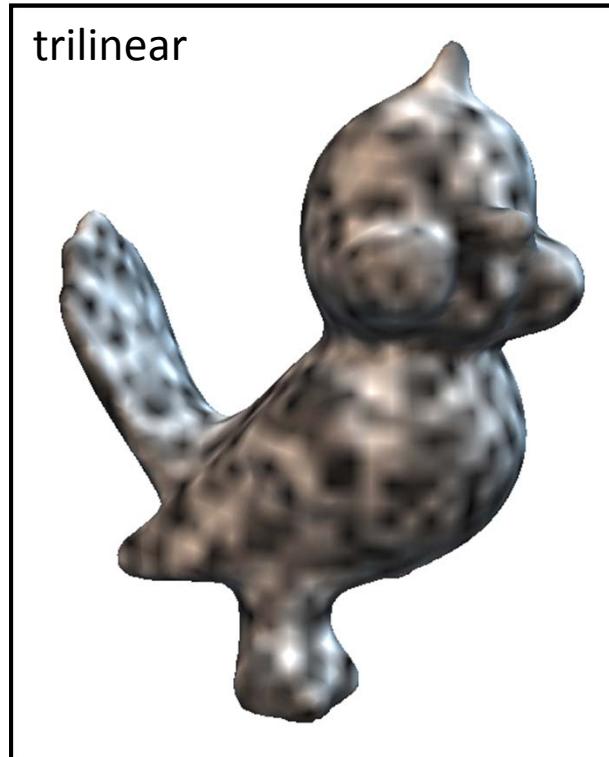
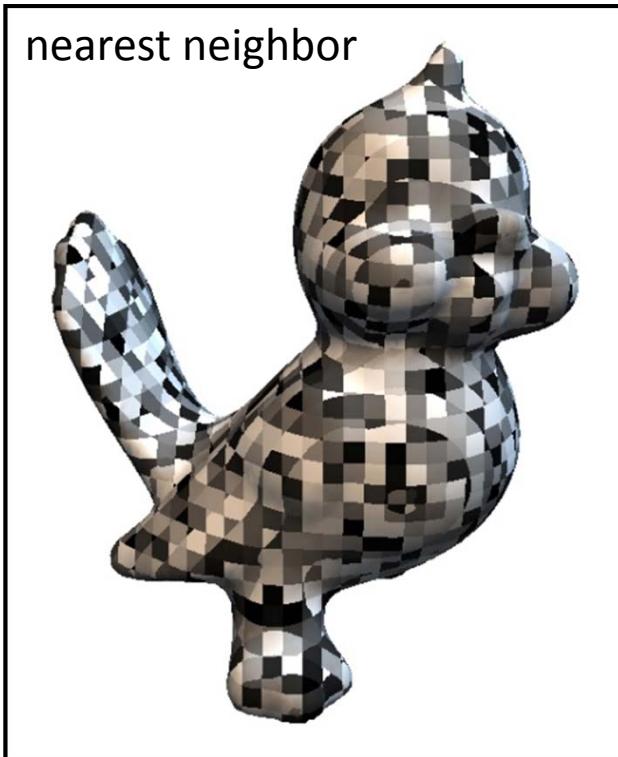
(Pseudo-)Zufallszahlen auf einem Gitter in der Praxis

- ▶ **naiver Ansatz:** erzeuge großes 2D/3D Gitter gefüllt mit Zufallszahlen
 - ▶ ist natürlich unpraktisch wg. großer Datenmenge und Periodizität
- ▶ **eleganter:** erzeuge einmalig ein 1D-Array mit Zufallszahlen aus $[-1; 1]$
 - ▶ z.B. float $rtab[256]$;
 - ▶ verwende Hash-Funktion um darauf zuzugreifen:
$$Index([x], [y], [z]):= P\left([x] + P\left([y] + P([z])\right)\right)$$
 - ▶ $P(\cdot)$ ist eine Permutation (z.B. bijektive Abbildung von i auf j mit $i, j \in [0.. 255]$), vermeidet sich wiederholende Muster
- ▶ **Noise-Funktion für weitere Schritte:**
$$Z(\mathbf{x}) := rtab[Index([x], [y], [z])] \text{ mit } \mathbf{x} = (x, y, z)$$

Lattice Value Noise

Beispiel: Interpolation einer 3D-Noise-Funktion

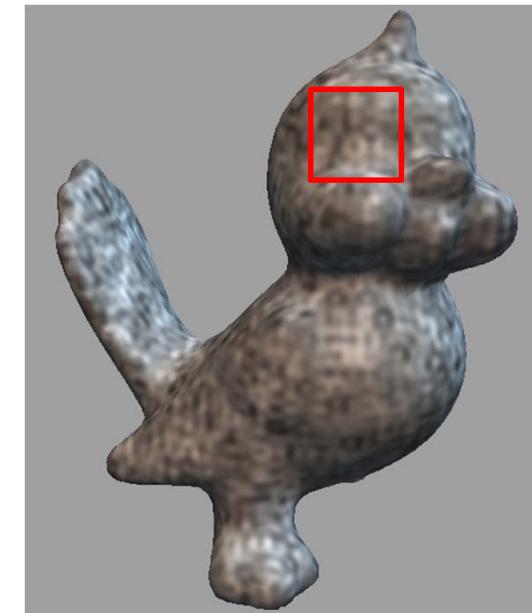
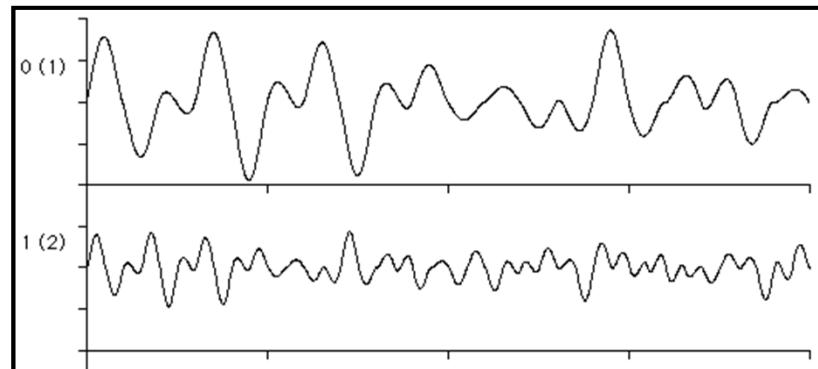
- ▶ das menschliche Auge nimmt keine Unstetigkeiten in der Krümmung (2. Ableitung) wahr
 - ▶ (bi-/tri-)lineare Interpolation verursacht Bandeffekte
 - ▶ kubische Interpolation ist zweimal stetig differenzierbar, also „glatt“



Rauschen

Direkte Anwendung von Noise-Funktionen zur Texturierung

- ▶ Auswertung der Noisefunktion an Position \mathbf{x} mit Frequenz f und Offset \mathbf{o} : $n(f \cdot \mathbf{x} + \mathbf{o})$
- ▶ Offset \mathbf{o} kann z.B. von der Zeit abhängen, anderer Offset gibt anderes Rauschen
- ▶ Frequenz f bestimmt die Größe der Strukturen



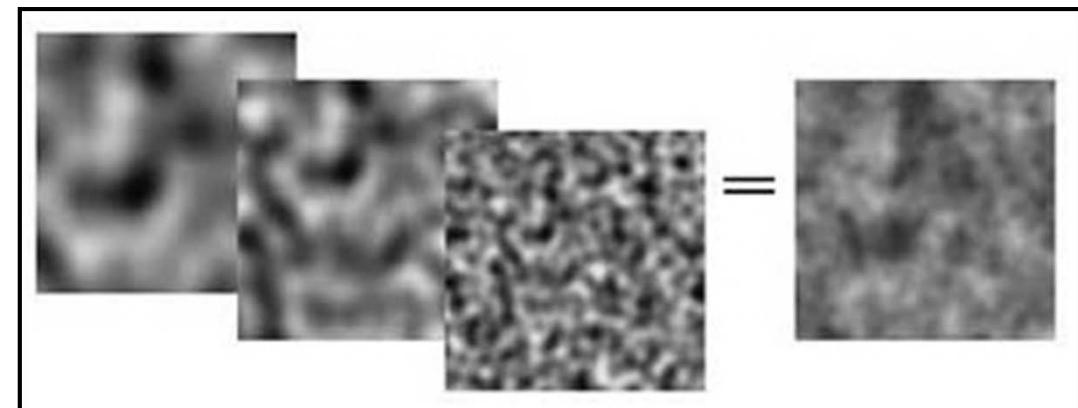
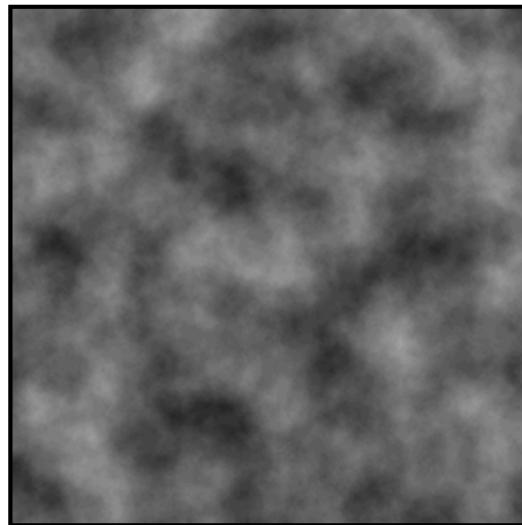
State of the Art-Report
über Noise-Funktionen
[http://www-sop.inria.fr/
reves/Basilic/2010/LLCDDELPZ10/
LLCDDELPZ10STARPNF.pdf](http://www-sop.inria.fr/reves/Basilic/2010/LLCDDELPZ10/LLCDDELPZ10STARPNF.pdf)



Turbulenz

Spektrale Synthese: Kombination unterschiedlicher Frequenzbereiche

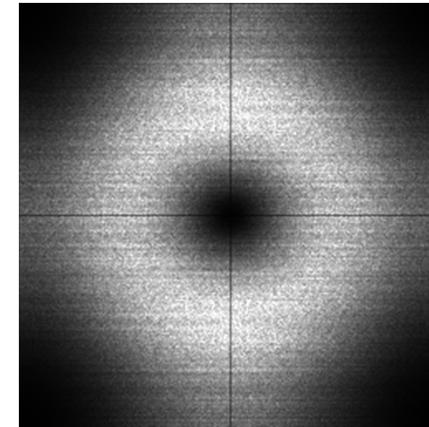
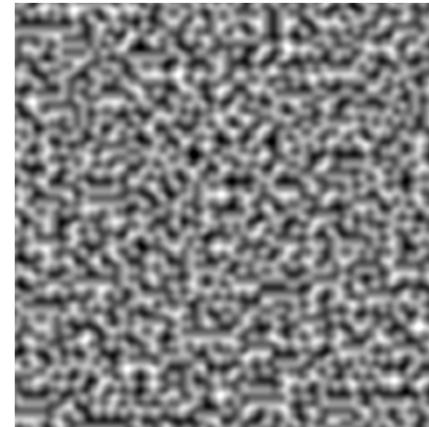
- ▶ $\text{turbulence}(\mathbf{x}) = \sum_k (\frac{1}{2})^k n(2^k \cdot \mathbf{x})$
 - ▶ Aufsummieren von k Oktaven, skaliert mit $(\frac{1}{2})^k$
 - ▶ addiere Details proportional zur Größe
(Selbstähnlichkeit, Fraktale, Brownsche Bewegung)
 - ▶ Frequenzspektrum $\frac{1}{f}$ (hier: $f = 2$), fraktales Rauschen



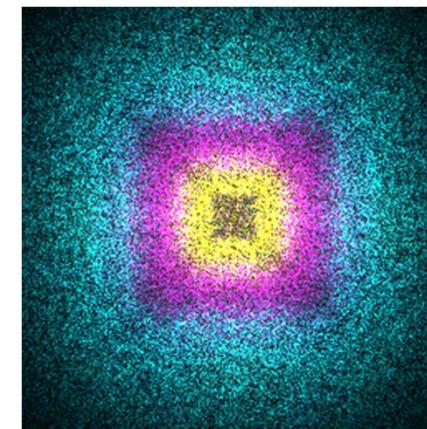
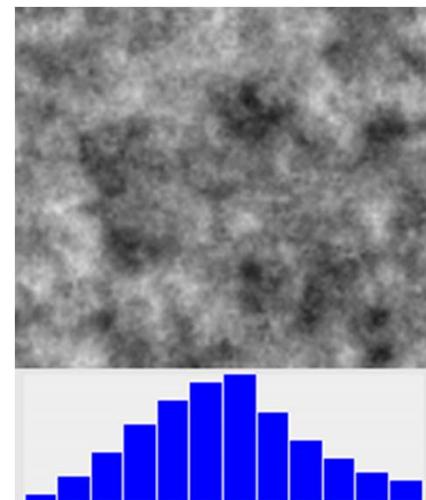
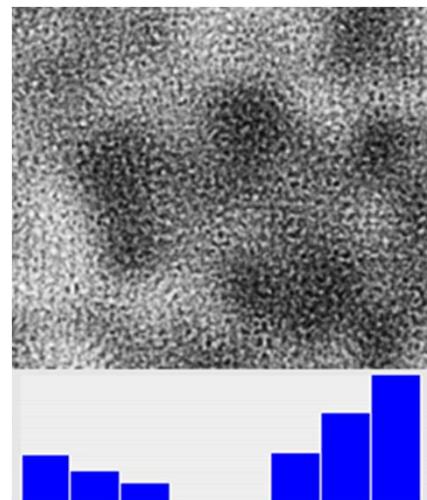
Turbulenz

Spektrale Synthese: Kombination unterschiedlicher Frequenzbereiche

- Bild und Frequenzspektrum einer Oktave



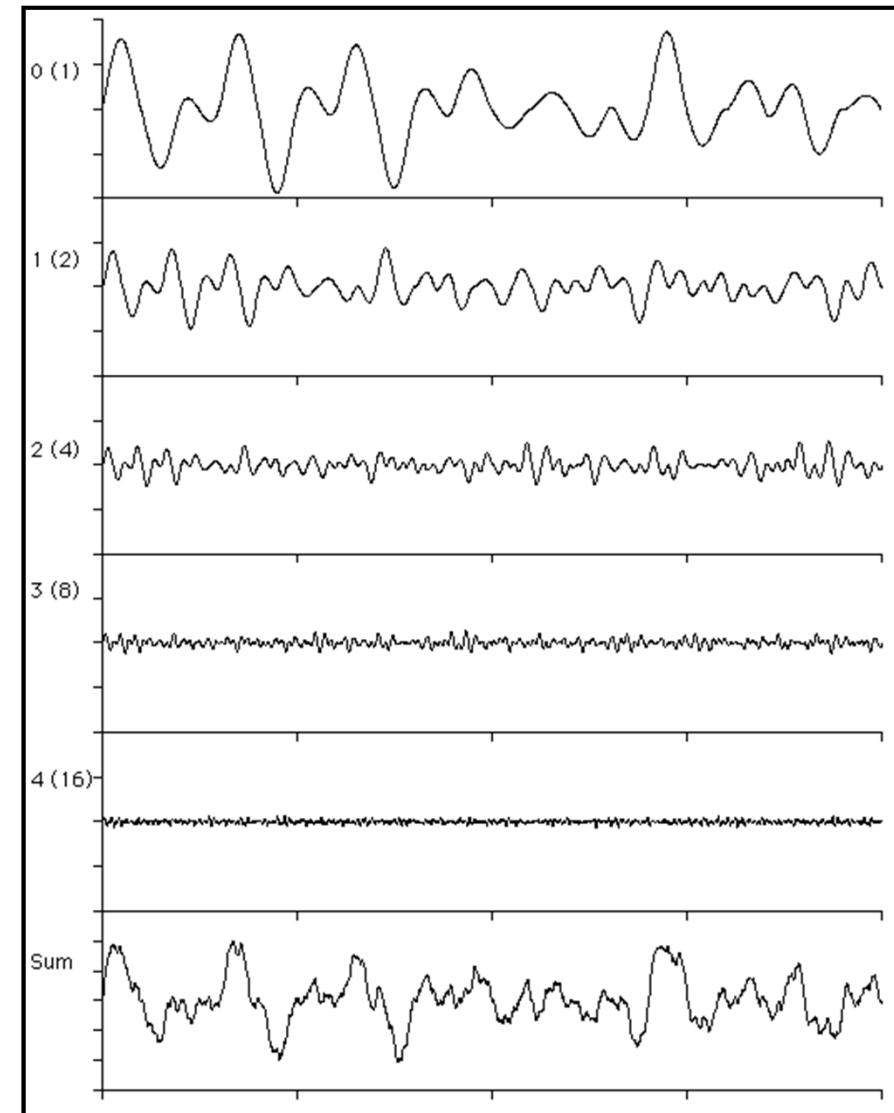
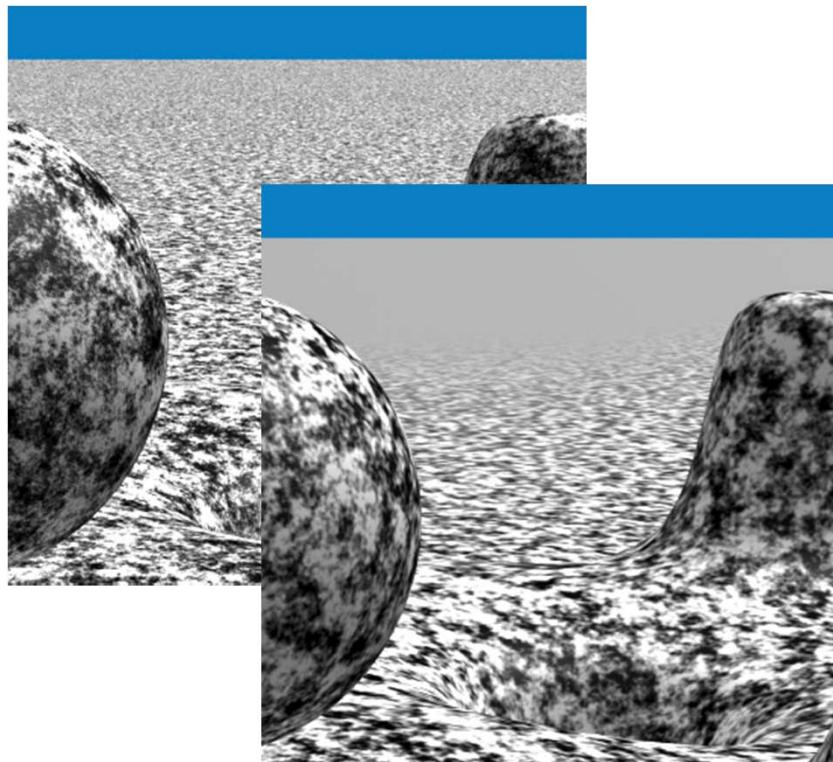
- gewichtete Summe mehrerer Oktaven (Spektren zeigen keine Turbulenz):



Turbulenz

Spektrale Synthese: Kombination unterschiedlicher Frequenzbereiche

- ▶ $\text{turbulence}(\mathbf{x}) = \sum_k (1/2)^k n(2^k \mathbf{x})$
- ▶ wie kann man hohe Frequenzen entfernen? (z.B. um beim Texturieren Aliasing zu vermeiden)
 - ▶ einfach Oktaven weglassen!



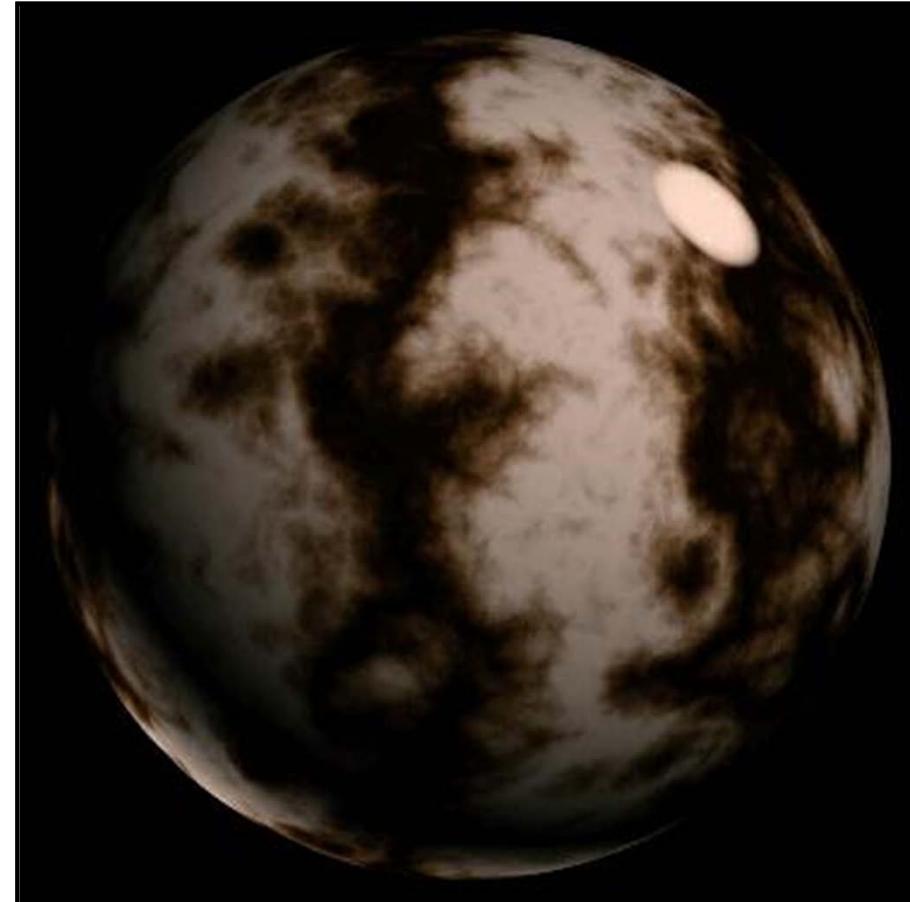
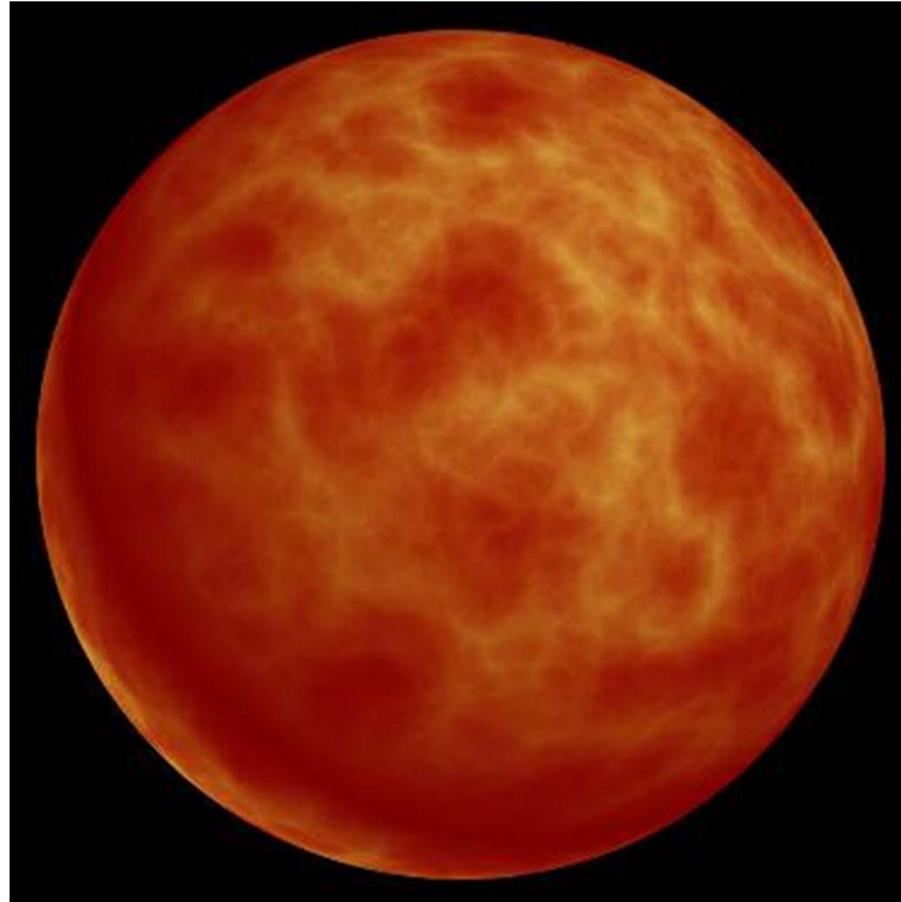
Turbulenz mit Strukturen (und Abb. auf Farbe)



- ▶ links: Turbulenz-Variante (Diskontinuitäten in Ableitung wg. Betrag)
- ▶ rechts: „Color-Mapping“ durch eine Sinus-Funktion

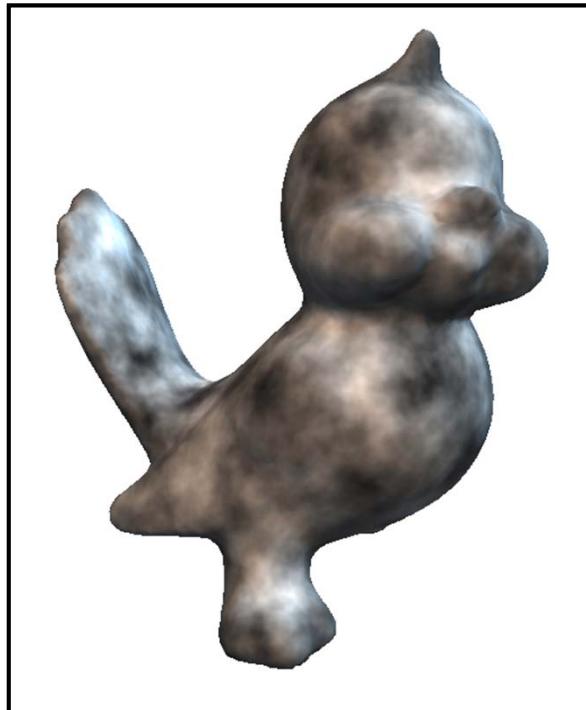
$$\sum_k \left(1/f\right)^k |n(f^k \mathbf{x})|$$

$$\sin\left(x + \sum_k \left(1/f\right)^k |n(f^k \mathbf{x})|\right)$$



Weiter Shader

- Noise als Basis für weitere Texturen (siehe Demo)



```
float turbulence(v,...)
```



```
frac( scale *  
turbulence(v,...) )
```



```
cos( v * sc1 +  
turbulence(v,...)*sc2 )
```

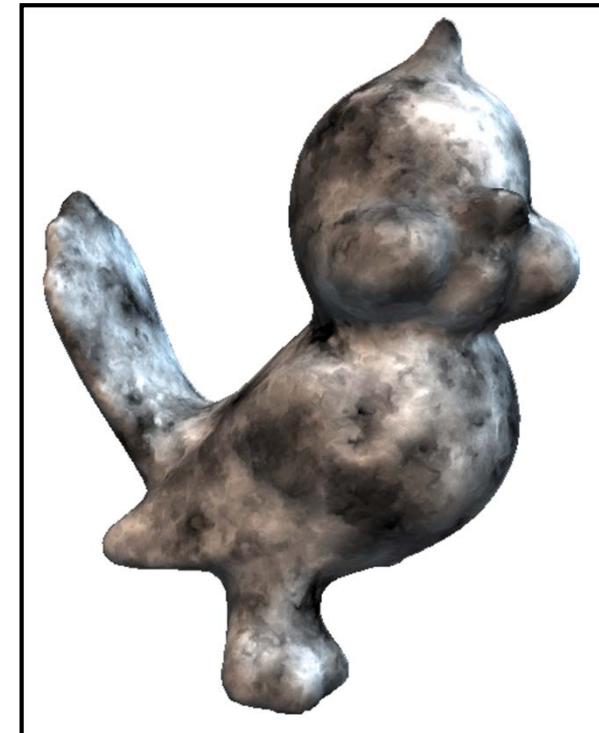
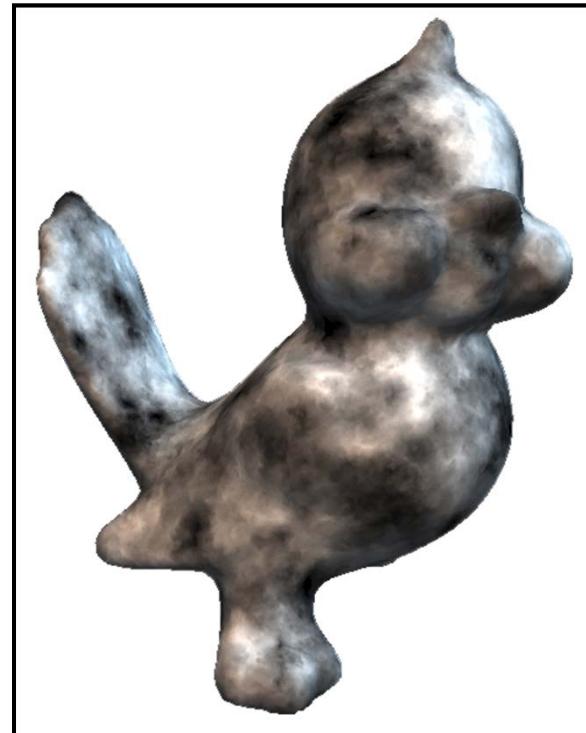
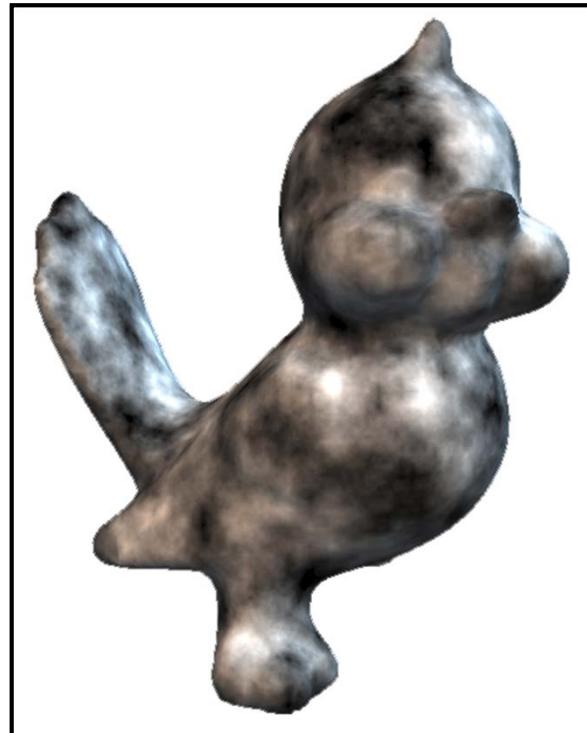
Kombination mehrerer Turbulenz-Texturen

- ▶ Bsp. ineinander verschachtelte Turbulenzfunktionen („Marmor“)

- ▶ links: $\text{turbulence}(\mathbf{x}) = \sum_k (1/2)^k |n(2^k \mathbf{x})|$

- ▶ mitte: $\text{turbulence}(\mathbf{x} + \frac{1}{2} \text{turbulence}(\mathbf{x}))$

- ▶ rechts: $\text{turbulence}(\mathbf{x} + \text{turbulence}(\mathbf{x}))$



„Natürliche“ Phänomene

Beispiel: Feueranimation

- ▶ 4D-Turbulenztextur mit mehreren Oktaven
 - ▶ Zugriff mit Position (3D) und Zeit (1D)
 - ▶ Turbulenz wird auf Farbwerte und Semitransparenz abgebildet
 - ▶ zusätzlich zu Turbulenz:
nach oben hin transparenter
- ▶ zeitliche Variationen
 - ▶ 3D-Schnitte aus 4D-Turbulenz-Textur
 - ▶ räumliche Kohärenz wird zu zeitlicher Kohärenz!
- ▶ andere Ansätze (siehe Hypertexture):
<http://blog.char95.com/post/fire/>
<http://developer.download.nvidia.com/SDK/10.5/direct3d/samples.html#PerlinFire> (Bild rechts)



Weitere Anwendungen

Bump/Displacement Mapping

- ▶ Interpretiere Noise-/Turbulenz-Funktion als Höhenfeld



Prozedurale Landschaften

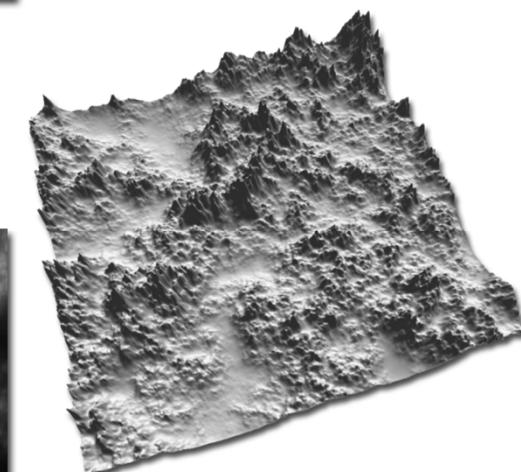
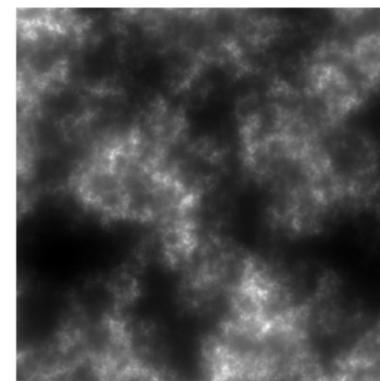
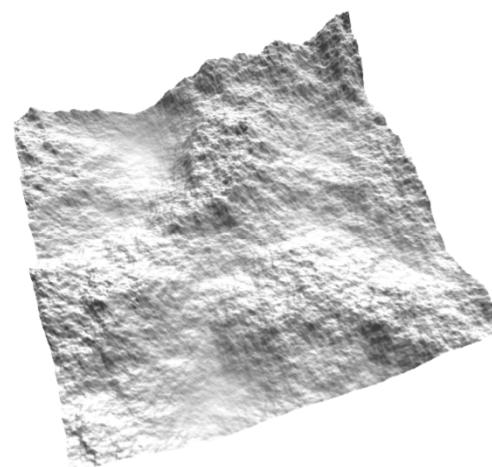
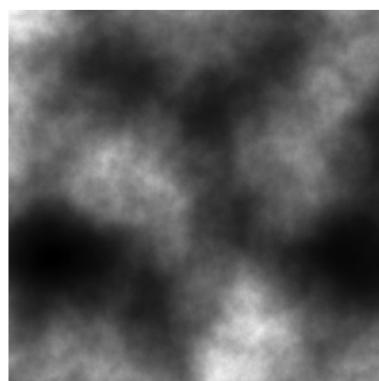
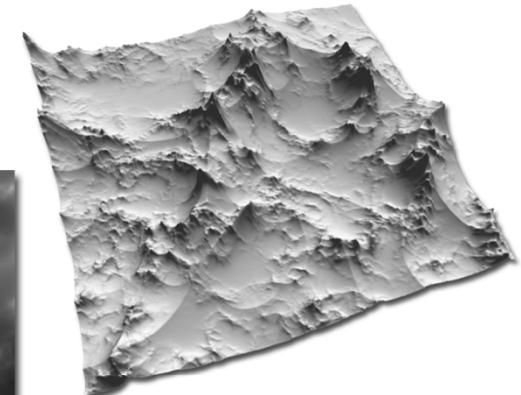
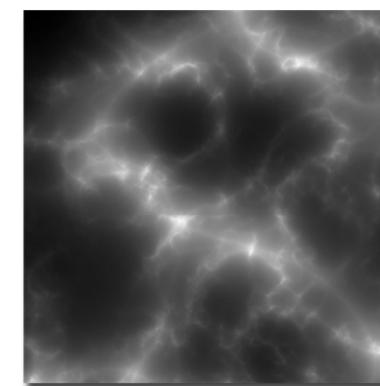
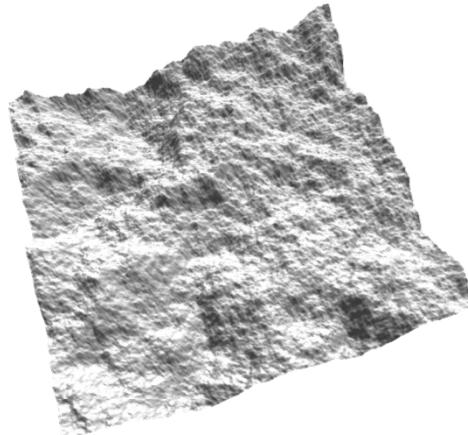
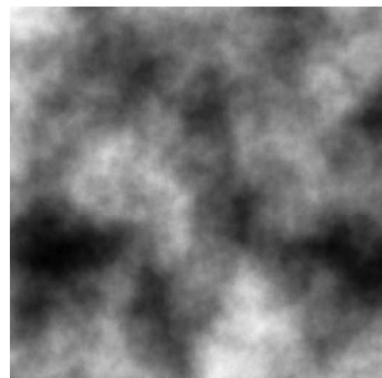
- ▶ Höhenfelder und Texturen generiert aus Noise-Funktionen



Beispiele: Höhenfelder aus Noise Funktionen



Unterschiedliche Verknüpfung (additiv, multiplikativ, ...) von Rauschfkt.



Prozedurale Landschaften



Prozedurale Texturierung (häufig verwendeter Ansatz, z.B. Terragen)

- ▶ keine Textur

- ▶ Texturierung abhängig von Höhe $h(\mathbf{x})$
$$\text{if } (h(\mathbf{x}) > 1000m) \text{ color} = \text{white};$$

- ▶ Textur mit Steigung $s(\mathbf{x})$ und Noise
$$\text{if } \begin{pmatrix} (h(\mathbf{x}) + n(\mathbf{x})) \text{ in } [h_{min}; h_{max}] \wedge \\ (s(\mathbf{x}) + n(\mathbf{x})) \text{ in } [s_{min}; s_{max}] \end{pmatrix} \text{ color} = \text{green};$$



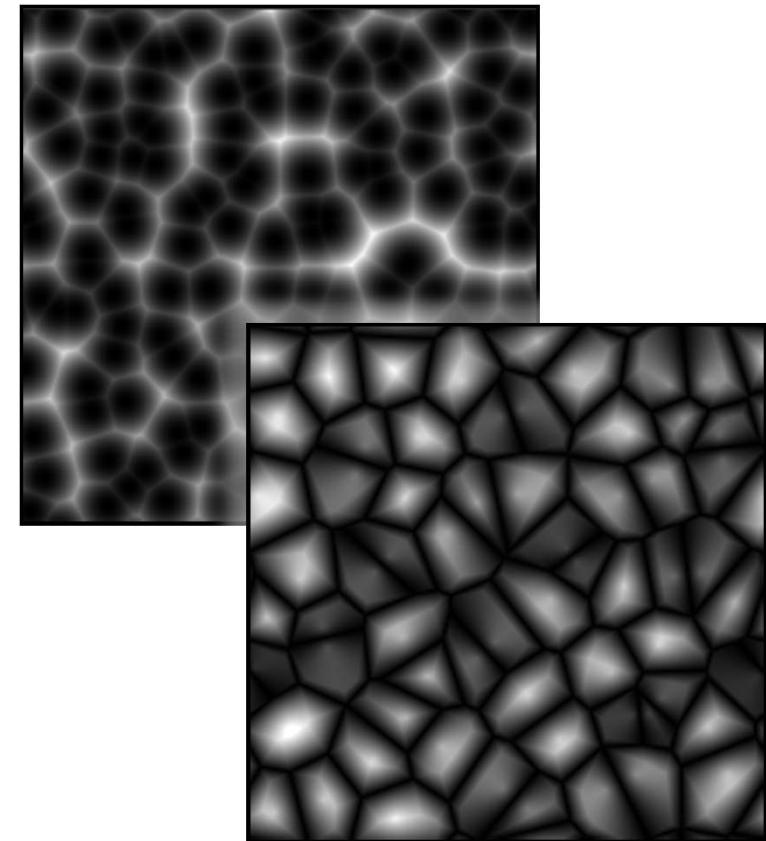
Prozedurale Landschaften

- Modelle für Texturen, Wolken, Wasserbewegung und -farbe, ...



Beispiele

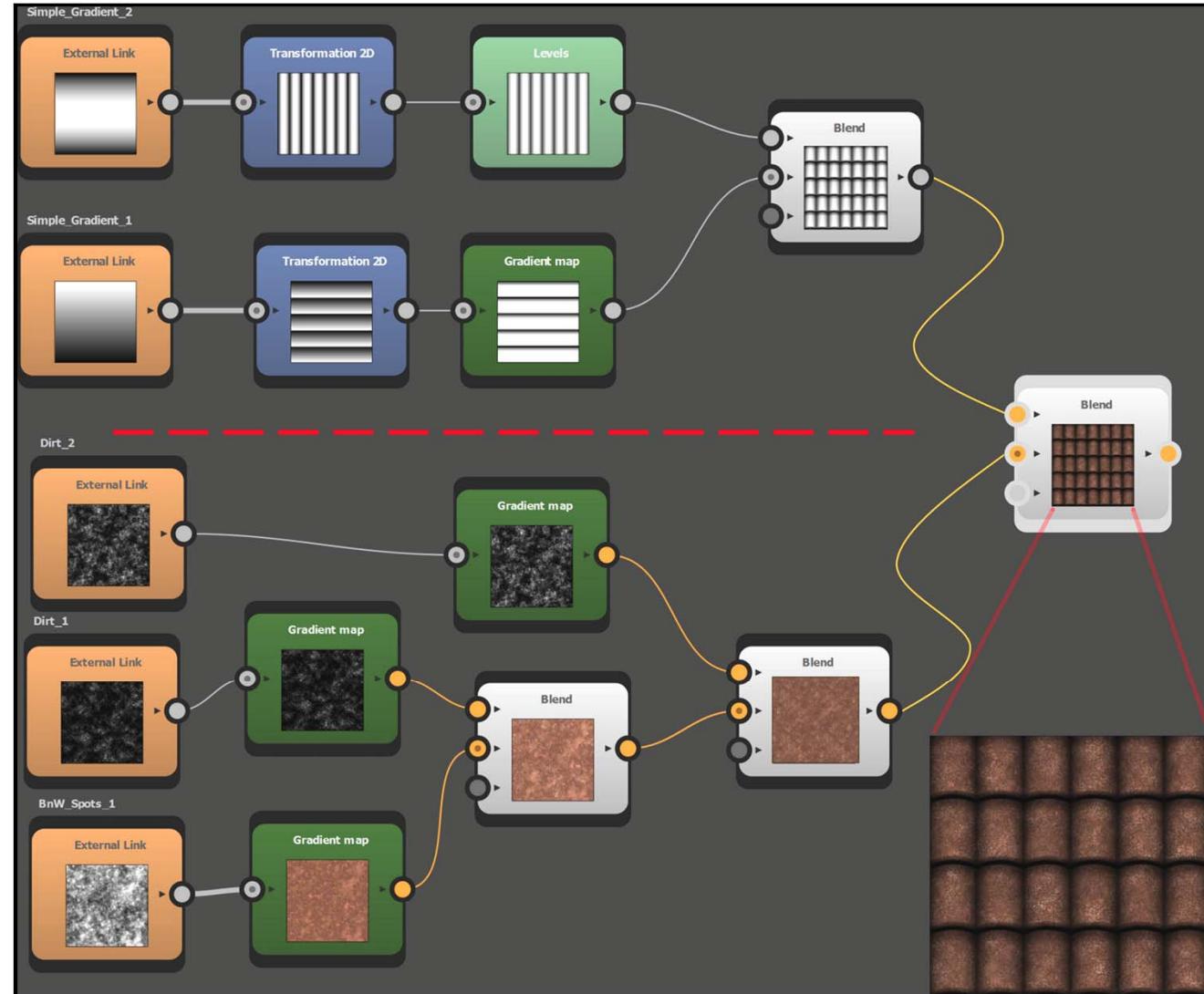
- ▶ viele weitere Beispiele und mehr Infos:
„Texturing and Modeling, A Procedural Approach“
David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley



Worley's Cellular Textures

Ausblick: prozedurale Texturen in der Praxis

- Kombination mit regelbasierten Systemen



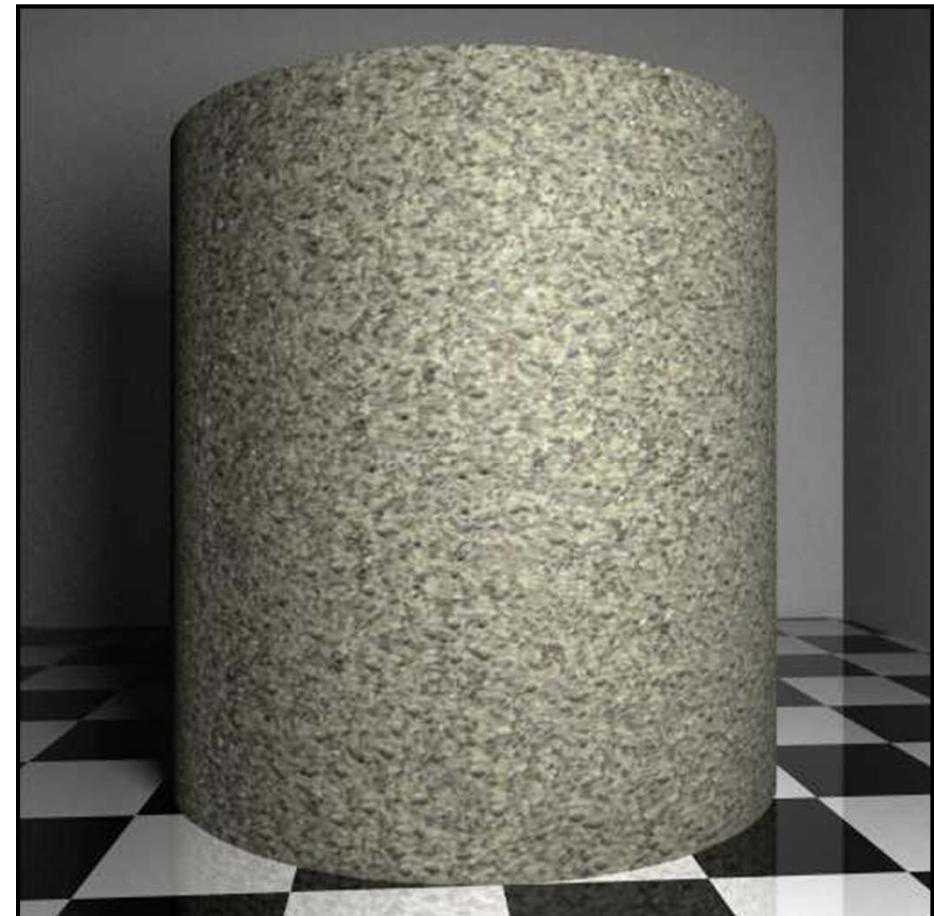
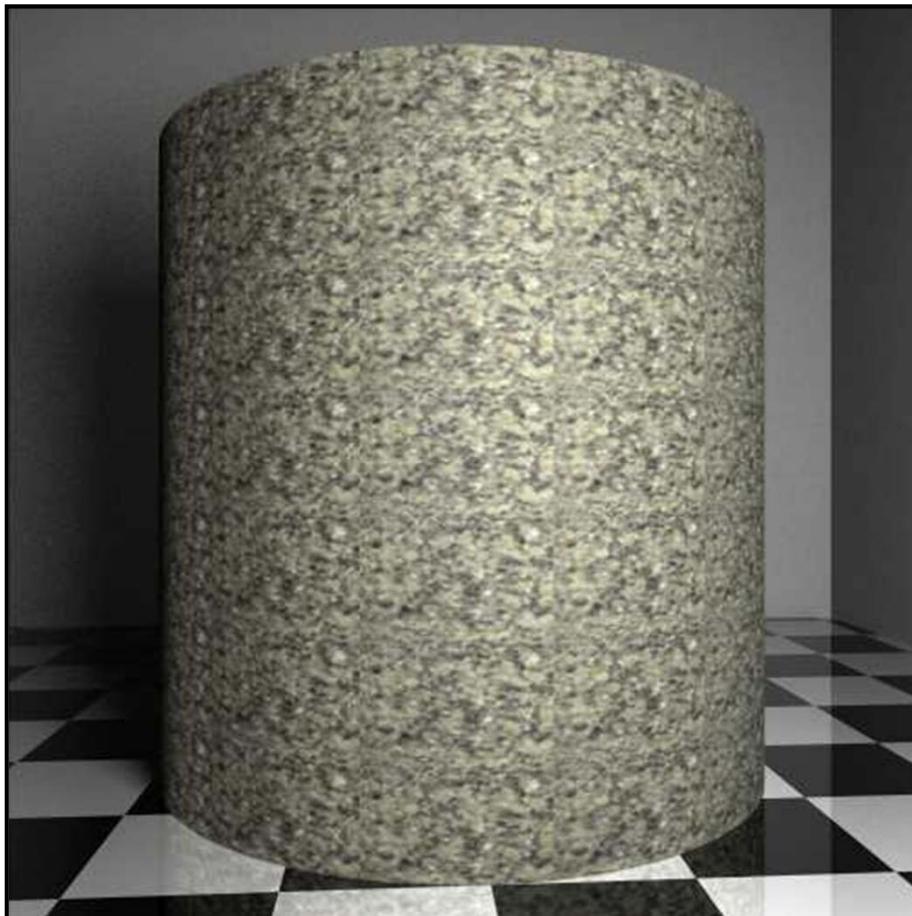
Prozedurale Texturen



- ▶ algorithmische Beschreibung von Texturen
 - ▶ kompakte Repräsentation (wenige Zeilen Code)
 - ▶ auflösungsunabhängig
 - ▶ für beliebig große Flächen (ohne sichtbare Wiederholungen)
 - ▶ parametrisierbar (z.B. mehrere Sorten Holz)
- ▶ Implizite Methode
 - ▶ Aufruf des Shaders für jeden Pixel/Schnittpunkt (wie eben)
 - ▶ benötigt evtl. Zugriff auf benachbarte Pixel (Ableitung)
 - ▶ Filterung ist nicht trivial (bei Turbulenz schon)
- ▶ Explizite Methode
 - ▶ generiere 2D/3D-Textur vor der Verwendung
 - ▶ verwende Texture-Mapping wie bisher
 - ▶ Vorteil: Texture-Filtering ist einfach

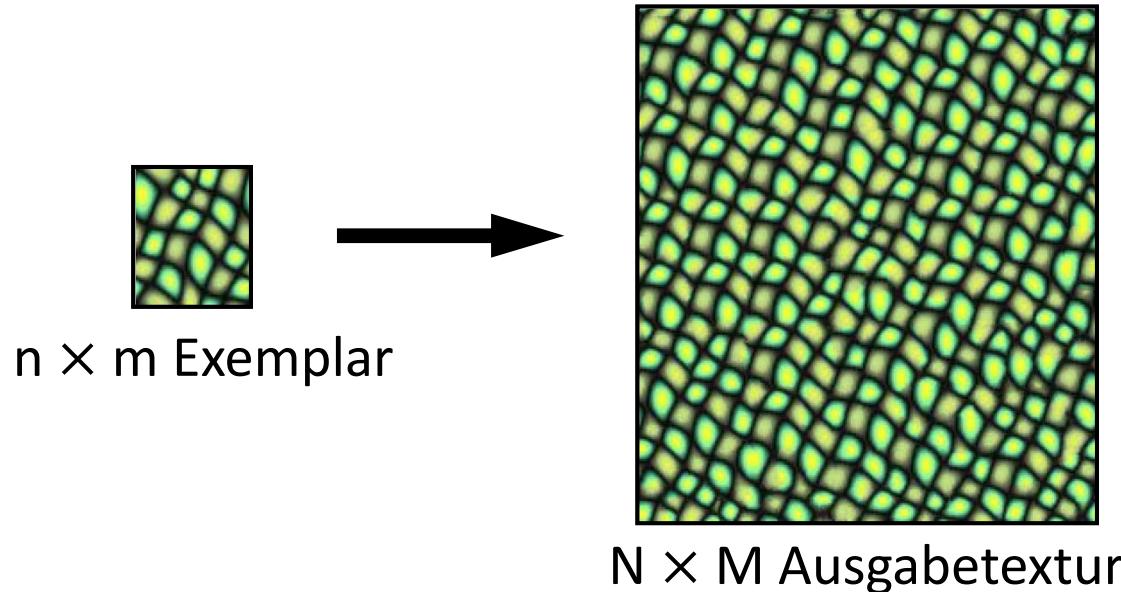
Textursynthese

- ▶ oft kleine Textur (Examplar) vorgegeben (z.B. extrahiert aus Fotografie)
 - ▶ Tiling (Kachelung) führt zu sichtbaren Wiederholungen
- ▶ Ziel: Berechnen (**Synthese**) einer großen Textur die „genauso“ aussieht
 - ▶ Verfahren in dieser Vorlesung langsam: keine Auswertung zur Laufzeit



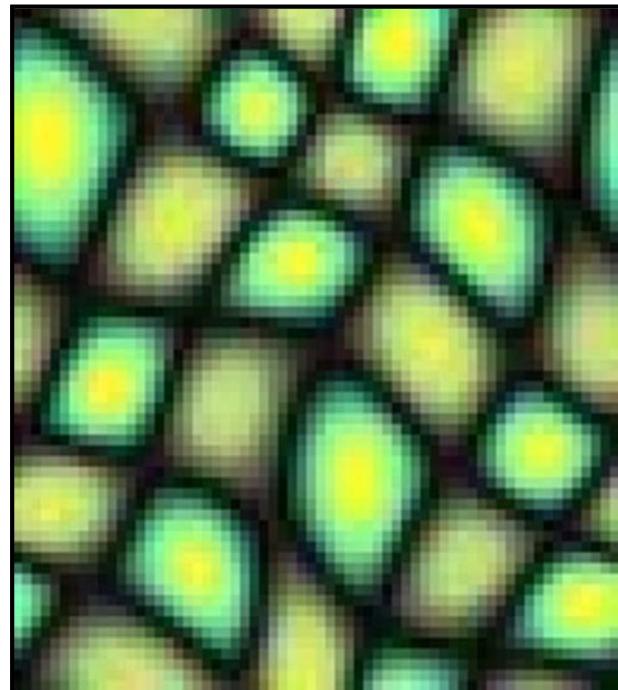
Textursynthese

- ▶ **Ziel:** synthetisiere eine Textur die genau wie die Eingabetextur aussieht
 - ▶ „gleich aussehen“ bedeutet „gleiche statistische Eigenschaften“
 - ▶ achte auch darauf genügend Variation zu erzeugen
- ▶ es geht nur um stochastischen Inhalt – Texturen mit semantischen Strukturen lassen sich so nicht erzeugen

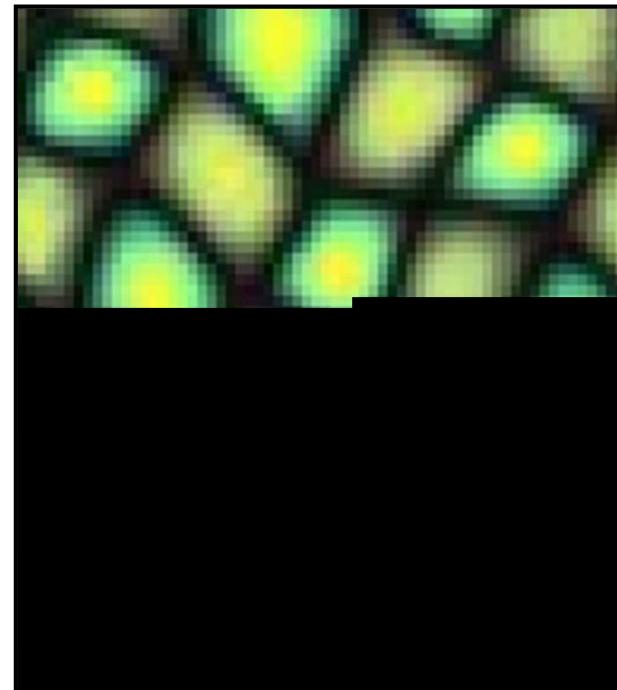


Pixelbasierte Textursynthese

- ▶ Ausgabetextur wird Pixel für Pixel erzeugt
 - ▶ von links nach rechts und oben nach unten
 - ▶ Annahme für die Erklärung:
wir haben schon einen Teil der Textur erzeugt
(Anm. in diesem Beispiel sind Exemplar und Ausgabe gleich groß)



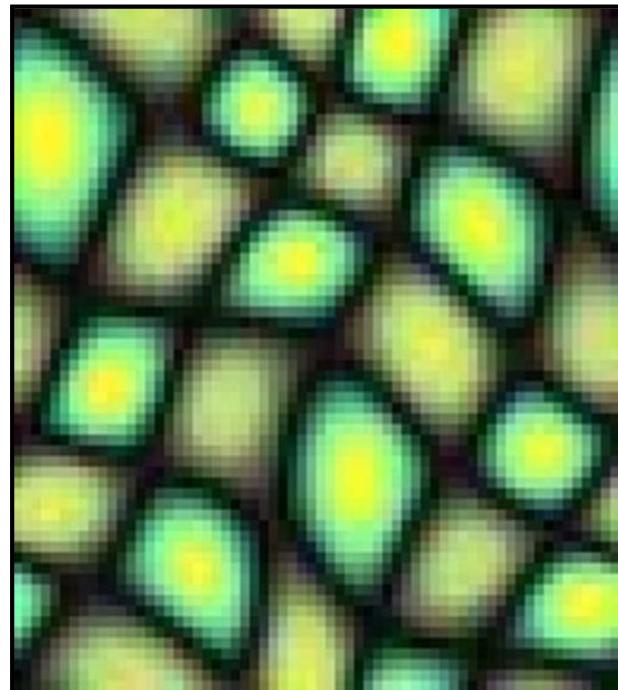
Exemplar



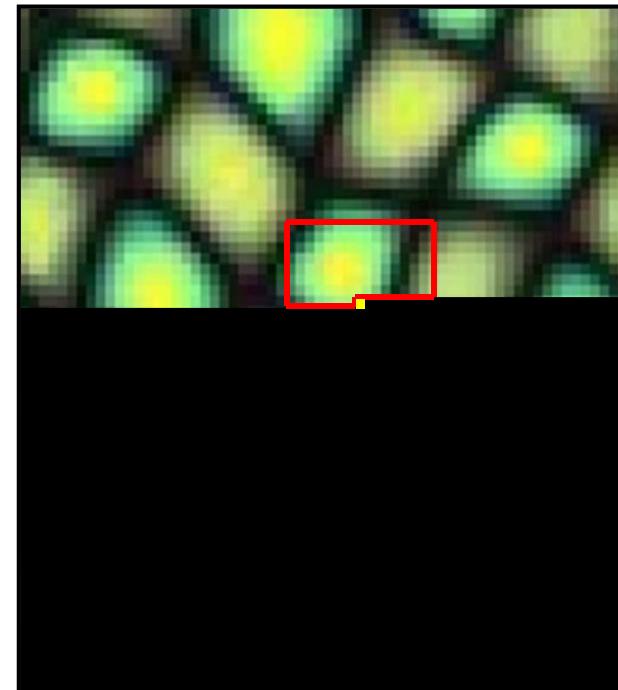
Resultat

Pixelbasierte Textursynthese

- ▶ Ausgabetextur wird Pixel für Pixel erzeugt
 - ▶ von links nach rechts und oben nach unten
- ▶ betrachte Nachbarschaft des nächsten zu erzeugenden Pixels
 - ▶ i.d.R. kleine Nachbarschaften ca. 20-50 Pixel
 - ▶ aber groß genug, um Strukturen in der Textur zu erfassen



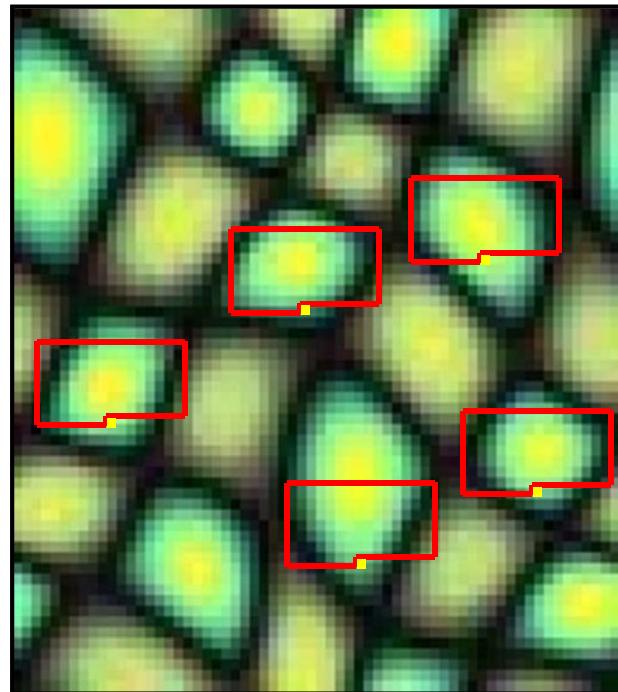
Exemplar



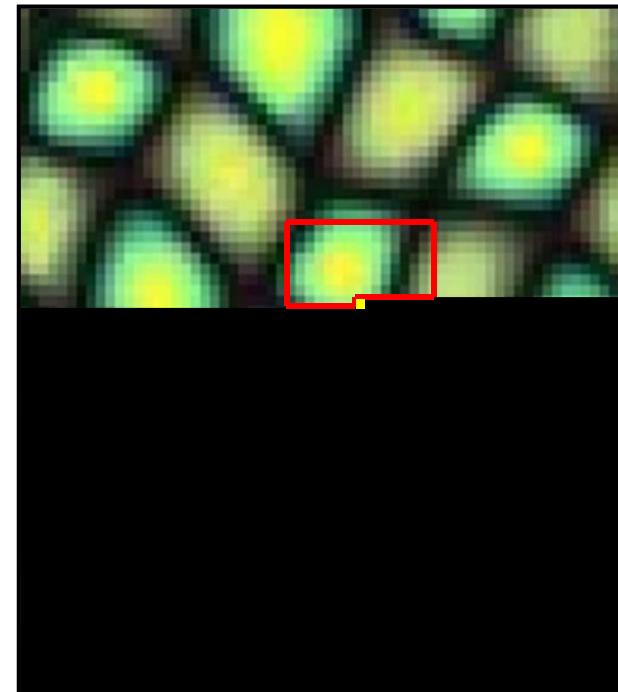
Resultat

Pixelbasierte Textursynthese

- ▶ Ausgabetextur wird Pixel für Pixel erzeugt
 - ▶ von links nach rechts und oben nach unten
- ▶ suche ähnliche Nachbarschaften in der Eingabetextur
 - ▶ ähnlich = Summe über alle Pixelfarbdifferenzen klein



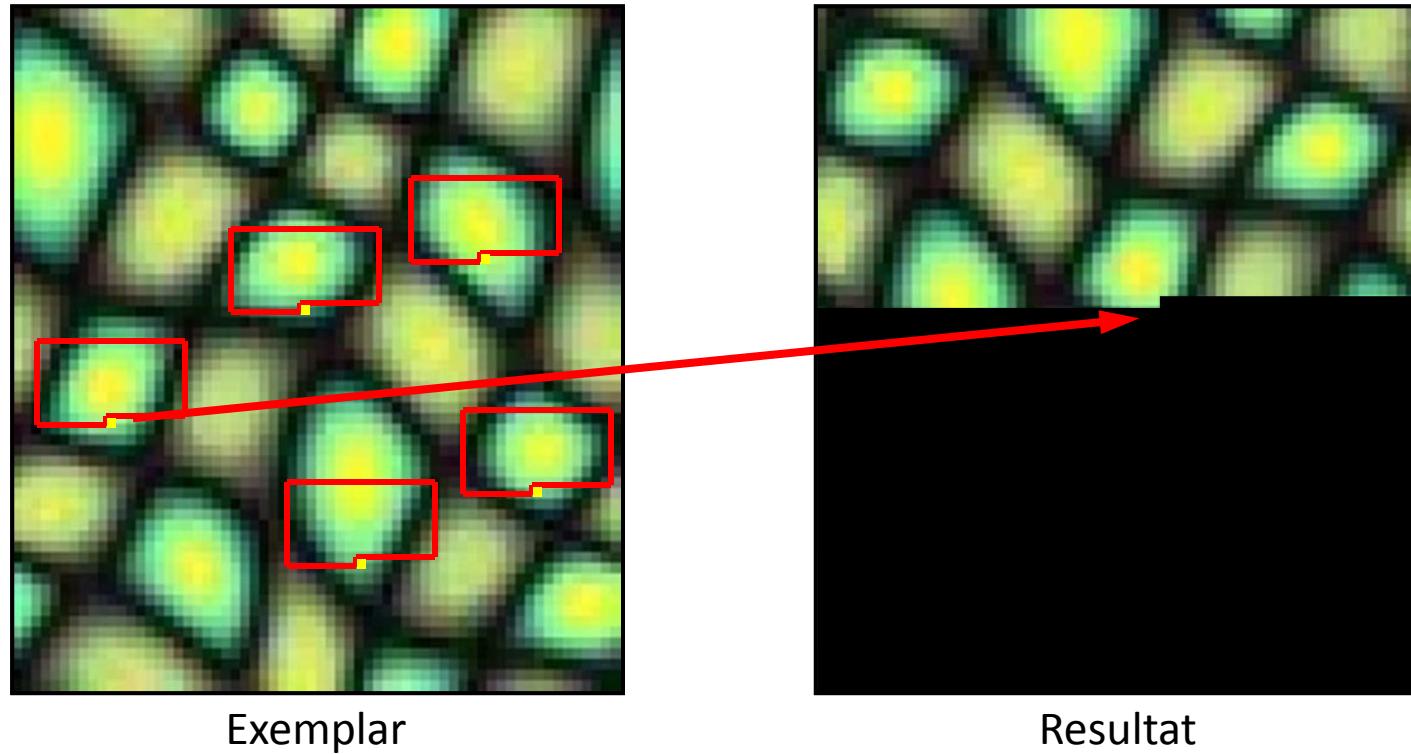
Exemplar



Resultat

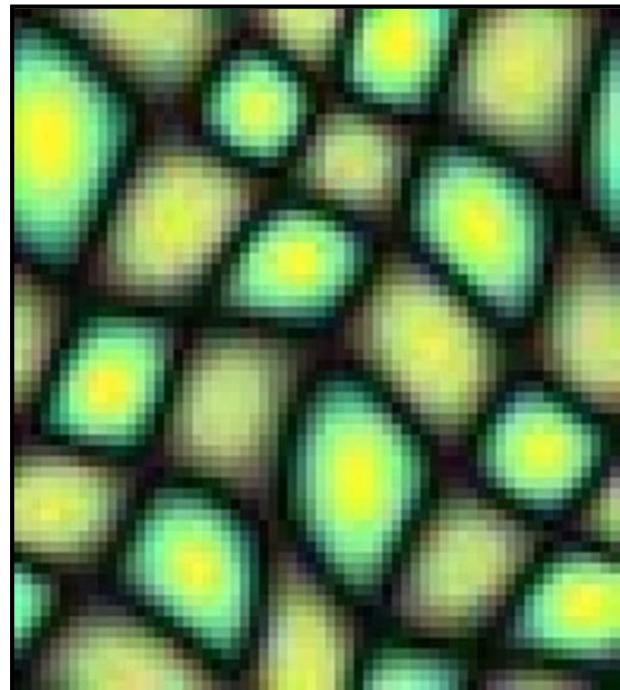
Pixelbasierte Textursynthese

- ▶ Ausgabetextur wird Pixel für Pixel erzeugt
 - ▶ von links nach rechts und oben nach unten
- ▶ kopiere den Pixel **einer der ähnlichsten** Nachbarschaften
 - ▶ stochastische Variation

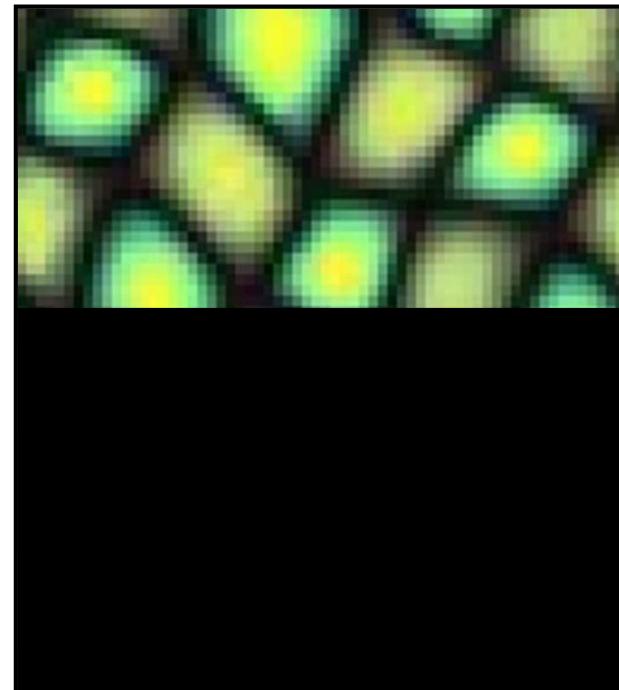


Pixelbasierte Textursynthese

- ▶ Ausgabetextur wird Pixel für Pixel erzeugt
 - ▶ von links nach rechts und oben nach unten



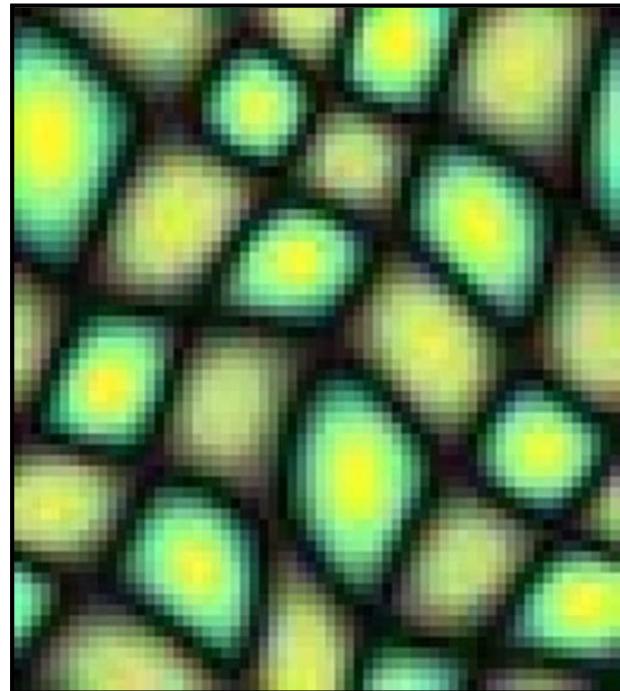
Exemplar



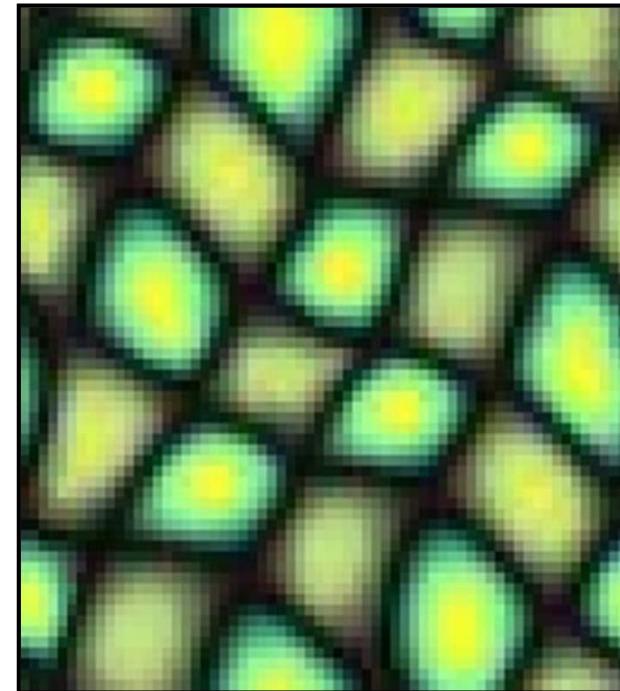
Resultat

Pixelbasierte Textursynthese

- ▶ Ausgabetextur wird Pixel für Pixel erzeugt
 - ▶ von links nach rechts und oben nach unten
- ▶ im Prinzip sehr einfach
- ▶ aber sehr langsam durch die Suche nach passenden Nachbarschaften

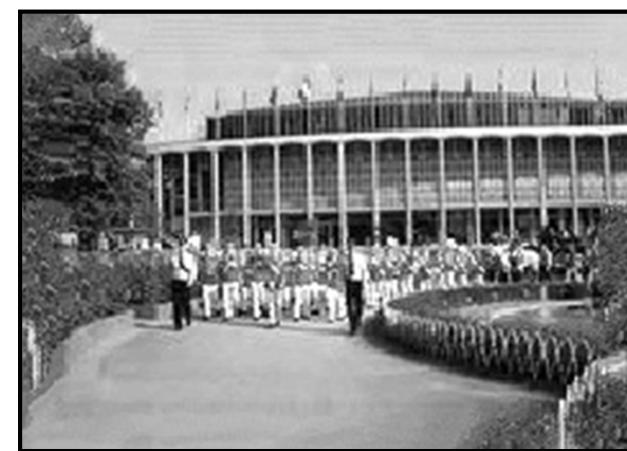
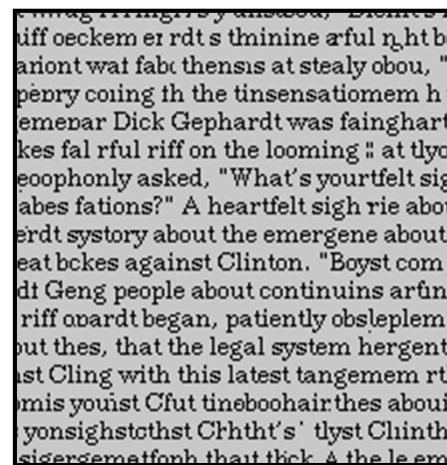
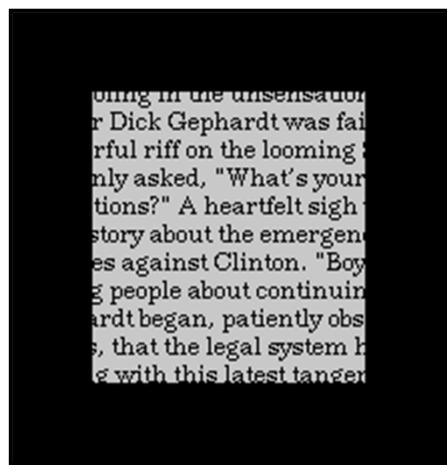
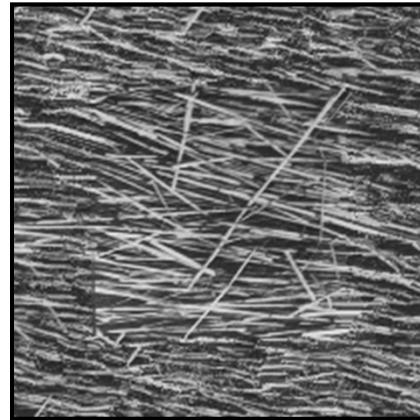
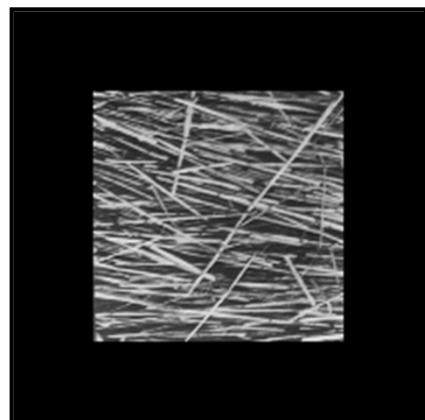
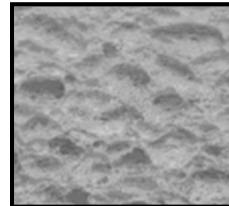
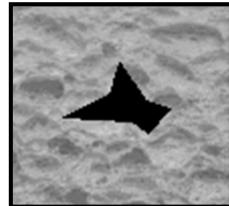


Exemplar



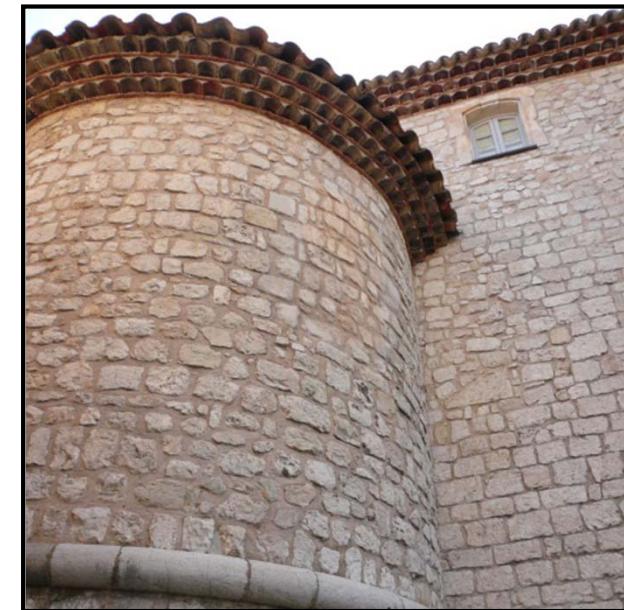
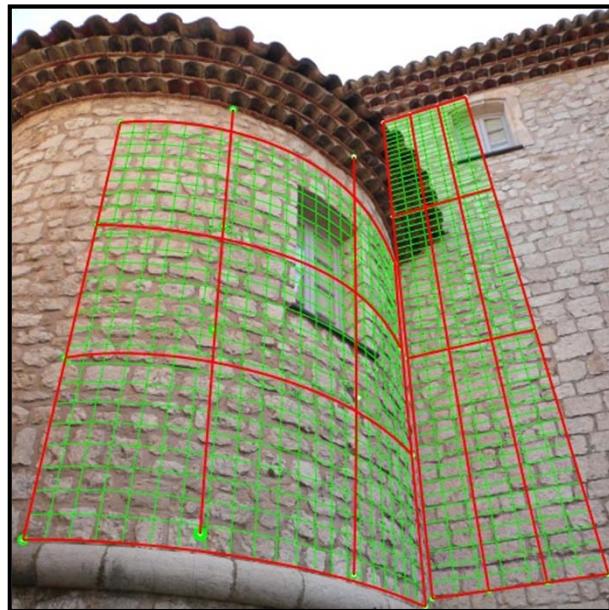
Resultat

Pixelbasierte Textursynthese [Efros99]



Pixelbasierte Textursynthese

- Berücksichtigung der Perspektive der Eingabe- und Ausgabetextur ist ebenfalls direkt möglich [Eisenacher et al. 08]



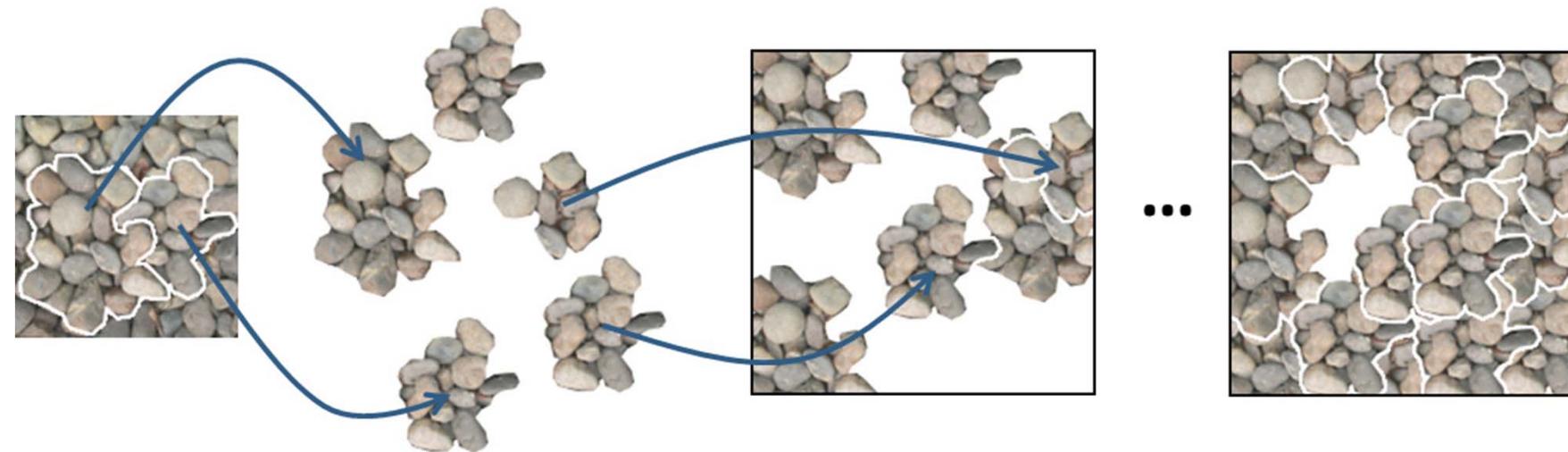
Anwendungsbeispiel

► Bild: Disney's Tangled



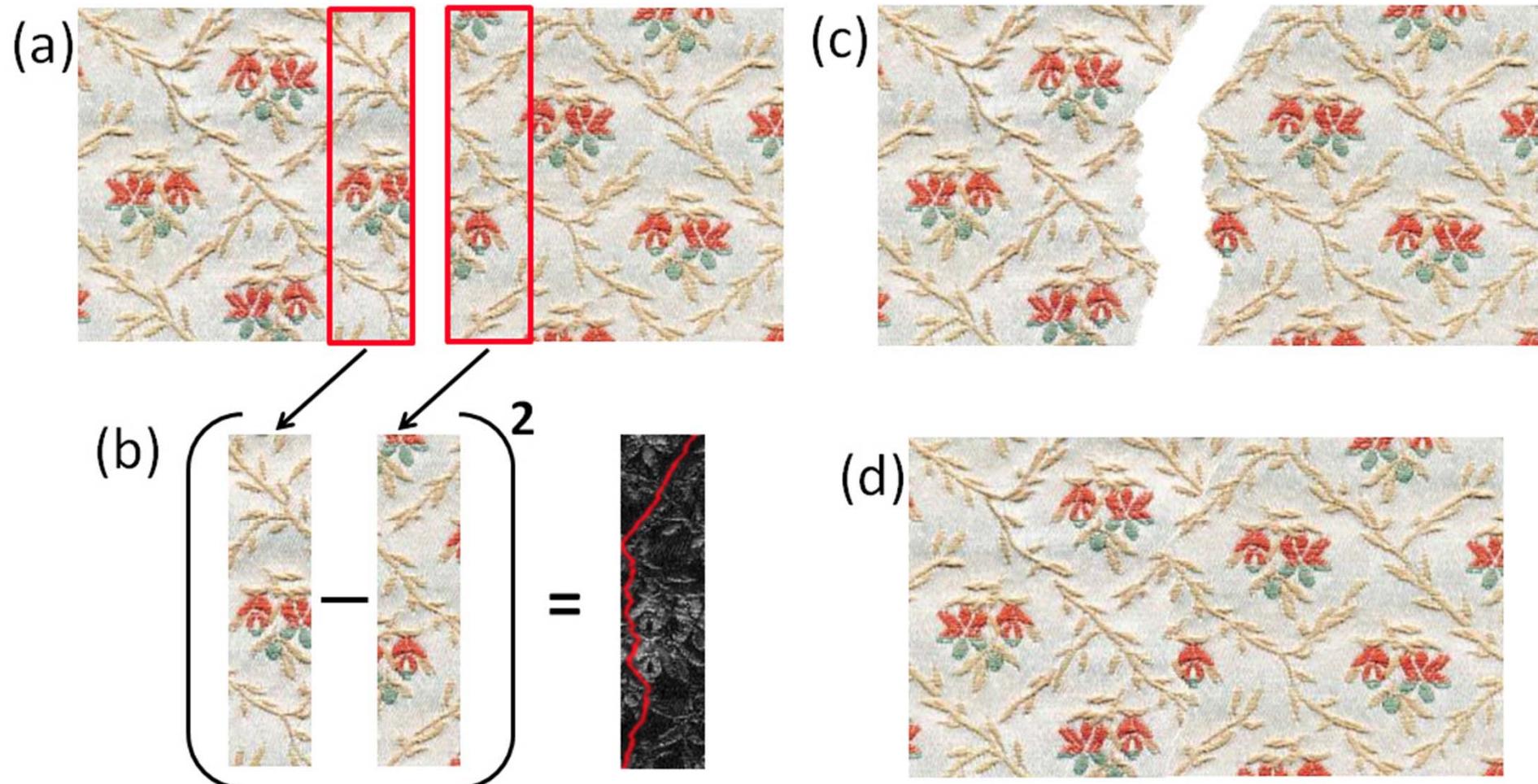
Exkurs: Patchbasierte Textursynthese

- Grundidee: verwende größere Bereiche des Exemplars



Exkurs: Patchbasierte Textursynthese

- ▶ Grundidee: verwende größere Bereiche des Exemplars
- ▶ ...was nicht passt, wird passend gemacht (Graph Cut)



Exkurs: Image Quilting (patchbasierte Synthese)

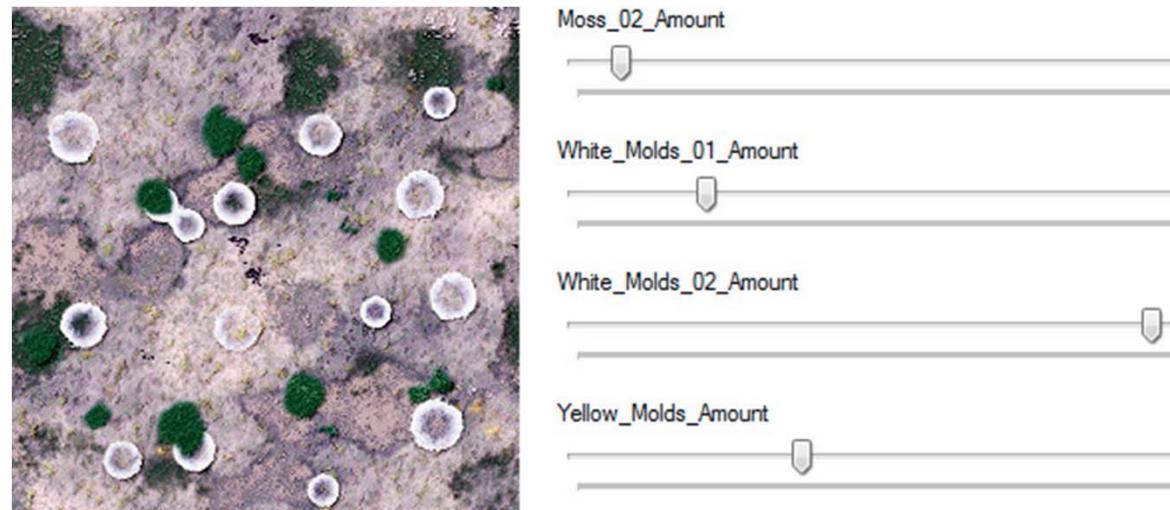


► <http://graphics.cs.cmu.edu/people/efros/research/quilting.html>



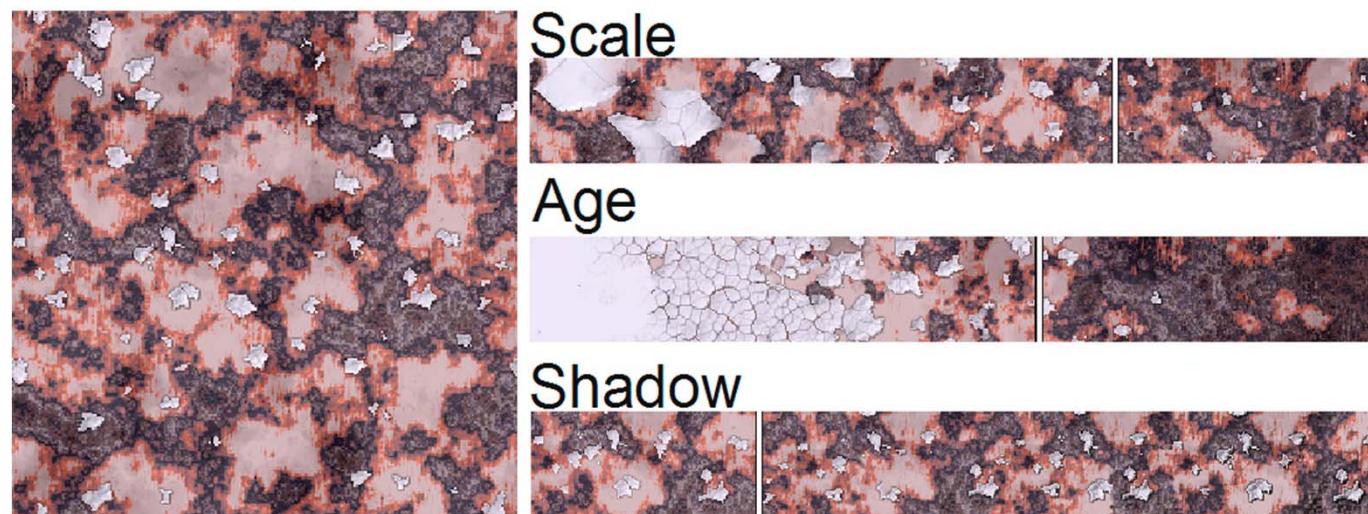
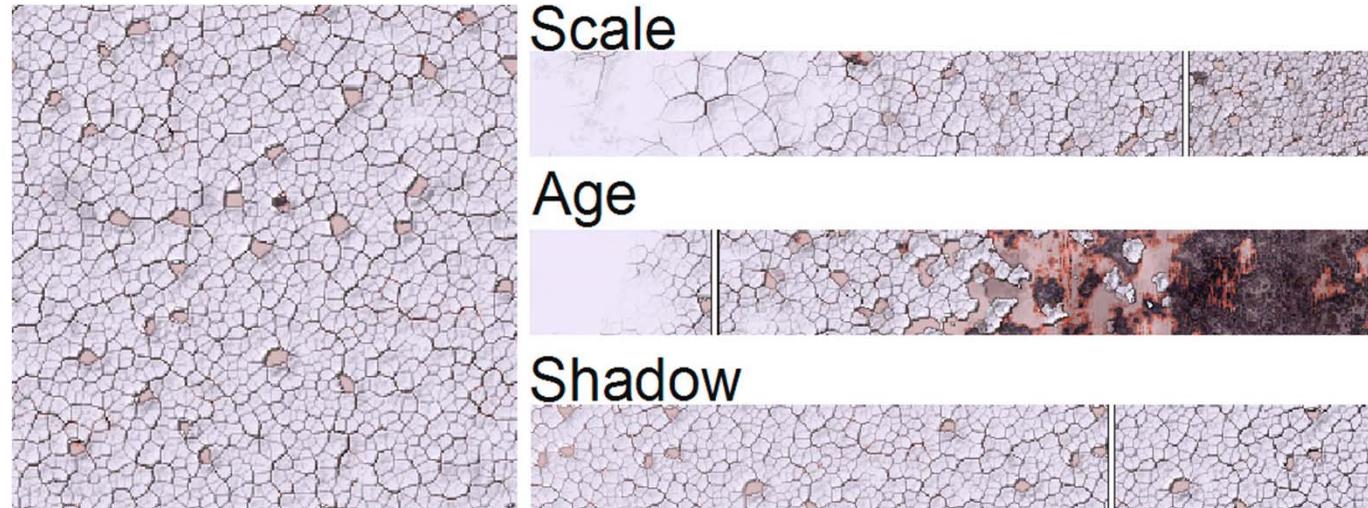
Ausblick: aktuelle Forschungsthemen

- Modelle mit semantischen Parametern
(Software Substance/Allegoricalmic)



Ausblick: aktuelle Forschungsthemen

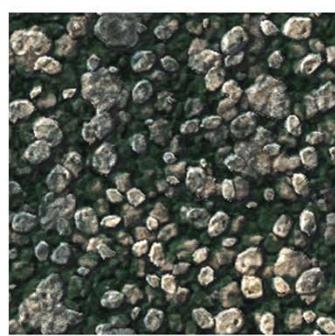
- ▶ Forschung: Synthese einer Textur für ein Preview der Auswirkungen
(A. Lasram und S. Lefebvre)



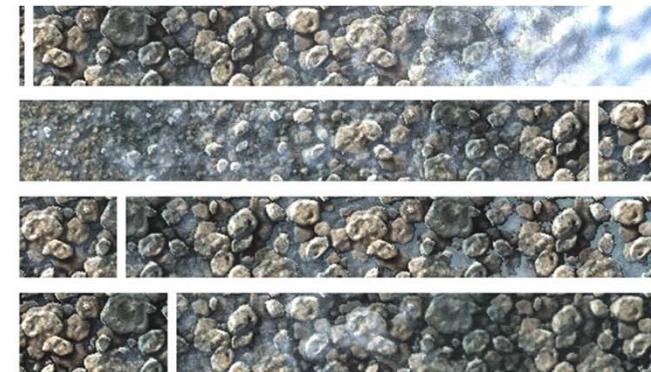
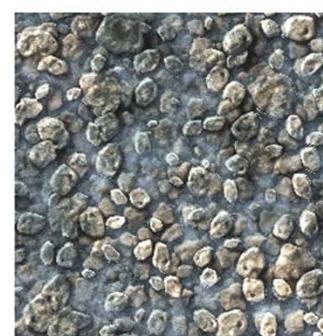
Ausblick: aktuelle Forschungsthemen



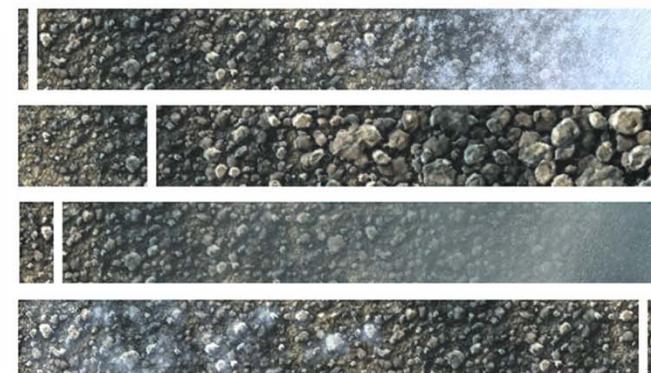
- ▶ Forschung: Synthese einer Textur für ein Preview der Auswirkungen
(A. Lasram und S. Lefebvre)



Snow
Sliders
Stones scale
Water level
Water moss



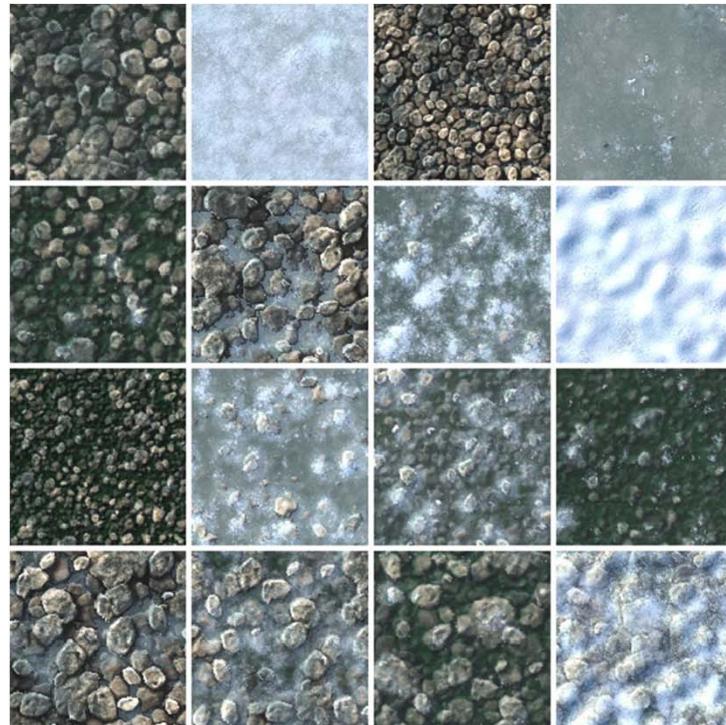
Snow
Sliders
Stones scale
Water level
Water moss



Ausblick: aktuelle Forschungsthemen

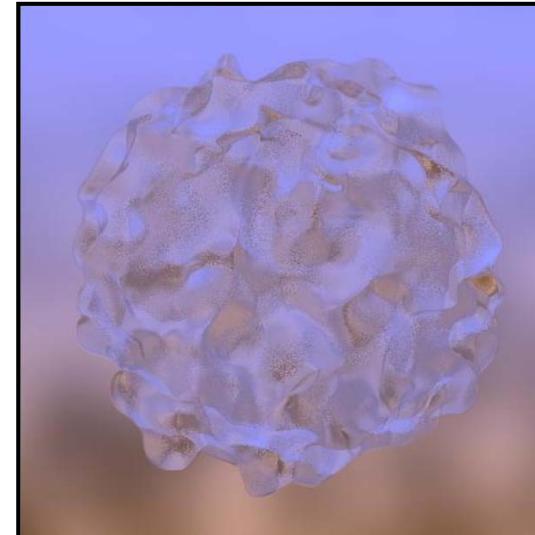


- Forschung: Synthese einer Textur für ein Preview der Auswirkungen
(A. Lasram und S. Lefebvre)



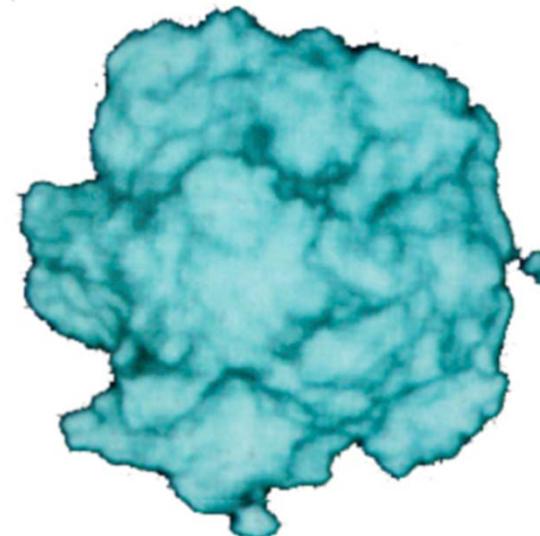
Hypertextures (Perlin [1989])

- ▶ Ziel: Erzeugung von komplexen 3D Texturen (z.B. Feuer, Wolken)
- ▶ Beschreibung eines Hypertexture-Objekts durch eine Dichtefunktionen (DF) im \mathbb{R}^3
 - ▶ innen $D(\mathbf{x}) = 1$, außen $D(\mathbf{x}) = 0$, Übergangsbereich $0 < D(\mathbf{x}) < 1$
 - ▶ komplexe Strukturen durch Noise-Funktionen und Modulation



Hypertextures

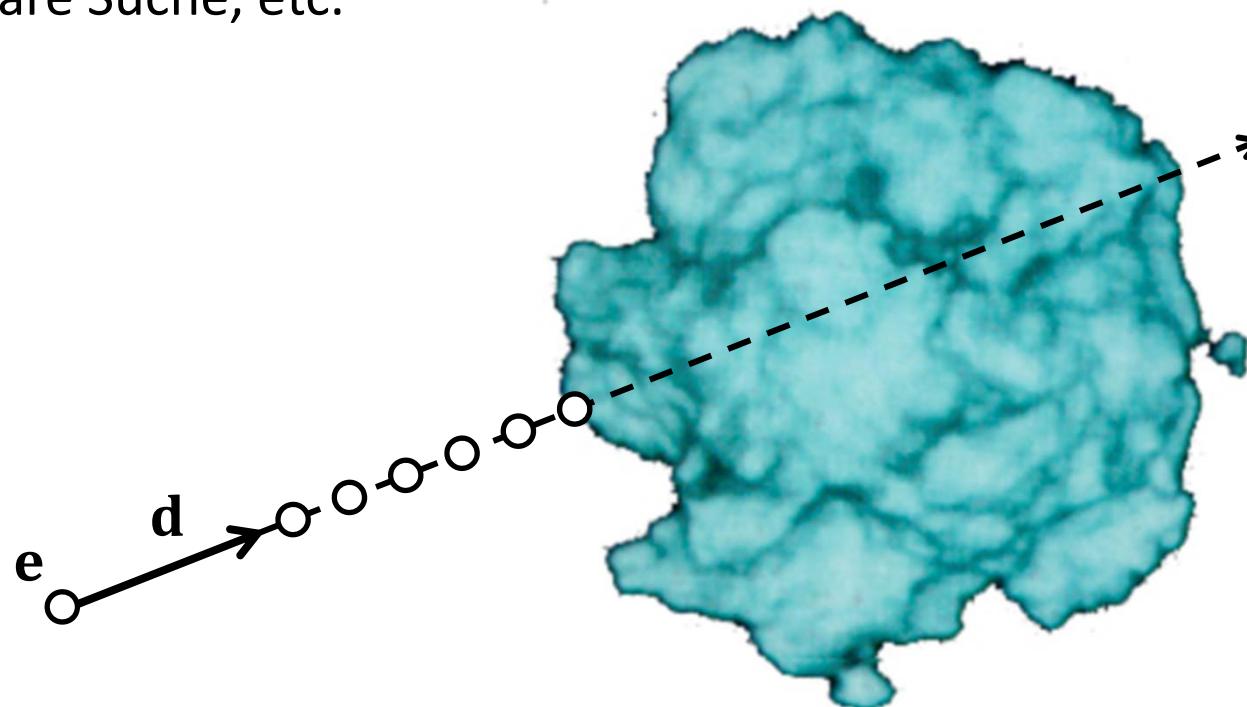
- ▶ Objekte werden beschrieben durch eine Dichtefunktionen (DF) im \mathbb{R}^3
 - ▶ innen $D(\mathbf{x}) = 1$, außen $D(\mathbf{x}) = 0$, Übergangsbereich $0 < D(\mathbf{x}) < 1$
- ▶ Bsp. Dichtefunktion ohne Transparenz
 - ▶ $D(\mathbf{x}) = \text{sphere} \left(\mathbf{x} \cdot \left(1 + \sum_i 1/_{2^i} n(2^i f\mathbf{x}) \right) \right)$
 - ▶ mit $\text{sphere}(\mathbf{x}) = \begin{cases} 1 & |\mathbf{x}| < 1 \\ 0 & \text{sonst} \end{cases}$



Hypertextures

Ray Marching (opake Objekte)

- ▶ gehe in kleinen Schritten entlang des Strahls
- ▶ teste: wenn $D(\mathbf{x}) = 1$, dann Schnittpunkt gefunden
- ▶ viele verschiedene Optimierungen
 - ▶ Anpassung der Schrittweite
 - ▶ binäre Suche, etc.



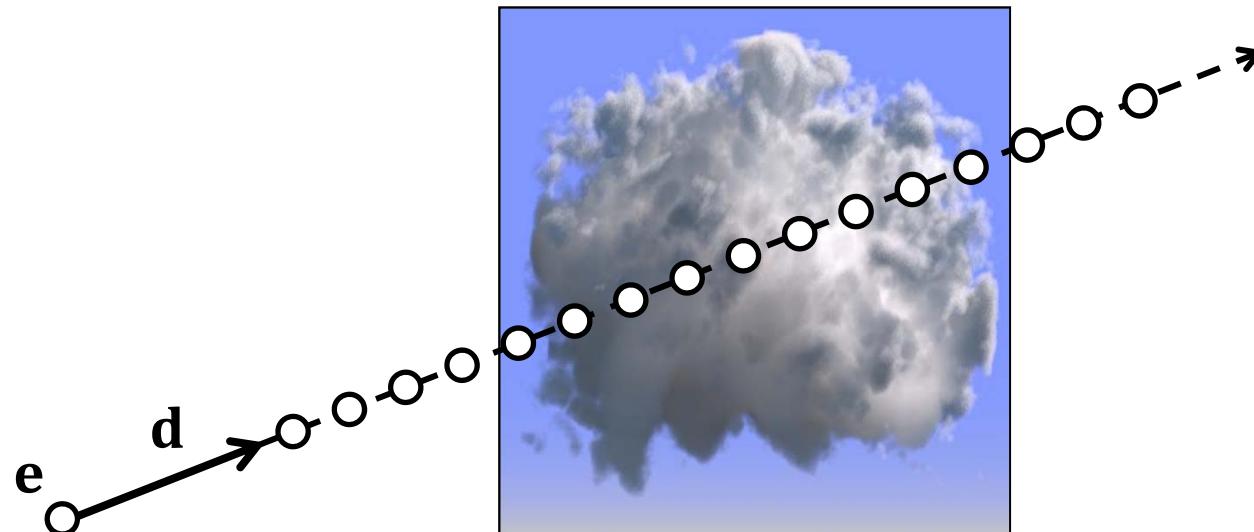
Hypertextures

Ray Marching (semitransparente Objekte)

- ▶ akkumuliere Opazität entlang des Strahls

$$\blacktriangleright D(\mathbf{x}) = \text{sphere} \left(\mathbf{x} \cdot \left(1 + \sum_i 1/2^i n(2^i f\mathbf{x}) \right) \right)$$

$$\blacktriangleright \text{mit } \text{sphere}(\mathbf{x}) = \begin{cases} 1 & |\mathbf{x}| < 1 \\ \max(0, 2 - |\mathbf{x}|) & \text{sonst} \end{cases}$$



Hypertextures



Ray Marching (semitransparente Objekte)

- ▶ akkumuliere Opazität entlang des Strahls

$$\blacktriangleright D(\mathbf{x}) = \text{sphere} \left(\mathbf{x} \cdot \left(1 + \sum_i 1/2^i n(2^i f\mathbf{x}) \right) \right)$$

$$\blacktriangleright \text{mit } \text{sphere}(\mathbf{x}) = \begin{cases} 1 & |\mathbf{x}| < 1 \\ \max(0, 2 - |\mathbf{x}|) & \text{sonst} \end{cases}$$

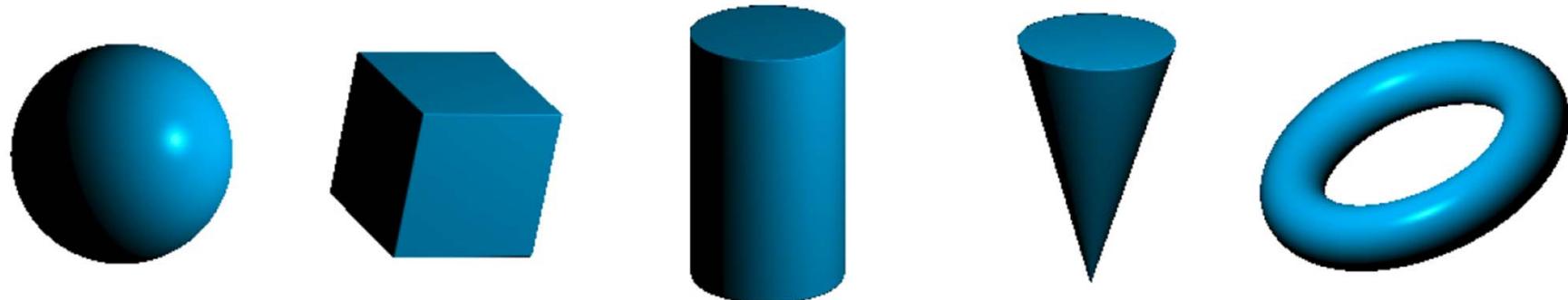
- ▶ DF können durch Modulationsfunktionen verändert werden
 - ▶ MFs sind beispielsweise bias, gain, noise, abs, sin
 - ▶ Hypertextur:

$$H(D(\mathbf{x}), \mathbf{x}) = f_n(\dots f_2(f_1(f_0(D(\mathbf{x}), \mathbf{x}))))$$

Distanzfelder, Distanzfunktionen

Implizite Darstellung von soliden Objekten

- ▶ **Distanzfunktion** $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$, die für jeden Punkt \mathbf{x} die kürzeste Entfernung zur Oberfläche $f(\mathbf{x}) = 0$ liefert
 - ▶ man spricht auch von der **Isofläche** für den Wert 0
- ▶ einfache Objekte, durch Distanzfunktionen beschreibbar
 - ▶ z.B. Kugel $f(\mathbf{x}) = |\mathbf{x} - \mathbf{m}| - r$ (Mittelpunkt \mathbf{m} , Radius r)

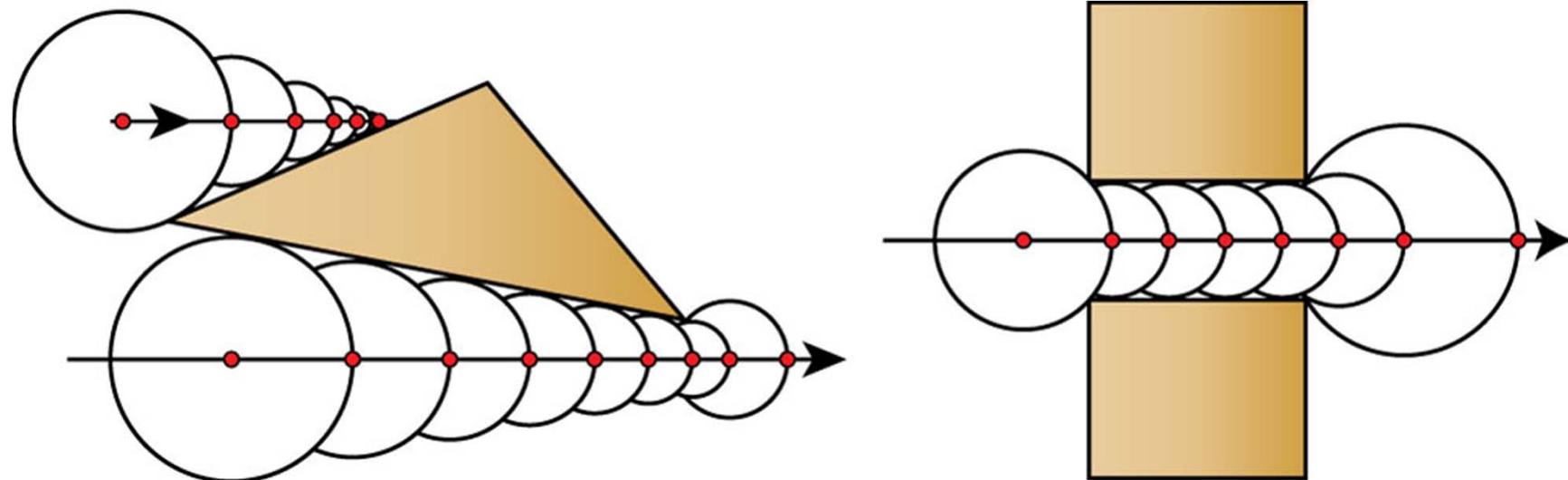


- ▶ idealerweise ist Abstand vorzeichenbehaftet, z.B. $f(\mathbf{x}) < 0$ im Inneren
- ▶ wird $f(\mathbf{x})$ diskret gespeichert (z.B. in 3D Textur) dann spricht man von einem **Distanzfeld**

Distanzfelder, Distanzfunktionen

Schnittpunktberechnung mit Sphere Tracing

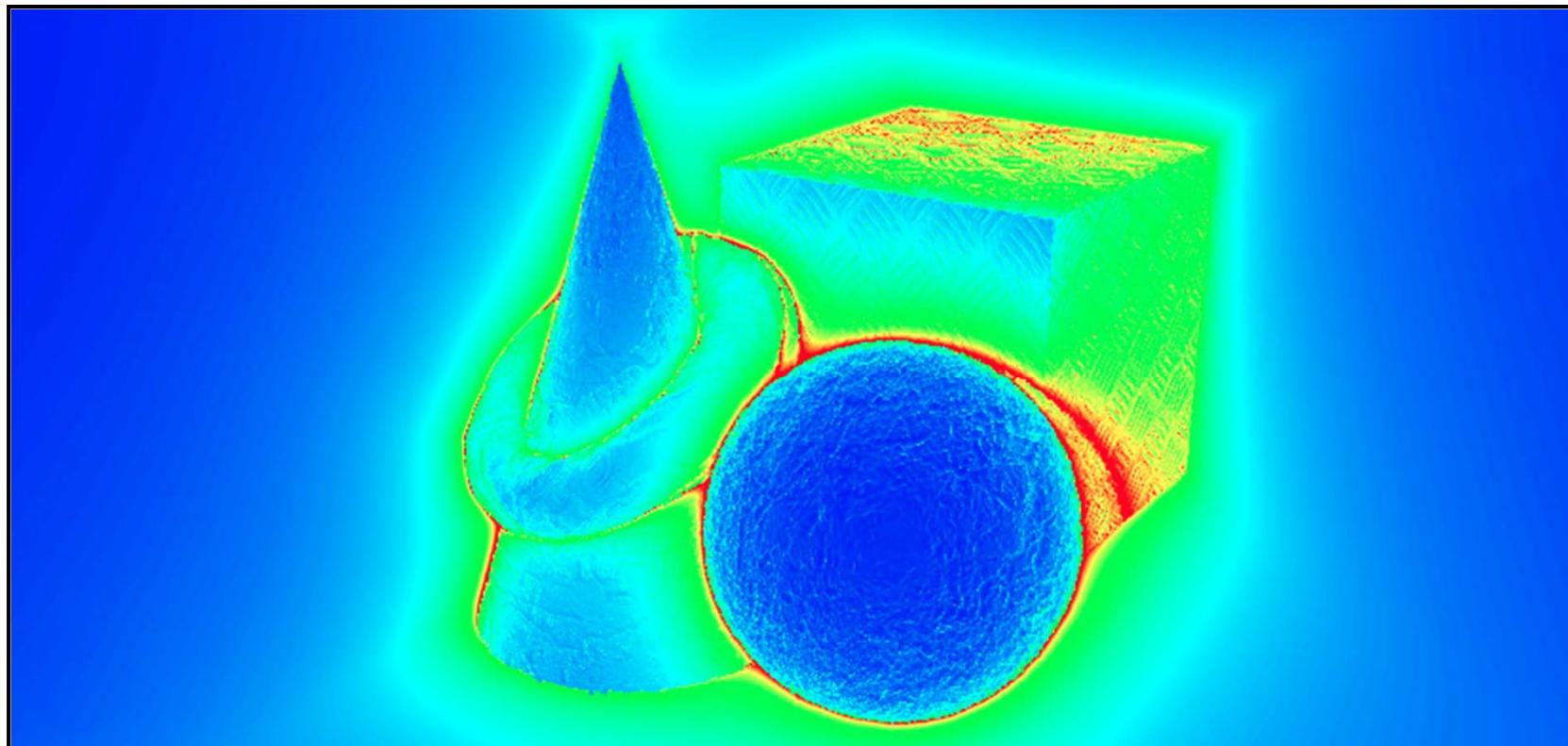
- ▶ **Sphere Tracing:** Finden des Schnittpunkts durch Ray Marching *und* Ausnutzen der Distanzfunktion/-felds für die Schrittweite
 - ▶ $f(\mathbf{x})$ liefert die kürzeste Entfernung zur nächsten Oberfläche
 - ▶ man kann immer eine Strecke $f(\mathbf{x})$ zurücklegen, ohne einen Schnittpunkt zu verpassen



Distanzfelder, Distanzfunktionen

Schnittpunktberechnung mit Sphere Tracing

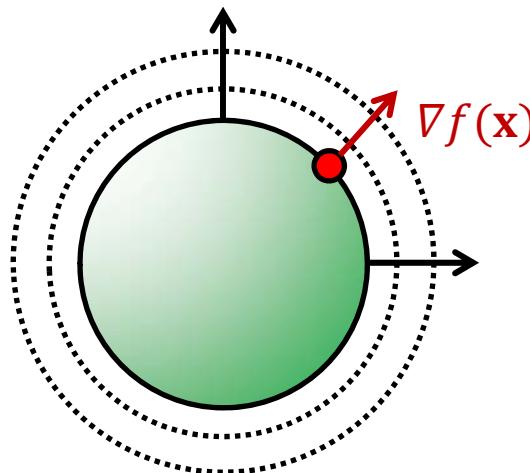
- ▶ Sphere Tracing: Finden des Schnittpunkts durch Ray Marching *und* Ausnutzen der Distanzfunktion/-felds für die Schrittweite
 - ▶ mehr Schritte für Strahlen, die Objekte knapp verpassen



Normale und Beleuchtungsberechnung

- ▶ der Gradient von $f(\mathbf{x})$ zeigt in die Richtung, in der die Distanz (also der Funktionswert) am schnellsten zunimmt
- ▶ auf der Fläche entspricht diese Richtung dem Normalenvektor
- ▶ Approximation des Gradienten über zentrale Differenzen:

$$\nabla f(\mathbf{x}) \approx \begin{bmatrix} f(x + \frac{1}{2}h, y, z) - f(x - \frac{1}{2}h, y, z) \\ f(x, y + \frac{1}{2}h, z) - f(x, y - \frac{1}{2}h, z) \\ f(x, y, z + \frac{1}{2}h) - f(x, y, z - \frac{1}{2}h) \end{bmatrix}$$



Verknüpfung von Distanzfunktionen



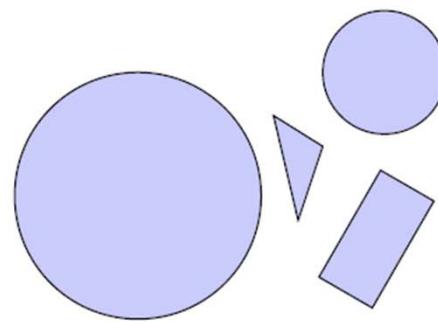
Constructive Solid Geometry (CSG)

- ▶ Boolesche Operatoren für Objekte

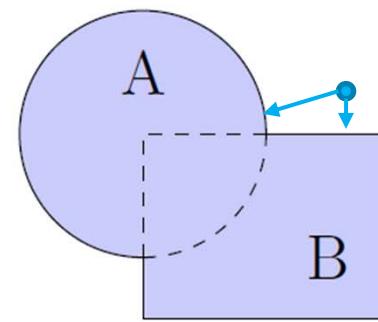
- ▶ Vereinigung: $f_{A \cup B}(\mathbf{x}) = \min(f_A(\mathbf{x}), f_B(\mathbf{x}))$

- ▶ Schnitt: $f_{A \cap B}(\mathbf{x}) = \max(f_A(\mathbf{x}), f_B(\mathbf{x}))$

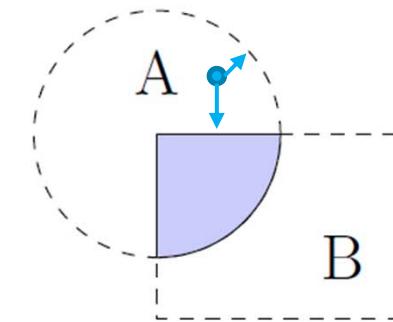
- ▶ Differenz: $f_{A \setminus B}(\mathbf{x}) = f_{A \cap -B}(\mathbf{x}) = \max(f_A(\mathbf{x}), -f_B(\mathbf{x}))$



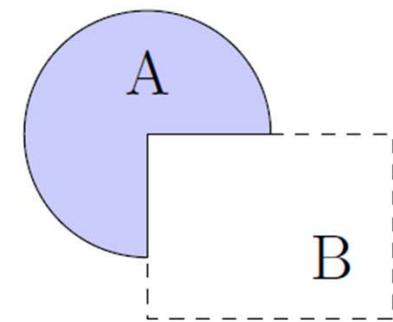
Vereinigung



$A \cup B$



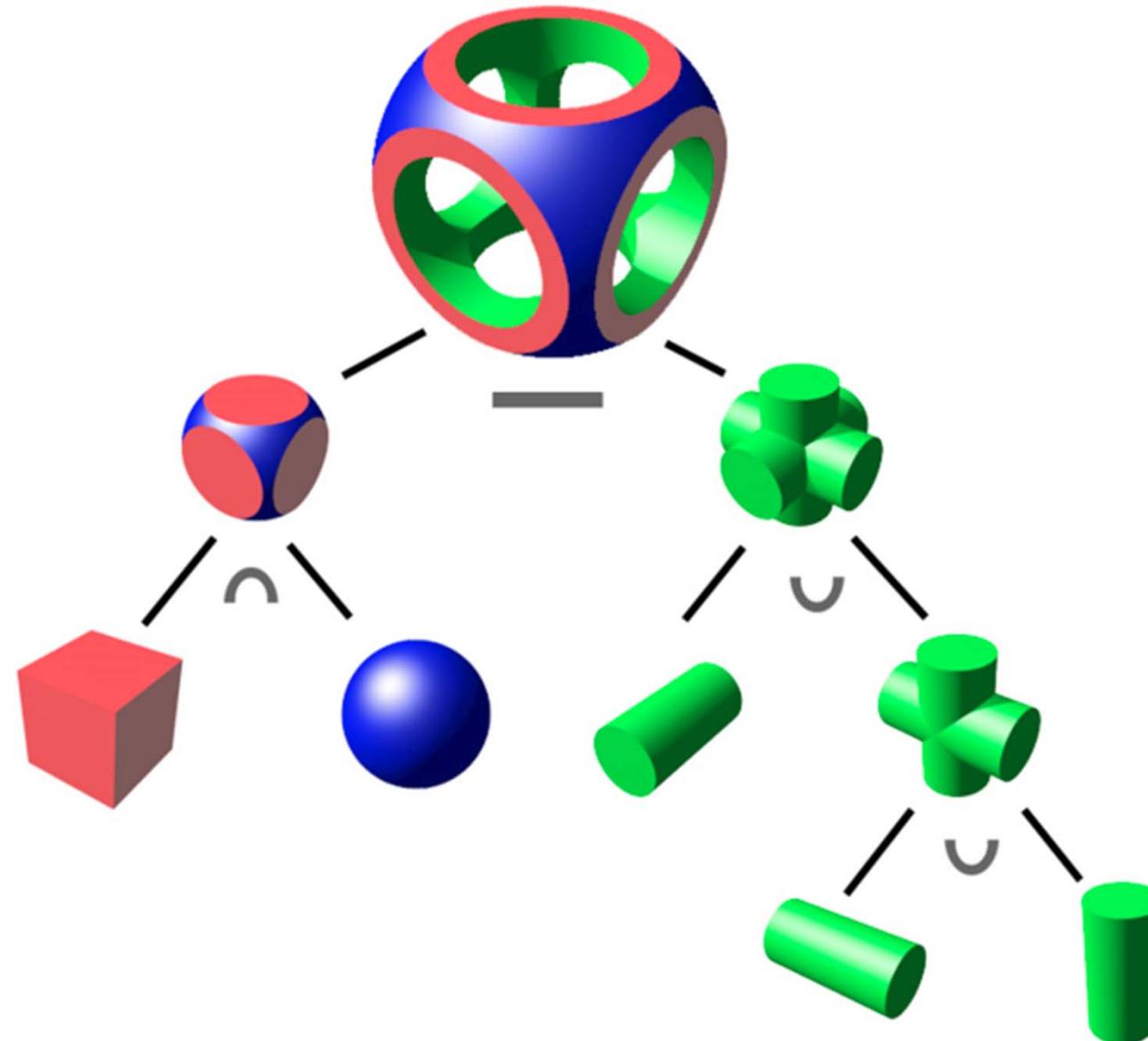
$A \cap B$



$A \setminus B$

- ▶ CSG ist auch mit anderen Repräsentationen, z.B. Dreiecksnetzen, möglich, aber u.U. aufwendiger
- ▶ CSG kann auch mit dem Stencil Buffer im Bildraum berechnet werden!

Constructive Solid Geometry



Ray Marching von DFs



► <http://iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm>



Ray Marching von DFs



► <http://iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm>



Vegetation

- manuelle Erstellung der Vegetation? 10^8 Dreiecke!



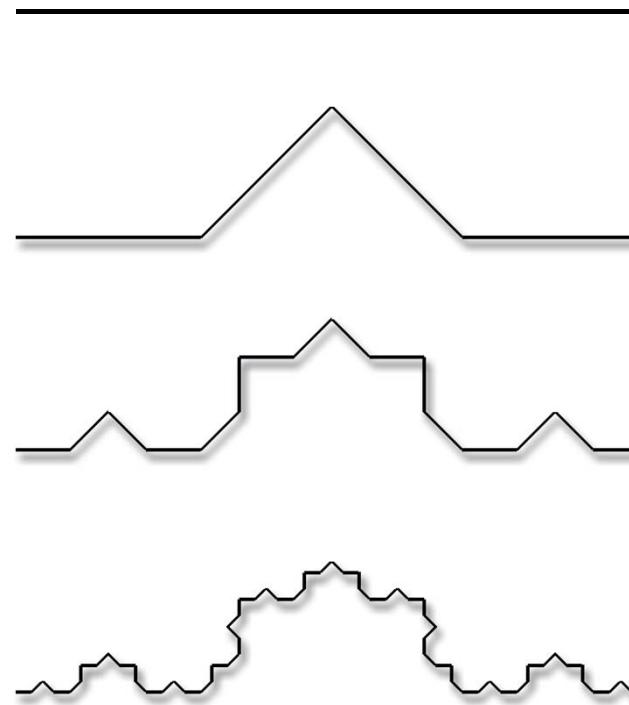
Lindenmayer-Systeme, L-Systeme



- ▶ theoretisches Modell für biologische Entwicklung und Morphogenese (= u.a. Entwicklung von Organismen)
 - ▶ nach Aristid Lindenmayer, 1968
 - ▶ basiert auf formalen Grammatiken
 - ▶ ursprünglich entworfen für die Beschreibung der Entwicklung von mehrzelligen Organismen
 - ▶ später: Erweiterung auf Pflanzen und Strukturen mit Verzweigungen
- ▶ Grundidee:
definiere ein komplexes Objekt durch sukzessives Ersetzen von Teilen eines einfacheren Objekts

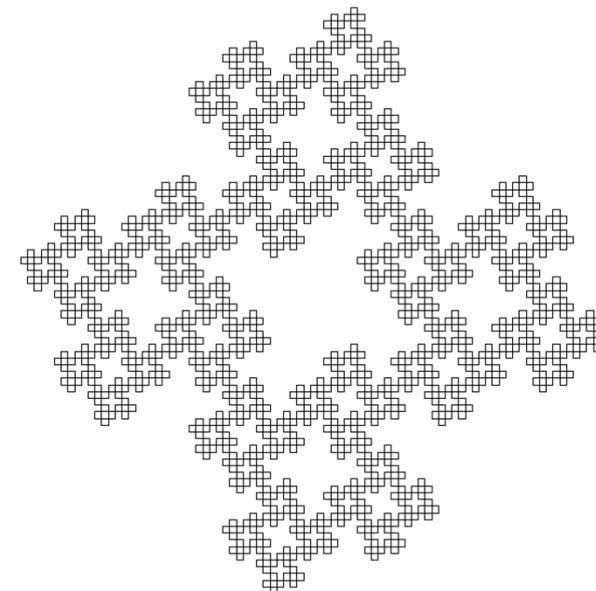
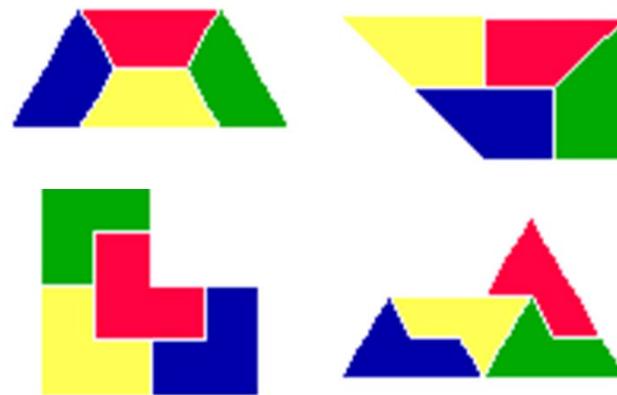
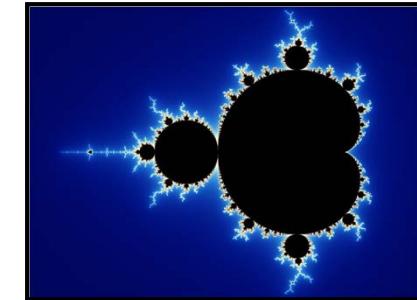
Selbstähnlichkeit

- ▶ Bsp. Kochsche Kurve
 - ▶ beginnt mit einem Liniensegment
 - ▶ in jeder Iteration: ersetze Linie durch 



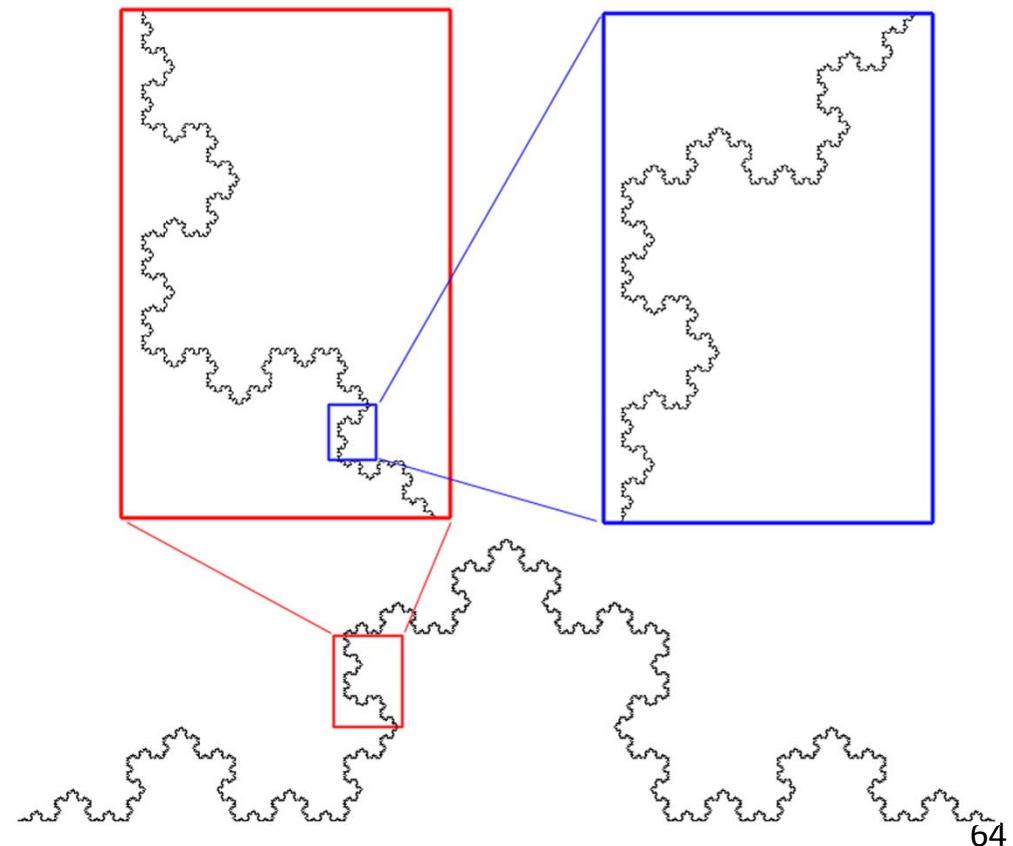
Selbstähnlichkeit

- ▶ “When a piece of a shape is geometrically similar to the whole, both the shape and the cascade that generate it are called self-similar”
(Benoît Mandelbrot, 1982)
- ▶ Fraktal (nach Mandelbrot): geometrisches Muster/Gebilde mit hohem Grad an Selbstähnlichkeit
- ▶ die rekursive Natur von L-Systemen (sukzessives Ersetzen) führt zu Selbstähnlichkeiten → fraktale Gebilde lassen sich daher gut mit L-Systemen beschreiben



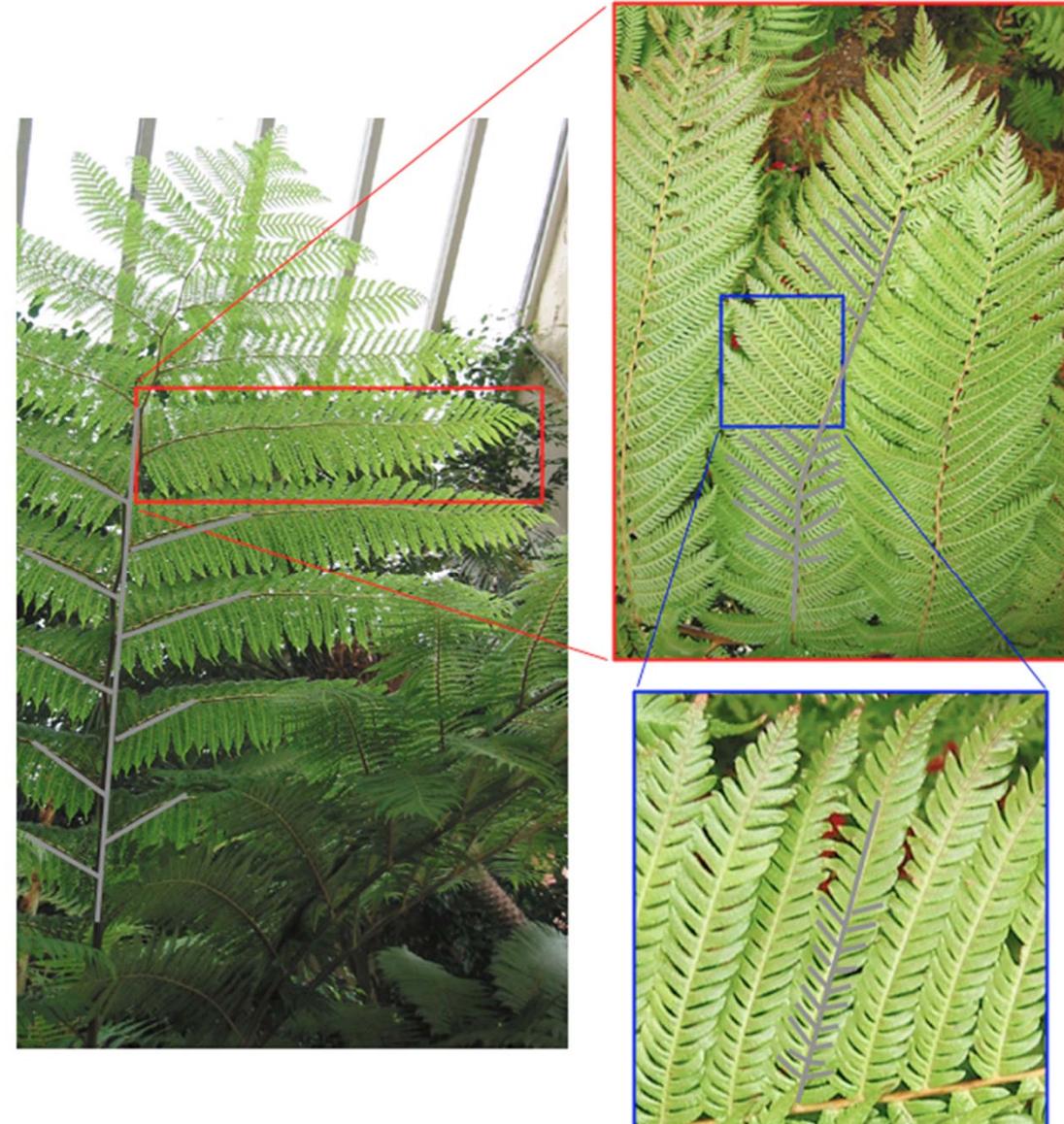
Selbstähnlichkeit in Fraktalen

- ▶ “When a piece of a shape is geometrically similar to the whole, both the shape and the cascade that generate it are called self-similar”
(Benoît Mandelbrot, 1982)
- ▶ Fraktal (nach Mandelbrot): geometrisches Muster/Gebilde mit hohem Grad an Selbstähnlichkeit
- ▶ Bsp. Kochsche Kurve
 - ▶ Selbstähnlichkeit ist hier exakt
 - ▶ die Form findet sich bei Vergrößerung immer wieder



Selbstähnlichkeit in der Natur

- ▶ ... ist nicht exakt
- ▶ tritt nur auf wenigen diskreten Skalen auf (hier 3)
- ▶ Selbstähnlichkeit bei Pflanzen ist ein Resultat der Wachstumsprozesse (Mandelbrot, 1982)



L-Systeme sind Ersetzungssysteme



- ▶ Grundidee: definiere komplexe Objekte durch sukzessives Ersetzen von Teilen eines einfacheren Objekts durch Ersetzungsregeln
- ▶ L-Systeme arbeiten wie **Chomskys formale Grammatiken**
- ▶ ein L-System ist definiert durch ein Quadrupel $G = (V, \Sigma, S, P)$
 - ▶ Alphabet (Zeichen) V
 - ▶ Terminalsymbole (Zeichen) $\Sigma \subset V$
 - ▶ Startwort/-symbol $S \in V^*$
 - ▶ Produktionsregeln $P \subset (V^* \setminus \Sigma^*) \times V^*$
 - ▶ **wichtiger Unterschied:** Produktionsregeln werden parallel angewendet, um die biologischen Prozesse nachzubilden
- ▶ ein L-System alleine macht noch kein geometrisches Objekt!
 - ▶ dazu wird noch eine Interpretationsvorschrift benötigt

D0L-Systeme



Beispiel für ein D0L-System (kontext-frei und deterministisch)

- $G = (V, \Sigma, S, P)$ mit
 - Alphabet $V = \{a, b\}$
 - Startsymbol $S = b$
 - Produktionsregeln $P = \{a \mapsto ab, b \mapsto a\}$
- Ableitung (5 Ersetzungsschritte, vorher festgelegt)
 - b
 - a
 - ab (jetzt beide Produktionsregeln gleichzeitig anwenden...)
 - aba
 - $abaab$
 - $abaababa$

Grafische Interpretation: Turtle-Grafik



- ▶ Schildkröte ist ein Cursor mit Position (x, y) und Richtung α
 - ▶ Stift, der auf und ab bewegt werden kann, Farbe kontrollierbar
 - ▶ wird eine Schrittweite d und ein Winkelinkrement δ vorgegeben, dann kann die Schildkröte mit wenigen Kommandos gesteuert werden

Typische Turtle-Kommandos

- ▶ F Vorwärtsbewegung mit Länge d
 - ▶ neue Position: (x', y') mit $x' = x + d \cos \alpha$ und $y' = y + d \sin \alpha$
 - ▶ gezeichnetes Liniensegment von (x, y) nach (x', y')
- ▶ f Vorwärtsbewegung mit Länge d ohne eine Linie zu zeichnen
- ▶ $+$ Rechtsdrehung um den Winkel δ
 - ▶ neue Orientierung: $\alpha' = \alpha + \delta$
- ▶ $-$ Linksdrehung um den Winkel δ
 - ▶ neue Orientierung: $\alpha' = \alpha - \delta$



Turtle-Grafik

Beispiel

► $G = (V, \Sigma, S, P)$ mit

► Alphabet

$$V = \{F, +, -\}$$

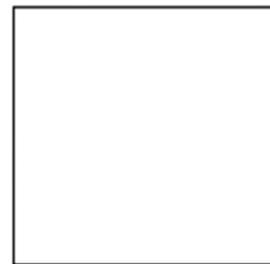
► Startwort

$$S = F + F + F + F$$

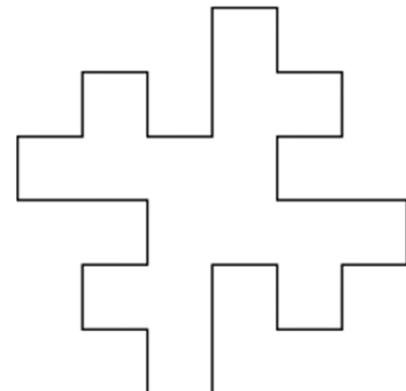
► Produktionsregel

$$P = \{F \mapsto F + F - F - FF + F + F - F\}$$

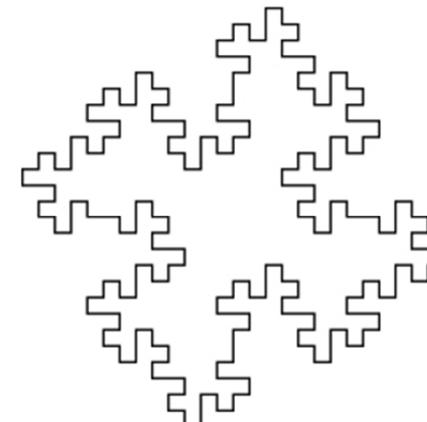
► $\alpha = 90^\circ$



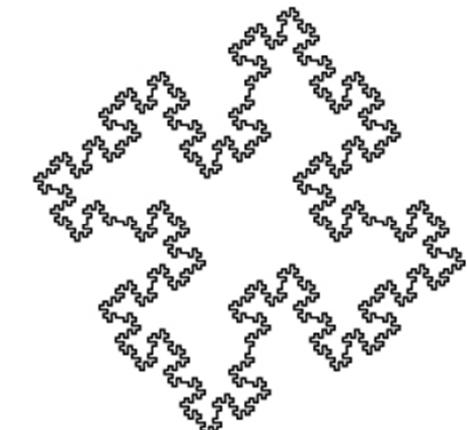
$n = 0$



$n = 1$



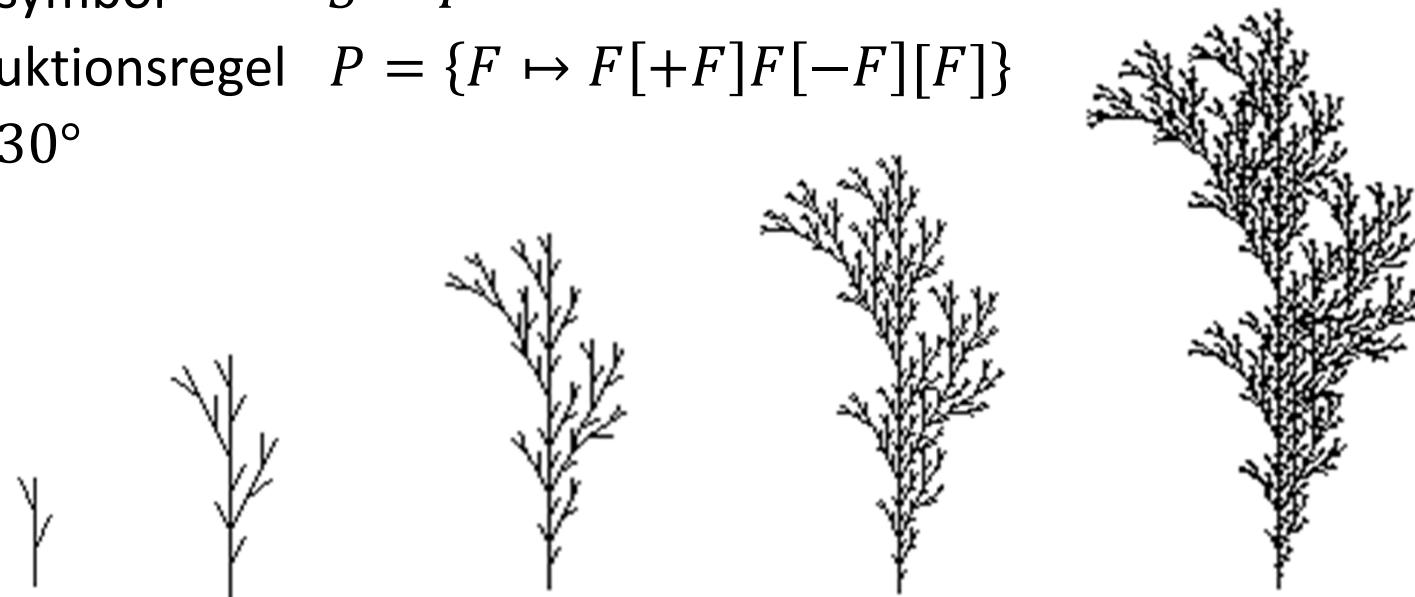
$n = 2$



$n = 3$

L-Systeme mit Verzweigungen

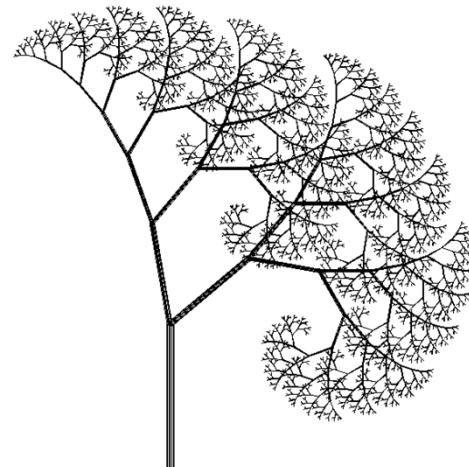
- ▶ um Verzweigungen darzustellen wird das Alphabet um zwei Symbole erweitert: „[“ und „]“
 - ▶ „[“ push: sichert den aktuellen Zustand (x, y, α) auf einem Stack
 - ▶ „]“ pop: holt den letzten gesicherten Zustand vom Stack (dabei wird keine Linie gezeichnet)
- ▶ Beispiel:
 - ▶ Alphabet $V = \{F, +, -, [,]\}$
 - ▶ Startsymbol $S = F$
 - ▶ Produktionsregel $P = \{F \mapsto F[+F]F[-F][F]\}$
 - ▶ $\alpha = 30^\circ$



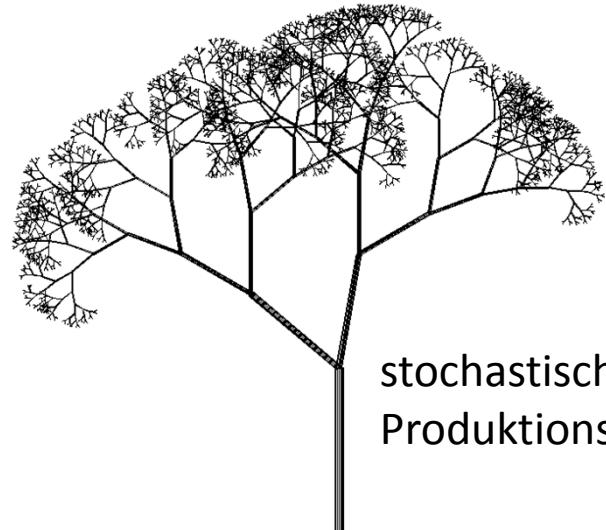
L-Systeme: Erweiterungen (siehe Bonus Material)



- ▶ 3D, parametrisierte Symbole (links-oben) und Mutationen (rechts),
stochastische Produktionsregeln (links-unten)



parametrisierte Symbole



stochastische
Produktionsregeln



$F[+F]+[+F-F-F]-FF[-F-F]$



$FF[+FF][-F+F][FFF]F$



$F[+F]+[+F-F-F]-F[-F][-F-F]$



$FF[+FF][-F+F][-F]F$

Ausblick: 3D L-Systeme mit Verzweigungen

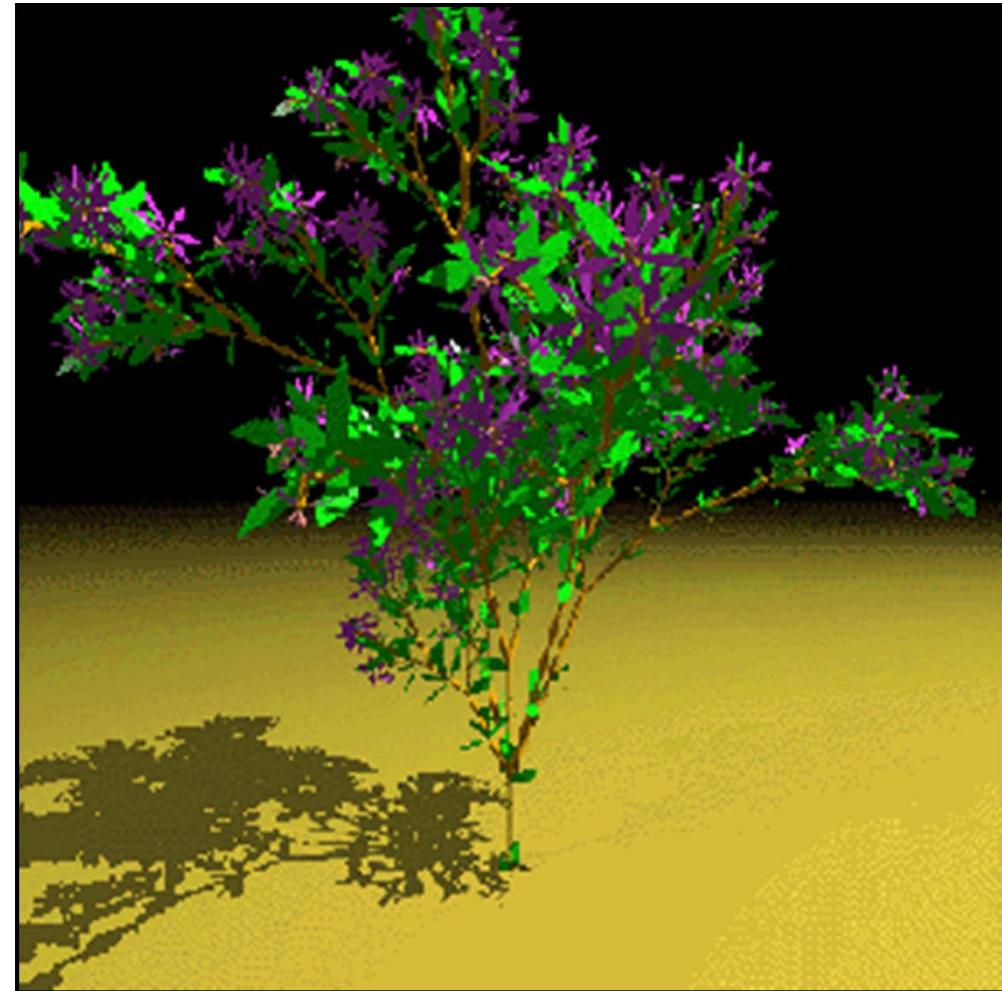


- ▶ The Algorithmic Beauty of Plants von Przemyslaw Prusinkiewicz und Aristid Lindenmayer
- ▶ Buch zum Download: <http://algorithmicbotany.org/papers/#abop>



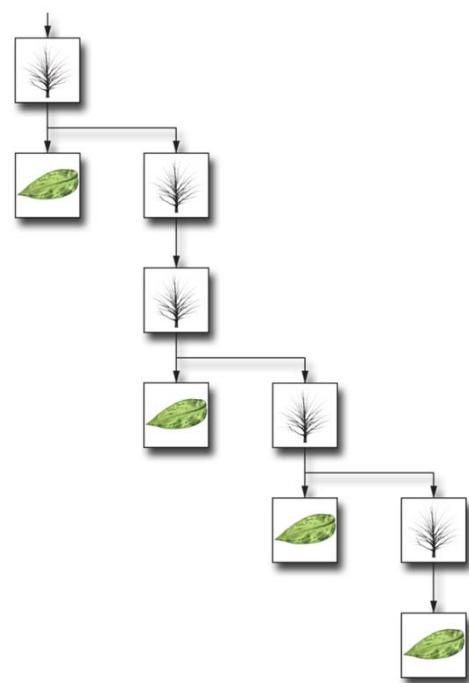
$n=7, \delta=22.5^\circ$

$\omega : A$
 $p_1 : A \rightarrow [&FL!A]////// [&FL!A]//////// [&FL!A]$
 $p_2 : F \rightarrow S // F$
 $p_3 : S \rightarrow F\ L$
 $p_4 : L \rightarrow [, , \wedge \wedge \{-f+f+f-f-| -f+f+f\}]$

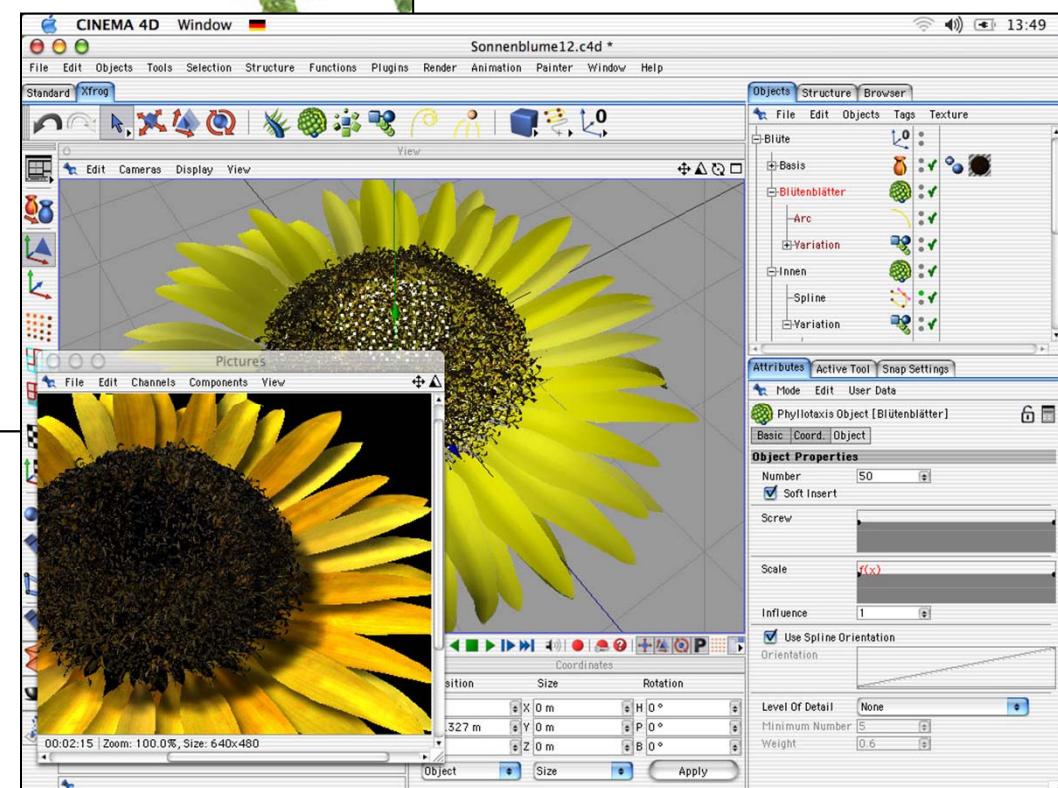
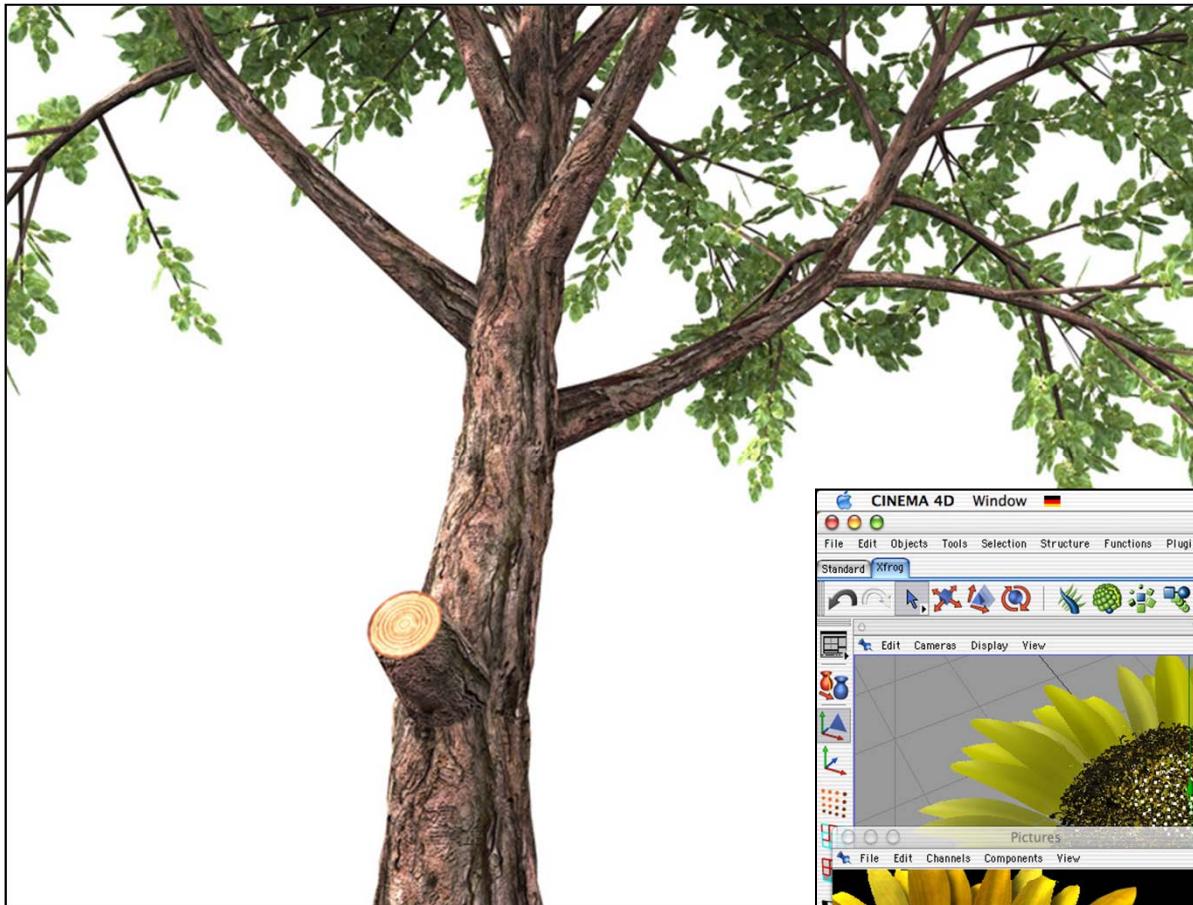


L-Systeme in der Praxis

- ▶ rein algorithmische Erzeugung bestimmter Pflanzen(teile), z.B. Knospenanordnung, ist manchmal unflexibel und erfordert viel Übung
- ▶ deshalb verfolgt man oft den Ansatz
 - ▶ parametrisierte, stochastische Ersetzungsregeln
 - ▶ Symbole mit komplexer Bedeutung
z.B. Symbol erzeugt Geometrie für ein vollständiges Blatt oder Blüte



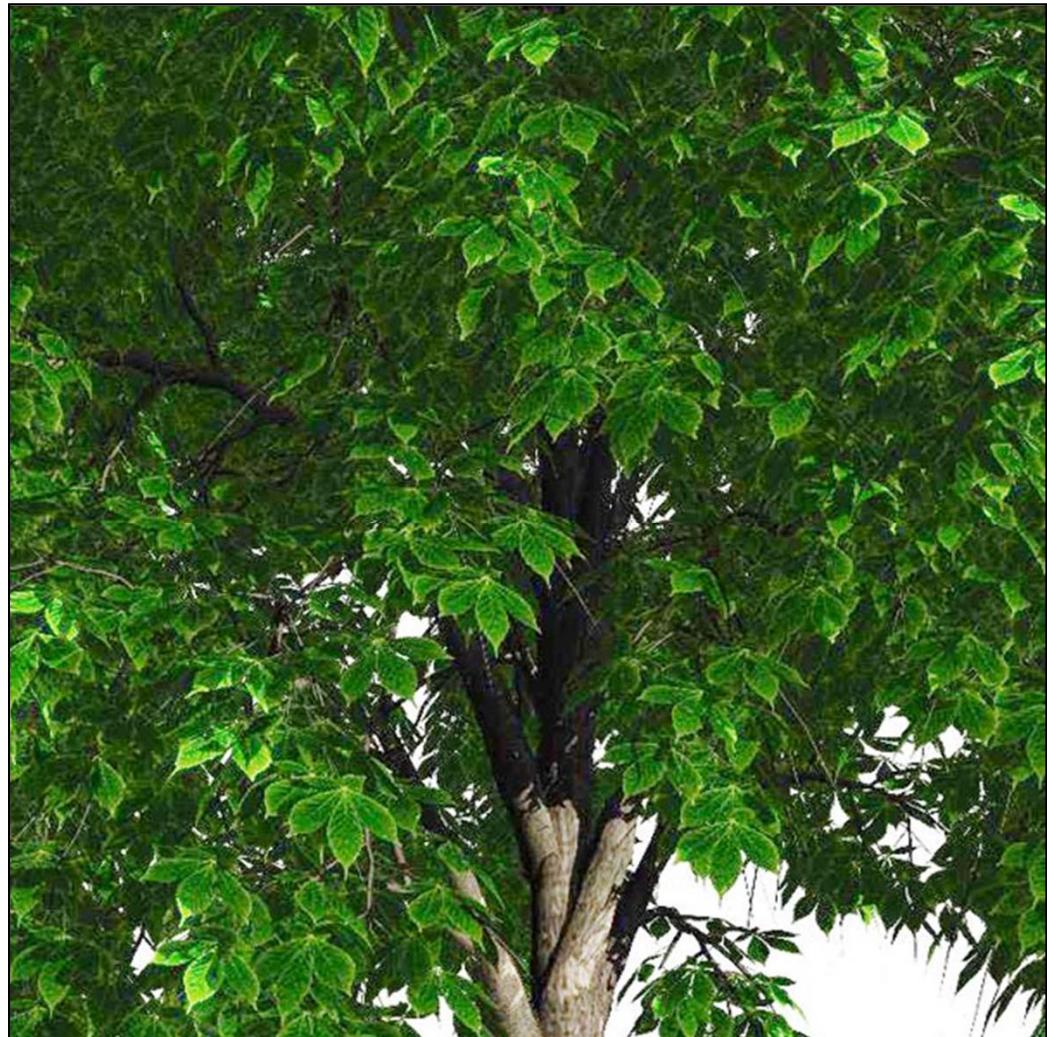
Xfrog



Xfrog



► Bilder: www.xfrog.com



Ausblick: aktuelle Forschungsthemen



Fazit

- ▶ prozedurale Modellierung ist ein mächtiges Werkzeug...
 - ▶ ...manchmal aber schwer zu kontrollieren
 - ▶ zunehmend wichtiger: die Menge des grafische Inhalts von Spielen, Filmen etc. steigt schnell
 - ▶ kompakte Beschreibung, ideale Form der „Datenkompression“



$n=7, \delta=22.5^\circ$

$\omega : A$
 $p_1 : A \rightarrow [&FL!A]/////' [&FL!A]////////' [&FL!A]$
 $p_2 : F \rightarrow S ///// F$
 $p_3 : S \rightarrow F L$
 $p_4 : L \rightarrow ['''\wedge\{-f+f+f-|-f+f+f\}]$